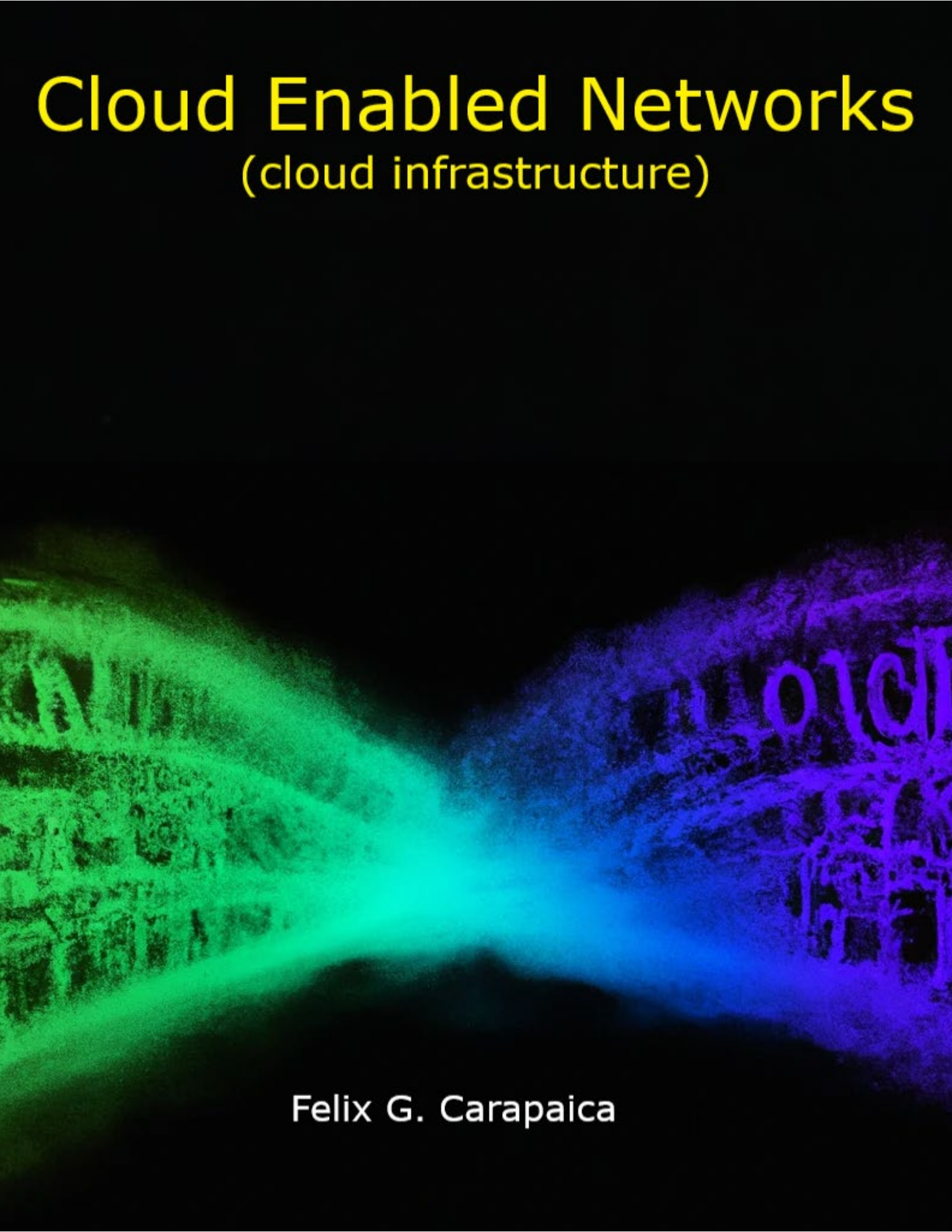
Cloud Enabled Networks

(cloud infrastructure)



Felix G. Carapaica

ISBN 978-0-99-702549-1

9 780997 025491 >

ISBN: 978-0-9939150-4-8

TELE 20483 Cloud Enabled Networks.

Felix G. Carapaica G.

Sheridan College, Oakville, Ontario, Canada.

Table of Contents

Learning Outcomes

- Determine the use cases for the different cloud models infrastructure, software, platform, on-prem and hybrid.
- Compare different cloud offerings.
- Describe the organization and general structure of a cloud provider.
- Evaluate the process and benefits of migrating services to a cloud provider.
- Write and run basic scripts to deploy resources and to retrieve resource information from the cloud environment.
- Deploy elastic virtual computers in a cloud networking environment.
- Configure webservers on elastic virtual computers.
- Correlate protocol encapsulation to security groups and network control access lists.
- Explain the methods of secure access to cloud virtual machines (instances).
- Configure elastic data storage in the cloud.
- Implement virtual endpoints to access cloud storage and other resources.
- Configure databases on elastic virtual computers.
- Configure webserver with databases on a LAMP stack.
- Configure elastic load balancers to distribute workload traffic.
- Deploy and scale a basic web application on Platform as a Service.
- Design and deploy virtual private clouds.
- Plan IPv4 and IPv6 address allocation for virtual private clouds.
- Configure IPv4 address subnetting in a cloud environment.
- Enable controlled public access to network resources and services.
- Configure private networks in a cloud.
- Configure virtual gateways to access resources in the cloud.
- Architect an organization in a cloud environment.

Chapter 1

Introduction to Cloud Computing

Description

The purpose of this section is to provide the student with the fundamentals of cloud reference models. There are four main models: Software as a Service, Platform as a Service, Infrastructure as a Service, On-Premises and one mixed or hybrid model. These are the principal models although there are many more combinations for particular cases. Knowing the classification and features of each cloud offering is vital to make decisions about cloud utilization and migration of services.

Learning Outcomes

- Determine the use cases for the different cloud models infrastructure, software, platform, on-prem and hybrid.
- Compare different cloud offerings.
- Describe the organization and general structure of a cloud provider.
- Evaluate the process and benefits of migrating services to a cloud provider

Main concepts

- Definition of cloud.
- Characteristics of cloud.
- The functional layers of the cloud model.
- Software as a Service.
- Platform as a Service.
- Infrastructure as a Service.
- On-premises.
- Hybrid model.

Activities

- Classify a list of services according to the cloud models.
- Case study: Sheridan College migration of services to the cloud.

Definition of Cloud

A cloud is a collection of computing resources available, on demand, over the Internet. These computing resources include compute, storage, databases, software, monitoring, management, analytics, networking and more. To access cloud resources, the consumer is presented with an interface that abstracts the details of the cloud operation. This enables the consumers to interact with the cloud to request and to obtain resources in a flexible manner.

These are the fundamental [1] characteristics of a cloud:

- **On-demand self-service**. The consumer of a cloud service obtains computing, storage, processing, memory and networking resources without the direct intermediation of another person or service party.

- **Usage alternatives**. In a subscription mode, the client of the cloud pays for the resources to be used on pre-arranged fixed fee. This is an important concept for an organization because for the use of the cloud becomes an expected cost for the planning of the operational budget. The alternative to the subscription mode is the pay-per-use. In this usage mode, the customer uses cloud resources as the need arises while paying only for the amount of usage.

- **Broad network access**. Capabilities are available over a private network or over the Internet and they can be accessed from a client platform, typically a web-based interface.

- **Resource pooling**. The cloud provider makes pools of resources available to multiple consumers. Even though the resources are shared, they appear to be logically separated. The physical and virtual resources are dynamically assigned and released on demand. The exact physical location and configuration of the resources is mostly opaque to the customer.

- **Instant elasticity**. The capacity of the resources can be increased instantaneously to match demand conditions. To the consumer, the capabilities available appear to be unlimited and can be taken as needed.

- **Measured service**. The usage of cloud resources can be controlled and optimized by monitoring a metric pertaining to the type of service. For example, for a database, the number of inputs/outputs (IOPS) is a metric. For a webserver, the amount of traffic and the CPU utilization can be metrics. If the metric reaches a certain threshold, an action is triggered to increase or decrease capacity.

Cloud Service Reference Models

There are several reference models for cloud depending on different areas of interest or viewpoints. One of the scopes used as a refence framework is the depth of access to the cloud structure. The typical components of a cloud are organized in layers from the applications at the top of the model to networking at the bottom. The position in the model of the access boundary defines the type of service that the cloud offers. Consequently, this reference model is used to guide organizations regarding their use of cloud services. This is the general **cloud as a service** layered model:

Applications	A program designed to carry out a specific task or service.
Data	The set of information that is required by the application to provide a function.
Runtime	The environment that supports the execution of an application. It includes the software that supports the running of code in real time.
Middleware	This is software that provides services to applications. These services are not available on the underlaying operating system.
Operating Systems	Management of the compute resource. Linux family, Apple macOS, Microsoft Windows, Google's Android OS, Apple iOS, etc.
Virtualization	Hypervisors create virtual Instances of computer hardware platforms, storage devices, and computer network resource.
Servers	The hardware platform that supports everything on the upper layers.
Storage	The hardware and the software that contain and manage the storage of data.
Networking	The layer that contains all the resources that carry information across locations.

Functional Layers of the Cloud Reference Model

This model describes cloud as a service depending on the depth of access to the layered stack. A cloud model allows organizations to differentiate the core aspects of their operations from the ones which are not core to their strategy. For example, if an organization purchases an existing general-purpose application, then it does not need to delve deep into the layers of the model. However, if the same organization wishes to build and manage a core custom application, then it needs greater access to the layers in the model.

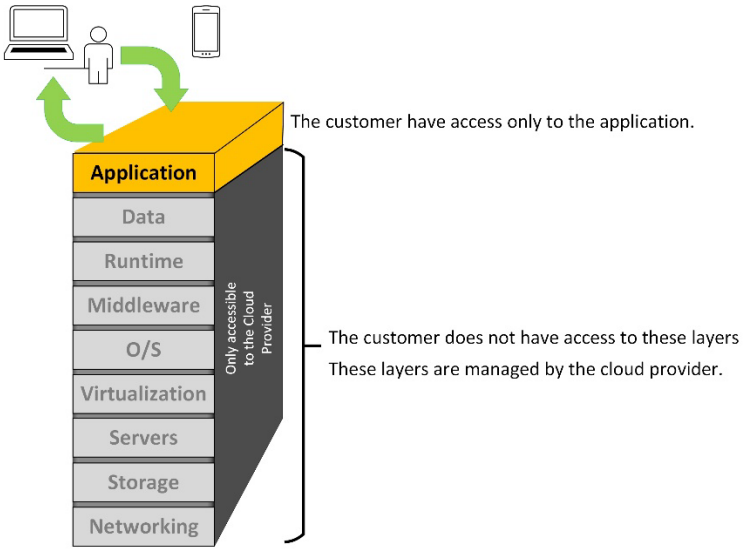
There are four general types of cloud [2] offering that stem from this model. They are:

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)
- On-Premise.

Then there are combinations of service profiles called Hybrid services.

Software as a Service (SaaS)

SaaS [2] is an application completely managed by the cloud provider. The customer is just a consumer of the application. Hence, it has no access to any of the cloud layers. SaaS is the most prevalent cloud offering with the most popular applications running on this model. Services such as Gmail, Teams, Zoom, Netflix, and Adobe Creative Cloud are Software as a Service.

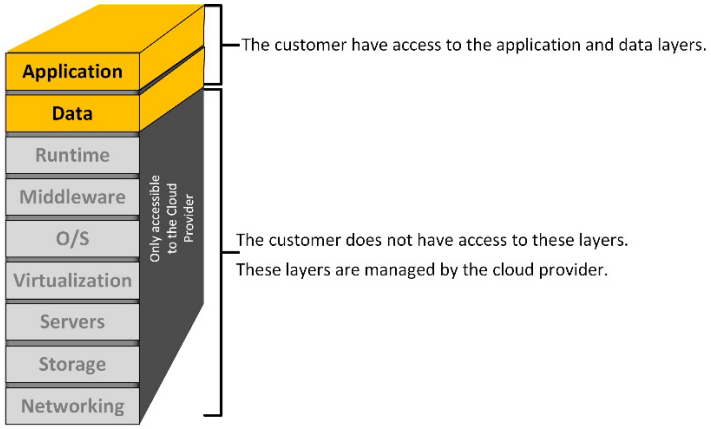


Software as a Service in the Cloud Layered Model

Software as a Service always amounts to an operating expense for an organization. There is nothing to build or maintain, instead that responsibility completely lays on the cloud service provider.

Platform as a Service (PaaS)

PaaS provides a complete development and deployment environment ready to be used. A PaaS cloud presents an interface to the customer with access to existing applications and related data. The interface hides the details of the platform.



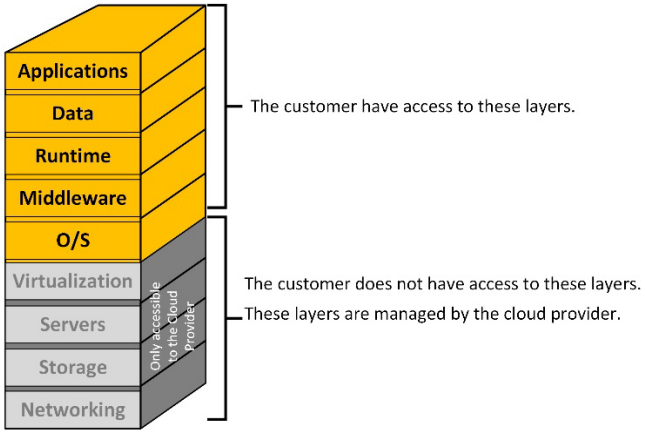
Platform as a Service in the Cloud Layered Model

The most common use case of this model is website hosting. For example, a customer who needs to deploy a webservice but does not want to deal with the technical details of deployment, operation, and infrastructure. The customer has no idea or even care how that works. PaaS provides the user with an application with easy to configure templates that enables the customization of the website and the upload of data such as pictures or videos. Services such as GoDaddy.com and Wix are examples of Platform as a Service.

Platform as a Service is heavily used by software developers too. The PaaS cloud fully manages the platform where to develop, test, and deploy applications. In this case, the main attractive of PaaS is that the apps can be delivered quickly and efficiently without the programmers being concerned with the underlay infrastructure that supports the application. The PaaS cloud handles all the orchestration of virtual servers and containers, their configuration, the network access, redundancy protection, the monitoring, the security and more. This sets the code developers free to just focus on coding, leaving the infrastructure to be taken care by the cloud. Examples of this service mode are Heroku and AWS Beanstalk.

Infrastructure as a Service (IaaS)

IaaS is the virtualization of a computing and network infrastructure inside a cloud service. The customers of this model deal with a configuration interface that allows them to create and maintain their own virtual infrastructure. The IaaS cloud providers manage large clusters of datacentres where resources are pooled together. These resources include compute, storage, security, appliances, networking and many more. The customers have access to such pool of resources on demand.



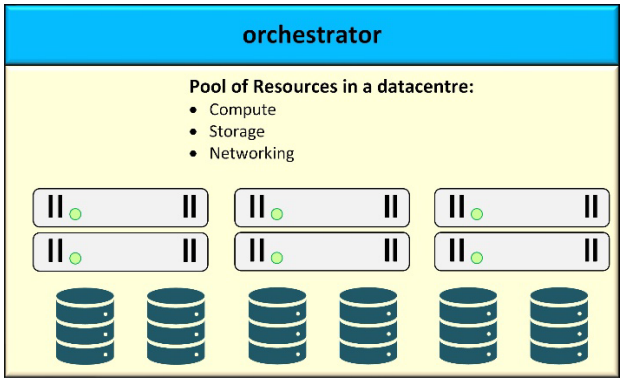
Infrastructure as a Service in the Cloud Layered Model

IaaS offers a pliable model where customers can replicate their current on-premises infrastructure inside an IaaS cloud provider, or they can migrate their current infrastructure to the cloud entirely or partially. With IaaS, the cloud customer has a great deal of freedom to develop its own computing platform that includes the networking, the security, the virtual resources of computing and storage and the degree of interconnection to the Internet. IaaS also enables the possibility of interconnecting private clouds in different providers enabling the concept of multicloud. For example, an organization could have resources in AWS, Azure, GCP and on premise, all interconnected via secure connections over the Internet or private links via service providers.

On-premises

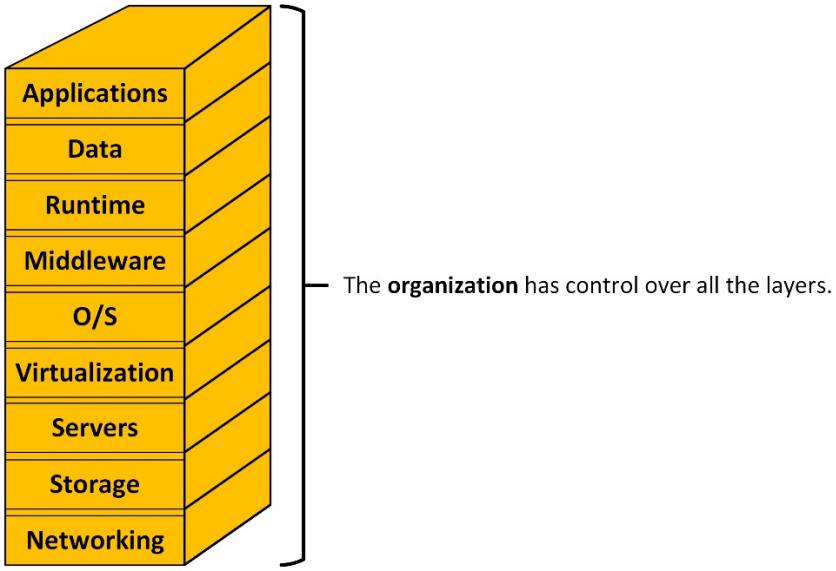
Cloud on-premises must not be confused with running a traditional datacentre. To be a cloud, the on-premises infrastructure must support an orchestration system that automatizes the deployment of services and instances elastically and on-demand from a shared pool of physical resources. For example, members of the organization can obtain computing and storage resources when they need it via a web interface. This is different than having servers running in a datacentre with one specific function.

An on-prem cloud involves a physical infrastructure that includes computers, storage servers, and network pulled together. This group of resources is managed by some orchestrator tool that deploys and configure virtual resources on top the physical resources. Examples of such tools are Openstack and VMware vCloud software.



A very simplified diagram for an on-prem cloud

The on-premise cloud implies that all the layers of the cloud model are under the control of the organization that owns the infrastructure which it is an IT department typically. It is necessary to clarify that the end-users of the on-prem cloud have limited access to the layers. Furthermore, the on-prem cloud can offer different access modes of software, platform and infrastructure within a company.



The On-prem Cloud Model

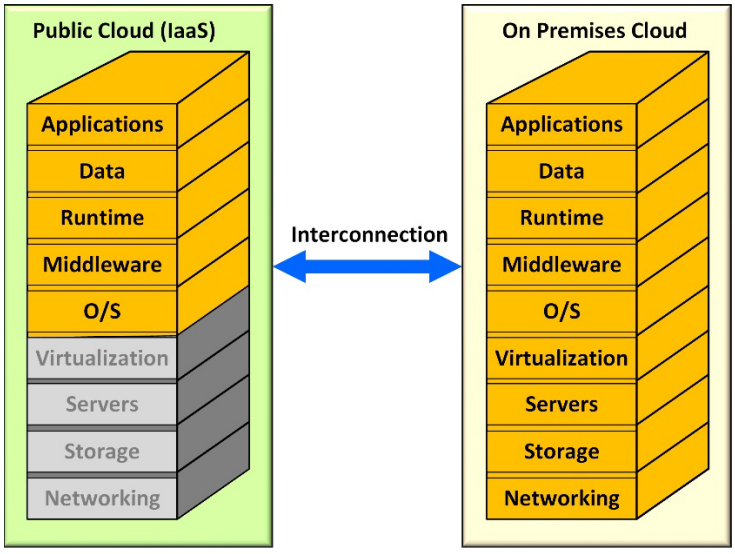
Hybrid cloud

The hybrid model, as the name indicates, is a mix between the Infrastructure as a Service and the On-premises models. There are several reasons that motivate this arrangement:

- An intermediate state in the migration to the cloud process.
- Keeping some services in-house while other services in the public cloud.
- Redundancy of services.
- Legal reasons.

The hybrid model is used for gradual migrations of resources from an organization into a cloud service. Once that the migration is complete, the in-house data centre is no longer required. However, sometimes some services can not be migrated. In such cases, a partial local data centre is maintained to keep the application running. In the case of data storage, an organization may choose to keep a replica of the data stored on-prem. In some cases, there might be laws that require that certain data must be kept local. Therefore, some services are moved into the cloud while some services remain behind.

The hybrid model requires a connection between the on-premises cloud datacenter and the cloud. This connection can be established over the public Internet using a secure channel (a VPN) or a dedicated, high data bit rate connection provided by a network carrier.



The Hybrid Cloud Model

Cloud Service Models Summary

The cloud models serve as a reference [2] when organizations need to make decisions regarding their use of cloud services. The cloud models help the organizations to separate the aspects of business that must be built and manage internally from the aspects that can be just contracted as a service. Business management is always trying to outsource services considered not to be part of the core mission of the organization. In this regard, cloud is seen as a business model where the internal talent focuses on the core strategy and the cloud providers concentrate in the operation and provisioning of the services and resources that the business needs. Essentially, this is all about specialization.

Process	Business Process Outsourcing (BPO)	Business Functions
Consume	Software as a Service (SaaS)	Application Data
Build	Platform as a Service (PaaS)	OS Middleware
Host	Infrastructure as a Service (IaaS)	Networking and Storage

Cloud as a business reference

SaaS [2,3] is the answer to consuming an available service or a ready to use application. Software as a Service is the answer when an organization sets a "cloud first" strategy. Basically, if the service exists, just pay to use it. In the SaaS model, the customer has nothing to set up. Everything is taken care of by the provider.

PaaS is for building applications without worrying about the details of the undelaying infrastructure. The most common usage of PaaS is building web sites. The customer only deals with the website looks and features while the cloud provider takes care of all the infrastructure.

IaaS is the usual profile chosen to host a service on a desired virtual infrastructure. This is common for enterprises with large and complex network infrastructure with diverse ways to access the resources. In such cases, the enterprise needs to keep the control of the logical topology and the type of resources present in the networks. The enterprise could do all that in a traditional datacentre; however, the cloud provides something different which is elasticity on demand. Resources can be added or removed dynamically when they are needed.

An on-premises model allows complete control of the stack. This implies that the organization has the talent and the resources to build its own cloud for either internal use or to provide a service to a customer.

Finally, there is the mixed format which is represented by the hybrid cloud model. This model enables enterprises to keep part of their business support in-house while another part is placed in the cloud. There are also cases with multi-cloud deployments where the organization IaaS is hosted in different cloud providers for reliability and strategic reasons.

SaaS	PaaS	IaaS	On-premises
Applications	Applications	Applications	Applications
Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware
O/S	O/S	O/S	O/S
Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

Summary of the different cloud models

Chapter 1 Coursework

Classify the following services according to the cloud models.

- Amazon Luna
- Cloudflare
- Google Domains
- Google Stadia
- GitHub
- Heroku
- Hashicorp Vault (vaultproject.io)
- JFrog
- Linode
- Netflix
- Nvidia GeForce Now
- Oracle PeopleSoft
- Presto card
- Squarespace
- D2L Brightspace (SLATE)
- Terraform Cloud
- Wix

Case study: Sheridan College migration of services to the cloud

For an organization, the decision to migrate from on-premises and data centres to the cloud requires a very serious study. An incorrect decision will result in high costs and loss of services. Every situation requires a previous analysis of the organization core functions and necessities. Once you graduate from college, it is likely that you will participate in this type of decision making as a technical consulting part.

The document [3] Sheridan College Cloud Computing Strategy v0.5 provides an excellent insight on this type of evaluation and the organization journey toward migrating resources and services to cloud. Read the document thoroughly and answer the following questions to participate in a class discussion.

- What are positive consequences of a successful cloud adoption strategies?
- What are negative consequences of an unsuccessful cloud adoption strategies?
- How can a cloud migration go wrong?
- What is the difference between the Cloud First strategy with respect Cloud Only?
- What is the cloud adoption strategy that the report recommends for Sheridan College?
- What are the characteristics of resources that can be successfully placed in the cloud?
- Sheridan College has two datacentres (Mississauga and Trafalgar) supporting services such as:
    - Sheridan's website hosting.
    - DNS name resolution.
    - Databases.
    - Registrar Office Systems.
    - Oracle PeopleSoft (Human Resource systems, policies, academics systems).

    - Voice over IP.
    - Wi-Fi services.
    - Video network services (Sheridan TV)
    - Custom applications (video editing, developer tools, etc).
- According to what these services do, which ones are more likely to be migrated to the cloud and what services are likely to stay in-house?
- Which ones would you say are the "good candidates to be migrated out to a cloud?
- What about the people who work supporting those systems? What will their role be?
- What does "competitively selected Cloud Platform of Record" mean in the paper?
- What are the services that Sheridan already have in clouds and what is the cloud model used for that?
- What are the benefits and disadvantages of?
    - On premise Enterprise IT application
    - IaaS
    - PaaS
    - SaaS
- What is the risk for Sheridan College when faculties or research units choose "to go their path to provisioning and supporting cloud services”?
- What is a goal, or at least a hope, of Sheridan College that cloud will bring about costs and budgeting? How does a cloud contribute to achieve this goal?

From the paper:

..."The migration of on-premise infrastructure to public cloud, and the divestment of responsibility that entails for hardware and software infrastructure, will over time (after migration and automation capabilities have been leveraged) allow technical team members to refocus on activities that provide a higher degree of value to the organization"...

- What would those activities be? Give examples.

The College plans to use as a principle: Optimize "Lift and Shift" Operations.

- If an application is already inefficient, will moving it to the cloud bring any benefit? What is the plan for those applications?
- How will the College protect "Continuity of Institutional Data?
- What does the report consider with respect applications that are: a) near end-of-life cycle, b) recently contracted?

In the Technical Criteria part of the report, in Application Utilization.

- How can a cloud scale an application up or down? Is it possible to do the same with a traditional service model, for example, physical servers?
- What are the changes expected for these IT roles?
    - System admins. database admins, developers, finance/assets management.

References

[1] Peter Mell, Timothy Grance. The NIST Definition of Cloud Computing, U.S National Institute of Standards and Technology. MD. 2011.

[2] IBM. IaaS vs. PaaS vs. SaaS. Understand and compare the three most popular cloud computing service models. Accessed. May 2021. [Online]. Available: https://www.ibm.com/cloud/learn/iaas-paas-saas. (2021)

[3] James Duncan. Sheridan College Cloud Computing Strategy v0.5. Oakville, 2020.

Chapter 2

Cloud Fundamental Structures

Description

The purpose of this section is to provide the student with a broad knowledge of the structures that compose clouds. It describes a cloud from simple to complex, starting with physical servers hosting virtual machines to tenant logical networks, datacentres, clusters of datacentres, regional groups of datacentres, to end with the global networks.

Learning Outcomes

- Describe the organization and general structure of a cloud provider.
- Execute basic scripts to retrieve resource information from the cloud environment.

Main concepts

- Fundamental Structure of clouds.
- Clouds made of datacentres.
- Resource virtualization.
- Sharing of physical resources.
- Logical isolation of tenants in the data centre.
- Overlay virtual interconnections.
- The elements of a cloud datacentre.
- Grouping of data centres into availability zones.
- Grouping of availability zones into regions.
- Global cloud networks.

Learning Activities

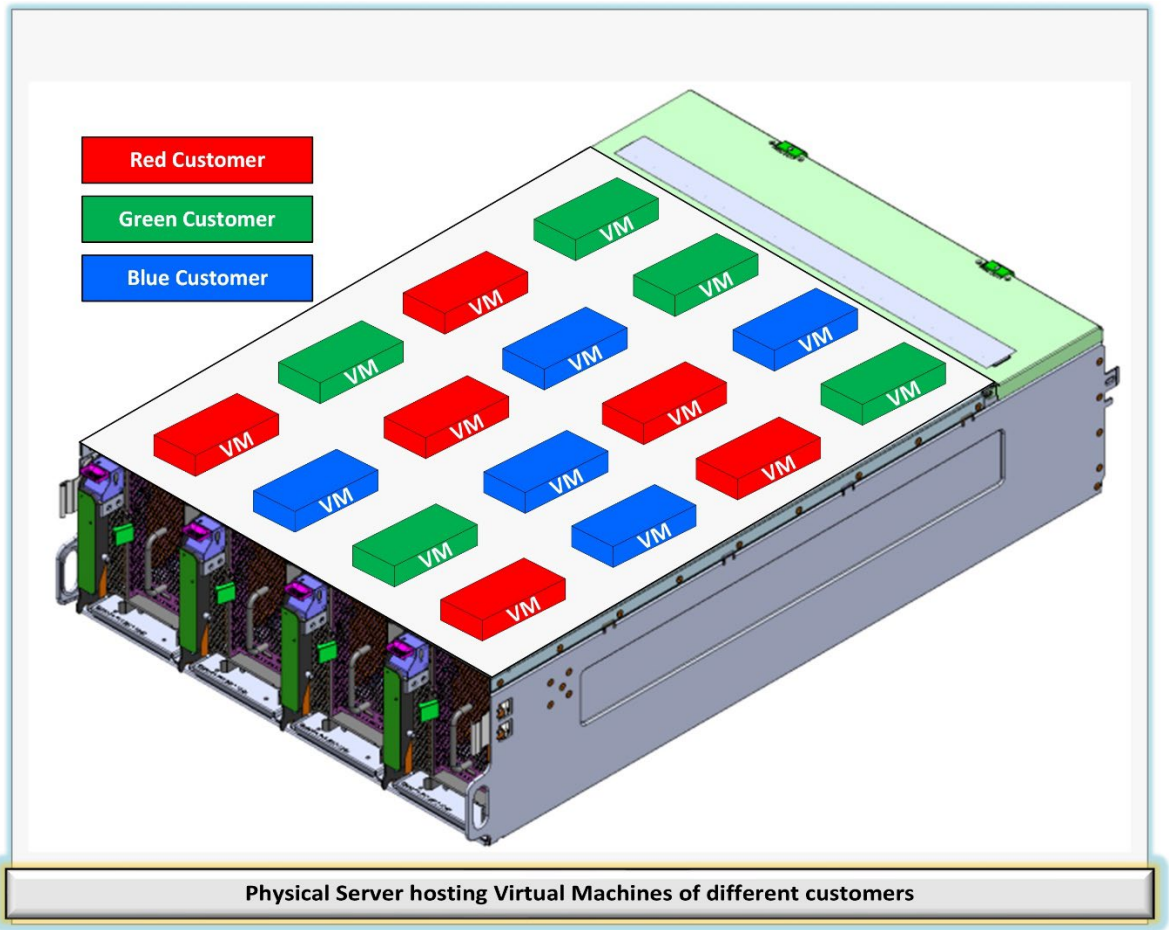
- Interacting with the cloud via GUI interface
- Interacting with the cloud programmatically
- Using AWS CLI to retrieve regions and availability zones information

The clouds

Cloud computing is the on-demand availability of computer services and resources over the Internet. There resources appear opaque, precisely as a cloud, from the viewpoint of the customers because they can only access a software interface that obscures the details of the operation. However, physically, these resources are hosted on massive datacentres which are interconnected by high-capacity networks.

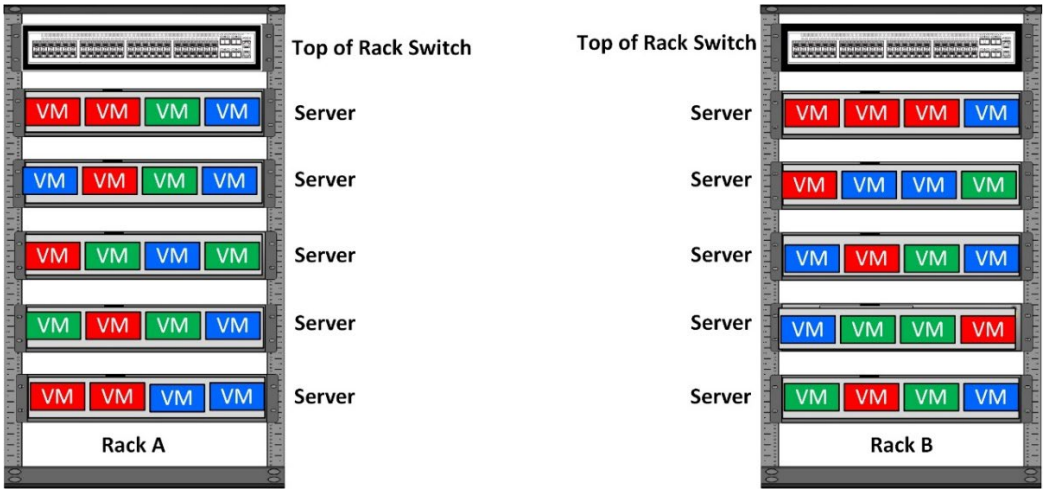
Building and maintaining the cloud datacentres requires a high degree of organization and control. To do so, the cloud providers follow very strict methods of procedure to operate the infrastructure. Furthermore, automation is used to deal with the vast scale of the operations and the only way to implement automation is that the components of the cloud follow reusable and repetitive patterns.

When a user creates a virtual resource in a cloud, an internal automatic procedure determines its location in the datacentre. An algorithm selects a physical server to host the virtual resource. The same server may be shared with the virtual instances of different customers. To illustrate the point, the following diagram shows a powerful physical server [1, 2] hosting the virtual machines for three different imaginary customers named red, green, and blue.



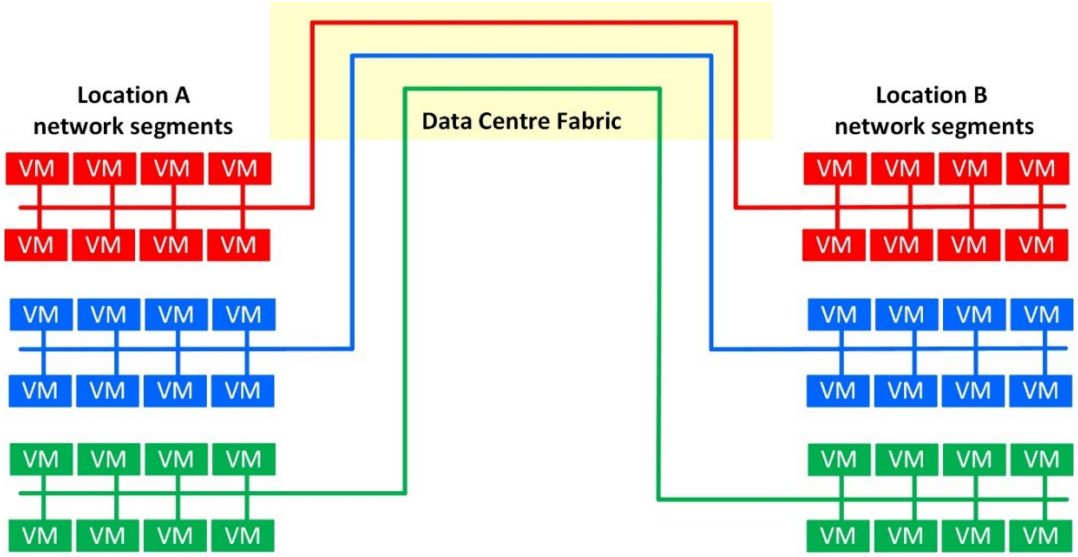
Physical Server hosting Virtual Machines of different customers

All the red instances belong to the same customer while the green and blue instances belong to different, completely unrelated **tenants**. These customers are not aware that they are sharing the same physical cloud computing resource because there is a virtual separation of the instances. Let's expand the scenario by showing two racks, A and B, in the next [3] diagram. Each rack has five physical servers and each server is hosting virtual machines (VMs) for the tenants.



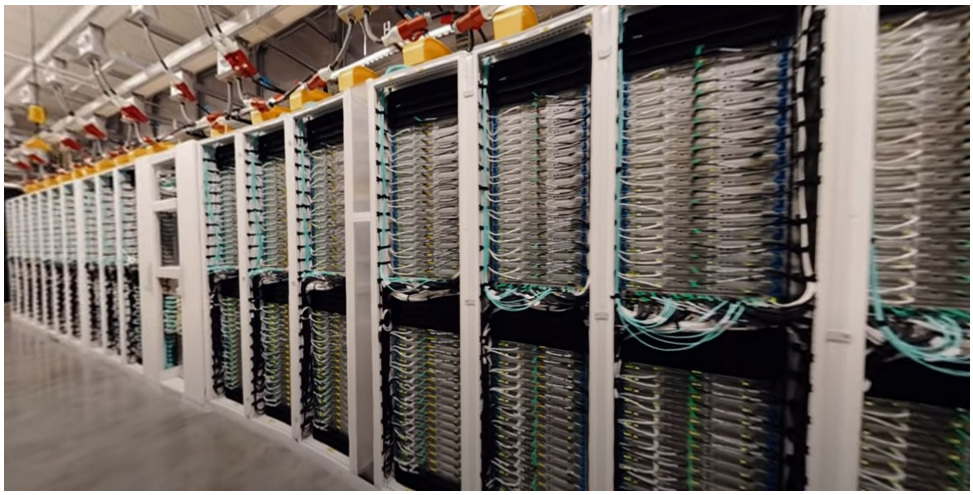
Generic view [3] of two racks of servers in the datacenter

Observe that at the **top of the rack (ToR)** there is a device which is a fast speed LAN switch. The physical servers interconnect to the rack's switch via high-speed interfaces. Each server hosts several virtual instances per tenant. All the VMs of the same colour belong to the same customer even if they are in separated physical racks and physical servers. Since they belong to the same customer, it is expected that they should be able to mutually communicate. The following figure [3] represents this concept.



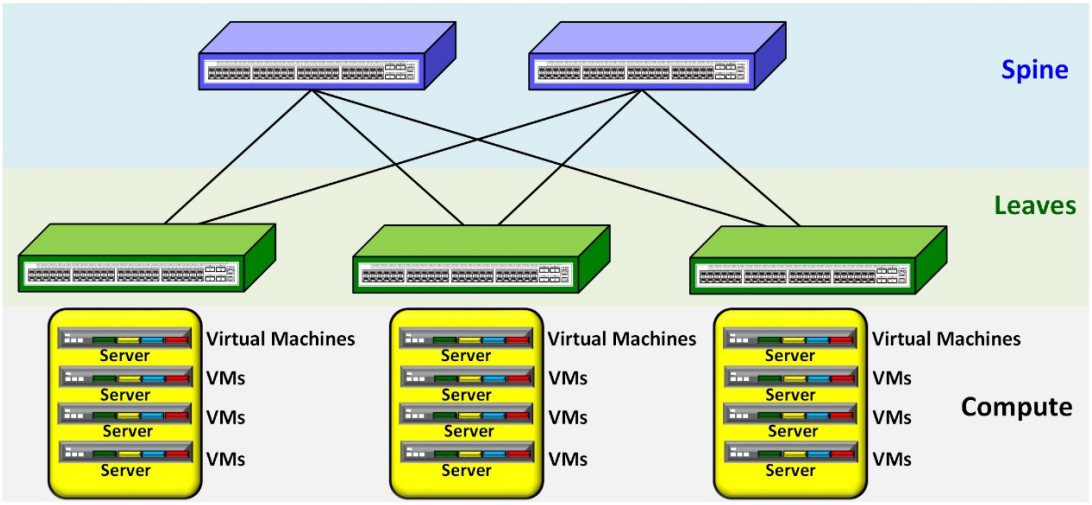
Intercommunication of logical resources [3] in the data centre

In the data centre, there are aisles filled with racks with a similar disposition as shown in the following picture taken from Microsoft Azure [4]. Notice that in this case example, the interconnecting LAN switch is in the (MoR) middle of the rack.



Partial View [4] of an aisle in a Microsoft Azure Datacentre

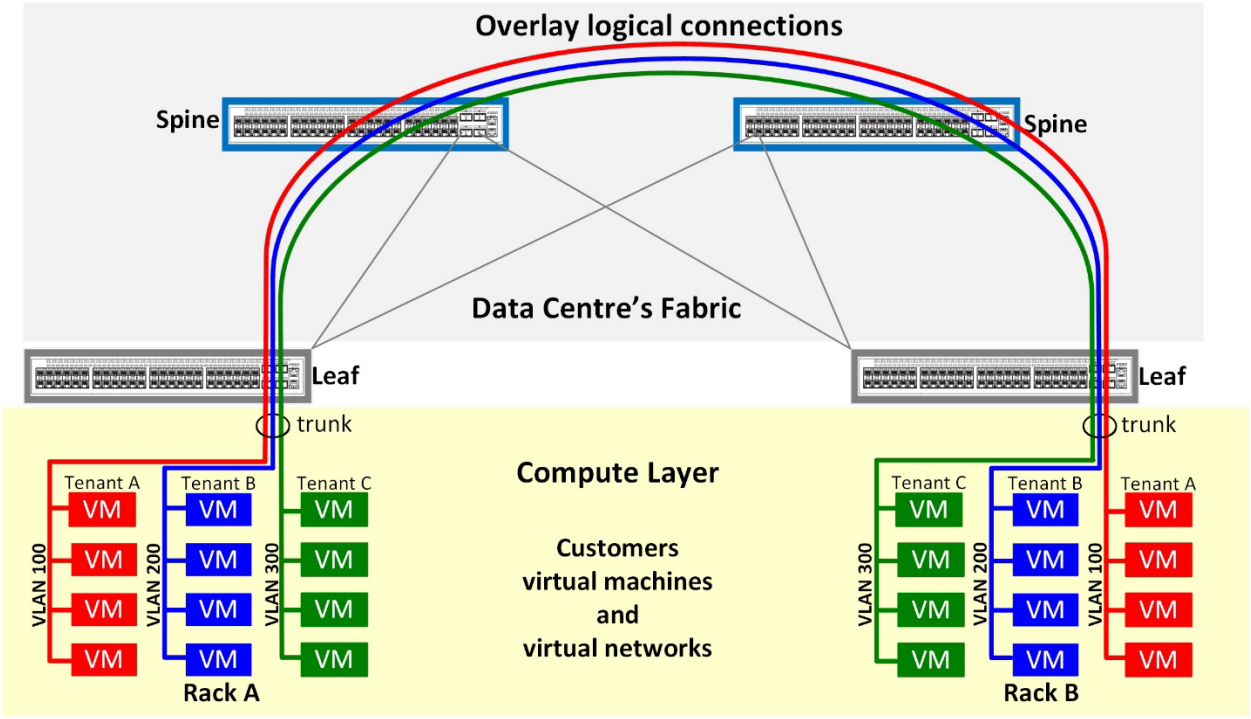
The customer's resources are spread across the cloud in different racks and even in different datacentres. The switches located in each rack interconnect with other switches in a higher tier so that the customer resources appear seemingly interconnected regardless of the physical location. A typical standard configuration for the modern data centre is as shown [3] below. This is called the **Leaf and Spine** or **Clos**:



Leaf and Spine arrangement [3] for a data centre

On each rack there is at least a leaf switch that is interconnected to spine switches using high-speed optical fibre links. This topology is common in data centres because it has been proven to offer the highest data transmission performance. Of course, this is a simplified topology for the sake of the explanation. In a complex data centre, there are multiple switching tiers; however, the fundamental idea remains the same. Let's refer to this the leaf and spine topology as the "data centre fabric" to hide the details a bit.

Multiple virtual logical connections are interweaved across the data centre fabric. The tenant virtual resources are interconnected with these virtual connections overlaid on the physical switching structure. From the viewpoint of the clients, it looks like they have dedicated networks as they can not see or interact with the other tenants sharing the resources. The following diagram [3] provides an oversimplified idea of the internal arraignment of a data centre.



Overlaid virtual interconnections [3] in a data centre

The deployment of the virtual resources, including the logical connections, is done with the orchestration and virtualization system that runs in the cloud provider. There are algorithms proprietary to the cloud vendors that determine where the virtual instances are going to be deployed. Things like assigning the physical server, the amount of memory and CPU type, all of that is done automatically. The customer does not have direct access to that orchestrator. Instead, the customer has access to some interface (typically a web interface) where configuration options are available. Once that a request for resource creation is made, **API (Application Programmable Interface)** calls are made to the cloud provisioners that implement the changes. What happens behind the scene is opaque to the customer and that is just fine. The cloud becomes a giant abstracted entity that provides resources on demand.

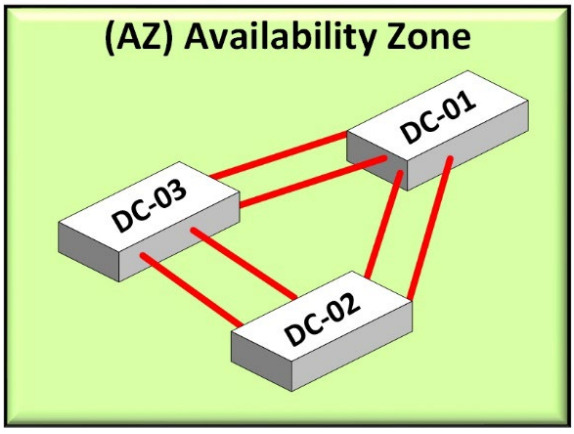
Cloud providers has perfected the construction of datacentres to the point that they have templated designs that allow them to build new infrastructure fast and efficiently. The following diagram [5] borrowed from AWS has a model datacentre with the ceiling removed to show the general structure.

AWS datacenter

Five sectors can be observed starting with the protection perimeter around the structure. In the bottom portion there is a power generation area. In the center of the building there are networking equipment rooms and dozens of double row aisles of racks that host the servers and storage devices. The back of the building is occupied by the cooling and fire protection equipment.

Availability Zones

Cloud providers do not rely to have their infrastructure on only one data centre but rather they build confederations of datacentres for reasons of capacity, redundancy, and access. A cluster of datacentres is known as an **Availability Zone**. The three major cloud providers, AWS, Azure and Google Cloud, use the same name and philosophy regarding the arrangement and location of their datacentres.



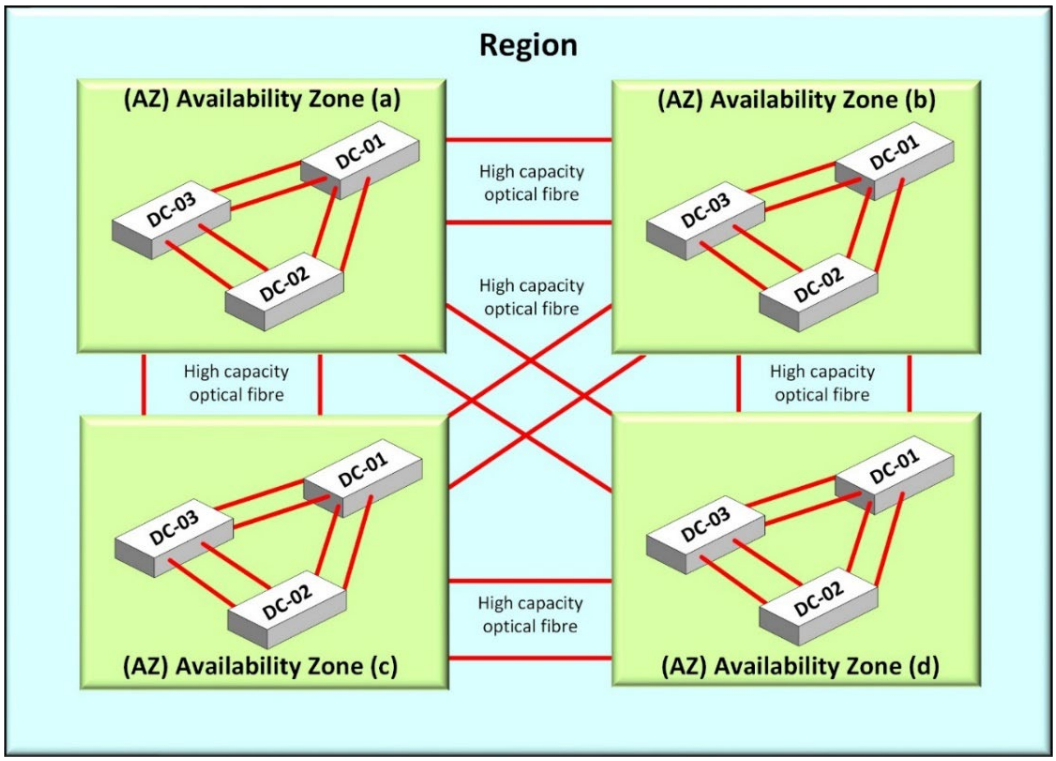
AWS datacenters [6] IAD-50, IAD-60, and IAD-71 in North Virginia

The cluster of datacenters is interconnected with high speed optical fibre links. The customer data is replicated among the datacentres that conform the group. This is to guarantee two of the most important cloud offerings: **reliability** and **availability**. User data and services can not be lost because of failures in the infrastructure. For that reason, the cloud providers replicate it in the datacentres of the availability zone to ensure reliable service. The data is always available even if a whole datacentre were to fail.

The datacentres that conform an availability zone are deployed within a perimeter determined by the maximum acceptable time that takes to exchange data on a round-trip basis. For AWS, the limit is 100 KM between [8] the datacentres of a region. For Azure, it is 2 milli-seconds of latency [6]. Either measurement way, distance or time, is a way to refer to transmission latency.

AWS Region

A **Region** is a cluster of availability zones within a geographical boundary. Therefore, a cluster of datacentres conforms an availability zone, and in turn, clusters of availability zones conform regions. To ensure high reliability, they are interconnected with multiple optical fibre links. The number of availability zones that compose a region depends on the regional market conditions. For example, there are regions that have only one availability zone while others have three, even six availability zones.



Schematic Representation of a Cloud Region

The cloud providers choose the region after evaluating several factors such as:

- Natural conditions (occurrence of events like flooding, hurricanes, and earthquakes).
- Proximity of markets.
- Energy supply.
- Access to the Internet.

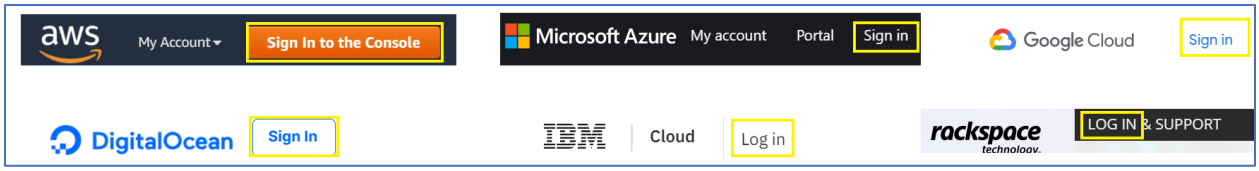
Difference between cloud and traditional datacenters

Datacenters

- Traditionally, datacenters follow two models: **in-house** and **collocated**.
- An **in-house** datacenter is a physical space located inside an enterprise. An internal IT organization manages and operates the datacenter.
- **Collocated** is the practice of leasing space in a third-party datacenter. In this case, an external organization leases physical space where the customers "co-locate" their equipment. The datacenter owner takes care of the physical security, environmental (cooling, heat dissipation, humidity), internal and external network connectivity, etc.
- Typically, a collocated datacenter sets a price per rack of service.
- The customer gets silos that contain their resources. They access these resources via remote access that in most cases are individual VPN (Virtual Private Network) connections.
- In both cases, either in house or collocated, provisioning of new services is somewhat a rigid process. For example, to deploy a web service, a physical server must be bought or leased. After procuring the machine, software packages must be downloaded, installed, and configured, and the service files must be provided and tested. Perhaps when the service was planned, the expectation was that one server would be enough to handle the requests. However, after being put in service, the requests are much higher than anticipated, therefore overwhelming the server. The solution would be to buy or lease another server and repeating all the steps above. This might take weeks or even months.
- The opposite also happens. For instance, a new application is expected to have high traffic and several servers are provisioned to handle such traffic. However, when the service is deployed, it fails the expectations, and the physical equipment is underused.
- The takeaway is that, in a traditional datacenter, services are deployed in a fixed manner. For example, a tenant leases and occupies a rack with 10 servers on it. If the tenant needs to expand its operations, it will need to add capacity by leasing another rack to provision more servers.

Clouds

- **Clouds** are also based on datacenters or rather a confederation of datacenters that appear to the customers as one unit.
- All the cloud providers offer an Internet available **interface** to request resources **dynamically**.



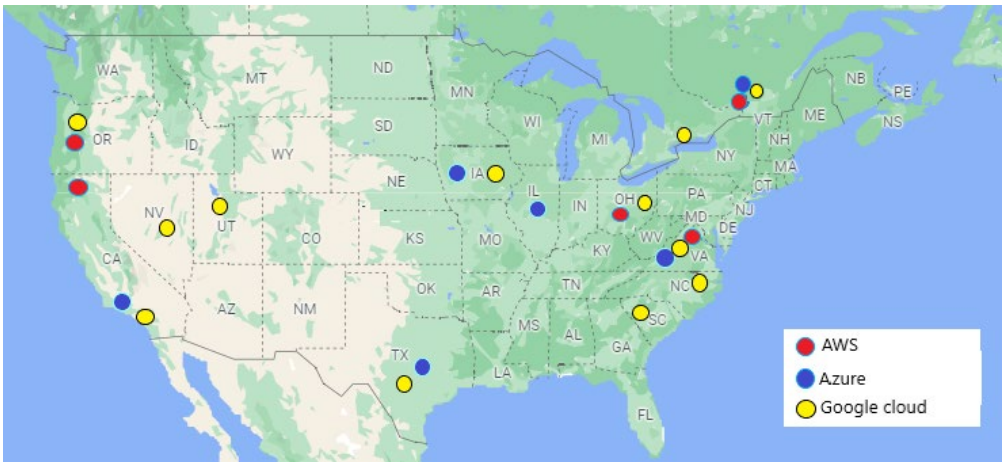
Login portals of cloud providers AWS, Azure, Google Cloud, Digital Ocean, IBM and Rackspace

- The services are provided **on-demand** or **subscription model** which is a major difference with respect the traditional collocated datacenter.
- If a service is requiring expansion, the customer can add more resources on the fly.
- If a service is overprovisioned, the customer can reduce the number of resources also.

- The resources are virtualized over a physical infrastructure.
- The cloud datacenter has enough capacity to serve the customer request.
- This provides **elasticity**, the customer can add or remove services **as needed**.
- The customer can add or remove resources in different ways. The customer can change the capacity of the services or add more resources to the services. This provides **scalability**.
- The customer pays for what it uses. The cloud provides an interface to keep track of cost per units. This provides **cost management**.
- How all of this operates is opaque from the viewpoint of the customer.
- The customer does not know exactly where its resources are located in the cloud. However, they appear to be omnipresent.
- The customer does not need to worry about physical devices failures. If a power supply fails in a server in the cloud-datacenter, the customer does not even notice that. There is enough redundancy to provide **reliable** service.

The three major clouds

The three major cloud providers according to market share [15] are AWS, Azure and Google Cloud in that order. Each of these clouds is a massive confederation of datacenters located strategically throughout geographical regions. The following map shows the regions in the U.S and Canada from information collected [8,9,10,11] in May 2022.



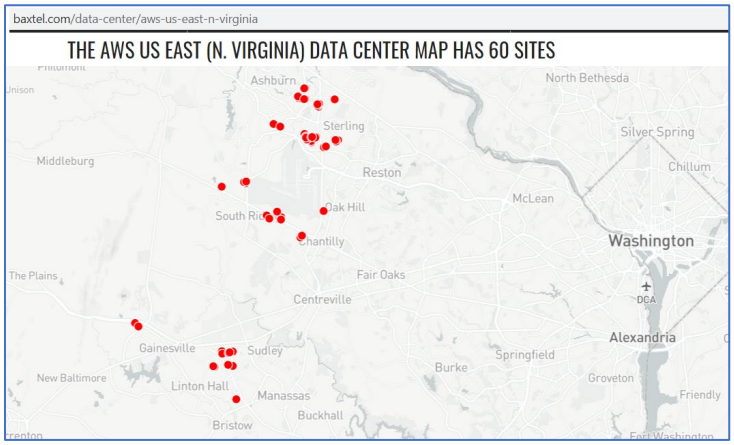
North America regions of the three largest cloud providers (May 2022)

In North America, AWS has 5 regions named us-east-1 (North Virginia), us-east-2 (Ohio), us-west-1 (Norther California), us-west-2 (Oregon), and one region Canada Central (Montreal). Beside these regions, there is one assigned exclusively to the US-government.

Microsoft Azure has six regions named east-us and east-us-2 (Virginia), west-us and west-us-2 (California), central-us (Iowa), north-central-us (Illinois), south-central-us (Texas) and Canada-central (Montreal). Also, beside these regions, there is one assigned only to the US-government.

Google Cloud has 11 regions named us-west-1 (Oregon), us-west-2 (Los Angeles), us-west-3 (Salt-Lake City), us-central-1 (Iowa), us-east-1 (South Carolina), us-east-4 (North Virginia), and two regions in Canada; North America-northeast-1 (Toronto) and North America north-east-2 (Montreal).

Even though the regions are denoted with dots in the previous map, they comprehend many datacentres. For example, AWS region us-east-1 has six availability zones that contain 60 sites according to the datacentre report from baxtel.com [10] (May 2022). Each site might have more than one datacentre.



AWS us-east-1 Region according to Baxtel's [6] data center report.

A common factor among AWS, Azure, and Google is that they have a region in North Virginia. In fact, AWS started its cloud business within this region. There are several reasons for choosing this location:

- A large volume of Internet traffic east-west and north-south traverses through North Virginia.
- The municipalities of the region have been receptive to the installation of the data centres.
- There is a large concentration of Internet and communications companies in the region.
- The proximity of the U.S government and the large cities in the east coast market.
- Reliable power supply.
- Low environmental risks (flooding, hurricanes, fires, and earthquakes).

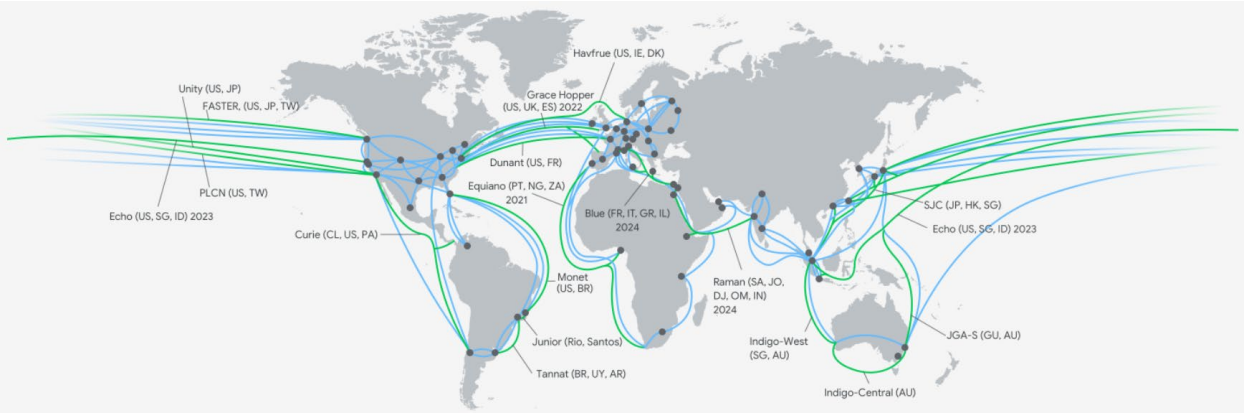
In Canada, the three providers [8,9,10,11] have a region in Montreal. This city is a preferred location for datacentres because it is in the path for multiple optical fibre links that carry Internet traffic across Canada, the United States and Europe. Another important factor is the abundant and stable Quebec hydro-electric power. The natural risks are very low in Montreal too except for the ice storms. Finally, not to be dismissed, it is easier and cheaper to cool down the datacentres in Montreal.

The cloud providers keep adding datacentres, zones and regions to their maps. On May 2022, AWS [9,11] has 26 regions and 84 Availability Zones across the world. (Canada west will open in Calgary by 2024).

AWS Global Regions [9] Infrastructure

Global Networks

Initially, cloud providers such as AWS and Azure, relied on the global Internet to interconnect their platforms. However, upon the consolidation of their cloud sites, they built their own network backbones [4,9] even bypassing the carrier networks in many cases. Google has been using its vast global backbone shown in the next map [12] from the very beginning to link its cloud infrastructure.



Google global network [12]

Reliable Clouds

Every cloud provider has a complex network of datacentres that is heavily interconnected and replicated. There is a pattern that the three major vendors, AWS, Azure, and Google, follow. They build redundant datacentres, then they define availability zones gathering clusters of datacentres. Finally, they group the availability zones into geographical areas.

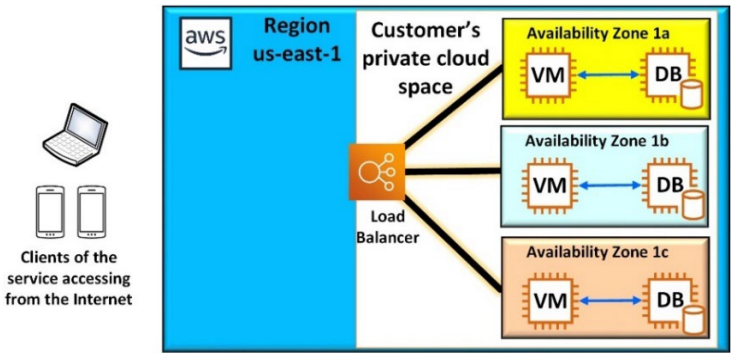
The datacentres are connected to multiple telecom carriers providing Internet access. Beside that, the major cloud providers also have their own network backbones with global reach so they can assure the customer that the data will travel inside their cloud networks.

The main reason behind this complexity is to guarantee availability and reliability. For the cloud vendors, it is critical that the customers have confidence that their data will always be available and that it will not be lost or corrupted. The service providers go to great lengths to offer service level guarantees because they have the infrastructure to back those claims up. For example, AWS offers [13] the so called “11 nines rule” (or 99.999999999 %) of reliability for some services such as data storage.

For the customer, it is very important to be aware of the cloud provider physical plant because it enables the design of resilient cloud platforms. For instance, a robust design of an application typically implies that several servers will be running with identical configuration. If one server goes down, the others pick up the load. Furthermore, each replicated server must be located in a different availability zone in the unlikely case that one of the zones becomes unreachable.

Another good example of robust design are cloud databases services. By default, the cloud providers offer elastic database services. That means that when the customer requests the creation of an elastic database the cloud automatically deploys copies of the database in several availability zones. The information stored in the database is replicated periodically ensuring its reliability and availability.

The following diagram illustrates a robust design of a service platform. A group of three identical virtual webservers and three identical databases provide the same service to the clients accessing an application from the Internet. A network load balancer acts as the service's interface between the back-end infrastructure and the clients of the application. The databases frequently replicate the stored data keeping the tables up to date. All these virtual instances have been deployed in different availability zones (us-east-1**a**, us-east-1**b**, and us-east-1**c**) to ensure reliability and availability. In the improbable event of a whole zone failing, the service would remain in operation because it was designed in a robust manner.



Robust service hosted in AWS Cloud

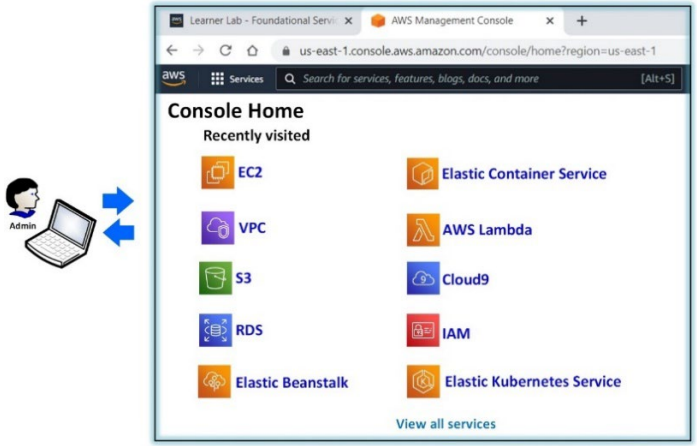
Cloud User Interfaces

There are several ways to interact with the cloud to deploy and manage the virtual resources. They can be classified in four types:

- **Graphic Console** via a web browser (GUI) interface.
- **Programmatically** via a programming language.
- **Command Line Interface (CLI)** via typing proprietary command lines on a text console.
- **Infrastructure as Code (IaaC)** via a specific software tool.

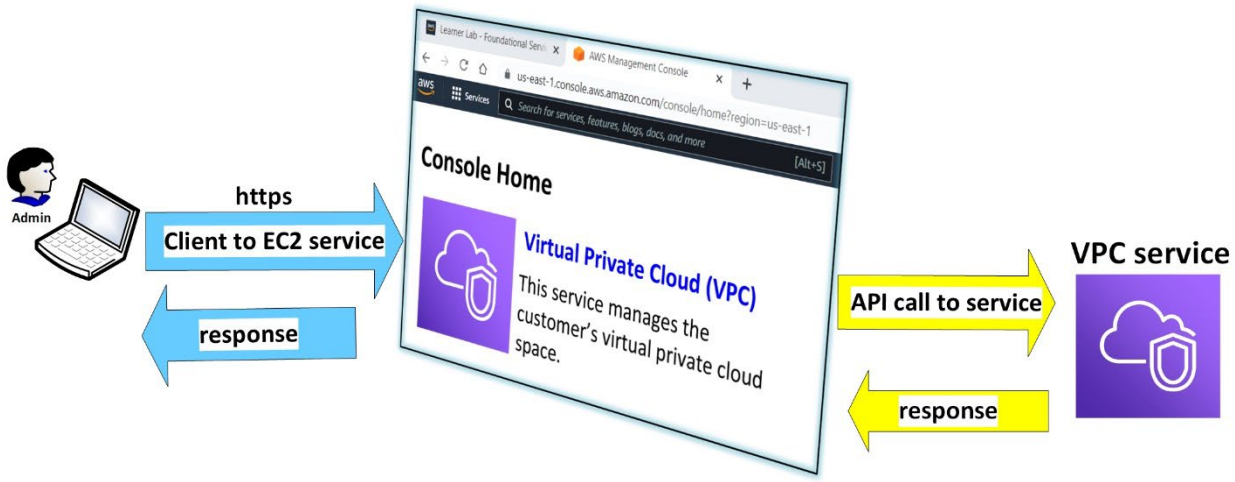
Graphic Console Interface

The graphic user interface provides a user friendly way to interact with the cloud by presenting the different services in a wizard configuration style. There are hundreds of services available in AWS.



View of the AWS Graphical User Interface

The client administrator's web browser interacts with the server portal using the secure web https protocol. Behind the scenes, the web server makes Application Programmable Interface (API) calls to the different services to retrieve, display, create, delete, or modify cloud resources. The web portal acts fundamentally as a translation interface to make the administrator's tasks easier.

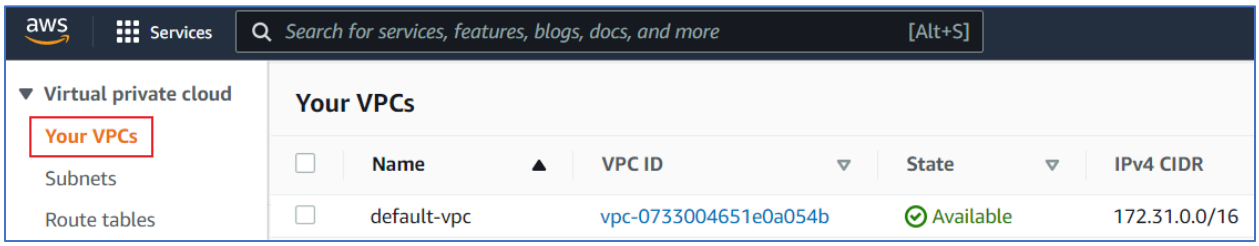


The GUI web server is an interface between the account admin and the cloud services.

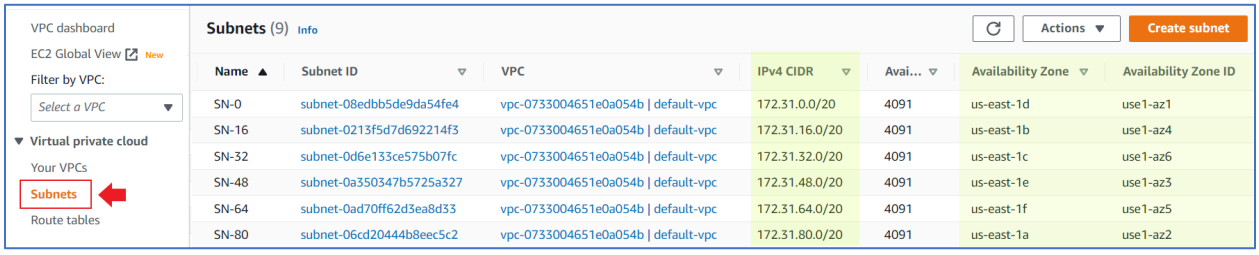
Learning Activity

Interacting with the cloud via GUI interface

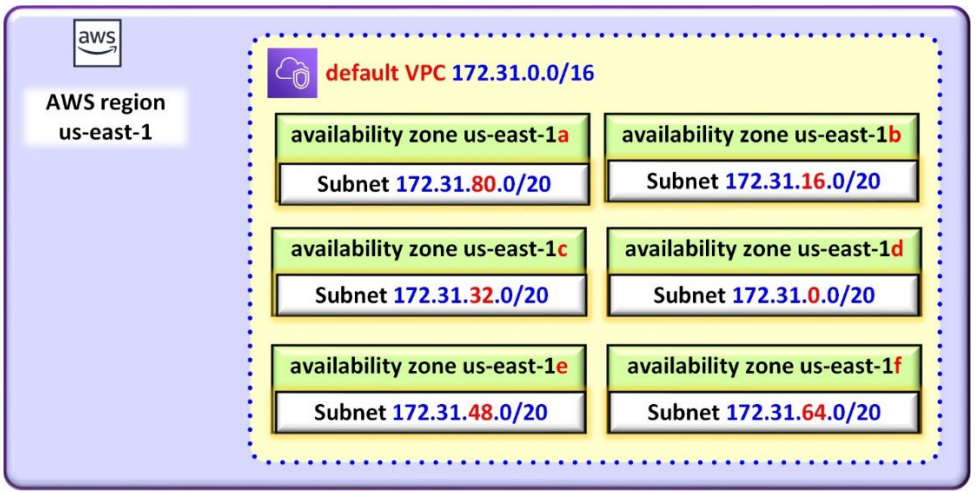
The learning objective of this activity is to begin the exploration of the AWS cloud using the graphic user interface. First, let's proceed to the virtual private cloud (VPC) service in the AWS account. There is always a default vpc with the IPv4 address space 172.31.0.0/16. This is where computer resources are deployed.



Also in the same VPC service, there is the Subnets option. When the user clicks on that button, it activates an API call that retrieves the detail description of every subnet existing in the VPC. Observe, that every subnet is allocated to a different availability zone.



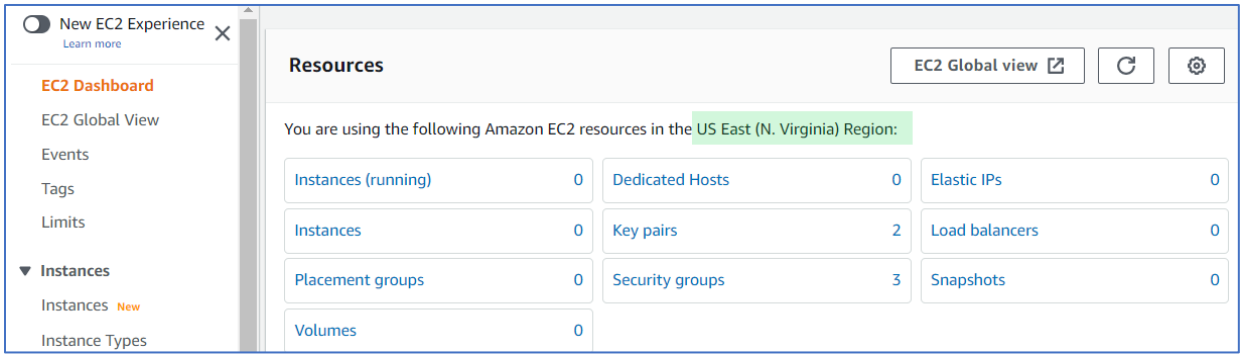
Pictorially, this is part of the information what the GUI returns for the current account regarding the six availability zones with the six subnets of the default VPC.



Representation of the default VPC.

Explore Availability Zones

- Search for EC2 Service, notice the Region that your account has assigned:

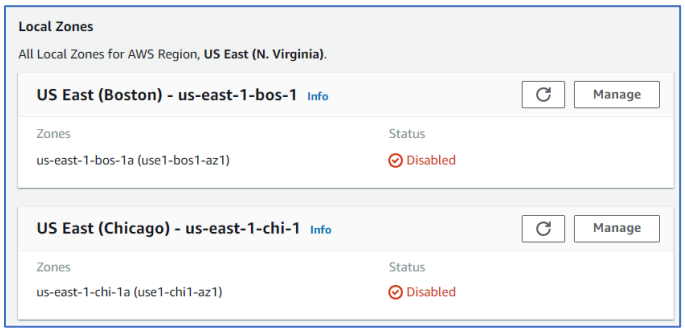


- Scroll down to see the Availability Zones of this Region:

- Click on Enable additional Zones to explore and learn more about the structure of AWS Zones.

Local Zones

A local zone is AWS infrastructure located near sites of interests such as a large city, a particular market, or places with concentration of IT industries. This snippet shows the local zones of Boston and Chicago.



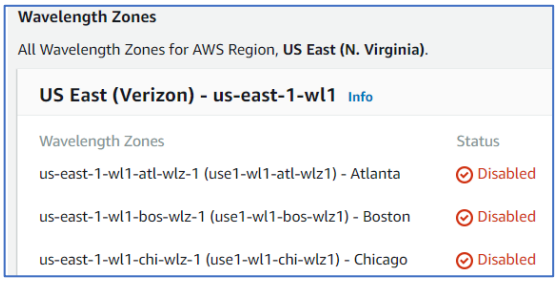
A local zone provides compute, storage, and database. Other services are available, but that depends on the constitution of the particular zone. The following snippet shows that the Boston and Chicago local zones offer the same services.

AWS Local Zones	Amazon EC2	Amazon EBS	Amazon ECS	Amazon EKS	Amazon VPC	Amazon FSx	Amazon ELB	Amazon EMR	Amazon ElastiCache	Amazon RDS
Boston	T3, C5d, R5d, and G4dn instances	General Purpose SSD (gp2)	✓	✓	✓		ALB			
Chicago	T3, C5d, R5d, and G4dn instances	General Purpose SSD (gp2)	✓	✓	✓		ALB			

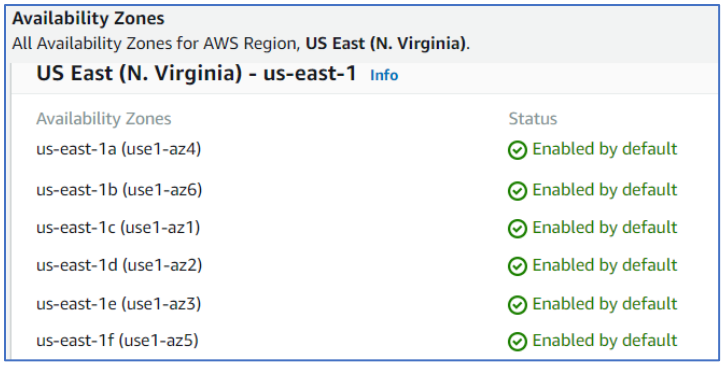
The main benefit obtained from having services deployed in a local zone is the low latency (under 10 ms) access because of the proximity of the resources. The local zones can be enabled from the management console without any additional cost. The costs incurred are for the services running in the zone.

AWS Wavelengths Zones

Wavelength Zones are an extension of the AWS cloud placed inside 5G telecommunication providers to provide the lowest latency service to the clients. This is the use case; an augmented reality application is hosted in an AWS standard zone in the Northern Virginia Region. A 5G client is located in Atlanta, Georgia, so its traffic will need to travel back and forth to the remote location causing an unacceptable delay that affect the user's experience. With wavelength zones, the user does not experience such delay because the application is hosted at the edge of the 5G network. The following snippet shows three cities where the telecommunications provider Verizon have AWS Wavelength Zones.



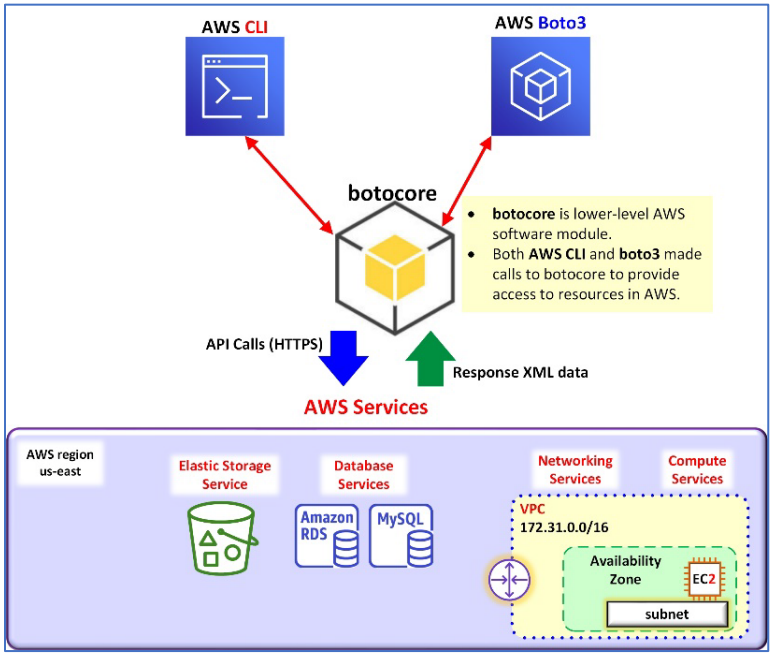
Finally, the console shows the us-east-1 region six availability zones. Each zone is a cluster of datacentres and it is denoted by a different letter and a zone-id. Also notice that they are all enabled by default.



Programmatical Interface AWS Python SDK Boto3

Boto3 is a **Software Development Kit (SDK)** for Python designed by AWS. This SDK contains the libraries, modules and tools to write Python programs that interact with AWS services. The programs can retrieve information, create and delete resources (**CRUD** verbs = **c**reate, **r**ead, **u**pdate and **d**elete).

AWS Boto3 (and AWS Command Line Interface too) interacts with a lower level software module which is called the **botocore**. For example, the calls from boto3 are translated into **API calls** using the secure web protocol HTTPS. These API calls are directed to the **endpoints** of the different AWS cloud services such as Elastic Compute Cloud (EC2), networking services (VPC), databases (RDS), storage (S3), functions and many more. The HTTPS protocol carries the answers back in the format of the XML language. Then botocore translates the answers to the boto3 requester.



AWS python SDK boto3 interaction with AWS services.

Boto3 has two main modules, **resource** and **client**. Resource is a higher level module that returns python classes. To work with this module, it is necessary to search in the documentation for the methods for the specific tasks. Frequently, programs written using the resource module are compact because the methods acts as wrappers for the execution of many tasks.

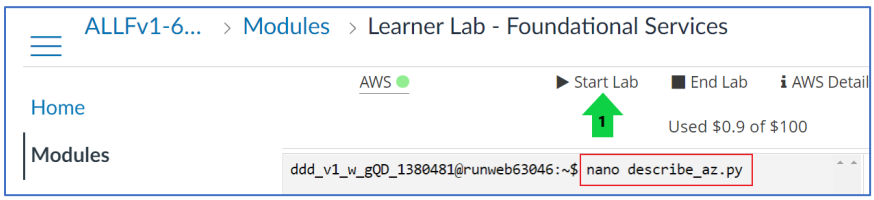
On the other hand, the module client returns python dictionaries. This implies that processing the data requires iterations over it. For example, searching the dictionaries to find lists and then looping over these lists to extract values. The module client is lower level but it is more python natural than the method resource. In any case, either method can do the same work. It is just than sometimes one problem might be easier to solve with one than the other, it depends. The following examples use the module client to retrieve information regarding availability zones, subnets and region information in general.

Learning Activity

Interacting with the cloud programmatically

Now, let's repeat the same transaction as it was done with the GUI console, but this time using the programming language python with the AWS SDK boto3.

- Go to your AWS Console View
- Use a text editor such as **nano** or **vi** to create the python scripts.



Describe Availability Zones.

The following script (describe_az.py) uses the **client module** to create an object named target. This object is referred to the EC2 service. The object has properties which are described in python structures predefined in the boto3 libraries. The method describe_availability_zones returns a python Dictionary.

```
# Python program describe_az.py retrieves the availability zones description.
import boto3
target = boto3.client('ec2')
response = target.describe_availability_zones()
print(response)
```

This is the response obtained when running the previous python script. There are two main outer keys in this dictionary, the key **AvailabilityZones** and the key **ResponseMetadata**.

```
$ python list_az_ids.py
{'AvailabilityZones': [{'State': 'available', 'OptInStatus': 'opt-in-not-required', 'Messages':
[], 'RegionName': 'us-east-1', 'ZoneName': 'us-east-1a', 'ZoneId': 'use1-az2', 'GroupName': 'us-
east-1', 'NetworkBorderGroup': 'us-east-1', 'ZoneType': 'availability-zone'}, {'State':
'available', 'OptInStatus': 'opt-in-not-required', 'Messages': [], 'RegionName': 'us-east-1',
'ZoneName': 'us-east-1b', 'ZoneId': 'use1-az4', 'GroupName': 'us-east-1', 'NetworkBorderGroup':
'us-east-1', 'ZoneType': 'availability-zone'}, {'State': 'available', 'OptInStatus': 'opt-in-not-
required', 'Messages': [], 'RegionName': 'us-east-1', 'ZoneName': 'us-east-1c', 'ZoneId': 'use1-
az6', 'GroupName': 'us-east-1', 'NetworkBorderGroup': 'us-east-1', 'ZoneType': 'availability-
zone'}, {'State': 'available', 'OptInStatus': 'opt-in-not-required', 'Messages': [],
'RegionName': 'us-east-1', 'ZoneName': 'us-east-1d', 'ZoneId': 'use1-az1', 'GroupName': 'us-east-
1', 'NetworkBorderGroup': 'us-east-1', 'ZoneType': 'availability-zone'}, {'State': 'available',
'OptInStatus': 'opt-in-not-required', 'Messages': [], 'RegionName': 'us-east-1', 'ZoneName': 'us-
east-1e', 'ZoneId': 'use1-az3', 'GroupName': 'us-east-1', 'NetworkBorderGroup': 'us-east-1',
'ZoneType': 'availability-zone'}, {'State': 'available', 'OptInStatus': 'opt-in-not-required',
'Messages': [], 'RegionName': 'us-east-1', 'ZoneName': 'us-east-1f', 'ZoneId': 'use1-az5',
'GroupName': 'us-east-1', 'NetworkBorderGroup': 'us-east-1', 'ZoneType': 'availability-zone'}],
'ResponseMetadata': {'RequestId': '777c41b3-1d4b-4963-84bb-91f69bcfa807', 'HTTPStatusCode': 200,
'HTTPHeaders': {'x-amzn-requestid': '777c41b3-1d4b-4963-84bb-91f69bcfa807', 'cache-control': 'no-
cache, no-store', 'strict-transport-security': 'max-age=31536000; includeSubDomains', 'vary':
'accept-encoding', 'content-type': 'text/xml;charset=UTF-8', 'content-length': '2973', 'date':
'Sat, 17 Sep 2022 19:00:07 GMT', 'server': 'AmazonEC2'}, 'RetryAttempts': 0}}
```

Python dictionaries are made of key:value pairs. In turn, the key AvailabilityZones contains, as a value, a **python list**. Thus, the previous program is modified to supply the AvailabilityZones key to obtain the list that contains all the availability zones.

```
# Python program describe_az.py displays the availability zones description only.
import boto3
target = boto3.client('ec2')
response = target.describe_availability_zones()
print(response['AvailabilityZones'])
```

Python lists are made of elements inside square brackets []. These elements can be of any data type, strings, numbers, lists, dictionaries, etc. In this case, every element is yet another dictionary. Thus, when the script is run the following results (with a bit of editing to show the structure better).

```
$ python list_az_ids.py
{'AvailabilityZones': [
{'State': 'available', 'OptInStatus': 'opt-in-not-required', 'Messages': [], 'RegionName': 'us-east-
1', 'ZoneName': 'us-east-1a', 'ZoneId': 'use1-az2', 'GroupName': 'us-east-1', 'NetworkBorderGroup':
'us-east-1', 'ZoneType': 'availability-zone'},
{'State': 'available', 'OptInStatus': 'opt-in-not-required', 'Messages': [], 'RegionName': 'us-east-
1', 'ZoneName': 'us-east-1b', 'ZoneId': 'use1-az4', 'GroupName': 'us-east-1', 'NetworkBorderGroup':
'us-east-1', 'ZoneType': 'availability-zone'},
{'State': 'available', 'OptInStatus': 'opt-in-not-required', 'Messages': [], 'RegionName': 'us-east-
1', 'ZoneName': 'us-east-1c', 'ZoneId': 'use1-az6', 'GroupName': 'us-east-1', 'NetworkBorderGroup':
'us-east-1', 'ZoneType': 'availability-zone'},
{'State': 'available', 'OptInStatus': 'opt-in-not-required', 'Messages': [], 'RegionName': 'us-east-
1', 'ZoneName': 'us-east-1d', 'ZoneId': 'use1-az1', 'GroupName': 'us-east-1', 'NetworkBorderGroup':
'us-east-1', 'ZoneType': 'availability-zone'},
{'State': 'available', 'OptInStatus': 'opt-in-not-required', 'Messages': [], 'RegionName': 'us-east-
1', 'ZoneName': 'us-east-1e', 'ZoneId': 'use1-az3', 'GroupName': 'us-east-1', 'NetworkBorderGroup':
'us-east-1', 'ZoneType': 'availability-zone'},
{'State': 'available', 'OptInStatus': 'opt-in-not-required', 'Messages': [], 'RegionName': 'us-east-
1', 'ZoneName': 'us-east-1f', 'ZoneId': 'use1-az5', 'GroupName': 'us-east-1', 'NetworkBorderGroup':
'us-east-1', 'ZoneType': 'availability-zone'}
]
```

The key AvailabilityZones supplies the python **list** with six elements (the six availability zones). A loop is needed to iterate over the elements on the list. First, a counter value is required. This can be extracted directly from the list using the python command **len** (it returns the length of the list)

```
import boto3
target=boto3.client('ec2')
response=target.describe_availability_zones()
count_az = len(response['AvailabilityZones'])
print('The list has', count_az,'elements')
```

The result of running this script is six elements for six availability zones.

```
aws_academy_console$ python length_az.py
The list has 6 elements
```

Now, let's use a **for loop** to go over the list. The command **range** receives the length value (6) to run the loop over the elements of the list using the indexes from 0 to 5.

```
import boto3
target=boto3.client('ec2')
response=target.describe_availability_zones()
az = response['AvailabilityZones']
count_az =len(az)
for n in range(count_az):
   az_id = az[n]['ZoneId']
   az_name = az[n]['ZoneName']
   print('The availability zone id is', az_id, 'and its name is', az_name)
```

After testing the previous program, more details can be added.

```
# This Python program prints the availability zones that AWS Academy use.
# Import AWS Python boto3 library.
import boto3
# Create an object
target = boto3.client('ec2')

# Retrieves the Availability Zones for the current Region
response = target.describe_availability_zones()

# Loop over data structure
for n in range(len(response['AvailabilityZones'])):
    az_name = response['AvailabilityZones'][n]['ZoneName']
    az_id = response['AvailabilityZones'][n]['ZoneId']
    state = response['AvailabilityZones'][n]['State']
    print(n+1,')','The availability zone', az_name, 'with ID', az_id, 'is', state)
```

The result of running the program are the names and ids of the six availability zones.

```
aws_academy_console$ python az_ids.py
1 ) The availability zone us-east-1a with ID use1-az2 is available
2 ) The availability zone us-east-1b with ID use1-az4 is available
3 ) The availability zone us-east-1c with ID use1-az6 is available
4 ) The availability zone us-east-1d with ID use1-az1 is available
5 ) The availability zone us-east-1e with ID use1-az3 is available
6 ) The availability zone us-east-1f with ID use1-az5 is available
```

Describe Subnets

Following the same logic, another python script is used to retrieve all the data that pertains to the subnets. The describe_subnets() method does exactly that in detail. Then the key word 'Subnets' supplies the list that contains every subnet dictionary.

```
import boto3
target=boto3.client('ec2')
response=target.describe_subnets()
print(response['Subnets'])
```

```
$ python describe_subnets.py (note: manual editing has been done to show the data better).

{'Subnets': [{'AvailabilityZone': 'us-east-1b', 'AvailabilityZoneId': 'use1-az4', 'AvailableIpAddressCount':
4091, 'CidrBlock': '172.31.16.0/20', 'DefaultForAz': True, 'MapPublicIpOnLaunch': True,
'MapCustomerOwnedIpOnLaunch': False, 'State': 'available', 'SubnetId': 'subnet-0213f5d7d692214f3', 'VpcId':
'vpc-0733004651e0a054b', 'OwnerId': '067251588034', 'AssignIpv6AddressOnCreation': False,
'Ipv6CidrBlockAssociationSet': [], 'Tags': [{'Key': 'Name', 'Value': 'SN-16'}],'SubnetArn': 'arn:aws:ec2:us-
east-1:067251588034:subnet/subnet-0213f5d7d692214f3'},

{'AvailabilityZone': 'us-east-1f', 'AvailabilityZoneId': 'use1-az5', 'AvailableIpAddressCount': 4091,
'CidrBlock': '172.31.64.0/20', 'DefaultForAz': True, 'MapPublicIpOnLaunch': True,
'MapCustomerOwnedIpOnLaunch': False, 'State': 'available', 'SubnetId': 'subnet-0ad70ff62d3ea8d33', 'VpcId':
'vpc-0733004651e0a054b', 'OwnerId': '067251588034', 'AssignIpv6AddressOnCreation': False,
'Ipv6CidrBlockAssociationSet': [],'Tags': [{'Key': 'Name', 'Value': 'SN-64'}], 'SubnetArn': 'arn:aws:ec2:us-
east-1:067251588034:subnet/subnet-0ad70ff62d3ea8d33'},

{'AvailabilityZone': 'us-east-1e', 'AvailabilityZoneId': 'use1-az3', 'AvailableIpAddressCount': 4091,
'CidrBlock': '172.31.48.0/20', 'DefaultForAz': True, 'MapPublicIpOnLaunch': True,
'MapCustomerOwnedIpOnLaunch': False, 'State': 'available', 'SubnetId': 'subnet-0a350347b5725a327', 'VpcId':
'vpc-0733004651e0a054b', 'OwnerId': '067251588034', 'AssignIpv6AddressOnCreation': False,
'Ipv6CidrBlockAssociationSet': [],'Tags': [{'Key': 'Name', 'Value': 'SN-48'}], 'SubnetArn': 'arn:aws:ec2:us-
east-1:067251588034:subnet/subnet-0a350347b5725a327'},

{'AvailabilityZone': 'us-east-1c', 'AvailabilityZoneId': 'use1-az6', 'AvailableIpAddressCount': 4091,
'CidrBlock': '172.31.32.0/20', 'DefaultForAz': True, 'MapPublicIpOnLaunch': True,
'MapCustomerOwnedIpOnLaunch': False, 'State': 'available', 'SubnetId': 'subnet-0d6e133ce575b07fc', 'VpcId':
'vpc-0733004651e0a054b', 'OwnerId': '067251588034', 'AssignIpv6AddressOnCreation': False,
'Ipv6CidrBlockAssociationSet': [], 'Tags': [{'Key': 'Name', 'Value': 'SN-32'}],'SubnetArn': 'arn:aws:ec2:us-
east-1:067251588034:subnet/subnet-0d6e133ce575b07fc'},

{'AvailabilityZone': 'us-east-1a', 'AvailabilityZoneId': 'use1-az2', 'AvailableIpAddressCount': 4091,
'CidrBlock': '172.31.80.0/20', 'DefaultForAz': True, 'MapPublicIpOnLaunch': True,
'MapCustomerOwnedIpOnLaunch': False, 'State': 'available', 'SubnetId': 'subnet-06cd20444b8eec5c2', 'VpcId':
'vpc-0733004651e0a054b', 'OwnerId': '067251588034', 'AssignIpv6AddressOnCreation': False,
'Ipv6CidrBlockAssociationSet': [], 'Tags': [{'Key': 'Name', 'Value': 'SN-80'}], 'SubnetArn': 'arn:aws:ec2:us-
east-1:067251588034:subnet/subnet-06cd20444b8eec5c2'},

{'AvailabilityZone': 'us-east-1d', 'AvailabilityZoneId': 'use1-az1', 'AvailableIpAddressCount': 4091,
'CidrBlock': '172.31.0.0/20', 'DefaultForAz': True, 'MapPublicIpOnLaunch': True, 'MapCustomerOwnedIpOnLaunch':
False, 'State': 'available', 'SubnetId': 'subnet-08edbb5de9da54fe4', 'VpcId': 'vpc-0733004651e0a054b',
'OwnerId': '067251588034', 'AssignIpv6AddressOnCreation': False, 'Ipv6CidrBlockAssociationSet': [], 'Tags':
[{'Key': 'Name', 'Value': 'SN-0'}],'SubnetArn': 'arn:aws:ec2:us-east-1:067251588034:subnet/subnet-
08edbb5de9da54fe4'}],
```

Show Regions

The following python script uses the client module and the describe_regions()to obtain the names of all the regions available for use in AWS Academy.

```
# This Python program prints the Regions that AWS Academy use
# This imports the AWS Python Library boto3
import boto3

# This creates an object to perform operations
ec2 = boto3.client('ec2')

# This retrieves and prints the region names
response = ec2.describe_regions()
print('List of AWS Regions available with AWS Academy')

for n in range(len(response['Regions'])):
    name = response['Regions'][n]['RegionName']
    print(n + 1,')','Region:', name)
```

Running the script shows all the regions where AWS Academy can be used.

```
python describe_regions.py
List of AWS Regions available with AWS Academy
1 ) Region: eu-north-1
2 ) Region: ap-south-1
3 ) Region: eu-west-3
4 ) Region: eu-west-2
5 ) Region: eu-west-1
6 ) Region: ap-northeast-3
7 ) Region: ap-northeast-2
8 ) Region: ap-northeast-1
9 ) Region: sa-east-1
10 ) Region: ca-central-1
11 ) Region: ap-southeast-1
12 ) Region: ap-southeast-2
13 ) Region: eu-central-1
14 ) Region: us-east-1
15 ) Region: us-east-2
16 ) Region: us-west-1
17 ) Region: us-west-2
```

Learning Activity

Using AWS CLI to retrieve regions and availability zones information

AWS CLI is a Command Line Interface tool to interact with AWS cloud. Resources can be queried for information, or they can be created by just typing AWS CLI commands. For example, try the following six commands [14] and compare the results with the previous python program outputs.

```
aws ec2 describe-regions

aws ec2 describe-regions --filters "Name=endpoint,Values=*us*"

aws ec2 describe-availability-zones

aws ec2 describe-subnets --query "Subnets[*].SubnetId"

aws ec2 describe-subnets --query "Subnets[*].CidrBlock"

aws ec2 describe-subnets --query "Subnets[*].AvailabilityZone"
```

Chapter Conclusion

All the major cloud operators concentrate vast amounts of compute and network resources in data centres which highly interconnected among themselves and across the Internet. These cloud providers methodically organize the resources according to patterns that enable their management using internal automation and orchestration tools.

The details inside the cloud are not all revealed to the customers of it. From the viewpoint of the customers, they get to interact with the clouds using interfaces and application endpoints. The users obtain a share of the vast resources that the cloud offers. They do so elastically and on demand.

The three major cloud providers, AWS, Azure and GPC, implement a similar design philosophy although they use different technologies, tools and names for the components of the cloud. Fundamentally, the clouds are made up of groups of datacentres. One or more datacentres interlinked within certain distance is regarded as an organizational zone. The main driver for such arrangement is to provide constant availability and thus reliability. Several zones within a geographical space conform a region. Furthermore, regions interconnected gives rise to a global cloud.

Chapter 2 Coursework

- Write a Python program that lists the IPv4 block (CidrBlock) of every subnet in the us-east-1 region.
- Modify the previous program to list every CidrBlock with the corresponding Availability Zone. For example, the output for one subnet would be: " Subnet 172.31.16.0/20 is in availability zone us-east-1b ". The program must list the six CidrBlocks of the us-east-1 region.
- Compare your results with the equivalent AWS CLI commands.

References

[1] Yan Zhao et al. Facebook Multi-Node Server Platform: Yosemite V2 Design Specification V1.00. 2016. Open Compute Project. Accessed. June 2022. [Online]. Available: https://www.opencompute.org/documents/multi-node-server-platform-yosemite-v05

[2] Mind Drip Media. What's inside a Facebook Datacenter Open Compute Rack? Accessed. May 2022. [Online]. Available: YouTube. https://www.youtube.com/watch?v=2l6gI-ksdKs

[3] Felix G. Carapaica, BGP Glue of the Internet, Canada, 2022.

[4] We live in the cloud. 2022. Microsoft Azure. Accessed. May 2021. [Online]. Available: https://news.microsoft.com/stories/microsoft-datacenter-tour/

[5] Amazon Web Services. Our Datacenters. Accessed. June 2022. [Online]. Available: https://aws.amazon.com/compliance/data-center/data-centers/

[6] Microsoft Azure. Microsoft Azure Regions and Availability Zones. 2022. Accessed. June 2022. [Online]. Available: https://docs.microsoft.com/en-us/azure/availability-zones/az-overview

[7] Microsoft Project Olympus. 2017. Accessed. June 2022. [Online]. Available: YouTube. https://www.youtube.com/watch?v=nNS6GFpUUjE

[8] AWS. Regions and Availability Zones. Accessed. June 2022. [Online]. Available: https://aws.amazon.com/about-aws/global-infrastructure/regions_az/

[9] AWS. Global Infrastructure. Accessed. June 2022. [Online]. Available: https://aws.amazon.com/about-aws/global-infrastructure

[10] Baxtel. AWS US East (N. Virginia) data center market. Accessed. May 2022. [Online]. Available: www.baxtel.com/data-center/aws-us-east-n-virginia/

[11] Danilo Poccia. Now Open - Third Availability Zone in the AWS Canada (Central) Region. Accessed. May 2022. [Online]. Available: https://aws.amazon.com/blogs/aws/now-open-third-availability-zone-in-the-aws-canada-central-region/

[12] Google Cloud. Meet our network. Accessed. June 2022. [Online]. Available: https://cloud.google.com/about/locations#network

[13] AWS. AWS S3 Storage Classes. Accessed. June 2022. [Online]. Available: https://aws.amazon.com/s3/storage-classes/

[14] AWS. AWS CLI Command Reference. Accessed. May 2022. [Online]. Available: https://docs.aws.amazon.com/cli/latest/reference/ec2/describe-regions.html

[15] Products In Cloud Infrastructure and Platform Services Market. Gartner Reports. January 2023. [Online]. Available: https://www.gartner.com/reviews/market/cloud-infrastructure-and-platform-services

Chapter 3

Cloud Computing

Description

This section focuses on the virtualization of cloud resources in the cloud. It begins with the fundamentals of virtualization. Then it proceeds to describe the role of the hypervisor and its two main types. It compares the modes of full virtualization and paravirtualization. Once that the main concepts are provided, the section continues in a learning by doing style. AWS Academy is used to create instances and along this process, the components and properties of the AWS VMs are explained.

Learning Outcomes

- Deploy elastic virtual computers in a cloud networking environment.
- Configure webservers on elastic virtual computers.

Main concepts

- Cloud compute virtualization service.
- The hypervisors.
- Full virtualization and paravirtualization.
- The cloud virtual machine VM.
- Elasticity on demand.
- The VM image.
- The VM virtual chassis.
- Secure shell access to a virtual machine in the cloud.
- Basic firewall.
- Configuration of basic webserver.

Learning Activities

- Launching an EC2 instance in AWS cloud to explore all its components.
- Bootstrapping EC2 instances with User Data.
- Create EC2 instances using AWS CLI.
- Deploy EC2 instances using Python Boto3 script.
- Deploy a Windows Server 2019 on AWS EC2 instance.

Virtualized Computing

Virtualized computing is the most fundamental service of the Infrastructure as a Service (IaaS) cloud model. In fact, this is the first service that was ever offered by clouds such as AWS and Azure. Fundamentally, the clouds make available vast pools of computer processing power and memory that the clients can lease, on demand, to make virtual computers that function, by all accounts, like physical computers.

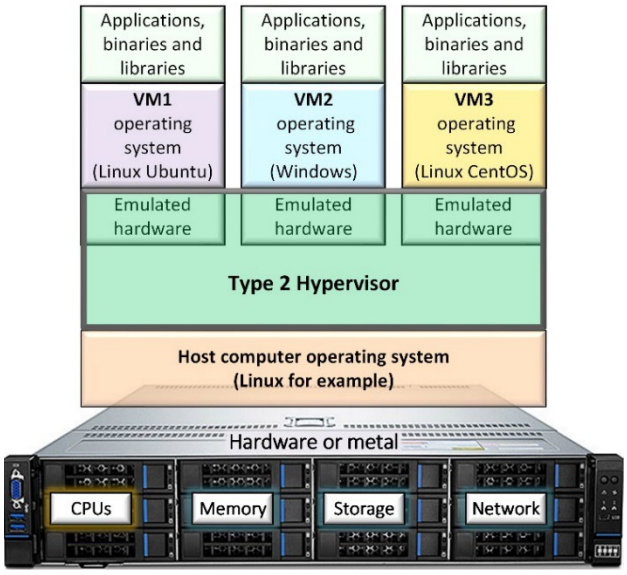
Currently, the IaaS clouds [1,2] have developed other models of computing beside virtual machines. That includes specific purpose services, serverless computing, containers and event triggered functions. Nevertheless, this section concentrates on the virtual cloud computer services of the cloud.

Virtual Machine

Generically, a **virtual machine (VM)** is the emulation of a computer device running on another machine. Even though the VM appears as a real computer to the user; it is mostly code running on the host. This is possible thanks to an intermediate translation layer between the virtual guest and the host machine. The **hypervisor** is the software layer [3,4] that enables the virtualization of computers. In general terms, the hypervisor intersects some of the guest requests for resource access and it handles the sharing of the host machine's CPU cycles and RAM memory. Also, the hypervisor provides environments that emulate networks, disks, and peripheral devices. Furthermore, the hypervisor allows the configuration of the network interfaces to grant access to external networks via the host computer. There are two main types of hypervisors, type 2 and type 1.

Type 2 Hypervisor

The kind of hypervisor that runs on an operating system [4] is called **hosted** or **type 2**. The next figure shows a host machine with the operating system Linux installed on it. The type 2 hypervisor needs the operating system to sustain three virtual computers sharing the host's physical resources.

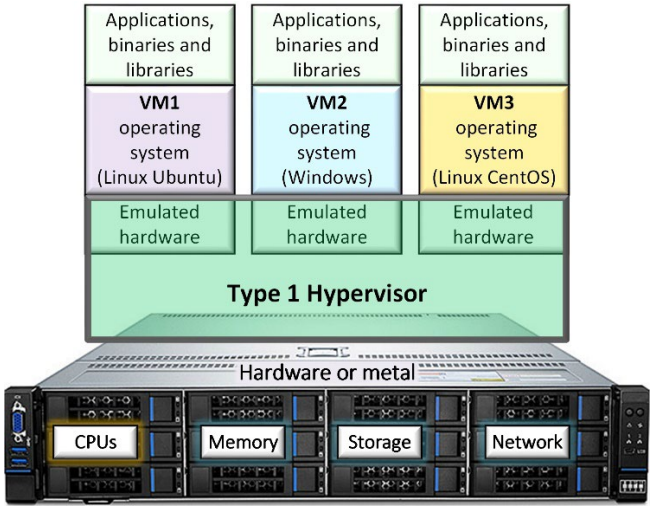


Type 2 Hypervisor

Examples of type 2 hypervisors are Oracle VirtualBox, QEMU and VMware Player. Typically, type 2 hypervisors are the correct fit for deployment of virtual machines on a personal computer and mid size enterprise. For a much larger scale, like the cloud, virtualization is taken to a different level.

Type 1 Hypervisor

This type of hypervisor operates directly on the server hardware without an underlaying operating system or a minimum operating system at the most. For that reason, a type 1 hypervisor [4] is also called **bare metal hypervisor**. VMware vSphere/ESXi, Red Hat RHEV are examples of type 1 hypervisors.



Type 1 Hypervisor

The cloud vendors use type 1 hypervisors to provide computer virtualization although they might use custom made or adapted hypervisors. The three largest providers, AWS, Microsoft Azure and Google Cloud [4,5,6] have so much engineering and economic power that they can design their own software and hardware. Even though they tend to be secretive with their internal infrastructure, the general knowledge is that:

- AWS ran **Xen hypervisor**[7, 9, 16] until 2017 when they switched into their own **Nitro Hypervisor**.
- Azure runs its own **Azure Hyper-V** (a version of Microsoft Hyper-V [1] specialized for cloud)
- Google Cloud [8] runs open-source **Kernel-based Virtual Machine (KVM)** hypervisor.

Running the hypervisor directly on the hardware allows the cloud provider to run virtualization more efficiently. The type 1 hypervisor turns the system hardware into a pool of logical computing resources that can be dynamically allocated to the VMs guest operating systems.

Virtualization modes, Full Virtualization

In full virtualization [10,11] mode, the hypervisor supports the guest (VM) operating systems without any modification. In such case, A hypervisor acts as a translation interface layer between the guest operating system and the metal. In principle, the hypervisor would have to completely emulate the hardware resources including the provisioning of virtual CPUs. But there is a problem with this approach.

In full emulation, the hypervisor directs calls from the applications running in the virtual machines to resources such as store, bios, and network cards. The hypervisor can do these tasks efficiently. Some instructions originated from the virtual machines can be executed directly on the hardware. In this case, the hypervisor just redirects those instructions to the underlay.

However, virtualization of the CPU by the hypervisor is an extremely difficult and intensive task. The instructions calls from the guest kernel operating system directed to the virtual CPU would have to be translated fast by the hypervisor. Beside that, there are kernel instructions from the guest operating system that can not be executed directly on the hardware because of privilege limitations for example.

This problem was an impediment for the development of the technology until CPU makers Intel and AMD took upon the task of making chipsets with extensions to do the hardware emulation of the processor rather than leaving that activity to the hypervisor. This simplified greatly the functioning of the hypervisor which now can concentrate on emulating disks, network, motherboard and PCI devices. This mode of full virtualization [6] is called **Hardware Virtual Machine** or **HVM**.

The benefit of full virtualization is that the guest operating system does not require any modification because it is decoupled from the hardware platform. The guest VMs are completely isolated from each other also providing a more secure platform.

Paravirtualization PV

Paravirtualization [6,8,9] on the other hand, involves the modification of the guest operating system kernel to replace privileged instructions with direct calls to the hypervisor. In essence, the operating system is aware that is running in a virtualized environment. Hence, the virtualization layer is leaner and more efficient because it does not need to do the translation of restricted calls. The downside of paravirtualization is that the guest operating system must be specially modified to run in the virtual environment. However, the user space applications and libraries do not require modification.

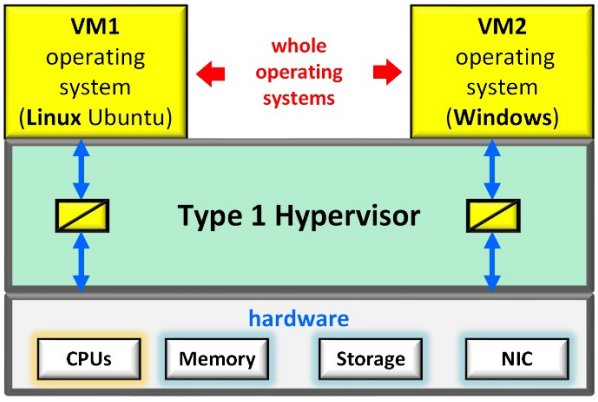
Knowing the difference between PV and HVM is particularly important if the cloud provider is AWS. Paravirtualization was initially preferred as the virtualization technology in the early clouds. Initially, AWS leaned toward paravirtualization of operating systems for the main reason that the original Xen hypervisor was designed to work with PV instances. Over time, improvements in the HVM technology and changes in the Xen hypervisor, erased the advantage that PV used to have. Lately, HVM has become the default recommendation for virtual machines in AWS.

All IaaS providers offer cloud computing services, but the names differ. AWS calls virtual computers **EC2 instances**. Microsoft Azure, Oracle Cloud and Digital Ocean just call them **virtual machines**. Google Cloud calls them **virtual machine instances**. In it fair to say that regardless the name, they are just virtual machines.

Operating Systems for the Cloud

By design, operating systems such as Linux and Windows are multi-user, multi-process and multi-purpose. Consequently, they are loaded with many tools and packages to support any kind of application. Furthermore, operating systems are designed to run on different hardware platforms. Therefore, they are loaded with a large number of drivers from chipset vendors. The more multi-functional an operating system is, the more complex and heavier it is. Knowing that, let's take a new look at virtualization but this

time emphasizing the aspect of the operating systems. The host computer and all the virtual machines run full operating systems even if they only use a fraction of their tools and packages.



Operating system in a type 2 hypervisor situation

Cloud providers take it a step further. They provide a menu of ready to use operating systems that the customer can choose from. These OS have been cleaned of unneeded packages and customized to run efficiently in the cloud. Some images of operating systems have been designed to work only with the particular cloud platform. An example of that is AWS AMI which is an AWS custom-made Linux based on components of open source CentOS and Fedora [17].

The main use case for cloud virtual machines is general purpose computing. This service offers a seamless migration path for customers who are lifting services from on-premises to the cloud. The VMs are designed to support any existing operating system. For an organization that has years of development on an operating system such as Linux, the natural transition is to move the applications to a similar environment. In the case of Microsoft software, the VMs can support Windows Server and all the roles that come with it including Active Directory. Thus, the network administration for an organization can be deployed completely in the cloud or in mixed mode on-prem and cloud.

There are other computing services available in the cloud such as serverless, containers, unikernel machines and event-triggered functions. These are aimed to be more cloud native solutions that require working mostly with code. This chapter does not cover those services, only the deployment of virtual machines in the cloud.

AWS Elastic Compute Cloud (EC2)

Elastic Compute Cloud (EC2) is the original and most fundamental AWS service. It is the realization of the cloud philosophy to offer **computing capacity on demand**. The **elastic** word is meaningful because it implies that this cloud service can be dynamically resized depending on the service requirements.

The fundamental EC2 unit is the **instance** which is a name for a virtual machine. The customer chooses the features of the VM from a menu like wizard. The main factors that are selected are the following: name, operating system, virtual chassis, optional storage, authentication access key, firewall protection, networking settings, payment format, and configuration data. Let's see each one these factors in detail by using an AWS Academy account.

Learning Activity

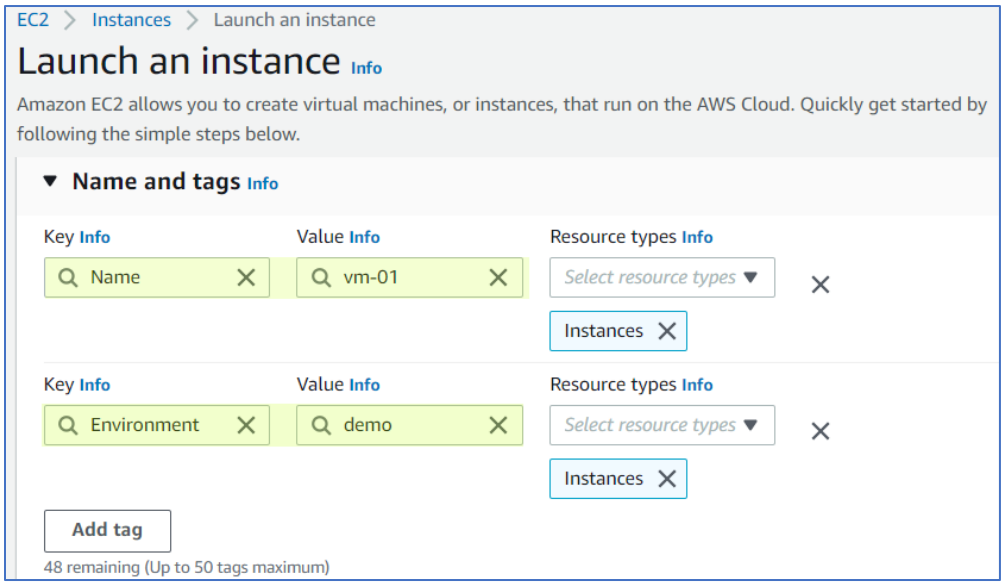
Launching an EC2 instance in AWS cloud to explore all its components

- Open your AWS Academy account and start a lab.
- Go into the AWS Console
- Search for EC2 service, go into the EC2 dashboard, instances, launch.

Note: the following examples use the older console mode because it shows more details.

Name

- The first step in the configuration of a new EC2 instance is to assign the name of instance and optional tags [12] like the environment of the VM in the organization.



It might sound trivial but assigning a proper name [12] to the VM is in reality very important and helpful for fleet operations. An organization that runs a large deployment of servers tends to group them according to their nature. For example, there are servers for staging, development, and production. Then a nomenclature is defined to name all the servers in staging with the prefix stage- followed by some characters. The servers in development could be named with the pattern dev-character and the ones in production could be prod-character. That results in names such as stage-01, stage-02, dev-01, dev-03, prod-01, prod-02, prod-03, and so on.

Having easy-to-follow patterns is a goal of any organization that runs many servers because it makes automation of repetitive tasks possible. There are tools such as Ansible, Puppet, Chef, Salt and Python which are used to write automation scripts for tasks such as retrieving information, updating configurations and installing packages on the servers. Hence, a (DRY) script with a prod-wildcard target can update all the production servers in one action instead of going manually server by server. This use case demonstrates that although the name tag of the VMs may appear unimportant, it is actually very useful. For that reason, organizations define naming nomenclatures for their compute resources.

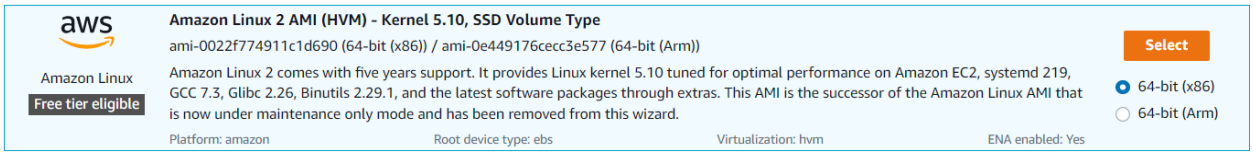
Image

An Amazon Machine Image (AMI) is the pre-formatted software needed to launch an EC2 instance of a particular flavour. An image is a template containing the operating system, server packages and applications. Instead of installing a generic operating system (which is possible as well) and then downloading and patching the packages, the customer can just choose the desired image from a menu. Let's take a look at the options:

Quickstart AMIs (45)	My AMIs (2)	AWS Marketplace AMIs (5622)	Community AMIs (500)
Commonly used AMIs	Created by me	AWS & trusted third-party AMIs	Published by anyone

- Quickstart AMIs are those images pre-built by AWS. They are optimized to work in this cloud.
- My AMIs are images created and saved by the customer.
- AWS Marketplace AMIs are images created by AWS and trusted third party participants.
- Community AMI are images made public by other contributors.

Under Quickstart AMIs, the first option that appears (pre-selected) is **Amazon Linux 2 AMI**:

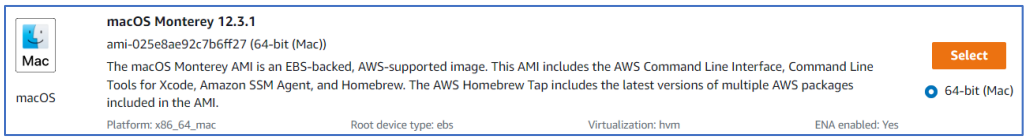


This is an image specially made by AWS to run efficiently in this cloud. It has the following characteristics:

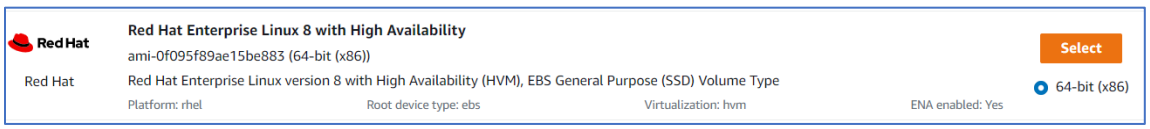
- This image is free tier which means there are no licensing or usage costs for 750 hours a year.
- Observe that it has a unique identification string (ami-0022f774911c1d690 in May 2022). This id string is Region-dependent, and it changes when the image is updated by AWS. This is important to know because programs that use the value to create EC2 instances should not be hardcoded, but rather use a variable to enter the value.
- It is fair to call this image AWS Linux because of the modifications done to the base Linux OS.
- It is designed for Hardware Virtualization Machines (HVM) or full virtualization.
- It runs on two choices of microprocessor: Intel x86_64 and ARM 64.
- The image boots from a root device volume which is in an Elastic Block Storage (EBS).

Quickstart AMIS contains many images to choose from. Let's explore a few:

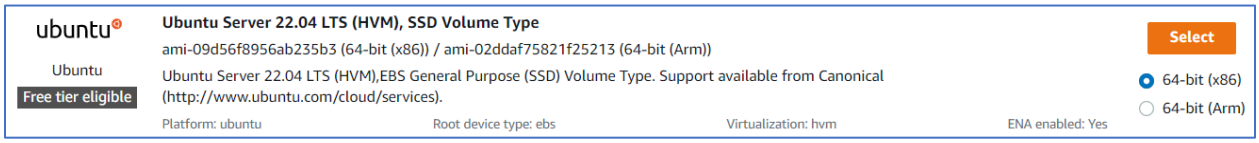
- MacOS



- Red Hat Linux

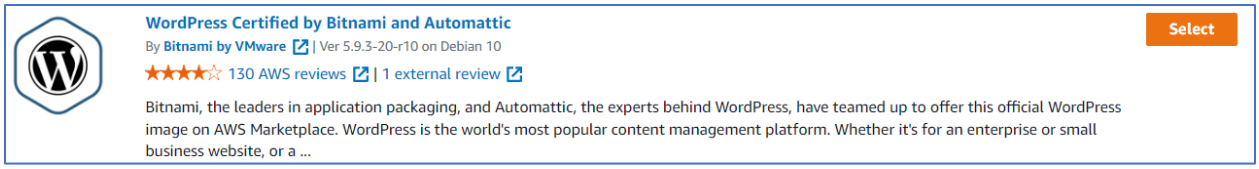


- Ubuntu Linux Server

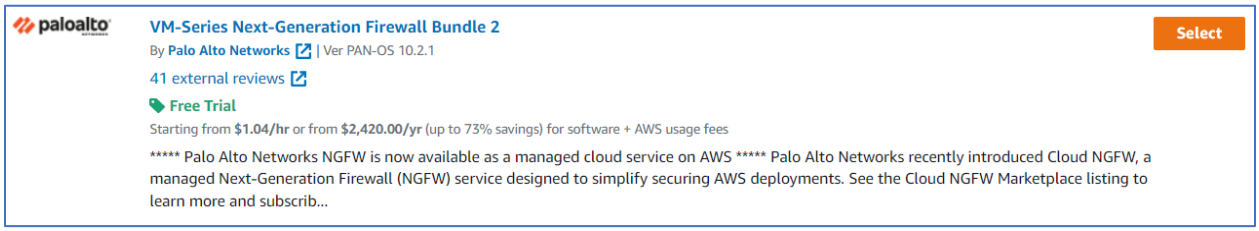


- Microsoft Windows Server

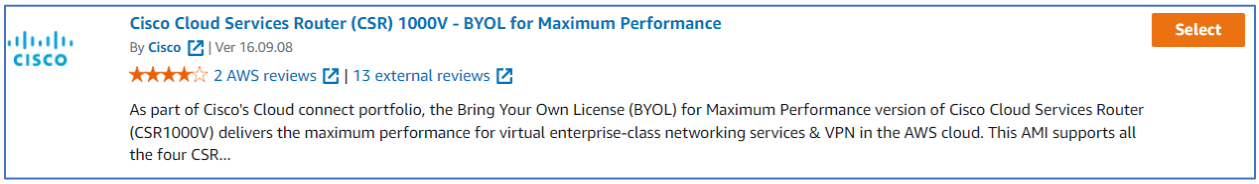
AWS Marketplace is where vendors and companies make their images available for lease. For example, this is an image of a **WordPress webserver** created by two companies.



- A **firewall image** by Palo Alto costs $1.04/hour or $2,420/year to lease (aside the AWS costs)

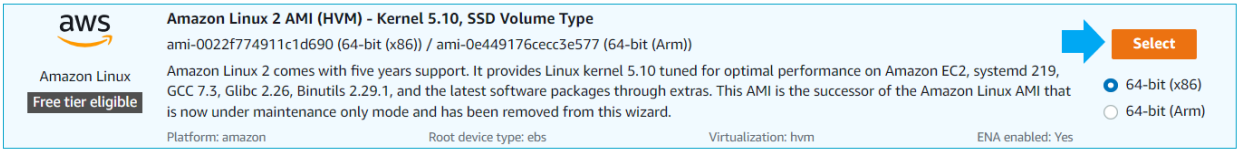


- An image for a **Cisco router**. The use case of the firewall above and the router image below is when a customer wants to use the same devices that the organization have been using on premises.



After exploring some of the choices, let's continue with the EC2 deployment.

- **Choose the image AMI** (Amazon Machine Image) **to create an EC2 instance**
- Choose the Free Tier AWS AMI Linux .



Type

The type is an abstraction of the hardware requested from the cloud pool of resources. The type can be seen as a virtual chassis model with allocation of vCPU, memory, storage, and networking. When the customer launches an EC2 instance, the cloud initiates an internal procedure to provision the virtualization of the resources that matches the selected instance type. AWS provides a menu of possible configuration sets to choose (CPU, amount of memory, storage, and networking).

- The instance type is part of the total cost. More resources, more expensive.
- The instances are classified according to the use cases:

Instance	Requirement
General Usage	Generic compute, memory, storage, and networking capacity.
Compute power	Faster processor and network access for web server apps and machine learning.
Memory	Higher memory available for data analysis and database operations.
Accelerated	3D graphics, visualizations, high-end financial analysis.
Storage	Large storage volumes with fast data access.

- The instances are denoted with letters according to their intended use:

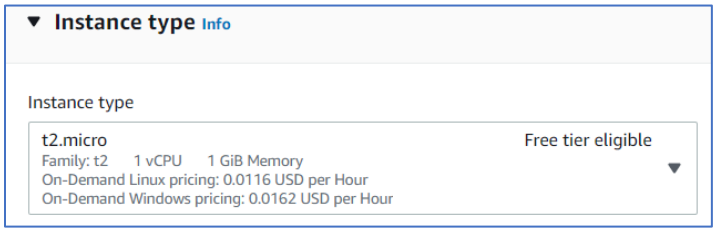
Instance	**Prefix letter**	**Types examples**
General Usage	t and m	t2, t3, m4, m5
Compute power	c	c4, c5
Memory	x and r, z	x1, r4, r5, z1d
Accelerated	f, p, g	f1, p2, p3, g3
Storage	h, i, d	d2, i3, h1

- The following table summarizes some of the types.
- Notice the relation between resource capacity and price.

Instance type	vCPUs	Architecture	Memory (GiB)	Storage (GB)	Storage type	Network (Gbps)	On-Demand Linux pricing USD per hour	On-Demand Windows pricing USD per hour
t2.nano	1	i386, x86_64	0.5		-	Low to moderate	0.0058	0.0081
t2.micro	1	i386, x86_64	1		-	Low to moderate	0.0116	0.162
t2.small	1	i386, x86_64	2		-	Low to moderate	0.023	0.032
t2.medium	2	i386, x86_64	4		-	Low to moderate	0.0464	0.0464
t2.large	2	x86_64	8	-	-	Moderate	0.0928	0.1208
t2.xlarge	4	x86_64	16	-	-	Moderate	0.1856	0.2266
t2.2xlarge	8	x86_64	32	-	-	Moderate	0.3712	0.4332
c1.medium	2	i386, x86_64	1.7	350	hdd	Moderate	0.13	0.21
c1.xlarge	8	x86_64	7	1680	hdd	High	0.52	0.84
d2.xlarge	4	x86_64	30.5	6144	hdd	Moderate	0.69	0.821
d2.2xlarge	8	x86_64	61	12288	hdd	High	1.38	1.601
f1.16xlarge	64	x86_64	976	3760	ssd	25	13.2	-
g2.2xlarge	8	x86_64	15	60	ssd	Moderate	0.65	0.767
x1e.32xlarge	128	x86_64	3904	3840	ssd	25	26.688	32.576
u-3tb1.56xlarge	224	x86_64	3072	-	-	50	27.3	37.604
u-12tb1.112xlarge	448	x86_64	12288	-	-	100	109.2	129.808
z1d.metal	48	x86_64	384	1800	ssd	25	4.464	6.672

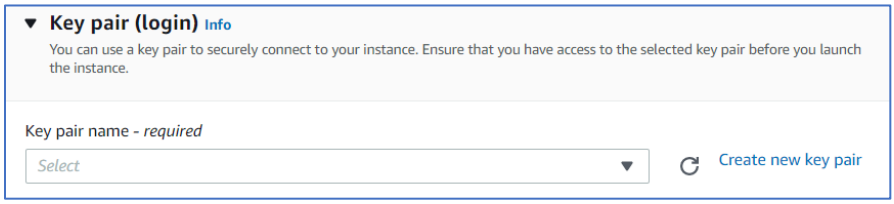
- The general use t2.micro with 1 virtual CPU, one GiB (Gibi-byte = 2^{30} bytes) of memory, and mid networking velocity, costs more than a cent per hour if the operating system is Linux.
- On the other hand, the monster u-12tb1.112xlarge with 448 vCPUs, 12,288 GiB of memory, network velocity of 100 Gbps, running Linux, costs US$ 109.2 per hour. That is US$ 956,592 per year without including the costs for image and licenses.

- Choose the free tier **t2.micro** (it is selected by default already)

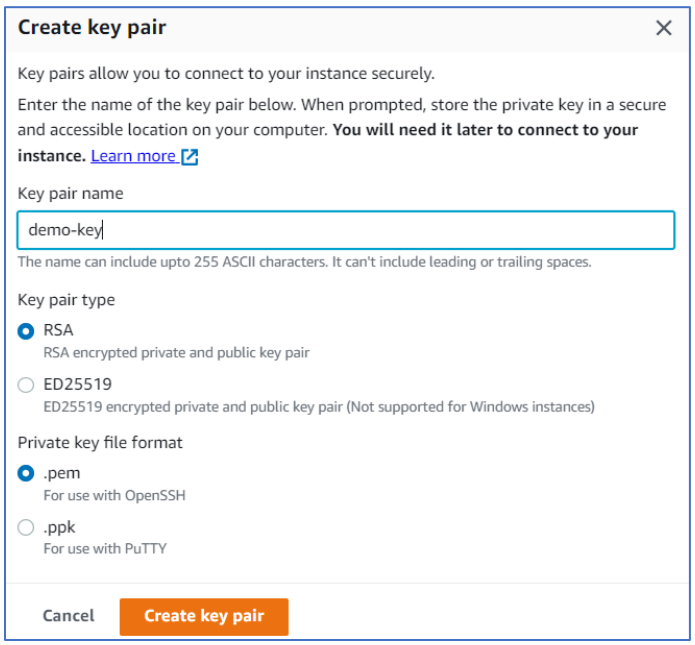


Login Authentication Key Pair

This is the key used to authenticate the access to the EC2 instance using secure shell (SSH) protocol. When the client attempt to login into the VM, a public-private key pair is used to prove authenticity.



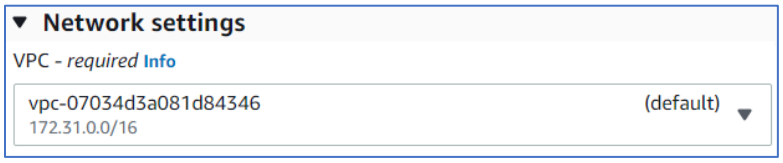
The AWS management console offers two options, either to reuse a key pair or to create a new one. In this example, a new key pair is created. By default, the cipher-algorithm is RSA (Rivest, Shamir, Adleman) and file type is PEM (Private Enhancement Mail). There is an optional algorithm ED25519 (Edwards-curve Digital Signature Algorithm) which is not usable with Windows computers. The file type PPK (PuTTY Private Key) is only used by the client's tool PuTTY.



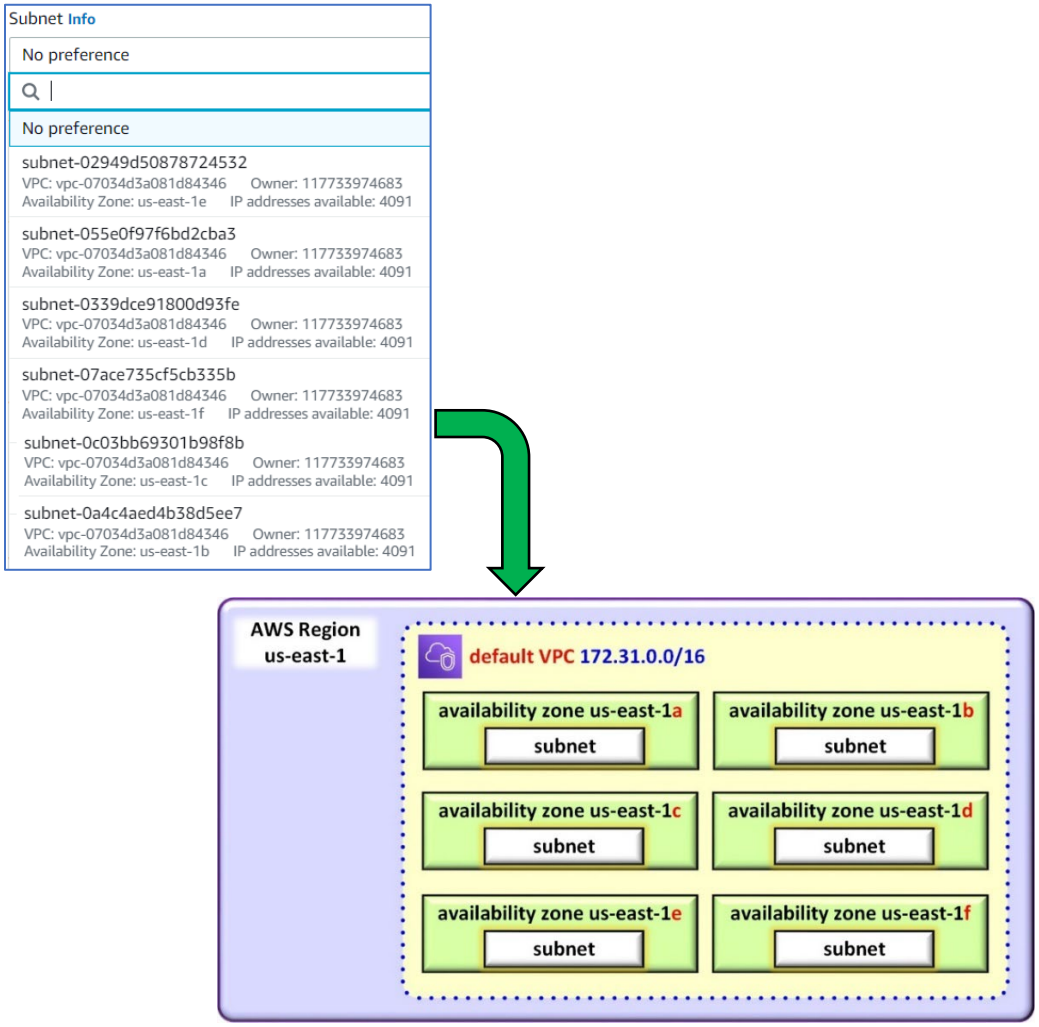
When the new key is created, a PEM file with the private key is automatically downloaded into the customer computer. It must be saved in a secure location. The public key remains in the EC2 instance. The system of secure access is the topic for another section where it is described in detail.

Network settings

Every AWS customer gets assigned a virtual private cloud (VPC). This is a virtual space in the cloud that belongs to the client to deploy its infrastructure. By default, the virtual space is assigned a private IPv4 address range 172.31.0.0/16. All the clients get assigned the same IPv4 range, but there is no conflict because there is address translation between public and private space in the same fashion that there is address translation for home Internet. This topic is also covered in greater detail in another section.



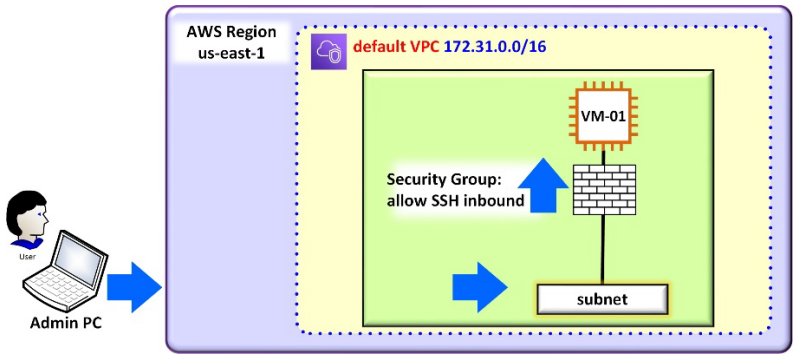
The default VPC, in the us-east-1 Region, contains six (6) default subnets. AWS places each subnet in different availability zones. That means that each subnet is provisioned in a different cluster of datacentres to ensure reliable access. Notice that, initially, each subnet has 4091 IPv4 addresses available. The following diagram shows a general view of the default VPC and its subnets.



The Virtual Private Cloud and the six default subnets and availability zones.

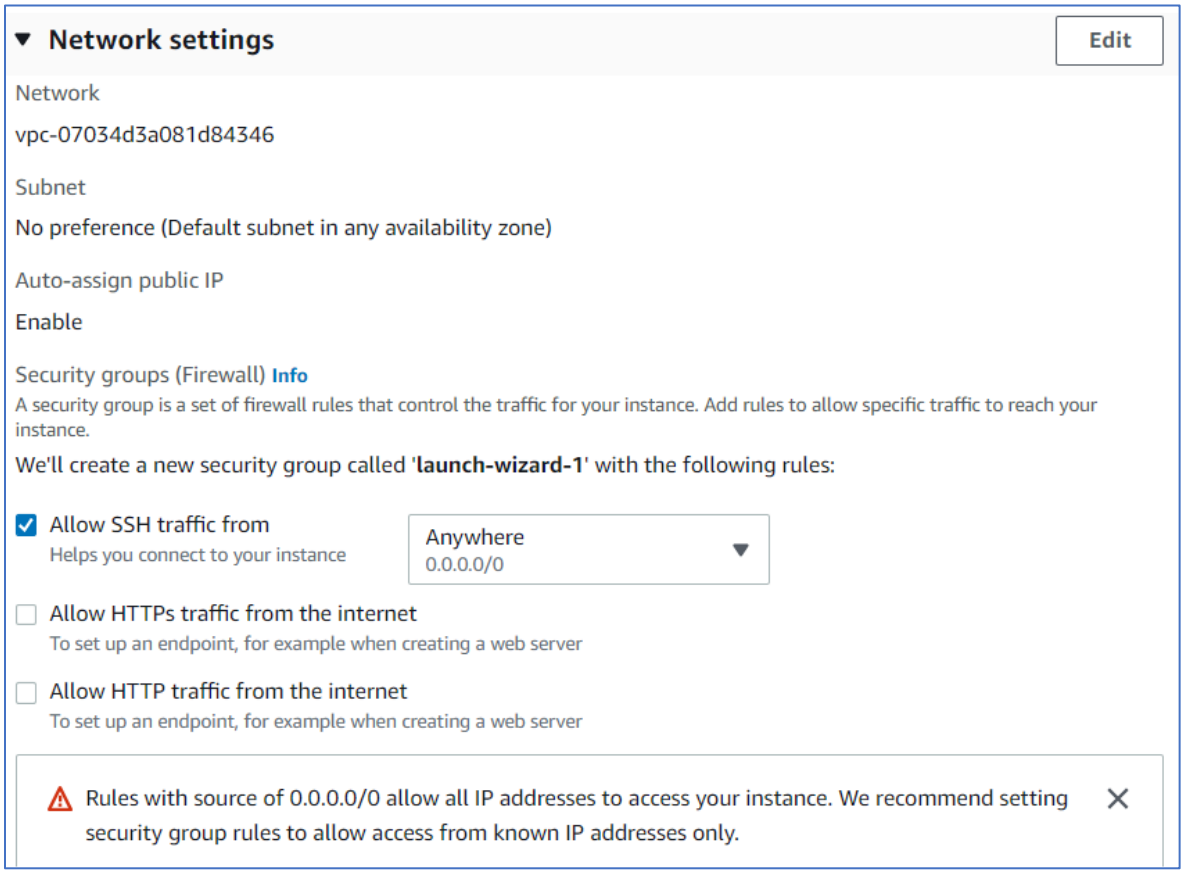
Security Group

A security group is simple firewall. It contains rules to allow access to the EC2 instance. The rules establish the source IP addresses that are allowed to access the resource and the destination protocols (the server applications) that can be accessed. By default, the only rule present in a new security group is to allow access using the SSH protocol from any address.



Security group firewall allowing only SSH inbound to the EC2 instance.

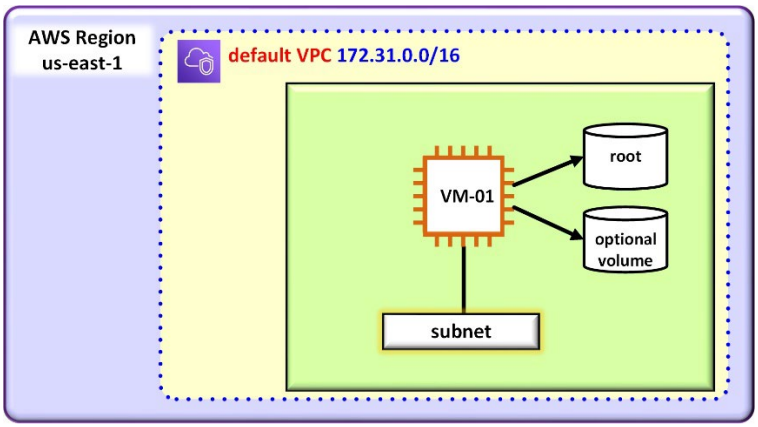
This rule makes sense because if administrative access were not allowed, there would be no way to manage the VM. Other rules can be added the time of creation or later such as allowing access to HTTP and HTTPS protocols by editing the security group. However, for now, the security group will be configured with access via SSH only (to prove a point during this demonstration).



Storage

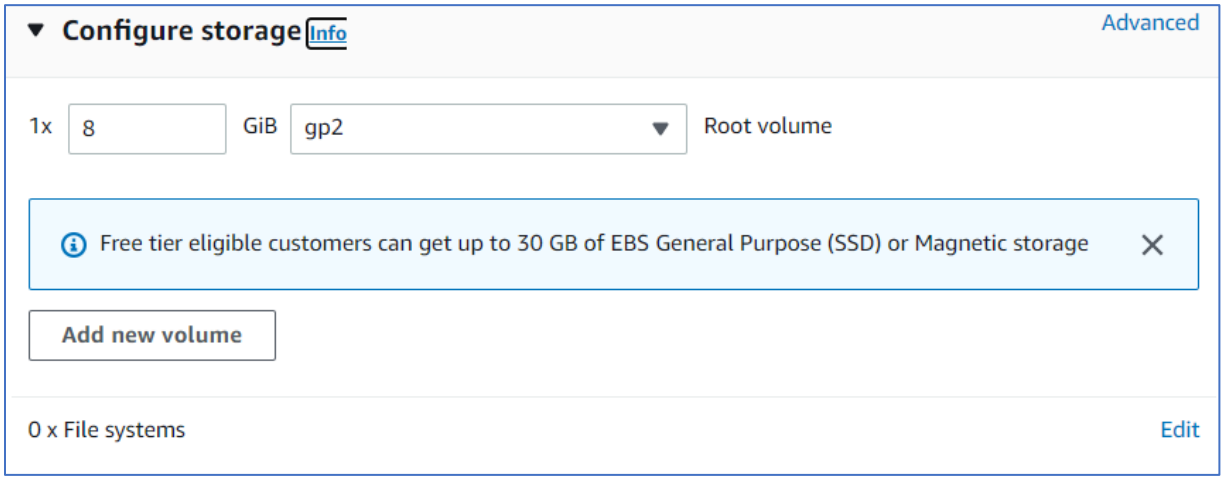
The new EC2 instance requires a minimum capacity to store the image with the operating system. When the EC2 instance is launched, it boots the image up from a root volume. This volume is provisioned by **AWS Elastic Block Store (EBS)**. Depending on the instance type, other EBS volumes can be optionally added for file systems and data. EBS is persistent storage so when the EC2 instance is stopped and restarted, the data will still be available.

Block storage is a technology that breaks data into blocks, assigns unique identifiers and then stores those blocks in storage servers across the datacentre. A dedicated Storage Area Network (SANs) is used to access the blocks quickly. When data is requested, the underlying storage system reassembles the data blocks hence from the viewpoint of the instance it looks like a storage device is attached to it.



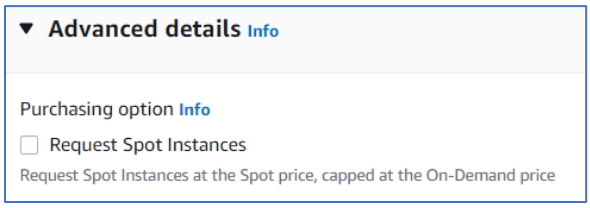
Representation of an EC2 instance with the EBS root volume and an optional EBS volume.

- By default, a t2.micro type is assigned 8 GiB of EBS root volume store.
- The default EBS volume type for AWS EC2 instances is **g**eneral **p**urpose **gp**2.
- The gp2 volumes are backed by solid-state drives SSD (flash memory).
- Other SSD device types are **io**1 and **io**2 for very high **i**nput/**o**utput accessing.
- Hard disk drives (HDD) are offered for low cost, large volumes (types st1 and sc1)
- Added new volumes require to be mounted within the operating system.



Advanced Details

Spot Instances

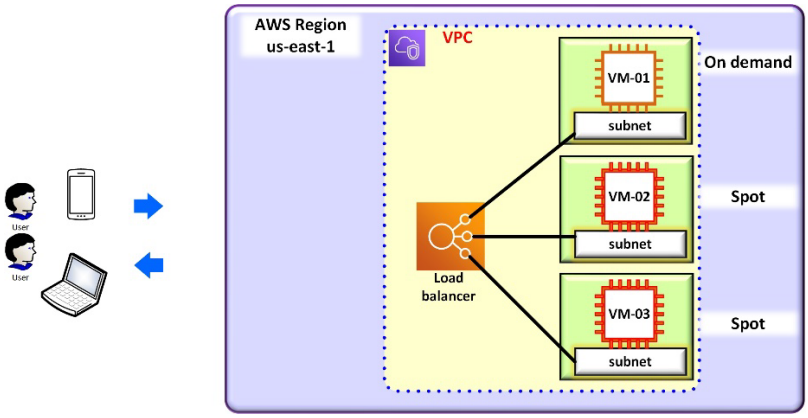


EC2 instance can be deployed in two availability modes: **on-demand** and **spot** [14]. When an EC2 instance is set up in the on-demand mode, it is always available, but it costs more money. The on-demand model is adequate for long running workloads. The spot option offers a reduced cost for applications that do not need to run all the time and applications that run processes over time periods.

A spot instance, on the other hand, runs only when there is enough idle EC2 capacity in the cloud. AWS continuously adjust a refence **spot price** depending on the supply and demand for spot resources. The customer sets a price that is willing to pay for and if that is higher than the current AWS spot price, then it gets the resources. However, the customer might lose the resource if beaten by another customer better offer or if the reference spot price increases.

Why would anyone want to use a system that does not guarantee the instance? There are several reasons. The spot pricing system offers deep discounts of up to 90% off the on-demand price, so that makes it attractive to run applications that can be interrupted such as data analysis, and batch jobs processing.

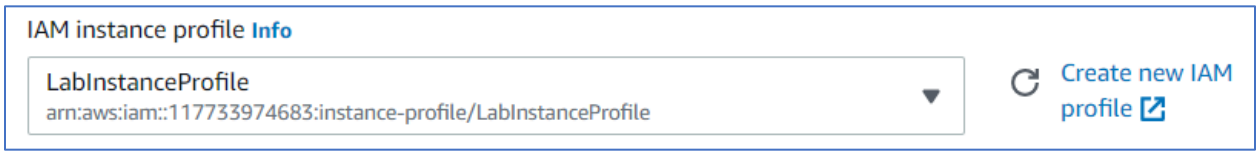
Another interesting usage is in elastic service deployment. Let's assume that a webserver application runs with a base topology of one EC2 instance on demand. Occasionally, there is a significant increase in traffic that this minimum setting can not take, therefore resulting in the degradation of the quality of the user experience. To overcome this problem, a group of three identical EC2s up is set up with a load balancer acting as the proxy access. This is possible because an image of the base server has been prepared (and stored in the Image, My AMIs folder). The base EC2 (VM-01) is configured on demand so it is always running to guarantee a minimum service level. The other two EC2s are configured on spot, if the price is right there will be deployed, otherwise, they are not, thus saving money. This deployment is called an auto-scaling group and it will be implemented in another section.



Auto scaling group

IAM Role

An Identity and Access Management (IAM) role [15] is a set of permissions to control access to other AWS resources. For example, an application running in the EC2 instance may need to access data stored in another AWS service. To do so, the application makes (Application Programable Interfaces API) calls that must be accepted by the receiving end. The user must write an IAM role policy to assigns the permissions required to make the calls. In this demonstration, there is no need to set up or use an IAM role; however, AWS Academy has a predesigned role called LabInstanceProfile or LabRole that can be used in many cases.



Instance auto-recovery

The EC2 instances auto-recover when their system status checks fail.

Shutdown behavior

An EC2 can be set to stopped or terminated when it is shutdown.

Stop - Hibernate behavior

In this case, the instance goes into hibernation meaning that the content of the RAM is saved into the root volume.

Termination protection

When this option is enabled, there is no way to terminate an instance until the condition is disabled.

Detailed CloudWatch monitoring

CloudWatch is an AWS monitoring service that collects information from the resource's status. CloudWatch collects metrics such as CPU and memory utilization, I/O writes to a database, networking usage and many others.

Elastic GPU

AWS Elastic Graphics is to attach graphics acceleration to the EC2 instance. Not all types are supported.

Elastic inference

Amazon Elastic Inference (EI) is an additional resource to accelerate the CPU for deep learning (DL) inference workloads.

Credit specification

This is a CPU credit specification which is only available for T2, T3, and T3a instances. The nature of some applications is that sometimes they are running at low utilization and in some other times, they need to burst CPU cycles to deal with heavy requests. When the EC2 instance is running idle, it accumulates CPU credits up to a permitted baseline set by AWS. When the need arises, the EC2 receives the accumulated CPU credits to burst them until they are depleted.

Placement group name

A placement group is a grouping of VMs all located nearby to reduce latency and to maintain high data bit rates. There are three formats, cluster, spread, and partition. A **cluster** placement group is when the group is in the same availability zone. A **spread** placement group guarantee that the instances are created in different hardware. Finally, the **partition** placement group places the instances in different disk partitions.

EBS-optimized instance

This option enables extended access to Elastic Block Storage EBS.

Capacity reservation

This is to reserve capacity for future expansion of the EC2 instance. This causes additional costs.

Tenancy

There are two types of tenancy: **default** and **dedicated**. The default mode is that resources are provisioned on shared hardware with other tenants. The client can request exclusively assigned hardware by paying a premium. The main driver of exclusive tenancy is typically government regulations.

RAM disk ID

Optional. A RAM disk drive is a way to use main system memory as a block device

Kernel ID

The customer can choose among available operating system kernels.

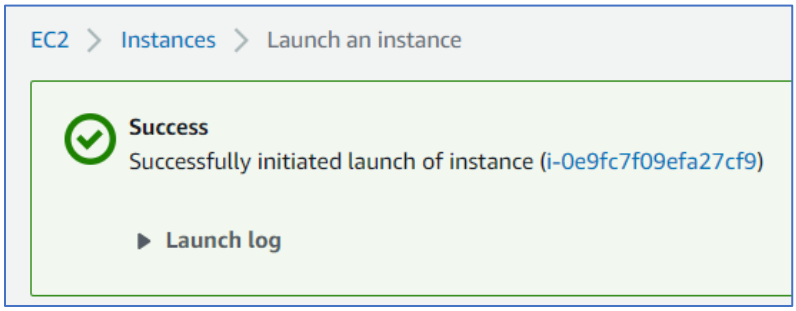
License configurations

The AWS License Manager service is associated with the instances at launch to enforce license compliance.

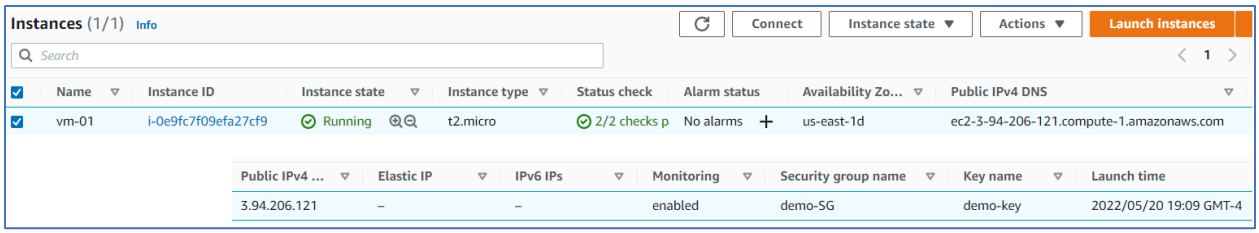
User data

This is a very helpful field. The user data provides a space to write a configuration script to bootstrap the launching of the instance. For example, a shell script can contain the commands to update, upgrade and download application packages and start a service. This field will be used in a second example in this guide, right now, it will be left blank.

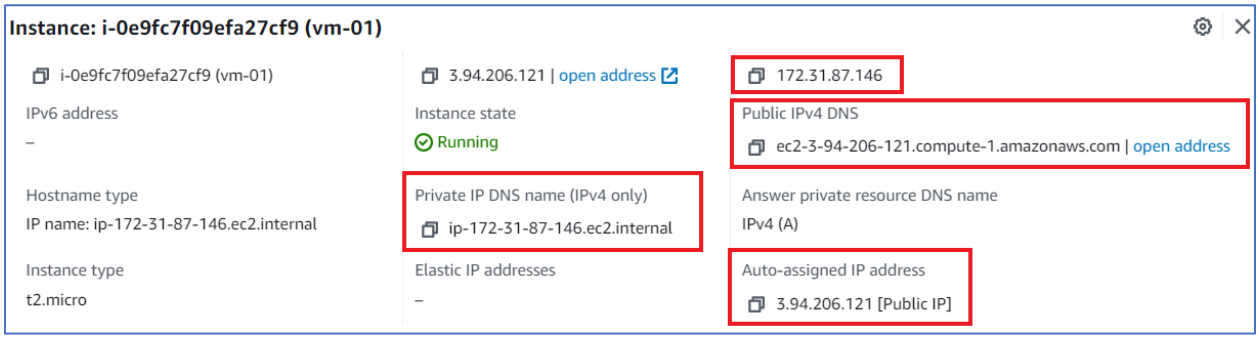
Proceed to launch the new instance.



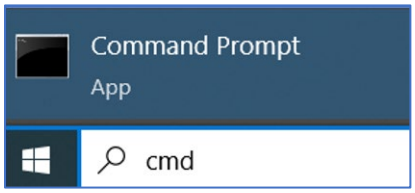
Observe the new instance in the EC2 instance dashboard.

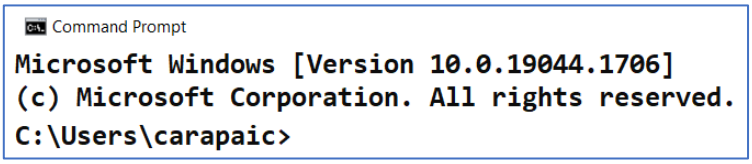


- The EC2 got a private IPv4 address 172.31.87.146 (for VPC internal use).
- A private DNS name ip-172-31-87-146.ec2.internal (for VPC internal use).
- A **public** IPv4 address **3.94.206.121** (reachable from the Internet).
- A public DNS name **ec2-3-94-206-121.compute-1.amazonaws.com** (Internet reachable).



Open the CMD tool (for Windows) or any application (like putty) that allows to use SSH.





Login into the EC2 instance using the SSH protocol

- The username for AWS AMI Linux is (always) **ec2-user**.
- (The username is ubuntu for a Linux Ubuntu image).
- The destination is either the public IPv4 address or the public DNS name.
- The authentication is done with the stored PEM file (already downloaded is previous step).
- The location of the PEM file is indicated by using the flag – i.

```
ssh ec2-user@3.94.206.121 –i C:Users\name\Downloads\demo-key.pem
```

Note: in real life, do not leave the pem file in an exposed place such as Downloads.

Note: typical causes for connection errors are a) white spaces in the path to the key destination, b) ssh application does not have rights to access the key, c) laptop's firewall blocks use of key. Solutions: a) enclose discontinued path with " ", b) assign rights to access the key, c) change firewall settings.

Proceed to login

```
ssh ec2-user@3.94.206.121 –i C:\Users\name\Downloads\demo-key.pem
The authenticity of host 'ec2-3-94-206-121.compute-1.amazonaws.com (3.94.206.121)' can't be established.
ECDSA key fingerprint is SHA256:mZC41x3VpwO3PbI1pLXTA8l1s9mvS/yYVh6EHfqFy64.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-94-206-121.compute-1.amazonaws.com' (ECDSA) to the list of known hosts.
Last login: Fri May 20 23:38:23 2022 from d226-76-239.home.cgocable.net
     __|  __|_  )
     _|  (     /   Amazon Linux 2 AMI
    ___|\___|___|
https://aws.amazon.com/amazon-linux-2/
2 packages for security, out of 5 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-87-146 ~]$
```

The welcoming message says that the system needs to be updated with the command **sudo yum update**.

- **sudo** (superuser do) gives super user privilege to the default user ec2-user to execute commands.
- **yum** (**y**ellowdog **u**pdater **m**odified) is an open-source package management tool for RPM (RedHat Package Manager) based Linux systems. It is used to update, install and remove software.
- Finally, **update** get the packages to the latest versions located in the Linux repositories. A Linux repository is a storage location from where the OS updates and applications are downloaded. Linux machines know where to find them because they come with a file the DNS names of the repositories. Since this is AWS Linux, its home URL is https://amazonlinux.com.

Proceed to update the operating system with the latest packages

```
[ec2-user@ip-172-31-87-146 ~]$ sudo yum update -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd, amzn2-core
Resolving Dependencies
--- > Running transaction check
--- > Package curl.x86_64 0:7.79.1-1.amzn2.0.1 will be updated
```

- Now, the EC2 instance is completely updated. Let's try some Linux commands.

Verify the hostname.

- Notice that the hostname format is: **ip-private-IPv4 addres.ec2.internal**.

```
[ec2-user@ip-172-31-87-146 ~]$ hostname
ip-172-31-87-146.ec2.internal
[ec2-user@ip-172-31-87-146 ~]$ cat /etc/hostname
ip-172-31-87-146.ec2.internal
```

Find out the release of the operating system.

- AWS Linux 2 is an operating system based on CentOS, Red Hat Linux, and Fedora.

```
[ec2-user@ip-172-31-87-146 ~]$ cat /etc/os-release
NAME="Amazon Linux"
VERSION="2"
ID="amzn"
ID_LIKE="centos rhel fedora"
VERSION_ID="2"
PRETTY_NAME="Amazon Linux 2"
HOME_URL="https://amazonlinux.com/"
```

Test external reachability by pinging an IPv4 address in Google (8.8.8.8)

```
[ec2-user@ip-172-31-87-146 ~]$ ping 8.8.8.8
64 bytes from 8.8.8.8: icmp_seq=1 ttl=107 time=0.958 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=107 time=0.992 ms
```

Test external reachability by pinging a public DNS name (Sheridan's website name)

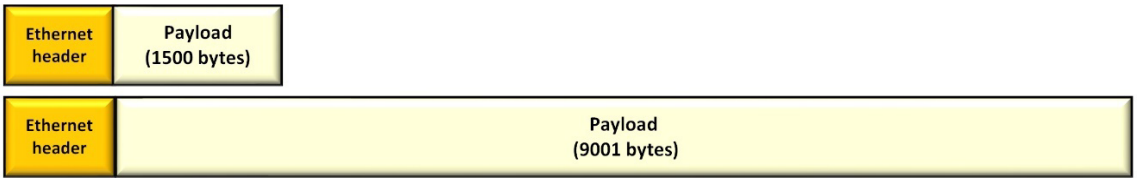
```
[ec2-user@ip-172-31-87-146 ~]$ ping www.sheridancollege.ca
64 bytes from www-traf.sheridanc.on.ca (142.55.7.60): icmp_seq=1 ttl=42 time=22.9 ms
64 bytes from www-traf.sheridanc.on.ca (142.55.7.60): icmp_seq=2 ttl=42 time=23.0 ms
```

Find out the interface networking information with the **ifconfig** command.

Note: the output has been edited for more clarity.

```
[ec2-user@ip-172-31-87-146 ~]$ ifconfig
    inet 172.31.87.146  mask 255.255.240.0  broadcast 172.31.95.255 (private IPv4 address)
    inet6 fe80::10f8:7eff:fe47:e835  prefix length 64  (Link Local IPv6 address)
    ether 12:f8:7e:47:e8:35  (MAC address Ethernet)
    MTU 9001 Bytes. (Maximum Transmission Unit)
```

- The private IPv4 address is 172.31.87.146 with a mask of 255.255.240.0.
- A link local IPv6 address is autoconfigured (fe80::10f8:7eff:fe47:e835 /64)
- The MAC address of the interface is 12:f8:7e:47:e8:35.
- To be noted is that the **maximum transmission unit (MTU)** is 9001 bytes. This means than the Ethernet frames can carry 9001 bytes of data in their payload. (The standard default is 1500 bytes). The larger payload increases the efficiency of data delivery in the data centre.



Comparison of the payload (MTU) size of default Ethernet frame versus Extended MTU frame.

Installation of webserver

Convert this EC2 instance into a webserver.

```
[ec2-user@ip-172-31-87-146 ~]$ sudo yum install httpd -y
```

- The package manager yum handles the installation of httpd by contacting the repositories.
- The word **httpd** indicates the software that is going to be installed. In this case, httpd is an open-source web server software released by the Apache Software Foundation.

Test the webserver functioning using the curl command.

- If the webserver is functioning, a reply will be obtained from the curl directed to an IPv4 address assigned to the machine. The EC2 instance has two local addresses.
- The address **127.0.0.1** is the universal **loopback address**. All computers this local address.
- The EC2 instance automatically got the private IPv4 address 172.31.87.146 from AWS.

```
[ec2-user@ip-172-31-87-146 ~]$ curl 127.0.0.1
curl: (7) Failed to connect to 127.0.0.1 port 80 after 0 ms: Connection refused
[ec2-user@ip-172-31-87-146 ~]$ curl 172.31.87.146
curl: (7) Failed to connect to 172.31.87.146 port 80 after 0 ms: Connection refused
```

- The test failed! Why?

Find out the status of the service.

```
[ec2-user@ip-172-31-87-146 ~]$ systemctl status httpd
```

- The newly configured **service must be restarted** with the following command.

```
[ec2-user@ip-172-31-87-146 ~]$ sudo systemctl restart httpd
```

Verify the status of the application.

```
[ec2-user@ip-172-31-87-146 ~]$ systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)
   Active: active (running) since Sat 2022-05-21 00:01:34 UTC; 27s ago
May 21 00:01:34 ip-172-31-87-146.ec2.internal systemd[1]: Starting The Apache HTTP Server...
May 21 00:01:34 ip-172-31-87-146.ec2.internal systemd[1]: Started The Apache HTTP Server.
```

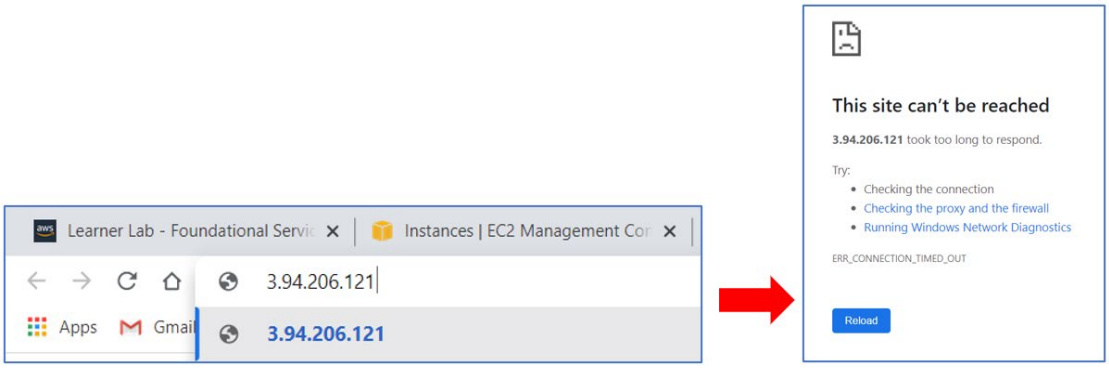
Test again with the curl command:

```
[ec2-user@ip-172-31-87-146 ~]$ curl 127.0.0.1
<head>  <title>Test Page for the Apache HTTP Server</title>
<p>This page is used to test the proper operation of the Apache HTTP server after it has been installed. If you can read this page, it means that the Apache HTTP server installed at this site is working properly.</p>
```

```
[ec2-user@ip-172-31-87-146 ~]$ curl 172.31.87.146
<head>  <title>Test Page for the Apache HTTP Server</title>
<p>This page is used to test the proper operation of the Apache HTTP server after it has been installed. If you can read this page, it means that the Apache HTTP server installed at this site is working properly.</p>
```

- The command returns show the default test page of Apache webservers. Both tests were successful.

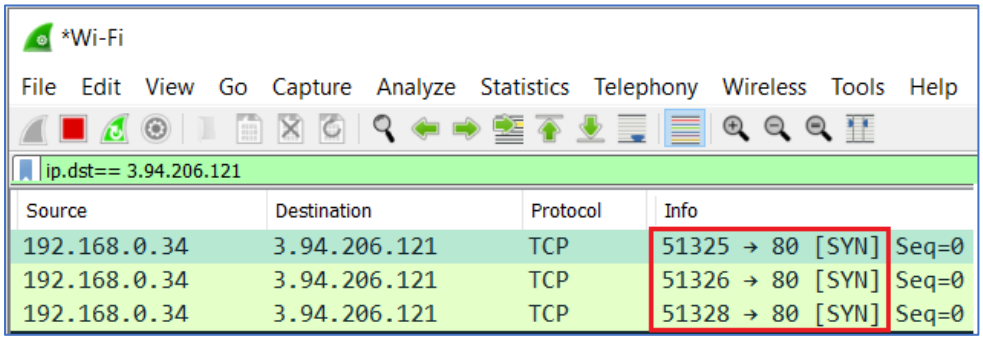
Test the webserver using a web browser and the public IPv4 address of the EC2 instance.



- It failed! **Why?**

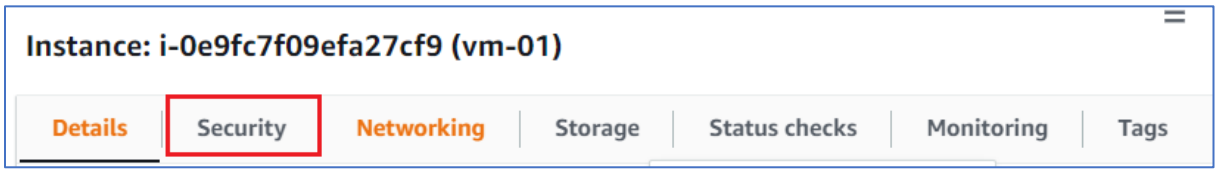
Troubleshoot using the traffic sniffer Wireshark.

- The expression **ip.dst==3.94.206.121** is a **filter** that only catches the desired IP packets.

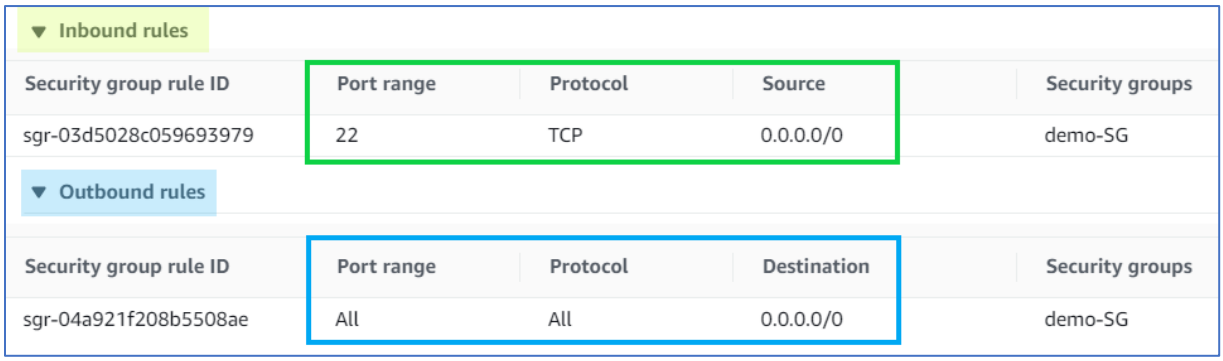


- The sniffed traffic shows that the local laptop (192.168.0.34) attempts to start a conversation with the remote destination 3.94.206.121.
- The destination TCP port is 80 which is the universally reserved port number for the HTTP protocol.
- The message sent is TCP SYN which means "open the port on your side".
- The local computer repeats the same message several times.
- But the other side does not reply. Why?
- The HTTP protocol was tested with a curl command internally and with the command systemctl status. They both yield positive results.

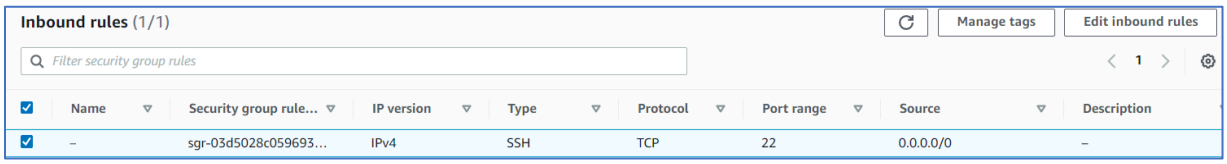
- The webserver is running and the message to open the conversation is being sent.
- What could be the reason that the webserver does not reply?
- Is there a component that could be blocking the incoming HTTP messages? The firewall.
- Let's review the firewall or the security group.
- In the EC2 dashboard, select the VM-01, under security.



- These are the current rules:

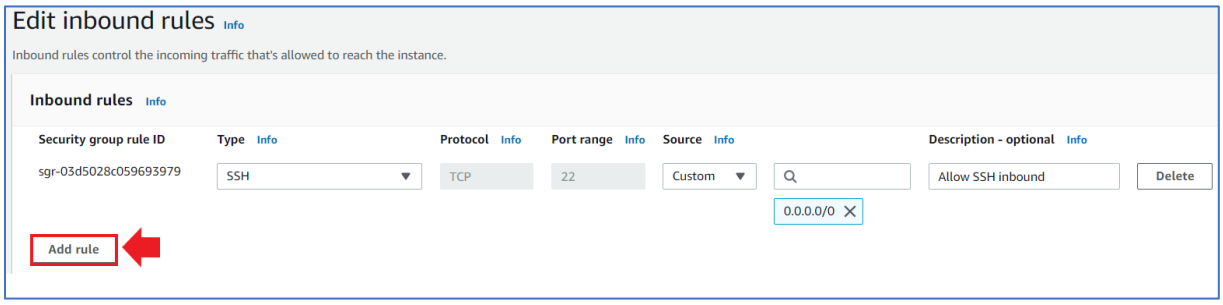


- The rules inbound only allow access to port 22 which is for SSH.

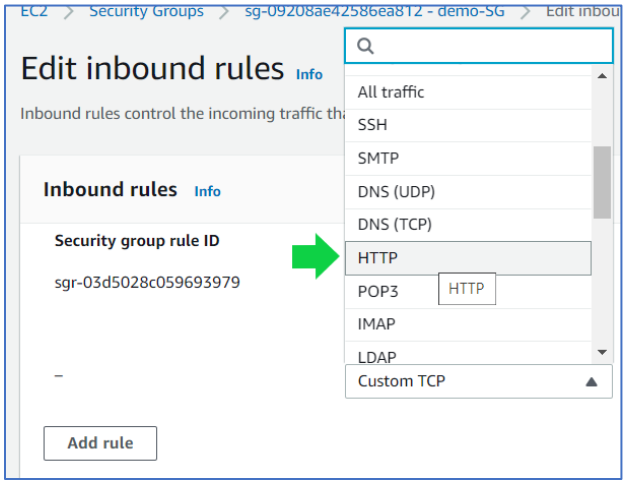


- The access to port 80 needs to be allowed.

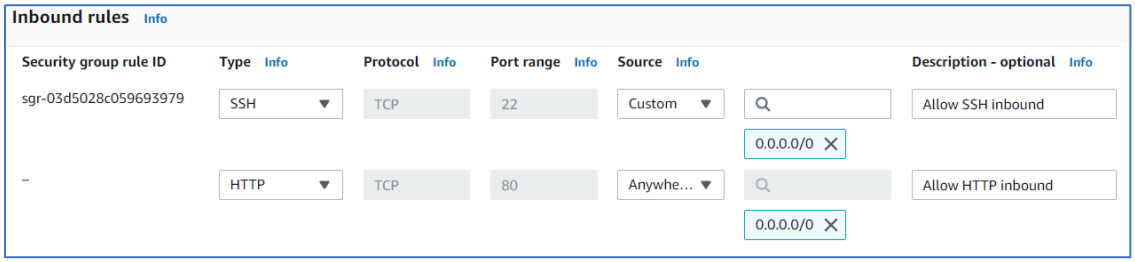
Add a new rule.



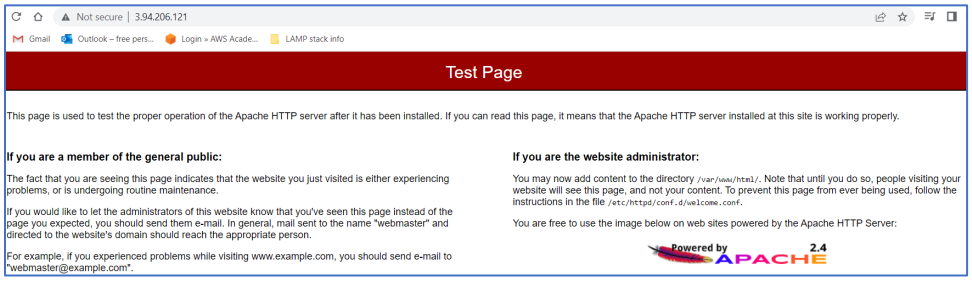
- The protocol to be added is HTTP (hypertext transfer protocol)



- This the modified security group.

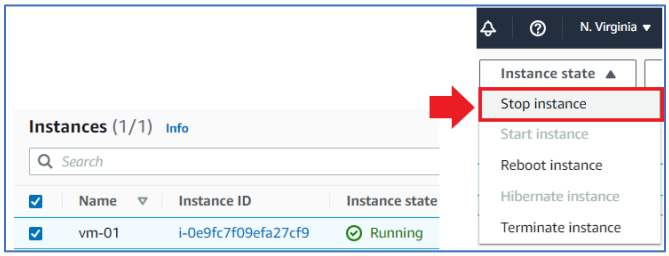


- The IPv4 address **0.0.0.0/0** has a universal meaning which is **all the addresses**.
- Hence, the new rule reads: allow from any source IPv4 address to destination port 80 (web HTTP).
- Now, web browsing works!

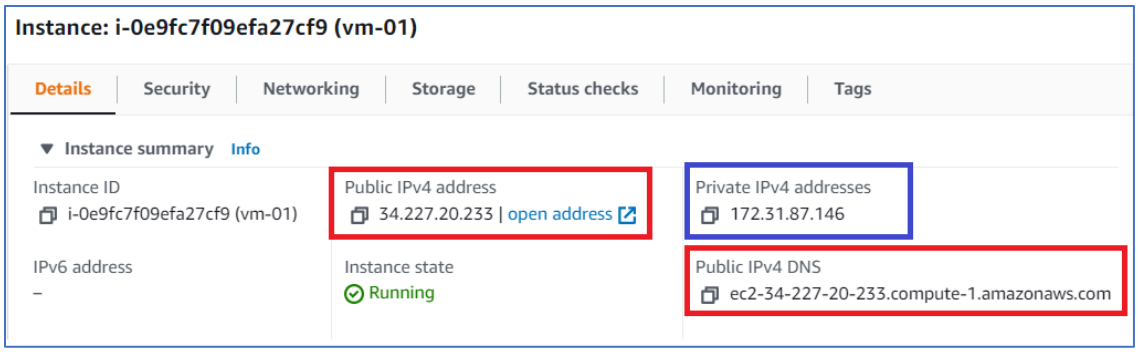


Note: this test page correspond to a previous version of HTTPD, the newer version just shows: It Works!

Stop the VM and then restart it.



Take a look at the IPv4 addresses of the EC2 instance.



- The private IPv4 address did not change.
- The public IPv4 address changed. Now, it is 34.227.20.233, before it was 3.94.206.121.
- The public DNS name also changed to ec2-34-227-20-233.compute-1.amazonaws.com.

Public IPv4 addresses are valuable assets because all the IPv4 address space has been depleted. AWS has control over many blocks of public IPv4 address to assign to its customers for Internet access. When a EC2 instance is created with public access enabled, an IPv4 address is drawn from the AWS address pool to be assigned to the VM. If the VM is not running, the address is returned to the pool. When the VM is restarted, a new address is assigned. The DNS name is also recreated because the IPv4 address is used to generate the unique identification part of the DNS name. In some cases, this is obviously an inconvenience but there are ways around this situation. For example, the EC2 instance could be fronted with a load balancer. In this case, the DNS name of the load balancer is used to access the service. Another way is to assign an **Elastic IP Address (EIP)** which is a permanent allocation of an IP address. An EIP costs money if the VM is not running. The deployment of a load balancer and EIP will be covered in detail in other sections.

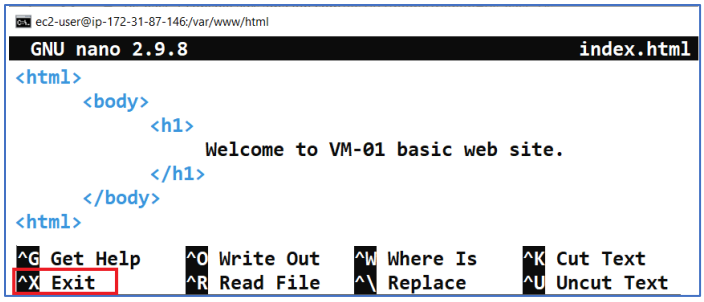
- Access the VM-01 using SSH
- Go inside the /var/www/html directory.
- List the content of the directory.

```
ec2-user@ip-172-31-87-146 ~]$ cd /var/www/html

[ec2-user@ip-172-31-87-146 html]$ ls

[ec2-user@ip-172-31-87-146 html]$ (there is nothing here)
```

- Let's create an **index.html** file.
- This index.html will replace the default Apache testing as the web landing page.
- Use a text editor either **nano** or **vi**.

```
ec2-user@ip-172-31-87-146 ~]$ sudo nano index.html
```

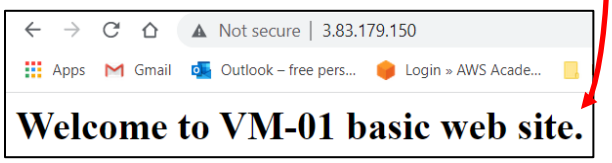
- Write a basic HTML script. Save with control-X.



Restart the application and test it with the Linux curl command:

```
[ec2-user@ip-172-31-87-146 html]$ sudo systemctl restart httpd
[ec2-user@ip-172-31-87-146 html]$ curl 127.0.0.1
<html>
      <body>
            <h1>
                  Welcome to VM-01 basic web site.
            </h1>
      </body>
<html>
```

Test it in the browser:



Let's practice a few helpful **Linux commands** now.

Stop the web service.

```
[ec2-user@ip-172-31-87-146 html]$ sudo systemctl stop httpd
```

Observe the hostname.

```
[ec2-user@ip-172-31-87-146 html]$ hostname
ip-172-31-87-146.ec2.internal
```

Verify who you are as a user.

```
[root@ip-172-31-87-146 html]# whoami
ec2-user
```

Become the root user.

```
[root@ip-172-31-87-146 html]# sudo su
[root@ip-172-31-87-146 html]# whoami
root
```

Verify your location.

```
[root@ip-172-31-87-146 html]# pwd
[root@ip-172-31-87-146 html]# /var/www/html
```

Remove the index.html file.

```
[root@ip-172-31-87-146 html]# ls
[root@ip-172-31-87-146 html]# index.html
[root@ip-172-31-87-146 html]# rm index.html
```

Recreate the index.html file with an echo command.

```
[root@ip-172-31-87-146 html]# echo "Hello from $(hostname –f)" >  index.html
```

The string $(hostname –f) has a special meaning. When it is executed with the echo command, the string gets replaced with the hostname of the machine. This is helpful to distinguish the VMs with unique names.

Restart the HTTPD service.

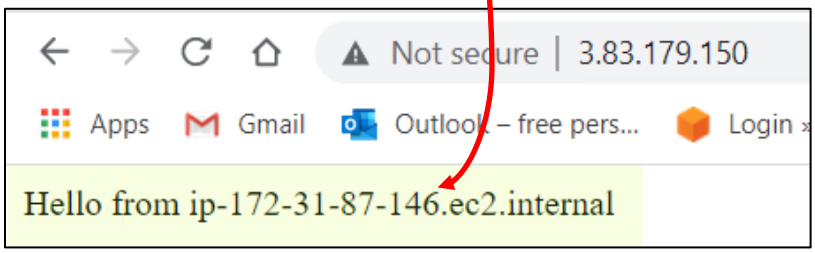
```
[root@ip-172-31-87-146 html]# ]# systemctl restart httpd
```

Leave the root user mode.

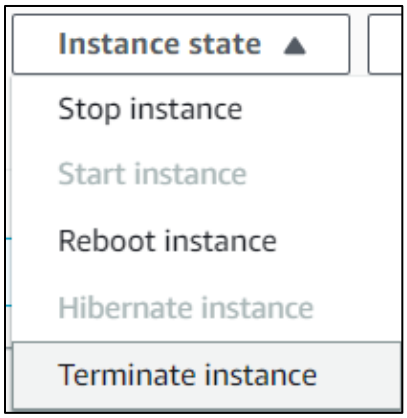
```
[root@ip-172-31-87-146 html]# exit
exit
[ec2-user@ip-172-31-87-146 www]$ cd
[ec2-user@ip-172-31-87-146 ~]$ whoami
ec2-user
```

Test the web server with an internal curl and with the web browser.

```
[ec2-user@ip-172-31-87-146 ~]$ curl 127.0.0.1
Hello from ip-172-31-87-146.ec2.internal
```



Terminate the EC2 instance in the EC2 service dashboard.

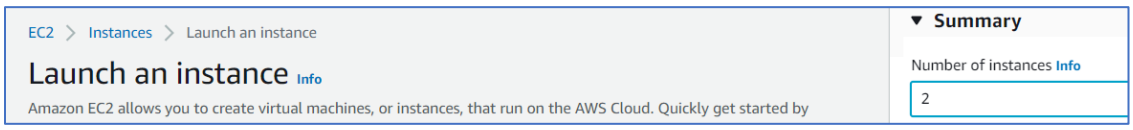


Learning Activity

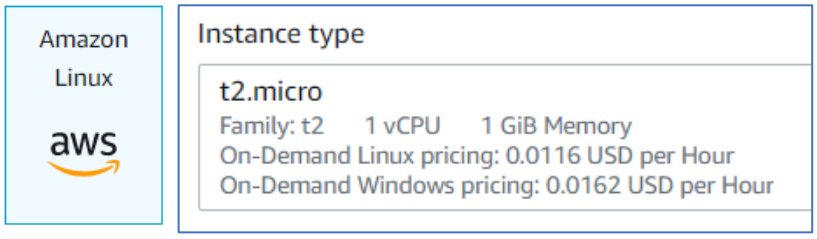
Bootstrapping EC2 instances with User Data

The User Data field under the advanced details section of the EC2 instance creation wizard is used to write configuration shell scripts. When the EC2 instance is created, the script is used to bootstrap the VM. To demonstrate this useful concept, two EC2 webservers will be deployed at once.

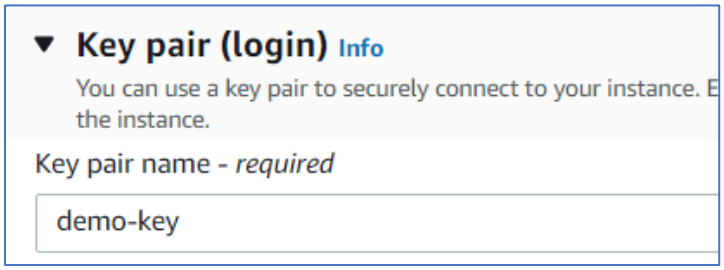
Create two EC2 instances at once.



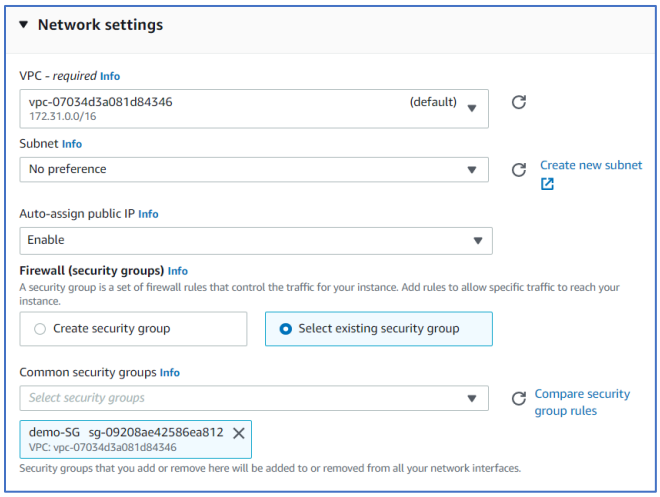
Use the default AWS AMI image and the t2.micro type.



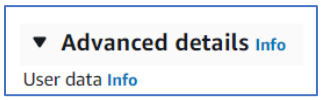
Select the key file to authenticate.



Choose the previously created security group (or make a new one with the right rules)



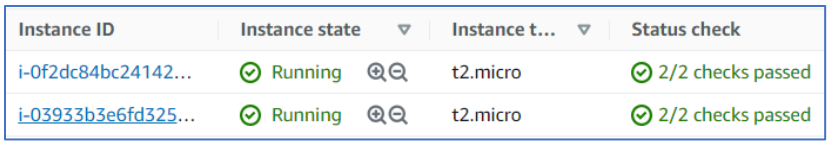
Under Advanced details, User data, write the bootstrap script.




```
#!/bin/bash
yum update -y
yum install httpd -y
cd /var/www/html
echo "<html><body><h1> Hello from $(hostname -f) </html></body></h1>" >
index.html
systemctl restart httpd
systemctl enable httpd
```

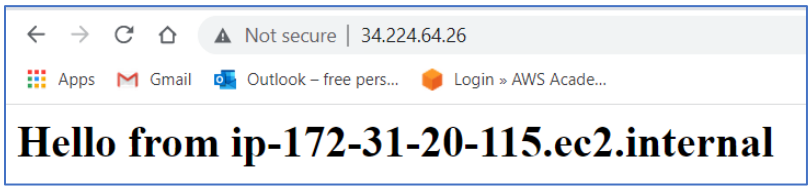
- The script begins with **#!/bin/bash** which is called a **shabang** (sharp # bang !) in Unix/Linux operating systems. The combination of the characters **#!** tells the operating system to read the text that follows as a shell script. Next, **/bin/bash** tells the operating system to use the **bash** command interpreter to execute the lines that follow.
- The first command calls the **yum** package manager to request updates and patches from the operating system (Amazon Linux) repositories.
- The second command asks yum to download and install Apache webserver (httpd). Upon the installation of http, several directories are created, one of them is /var/www/html where the index.html welcome page will be located.
- The command **cd** is used to changing the directory from home to /var/www/html.
- Then the command echo executes a string of html text that will be saved as index.html. The string $(hostname -f) is interpreted to be replaced by the machine hostname ip-private-IPv4 addres.ec2.internal.
- Finally, systemctl restarts the webserver application. The command systemctl enable ensures that the webserver will restart by itself when the VM is rebooted or stopped.

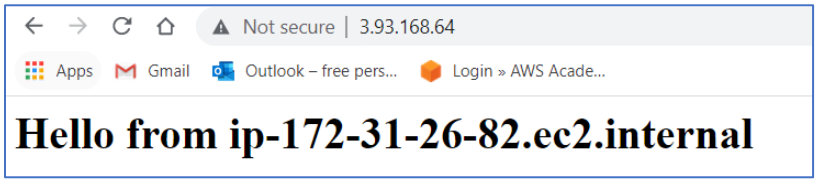
Wait for the EC2 instances to start running.



Instance ID	Instance state	Instance t...	Status check
i-0f2dc84bc24142...	Running	t2.micro	2/2 checks passed
i-03933b3e6fd325...	Running	t2.micro	2/2 checks passed

Grab the public IPv4 address of each machine and test.





Learning Activity

Deploy website configured with uploaded files

- Obtain or create a JPEG image. In this example, the image is called testpattern.JPG (a monitor and TV test pattern) and it looks like this:

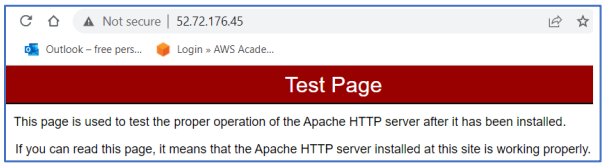
In your laptop, write this file and save it as index.html. (Adjust the image name accordingly)

```
<html>
    <head>
        <title> TELE 20483 </title>
    </head>
    <body>
        <h1> This is a web server example </h1>
        <p>  Cloud Enabled Networks </p>
    </body>
    <img src="testpattern.JPG" width=500/>
</html>
```

Create an EC2 instance, bootstrap it like this:

```
#!/bin/bash
yum install httpd -y
systemctl restart httpd
systemctl enable httpd
```

Test that it works, it should show the default Apache test page. (**note**: the latest version of HTTPD just shows the phrase: It Works instead of the image shown here)



Use the scp command to upload the index.html file and the JPEG image.

- The format of the command is:

```
scp –i authentication-key-location file-to-upload ec2-user@ip-address:/home/ec2-user
```

```
scp –i C:\Users\name\Downloads\demo-key.pem
C:\Users\name\Downloads\testpattern.JPG ec2-user@52.72.176.45:/home/ec2-user
The authenticity of host '52.72.176.45 (52.72.176.45)' can't be established.
ECDSA key fingerprint is SHA256:q5q9Jxoa4F5zQ771WV33Xa165Al1B3m/QhCBNWF13bI.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
Please type 'yes', 'no' or the fingerprint:
Warning: Permanently added '52.72.176.45' (ECDSA) to the list of known hosts.
testpattern.JPG                                   100%  224KB   1.2MB/s   00:00

scp –i C:\Users\name\Downloads\demo-key.pem "C:\Users\name\Documents\VSC HTML
\index.html" ec2-user@52.72.176.45:/home/ec2-user
index.html                                        100%  238     5.7KB/s   00:00
Notice: that the path to the index.html file has gaps in the name. Use " " in such case.
Notice: that Windows uses back slash (\) and Linux uses forward slash (/)
```

SSH into the EC2 instance and verify that the two files are present in the home directory.

```
[ec2-user@ip-172-31-29-36 ~]$ ls
index.html    testpattern.JPG
```

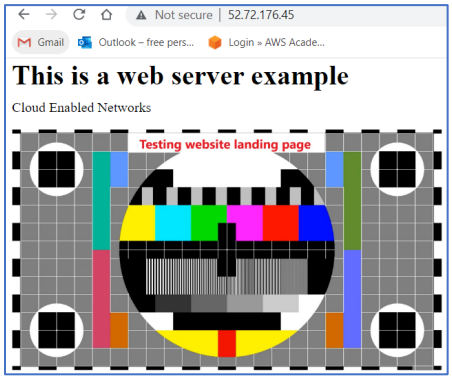
Move the two files to /var/www/html.

```
[ec2-user@ip-172-31-29-36 ~]$ sudo mv index.html /var/www/html
[ec2-user@ip-172-31-29-36 ~]$ sudo mv testpattern.JPG /var/www/html
[ec2-user@ip-172-31-29-36 ~]$ sudo systemctl restart httpd
```

Restart the web service.

```
[ec2-user@ip-172-31-29-36 ~]$ sudo systemctl restart httpd
```

Test with the web browser.



Learning Activity

Create EC2 instances using AWS CLI

AWS CLI is the command line interface method to interact with AWS instead of using the web interface. For every configuration GUI, there is always a CLI version that administrators can use instead. However, to obtain the maximum benefit of a CLI, organizations and administrators keep repositories of script templates to be used for repetitive tasks. This is a brief introduction to AWS CLI.

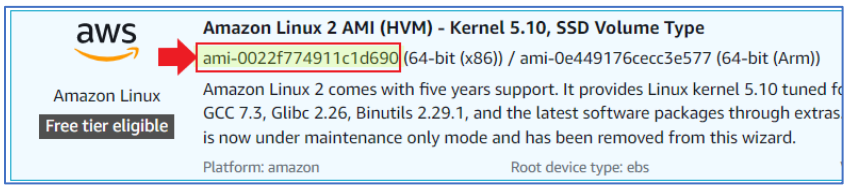
Creation of an AWS EC2 Instance

- The following is the base script to create EC2 instances.
- The command is split into several lines for readability. (Issue the command in one line though).

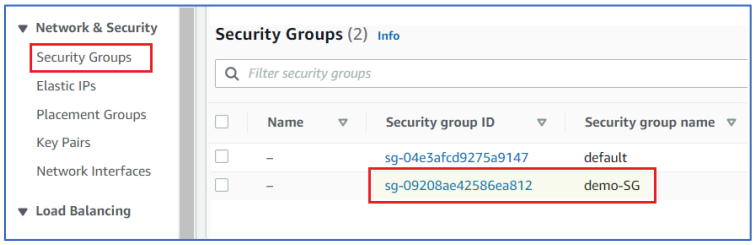
```
aws ec2 run-instances
--image-id ami-xxxxxx
--count 1
--instance-type t2.micro
--key-name xxxxxx
--security-group-ids xxxxxx
--subnet-id xxxxxx
```

To complete the script some information needs to be collected from the AWS management console.

- The Image Identification string (this string changes over time and it differs across AWS regions).



- The Security Group identification string.

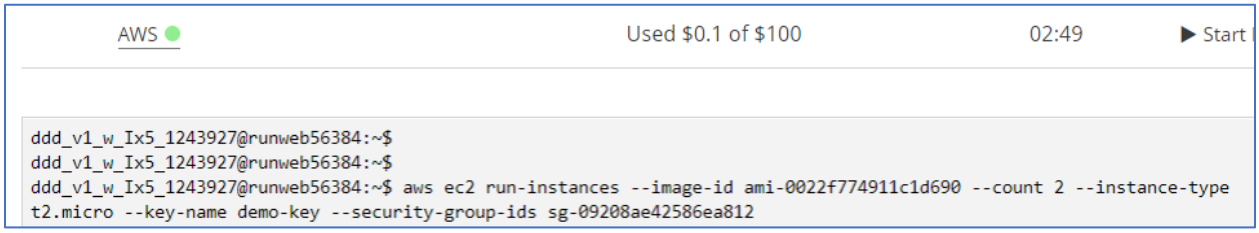


- That is enough information for now to create an EC2 instance.
- This is the script now to create two EC2 instances:

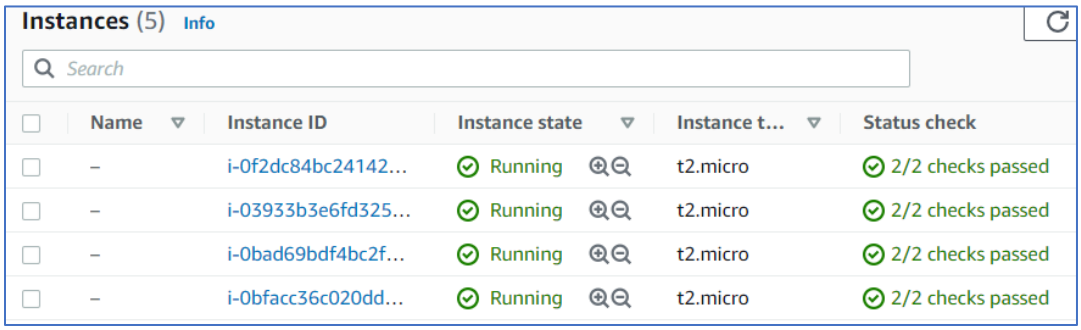
```
aws ec2 run-instances
--image-id ami-0022f774911c1d690
--count 2
--instance-type t2.micro
--key-name demo-key
--security-group-ids sg-09208ae42586ea812
```

Run the script as one line.

```
aws ec2 run-instances --image-id ami-0022f774911c1d690 --count 2 --instance-type
t2.micro --key-name demo-key --security-group-ids sg-09208ae42586ea812
```



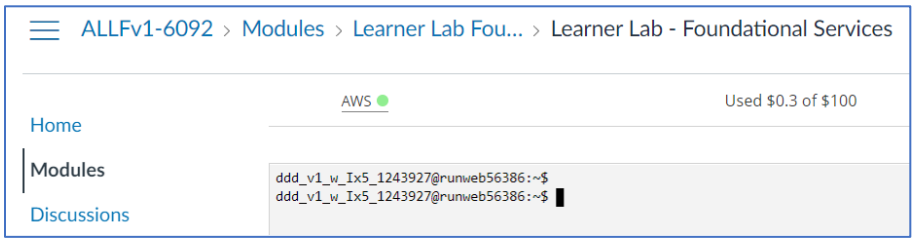
- There are now four (4) EC2 instances as seen in the AWS Console.



Learning Activity

Deploy EC2 instances using Python Boto3 script

- In the AWS Academy console.



Make a directory to store all your python scripts.

```
ddd_v1_w_Ix5_1243927@runweb56386:~$ mkdir boto3
ddd_v1_w_Ix5_1243927@runweb56386:~$ cd boto3
```

Use text editor nano to create a python script.

```
ddd_v1_w_Ix5_1243927@runweb56386:~$ nano create_ec2.py
```

- This is the script:

```
import boto3
# Variables receive values from the user:
quantity = int(input('Enter the number of EC2s to be created: '))
ami = (input('Enter the Image ID: '))
keyname = (input('Enter the key name: '))
# This function creates EC2 instance:
def create_instance():
    ec2_client = boto3.client("ec2", region_name="us-east-1")
    instances = ec2_client.run_instances(
        ImageId = ami,
        MinCount = quantity,
        MaxCount = quantity,
        InstanceType = "t2.micro",
        KeyName = keyname,
        TagSpecifications=[{'ResourceType': 'instance','Tags': [{'Key': 'Name','Value': 'VM'}]}])

# The created instances ID are printed:
    for n in range(quantity):
       print(instances["Instances"][n]["InstanceId"])

create_instance()
```

Save the program.

AWS ● Used $0.3 of $100 03:48 ▶ Start Lab

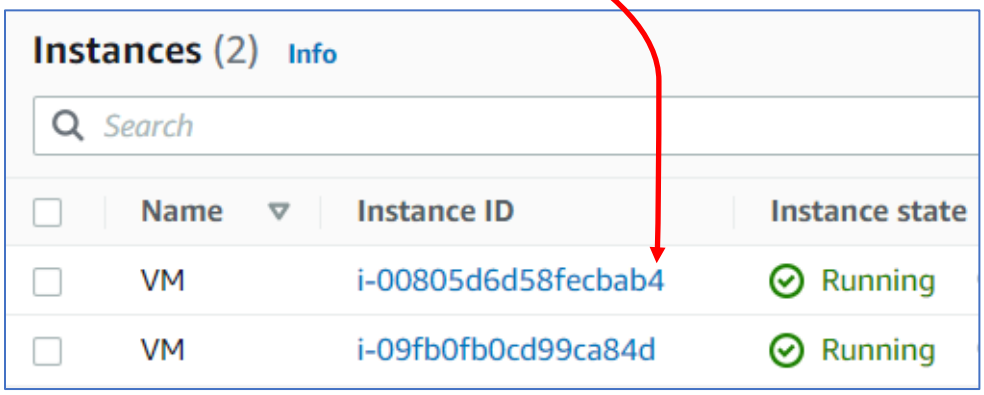
GNU nano 2.5.3 File: create_ec2.py

```
import boto3
# ami = input('Enter the AMI:')
quantity = int(input('Enter the number of EC2s to be created:'))
ami = (input('Enter the Image ID:'))
keyname = (input('Enter the key name:'))

def create_instance():
    ec2_client = boto3.client("ec2", region_name="us-east-1")
    instances = ec2_client.run_instances(
        ImageId = ami,
        MinCount= quantity,
        MaxCount= quantity,
        InstanceType = "t2.micro",
        KeyName= keyname,
        TagSpecifications=[{'ResourceType': 'instance','Tags': [{'Key': 'Name','Value': 'VM'}]}])

    for n in range(quantity):
       print(instances["Instances"][n]["InstanceId"])

create_instance()
```

Run the program.

- Compare the EC2 instances ID from the program printout with the AWS management console.

```
ddd_v1_w_lx5_1243927@runweb56386:~/boto3$ python create_ec2.py
Enter the number of EC2s to be created: 2
Enter the Image ID: ami-0022f774911c1d690
Enter the key name: demo-key
i-00805d6d58fecbab4
i-09fb0fb0cd99ca84d
```



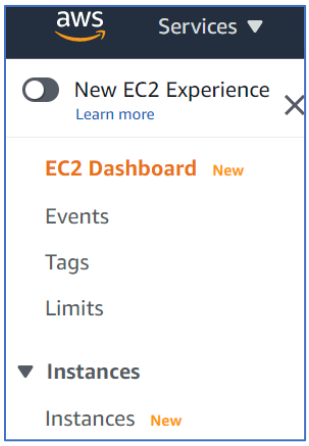
- Remove the EC2 instances once finished.

Learning Activity

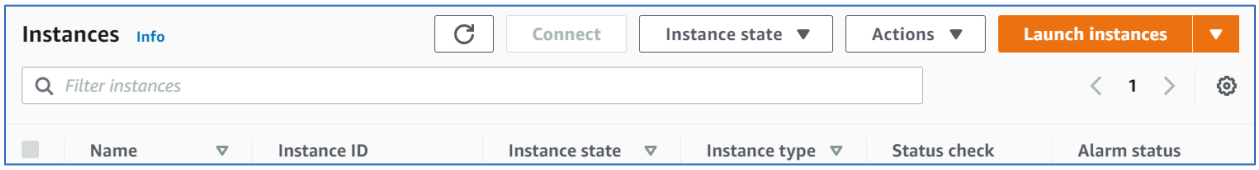
Deploy a Windows Server 2019 on AWS EC2 instance

In this activity, a virtual machine is deployed supporting a Windows Server 2019 in Amazon AWS Elastic Cloud Services (EC2).

- In the EC2 dashboard.



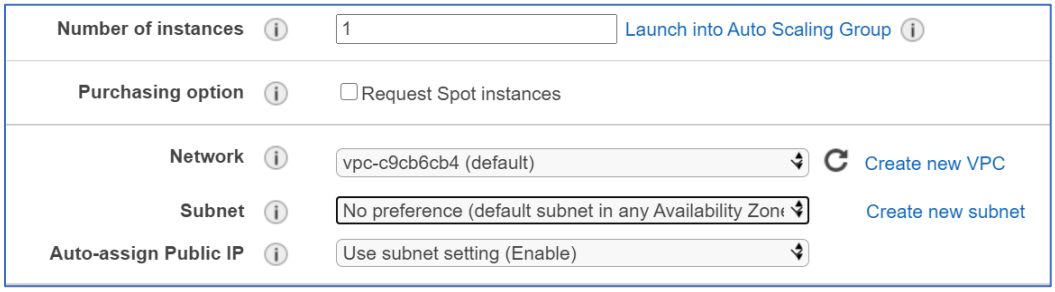
- Launch Instance



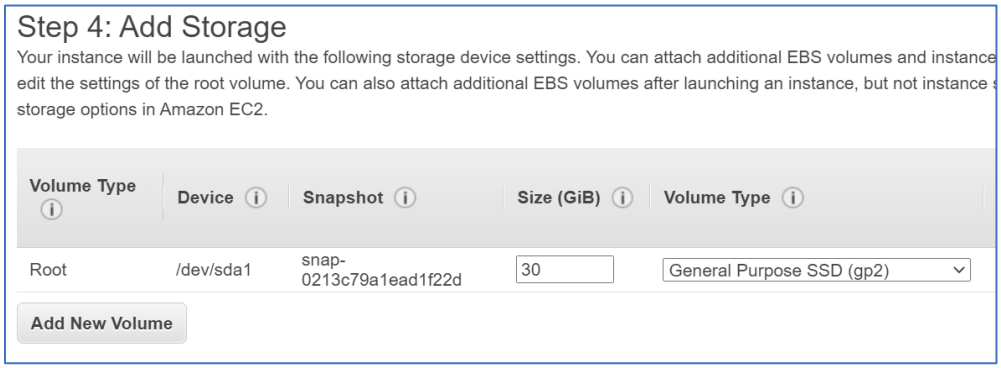
- Select the Image for Microsoft Server 2019 Base

- Choose a platform (t2.micro in this case)

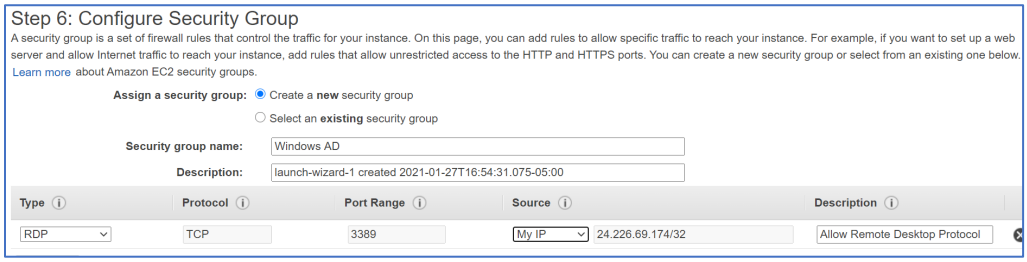
- Select preferences



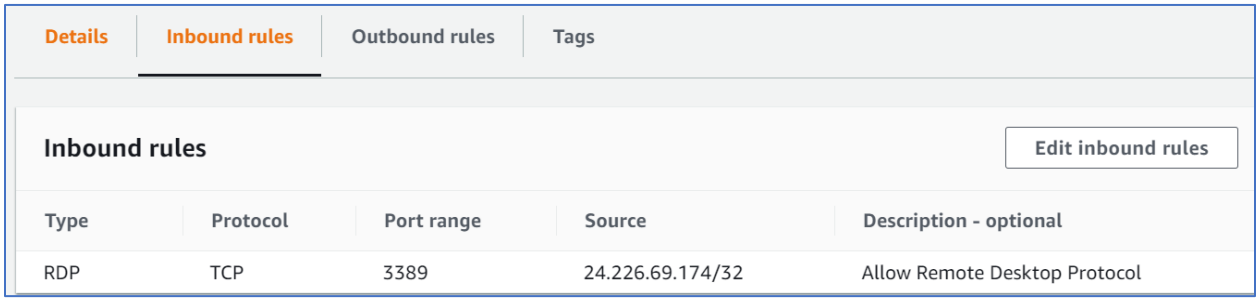
- Add storage (it needs certain size being a Windows server)



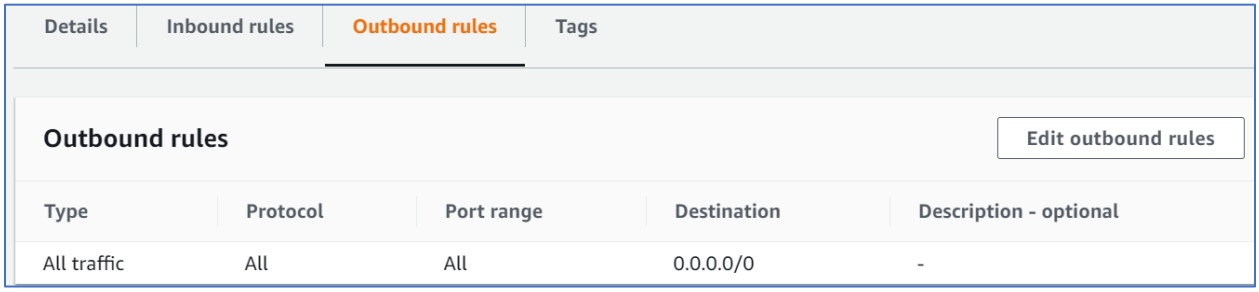
- A new Security Group is created



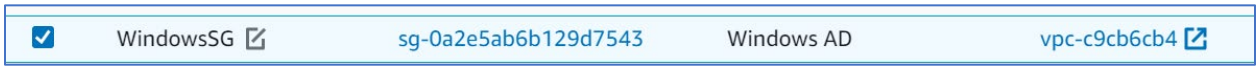
- Allow access to Remote Desktop Protocol (port 3389).



- Allow all access outbound.



- Done.



- The VM has been deployed.

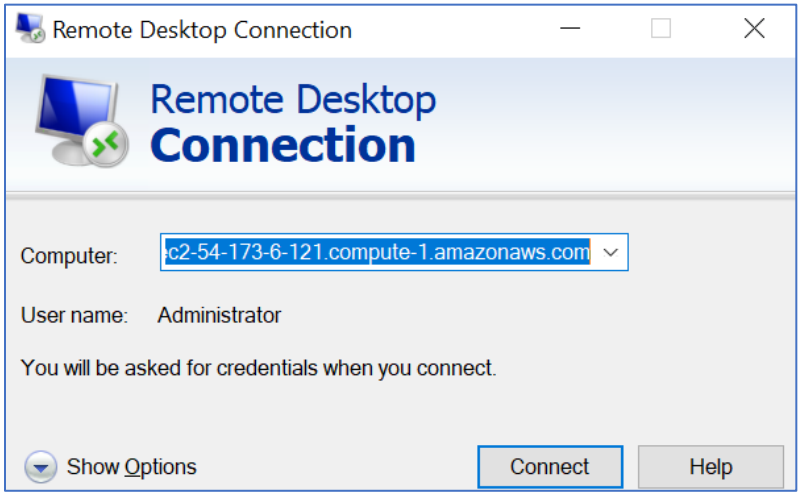
Name	Instance ID	Instance state	Instance type
Windows-01	i-0d25df1907ca15039	Running	t2.micro

- The server got the public address 54.159.1.242.

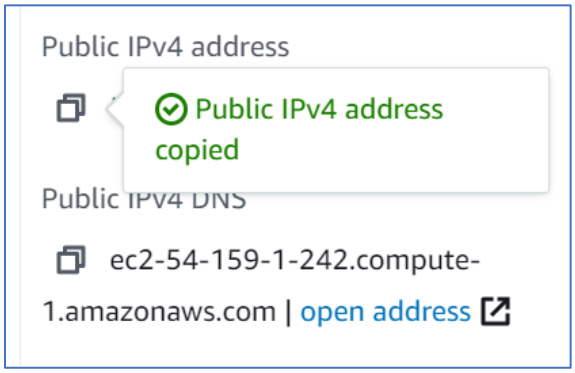
Public IPv4 address	Elastic IP	IPv6 IPs	Security group name	Key name
54.159.1.242	–	–	Windows AD	DEMO-CLASS1

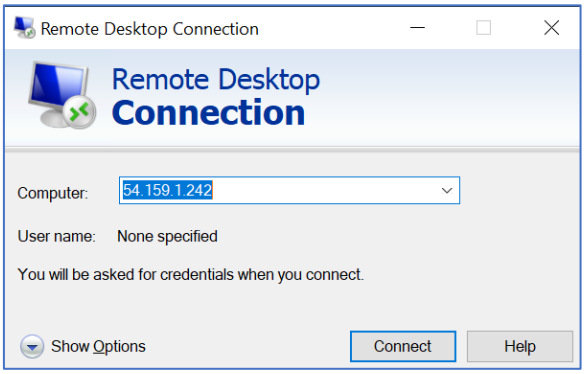
- Connect to the VM remotely (from your laptop)

- The DNS name can be used to establish the connection.

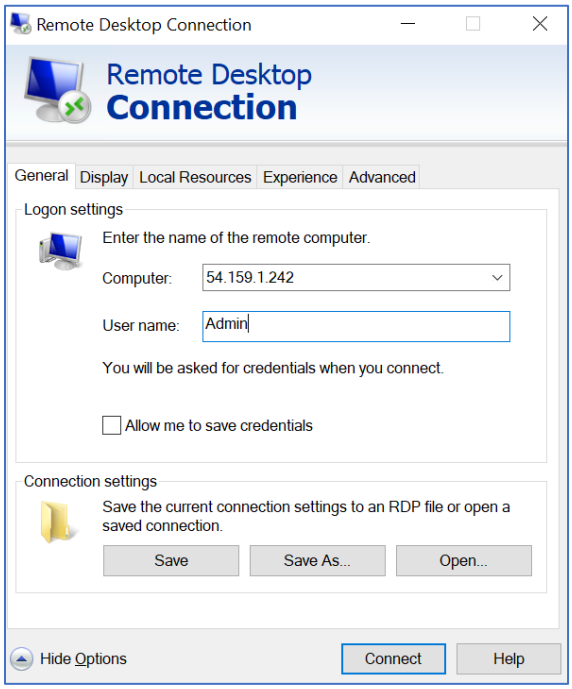


- This works with the public IPv4 address also, of course.

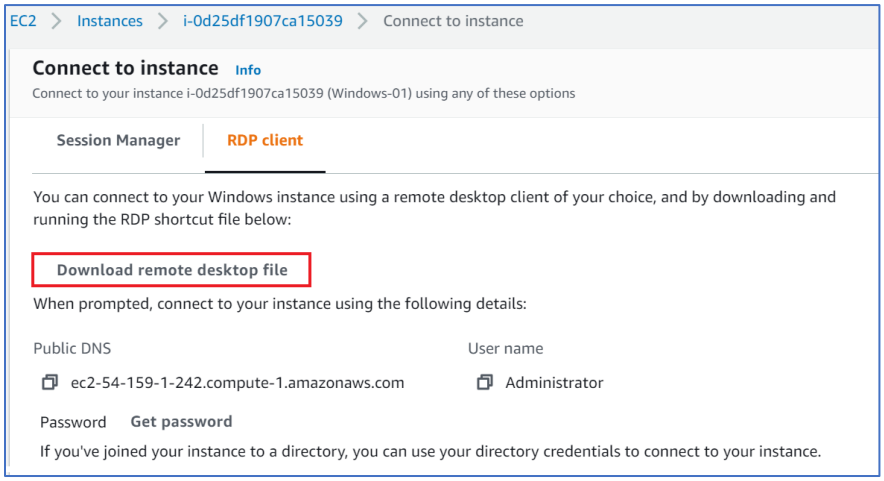




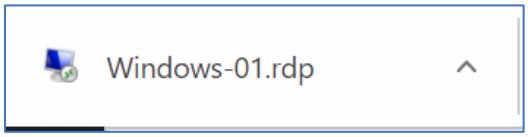
- The server username is **Administrator**.
- (You will be asked for Credentials when you connect).



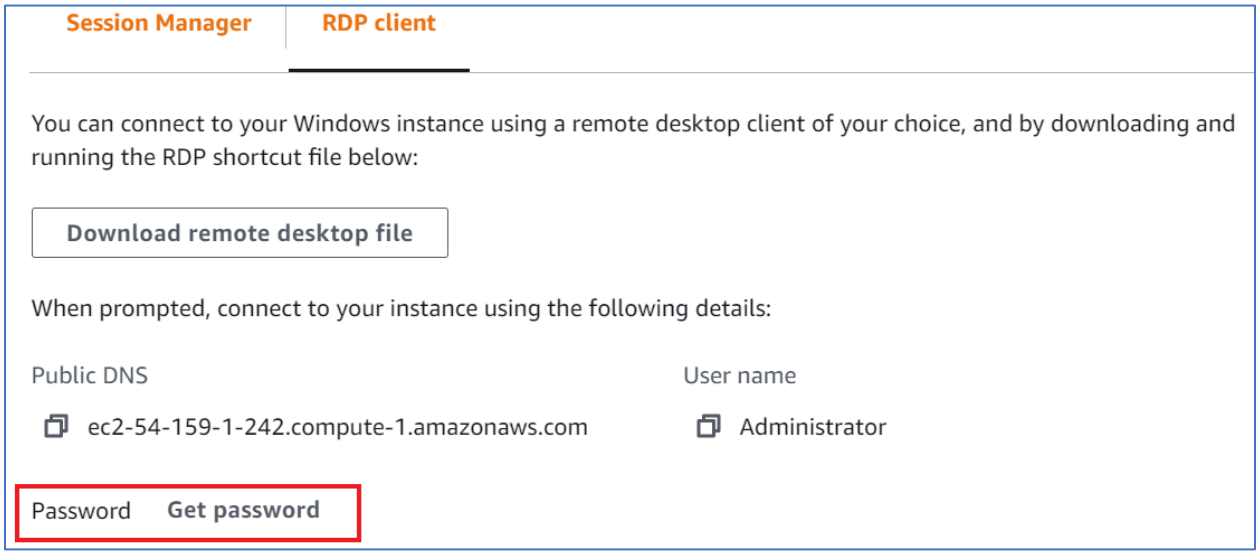
- A desktop client file is available, but this is not needed if RDP is already installed in the client's host computer (it is already installed in this laptop).



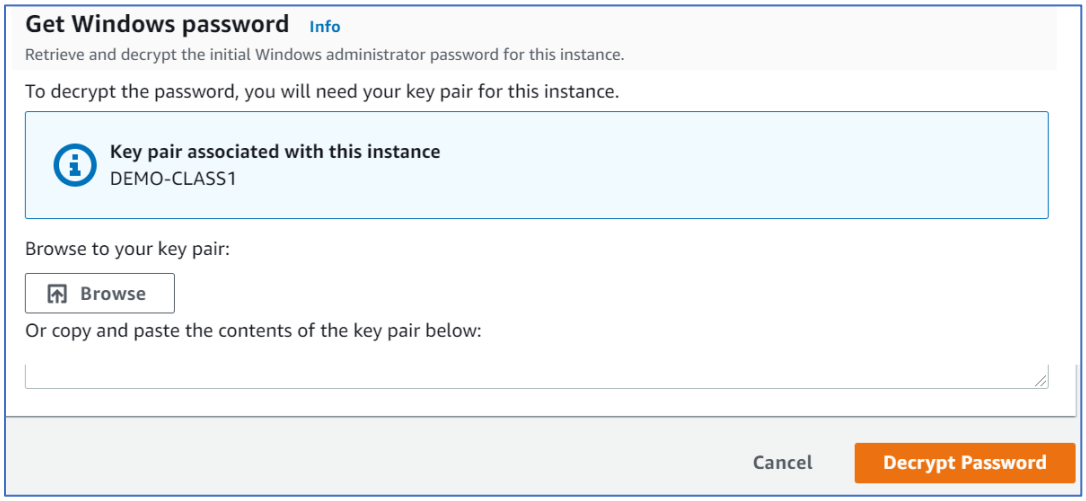
- The file is Windows-01.rdp (just in case that there is the need to download it. Again, not needed if the PC has RDP already installed.)



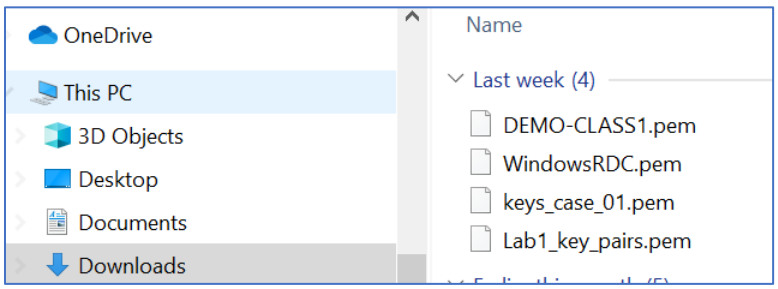
- The Credentials file must be obtained in **Password Get Password**.



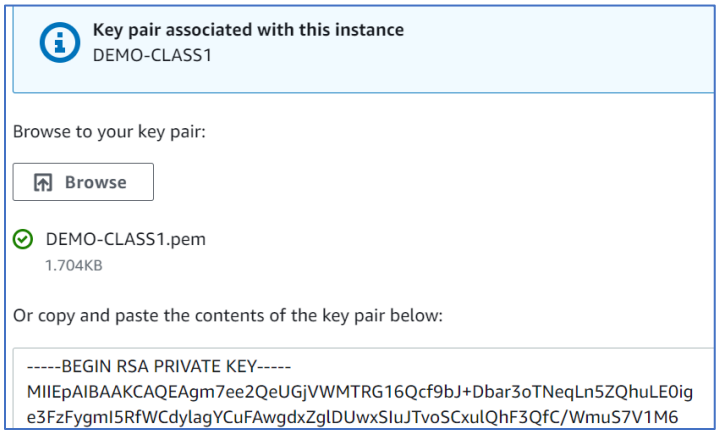
- Get the Windows password.
- Browse to the location of the key pair associated with this instance.



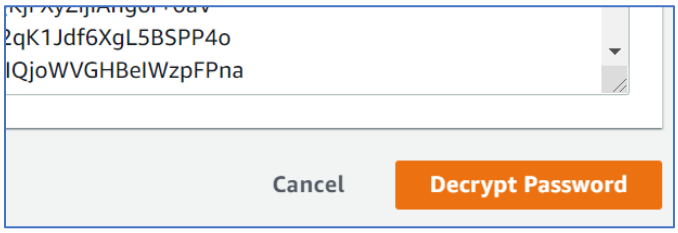
- Find the corresponding PEM file with the private key.
- In this example, it is DEMO-CLASS1.pem



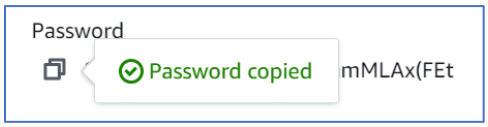
- Obtain the key.pem file.



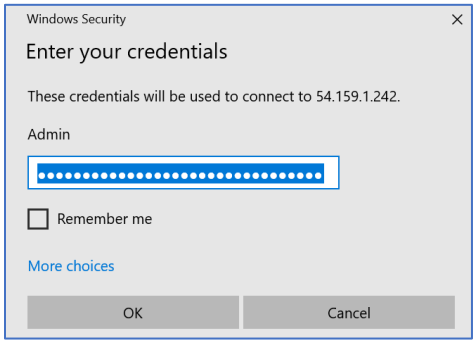
- Then hit **Decrypt Password**.



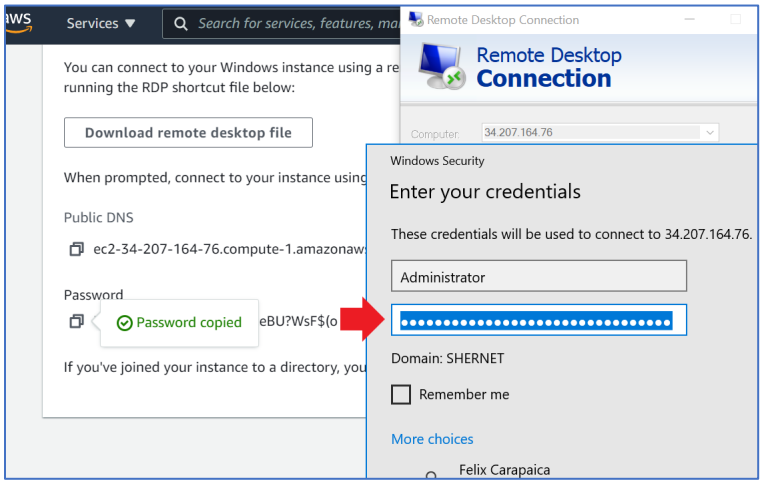
- Copy the password now.

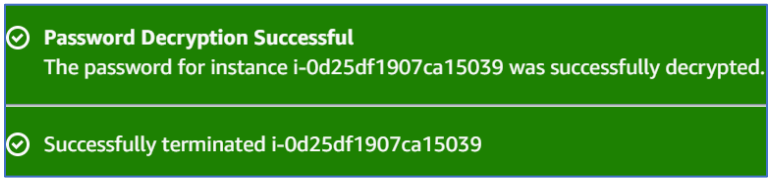


- Paste the password.

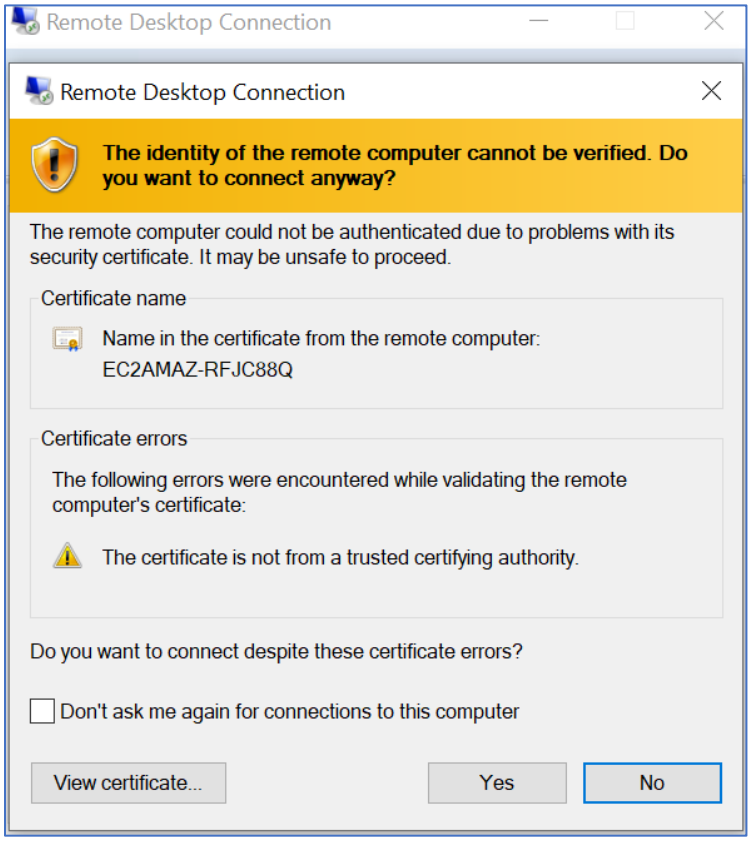


- (Copy from left to right)

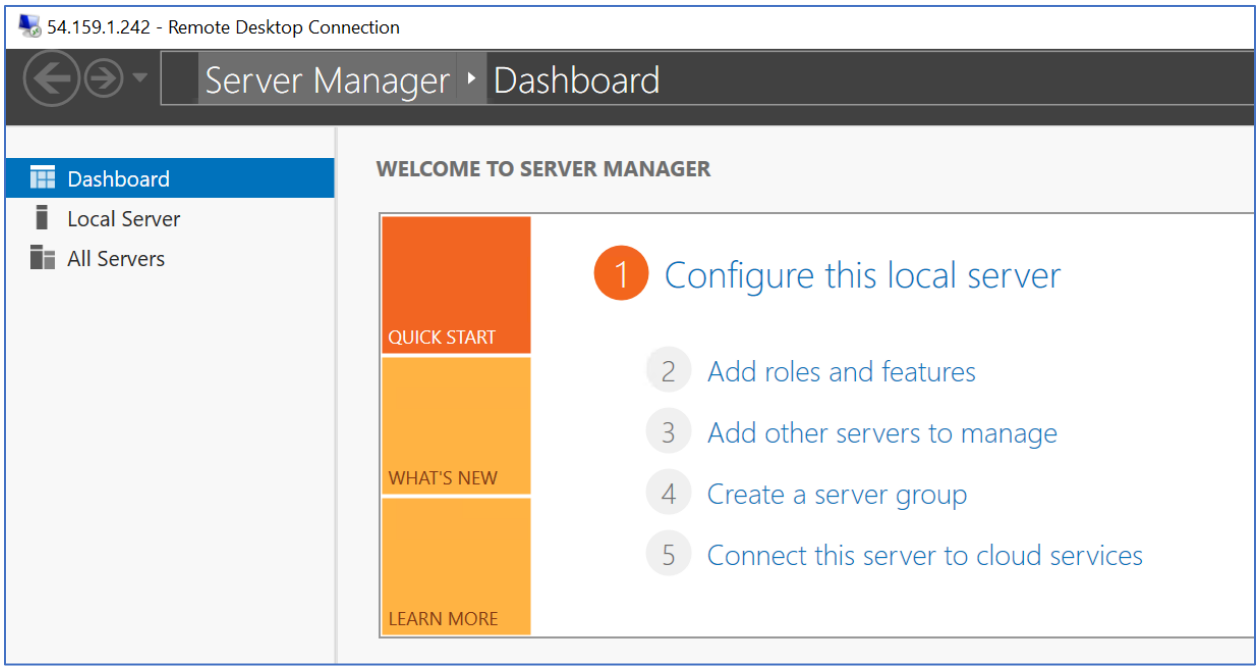




- Proceed by accepting the security certificate.



This is the Windows server 2019.



Chapter Conclusion

This section introduced the concept of virtualization of resources which is the main building block for cloud infrastructure. All the cloud vendors implement type 1 virtualization even though the technology used differs. They deploy the hypervisors directly on the hardware because of its greater efficiency.

The field of hypervisor technology is an extreme cutting-edge expertise. Open Xen hypervisor is an open-source project that organization such as AWS uses [9] to implement virtualization in its datacentres. On the other hand, Azure [1] uses its own Hyper-V hypervisor for cloud virtualization and Google Cloud [8] use Linux KVM. In all the cases, the vendors customize their hypervisors to match their technological and commercial goals.

For the customer, it is opaque how all this work and that is just fine because configuration and management interfaces are provided to deploy and handle virtual machines. In this section, three different management interfaces were experienced: the AWS Management Console, AWS CLI and Boto3 Python. The AWS Console is a web-based management tool while AWS CLI and Boto3 Python are scripting tools.

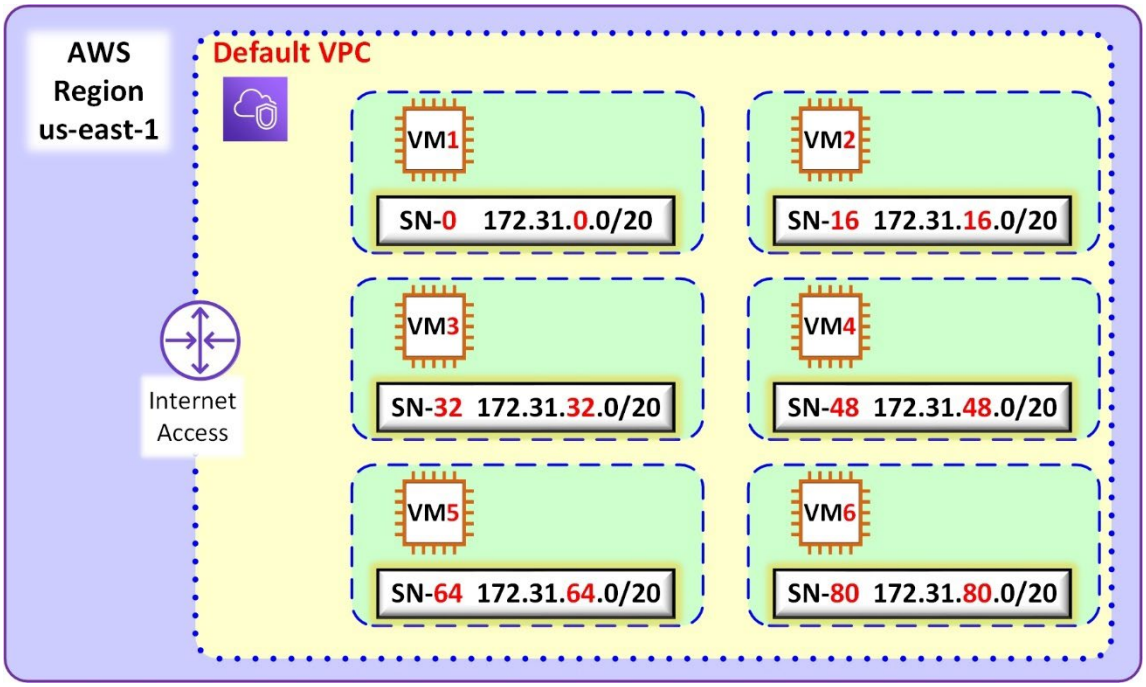
Virtualization of computers is offered by all the Infrastructure as a Service providers even though the VMs are called with different names. AWS calls this service Elastic Compute Cloud or EC2. This service presents a menu-like of choices of image, virtual chassis, storage, networking, access security and other features that the customer selects to deploy customized servers.

The most significant concept that distinguishes clouds is elasticity on demand. The virtual machines are deployed on command, and they can be expanded in capacity without major disruption of the services. In this section, webservers were deployed on Elastic Compute Cloud instances running AWS Linux. Finally, a Microsoft Windows server 2019 was also deployed.

Chapter 3 coursework

Assignment deploying webservers on EC2 instances using different ways

- Deploy the following infrastructure according to the specifications below:



Instance	Image	Function	Deployment mode	Placement Subnet	Security Group
VM1	AMI	website	AWS console	172.31.0.0/20	Allow SSH, HTTP
VM2	Ubuntu server latest	website	AWS console	172.31.16.0/20	Allow SSH, HTTP
VM3	AMI	website	AWS CLI	172.31.32.0/20	Allow SSH, HTTP
VM4	Ubuntu server latest	website	AWS CLI	172.31.48.0/20	Allow SSH, HTTP
VM5	AMI	website	Python boto3	172.31.64.0/20	Allow SSH, HTTP
VM6	Ubuntu server latest	website	Python boto3	172.31.80.0/20	Allow SSH, HTTP

Description (explanation of the previous table)

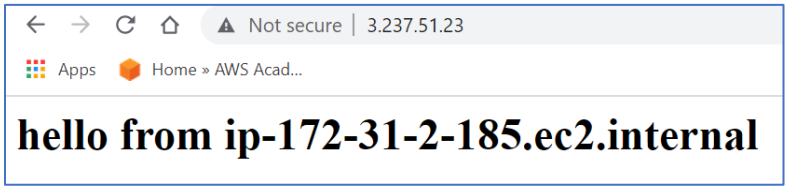
- Deploy six (6) EC2 instances (VMs) using the indicated methods in the previous table.
- These VMs will be named as indicated in the previous table (from VM1 to VM6).
- The Operating Systems of each EC2 instance are indicated in the previous table.
- These six (6) VMs are webservers running HTTP.
- It is imperative that the VMs be deployed in the indicated subnets.
- Create one Security Group that allows SSH and HTTP.
- Apply this Security Group to all these VMs.
- Bootstrap these VMs so they indicate their IPv4 private address in their landing web page.
- All the EC2 instances must be configured using the user_data.

- Use this user data for the deployment of the Ubuntu servers. Notice that the Ubuntu packet manager is apt-get (not yum). NGINX is an open source software that runs web services.

```
#!/bin/bash
apt-get update
apt-get install nginx -y
cd /var/www/html
echo "<html><body><h1> Hello from $(hostname -f) </h1></body></html>" > index.html
systemctl restart nginx
systemctl enable nginx
```

Submit

- The CLI command used to deploy VM3 and VM4.
- The python script used to deploy VM5 and VM6.
- The screenshot of the EC2 instances from the AWS console.
- A screen shot for every website home page (from your browser). For example,



- Complete this table with the corresponding information.

Instance	Instance id	Private IPv4	Public IPv4	Subnet ID	Availability Zone	Web deployment
VM1						
VM2						
VM3						
VM4						
VM5						
VM6						

Assignment Interacting with the cloud programmatically

Introduction:

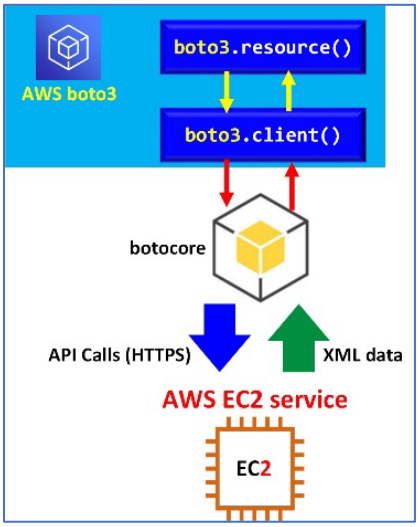
There are several ways to interact with the cloud resources:

- **Graphic Console** via a web browser interface.
- **Programmatically** via a programming language.
- **Command Line Interface (CLI)** via typing command lines on a console.
- **Infrastructure as Code (IaaC)** via a specific software tool such as CloudFormation and Terraform.

The **objective** of this assignment is to programmatically deploy EC2 resources using AWS python boto3. Afterwards, the resources deployed will be verified both programmatically and via the graphic console. The assignment is meant to reinforce the student's knowledge regarding the structure of the cloud, the ability to deploy resources in varied ways and to learn how to retrieve information off the cloud.

Boto3 offers two modules, resource and client. The high-level resource module returns **classes** while the lower-level client module returns **dictionaries**. Some tasks are simpler to do with resource while others are simpler to do with client. AWS maintains a vast documentation about both modules.

Python boto3 interacts with the lower level module botocore which acts as an intermediary to access the cloud resources such as the Elastic Compute Cloud service (EC2). In this case, botocore translates the upper level instructions from boto3 to HTTPS API calls directed to the endpoints of the EC2 service. Then, this service answers with data formatted in the Extensible Markup Language (XML). For example, the following diagram illustrates this principle with a python script retrieving information about an EC2 virtual machine. Furthermore, python scripts can also create, delete, or modify EC2 instances.



Python boto3 functional diagram.

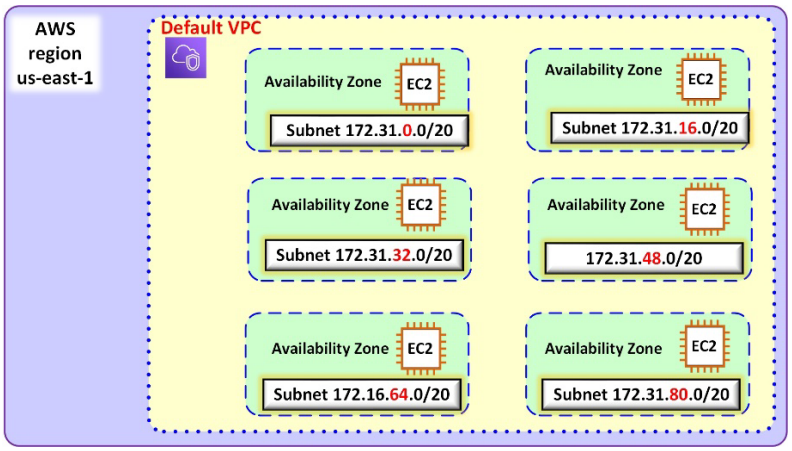
Organizations maintain repositories of programs to manage their resources because dealing with the cloud infrastructure in a structured, programmatic way is advantageous.

Task

In this assignment, the student is to write a python script that:

- will retrieve the **identification** of the **six subnets** in the default virtual private cloud (VPC).
- Then using such information, the same script will deploy **one EC2 instance per each subnet**.
- The deployment of the EC2 will include User Data to configure httpd websites.
- The website landing page should say Hi from (VM's hostname).
- Finally, the script will list the identification of the EC2 and its private IPv4 address.

The following diagram shows the requested infrastructure. By the end of the script execution, there will be a total of six websites deployed across the six default subnets.



Running the python script results in six (6) websites.

Summary of EC2 instances:

Instance	Image	Function	Deployment	Subnet (location)	Security Group
VM1	AMI Centos	website	Phyton	172.31.0.0/20	Allow SSH, HTTP
VM2	AMI Centos	website	Phyton	172.31.16.0/20	Allow SSH, HTTP
VM3	AMI Centos	website	Phyton	172.31.32.0/20	Allow SSH, HTTP
VM4	AMI Centos	website	Phyton	172.31.48.0/20	Allow SSH, HTTP
VM5	AMI Centos	website	Phyton	172.31.64.0/20	Allow SSH, HTTP
VM6	AMI Centos	website	Phyton	172.31.80.0/20	Allow SSH, HTTP

Description (explanation of the previous table)

Step 1

- Deploy six (6) EC2 instances (VMs) using Python boto3.
- The Operating Systems of each EC2 instance are indicated in the previous table.
- These six (6) VMs are webservers running HTTP (application Apache2 or HTTPD.)
- Bootstrap (User Data) these VMs so they show their hostname (which includes the IPv4 private address) in their landing web page.
- It is imperative that the VMs to be deployed in the indicated subnets.
- Use the same Security Group (allowing SSH and HTTP.)
- Apply the same Security Group to all these VMs.

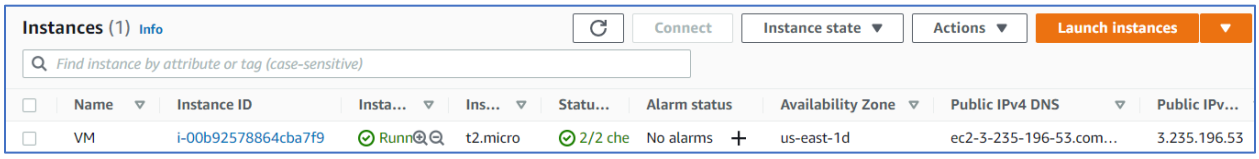
Deliverable/Submission:

In a Word document or PDF:

- **1) Submit your python code.**
- **2) Submit the proof that it works.**
    - Submit the snippet of the result of the execution. For example:

```
aws_academy_console:$  python3 create_and_describe_all_the_ec2s.py
########################################
Instance id is i-03001c0bb3023457a
Private IPv4 is 172.31.0.198
Etc…
########################################
```

- **3) Submit the screen shots of the AWS EC2 Service** showing all the created EC2 instances. For example,



- **4) Submit the screen shots of every webserver** being accessed via a web browser. For example,

Marking

Task	Task name	Evaluated by	Weight
1	Program	Quality of program, functionality, result.	50
2	Snippet working proof		20
3	AWS EC2 Service snippet		10
4	Proof of access to web sites		20
Total			100

References

[1] AWS to Azure services comparison. Accessed. June 2022. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/aws-professional/services

[2] Ellis Fitch. What are the different types of computing in AWS Services? Accessed. June 2022. [Online]. Available: https://www.nhphoenix.com/blog/what-are-the-different-types-of-computing-in-aws-services/

[3] VMware. What is a Cloud Hypervisor? Accessed. June 2022. [Online]. Available: https://www.vmware.com/topics/glossary/content/cloud-hypervisor.html

[4] AWS. Linux AMI virtualization types. Accessed. June 2022. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/virtualization_types.html

[5] Russell Pavlicek -- Next Generation Cloud. Accessed. June 2022. [Online]. Available: https://www.youtube.com/watch?v=8UgiPODw3CY

[6] Vineet Badola. AWS AMI Virtualization Types: HVM vs PV (Paravirtual VS Hardware VM). Accessed. June 2022. [Online]. Available: https://cloudacademy.com/blog/aws-ami-hvm-vs-pv-paravirtual-amazon/

[7] The Xen Project, Unikernels. Accessed. May 2022. [Online]. Available: https://wiki.xenproject.org/wiki/Unikernels

[8] Andy Honig and Nelly Porter. 7 ways we harden our KVM hypervisor at Google Cloud Accessed. June 2022. [Online]. Available: https://cloud.google.com/blog/products/gcp/7-ways-we-harden-our-kvm-hypervisor-at-google-cloud-security-in-plaintext

[9] Bhanu P Tholeti. Dive into the Xen hypervisor. 2011. [Online]. Available: https://developer.ibm.com/articles/cl-hypervisorcompare-xen/

[10] VMware. Understanding Full Virtualization, Paravirtualization, and Hardware Assist. 2007. [Online]. Available:https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf

[11] Xen Project. Understanding the Virtualization Spectrum. 2014. [Online]. Available: https://wiki.xenproject.org/wiki/Understanding_the_Virtualization_Spectrum

[12] AWS. Tag your Amazon EC2 resources. Accessed. June 2022. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Using_Tags.html

[13] AWS. Amazon EC2 instance root device volume. Accessed. May 2022. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/RootDeviceStorage.html.

[14] Amazon EC2 Spot Instances. AWS. Accessed. May 2022. [Online]. Available: https://aws.amazon.com/ec2/spot/.

[15] AWS. IAM roles for Amazon EC2. Accessed. June 2022. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html.

[16] AWS Nitro System. Accessed. June 2022. [Online]. Available: https://aws.amazon.com/ec2/nitro

[17] Amazon Linux 2022. Accessed. June 2022. [Online]. Available: https://docs.aws.amazon.com/linux/al2022/ug/relationship-to-fedora.html

Chapter 4

Cloud Network Communications

Description

This section provides a general explanation of the process of communication between a local client and a server in the cloud. This is fundamental knowledge needed to deploy and to maintain resources in the cloud. There are several processes and protocols involved in the transference of information between a client computer to reach a webserver in the cloud. These protocols are outlined in this chapter.

Learning Outcomes

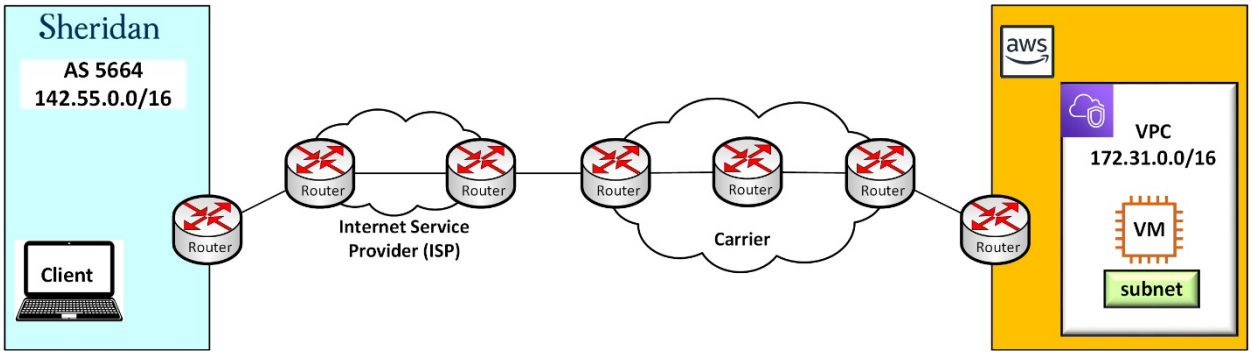
- Compare different cloud offerings.
- Describe the organization and general structure of a cloud provider.
- Deploy elastic virtual computers in a cloud networking environment.
- Configure webservers on elastic virtual computers.
- Analyze the delivery of information throughout the Internet.

Main concepts

- How a client computer communicates with a service running in a cloud provider.
- How messages are encapsulated and delivered from one site to another across the Internet.
- The fundamental role of the networking protocols in the delivery of traffic.

Client to Cloud across the Internet

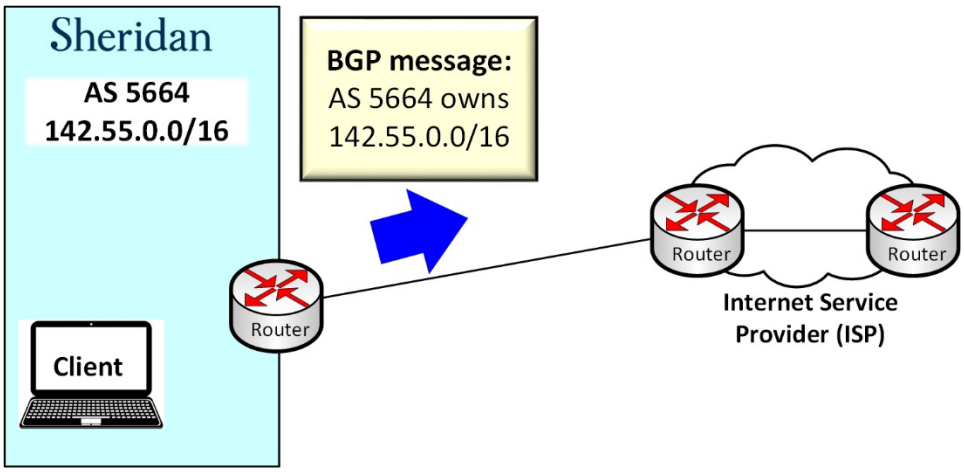
The purpose of networking is the delivery of information from one place to another. Whether the communication happens among computers, mobile devices, or the cloud, this principle remains the same. In all cases, information is packetized and delivered from one location to another. The following high-level network diagram will be used to explain the travel of data between a client located in an enterprise network (Sheridan College) and a webserver running in an EC2 instance in a VPC in the AWS cloud.



From client to server in the cloud

Sheridan College owns the autonomous system number 5664 and the **IPv4 address range** 142.55.0.0/16. An **autonomous system** is a network domain, and 5664 is the unique global number that identifies Sheridan College as a participating organization on the Internet. The IPv4 address 142.55.0.0/16 means that Sheridan owns all of the IPv4 addresses that begin with 142.55 (from 142.55.0.0 to 142.55.255.255).

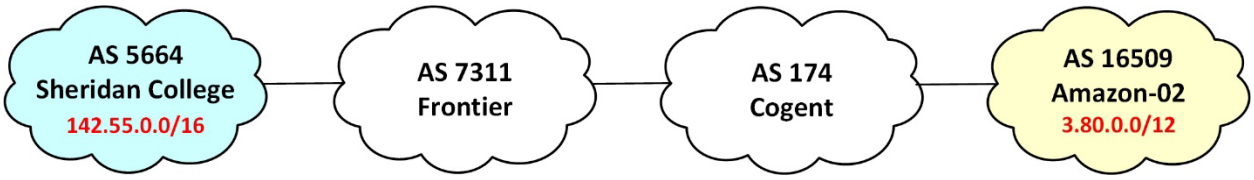
Sheridan connects to several network providers. In the diagram, there is a router sitting at the border of Sheridan's routing domain which is directly connected to another router located in the Internet Service Provider (ISP). These routers speak a common language to advertise routing knowledge. This language is the Border Gateway Protocol (BGP). Sheridan's border routers use this routing protocol to advertise the public IP address space inside the local autonomous system as illustrated in the next diagram.



Sheridan's border router using BGP to advertise the domain's public IP address space

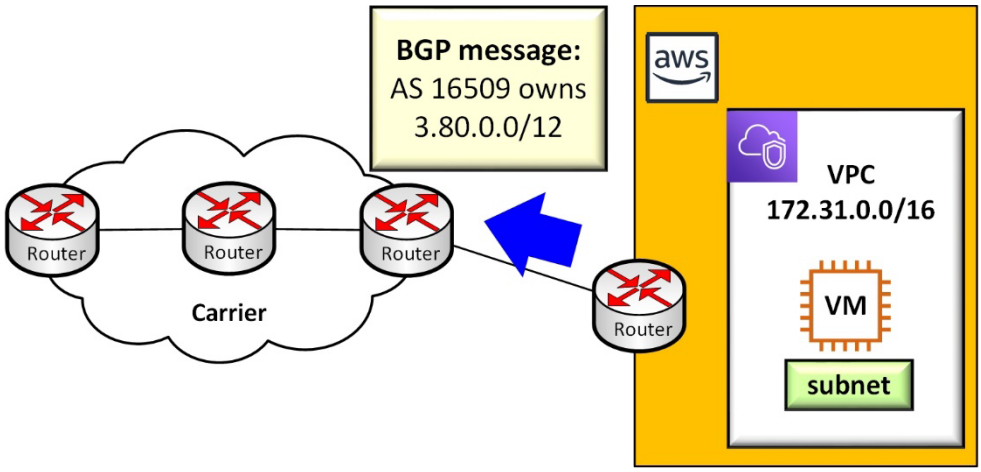
Consequently, the ISP learns the location of all the 142.55.0.0/16 addresses. The router that does the learning, install the information in its routing table and then it proceeds to re-advertise the knowledge to its neighbour router. The process is repeated from neighbour to neighbour, so the information keeps propagating throughout the Internet. Nevertheless, it is important to clarify that BGP only advertises new information. If nothing changes in the network, then there is nothing to advertise.

Between Sheridan College and AWS there are at least two other AS domains. The first one is Frontier [2] which is the Internet provider. The next domain is Cogent [3] which is one of the largest carriers in the world. Amazon owns several autonomous systems with many blocks of IPv4 addresses. One of these IPv4 blocks is 3.80.0.0/12 which is in AS 16509 according to information gleaned from search sites [1,2,3]. Thus, this address range will be used in this example.



High-level view of the Internet path between Sheridan and AWS

Between Sheridan and AWS there are dozens of routers keeping this part of the Internet glued together. They propagate the location of AS 5664's 142.55.0.0/16 to AWS. From the other side, AWS advertises to Cogent its address blocks, in the example only 3.80.0.0/12 is presented.

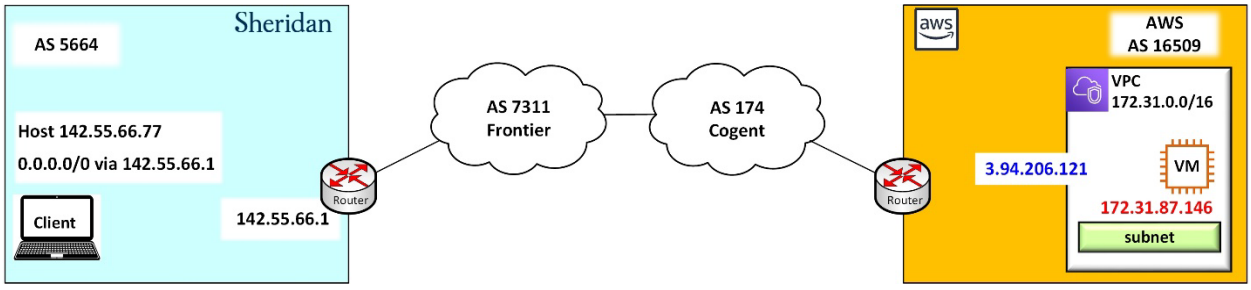


AWS's border router advertising the domain's IP address space

The carrier Cogent propagates the knowledge to its neighbours. Frontier learns the information too and it advertises 3.80.0.0/12 to Sheridan. Again, this happens only once if the conditions of the network do not change. If a new address block is enabled or if an existing network goes down, BGP advertises the event, otherwise, there is nothing to say.

The main point is that Sheridan knows where 3.80.0.0/12 is located, and AWS knows where 142.55.0.0/16 is located. The Internet organizations in the path also know how to get to these destinations. Thus, IP traffic can flow from one end to the other.

Let's assume that in Sheridan, the host computer has been assigned the public IPv4 address 142.55.66.77. When the computer connected to the network, a DHCP server assigned the IPv4 addresses to the host PC. Additionally, it provided the information of "0.0.0.0/0 via 142.44.66.1" which literally means: "if you need to send traffic outside of this local area network send it to the default gateway router which is located at 142.55.66.1". **Proviso**: this is a simplification of the network for the sake of the explanation. In reality, there are more devices and processes involved. The IPv4 addresses are real and they were just chosen for this example.

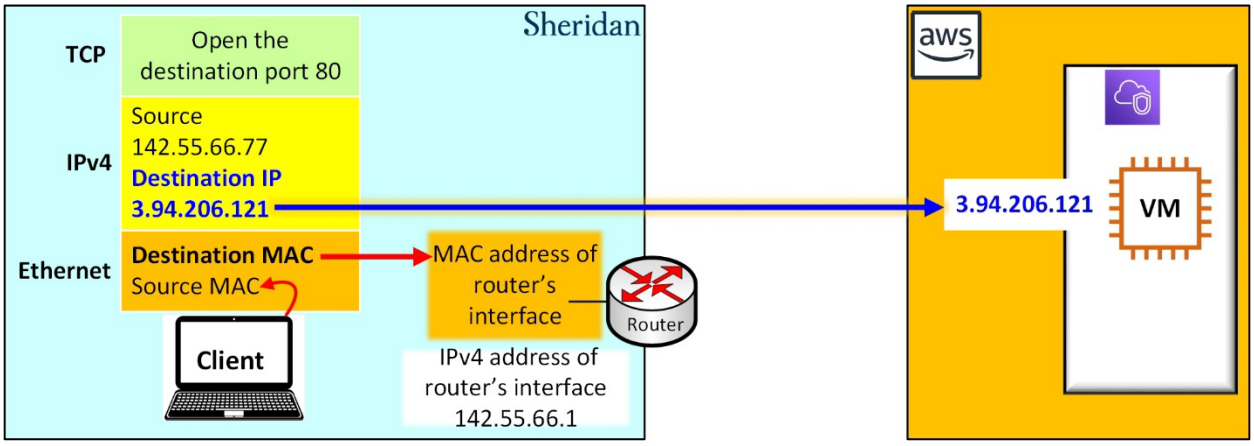


End nodes with specific IPv4 address information

Let's suppose that in AWS an EC2 instance has been deployed running a web service. This VM has the public IPv4 address 3.94.206.121 which is in the block of addresses 3.80.0.0/12. In a similar fashion, the EC2 instance knows how to send traffic outside of AWS because it has a default route pointing to an exit.

The user at the Sheridan host opens the web browser and types the IPv4 address of the web server. This action generates a message to start the session with the server. The information is encapsulated like this:

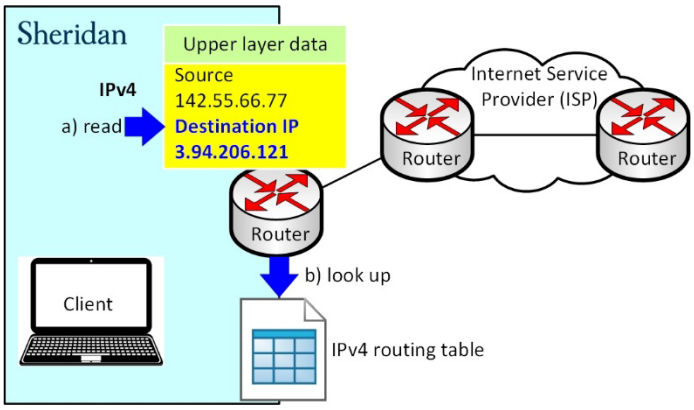
- The topmost data contains a TCP message to open the destination port 80 which is HTTP.
- The IPv4 packet contains the destination address of the webserver 3.94.206.121 in AWS.
- The destination MAC address of the Ethernet header is the next hop where the frame will be delivered. In the figure, it points to the physical interface of the default gateway router.



First step in the delivery of traffic from client to server

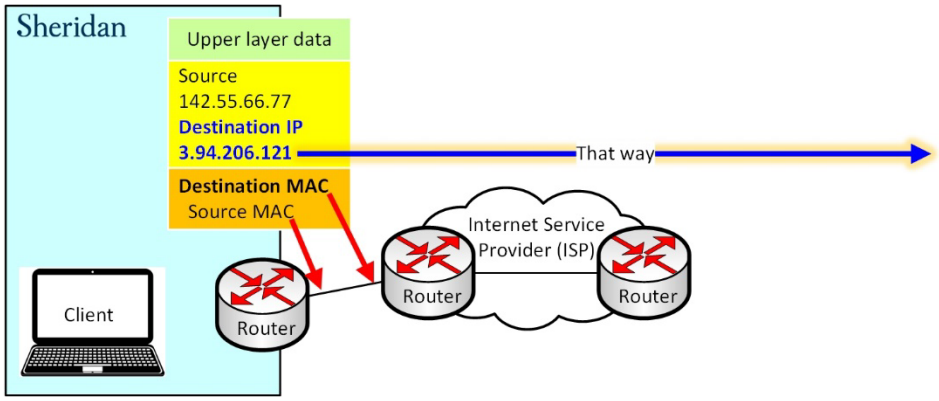
The Ethernet frame containing all the encapsulated data is forwarded to the next hop. The router receives the frame and removes the Ethernet header. The function of the frame header is to move the data **hop by hop** hence its function has been fulfilled. The router then proceeds to examine the information in the

IPv4 header. There are several fields that the router check, but in this explanation the only relevant is the destination IPv4 address.



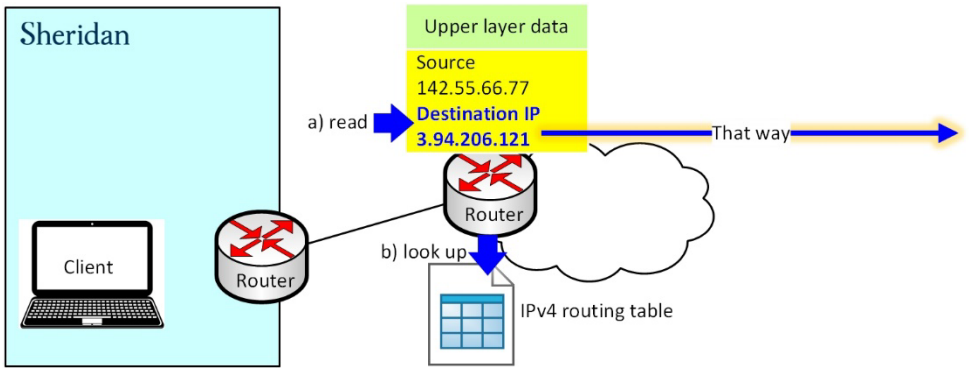
Second step in the delivery of traffic from client to server

The router looks 3.94.206.121 up in its IPv4 routing table. It knows, from BGP, that the address is located going the way of the next hop router located in the Internet Service Provider. It looks up the destination MAC address to reach the next hop, builds a new frame, and forwards the frame to its neighbour.



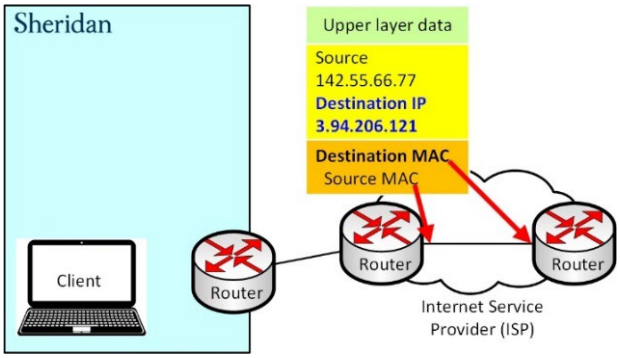
Third step in the delivery of traffic from client to server

The frame arrives to the next hop. The router removes the frame and it reads the fields in the IPv4 packet.



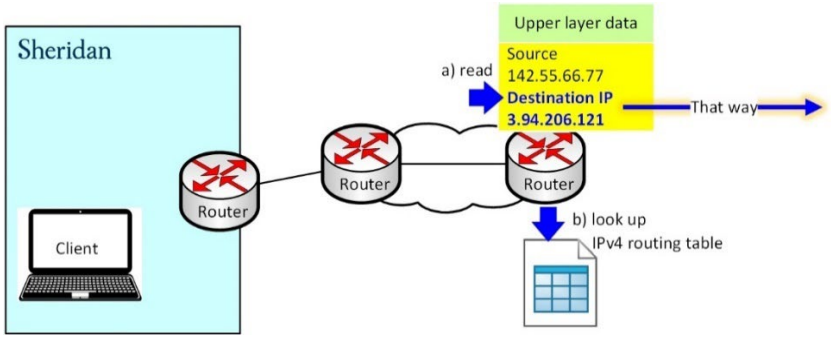
Fourth step in the delivery of traffic from client to server

The router read the destination IPv4 address, looks up the routing table, finds the next hop, and proceeds to build a frame to forward the data to the next router. Notice that the routers do not look into the upper layer data. They are only concerned with their job of forwarding traffic.



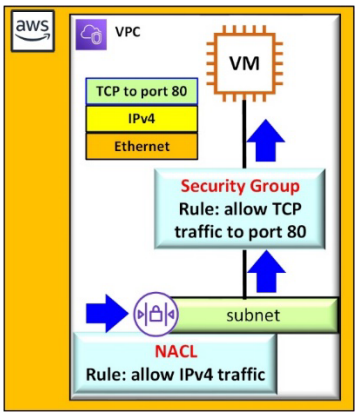
Fifth step in the delivery of traffic from client to server

Every router along the path does the same procedure: receive the frame, read the IPv4 header, look up the destination address, then forward the traffic to the next hop.



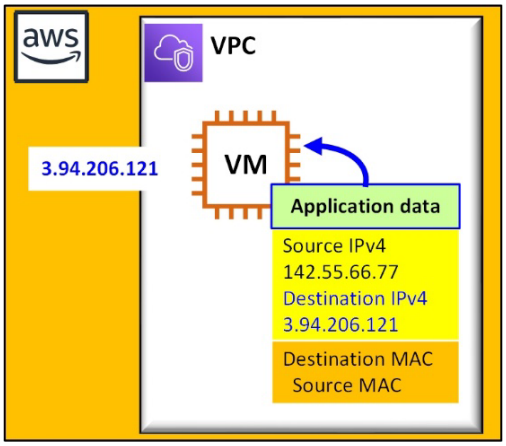
Sixth step in the delivery of traffic from client to server

The information eventually arrives to the VM running the webserver. Before, the data passes through two filters in the AWS cloud. The first filter is a **Network Access Control List (NACL)** and the second is the **Security Group**. The NACL controls the access of traffic at the subnet level while the security group controls the access to the EC2 instance. Both basic firewalls are explained in detail in another section.



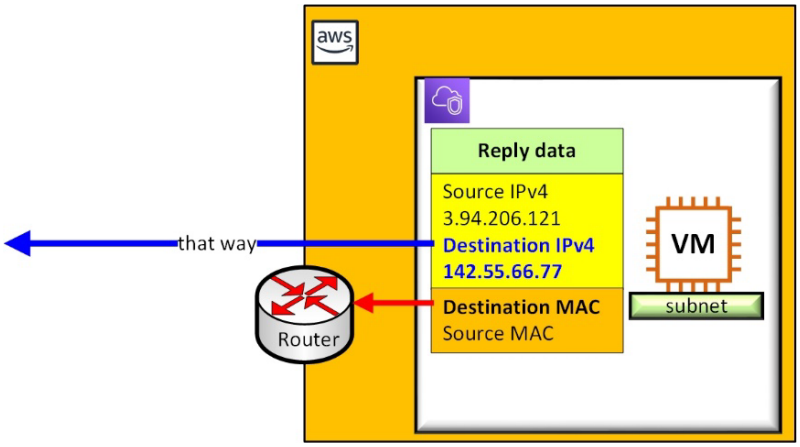
Traffic passing through the security filters in the VPC

Finally, the TCP request to open the server's port 80 (HTTP) is received by the EC2 instance. HTTP is the protocol used to exchange web information. The webserver accepts the request to start a HTTP conversation with the client.



Final step in the delivery of traffic from client to server

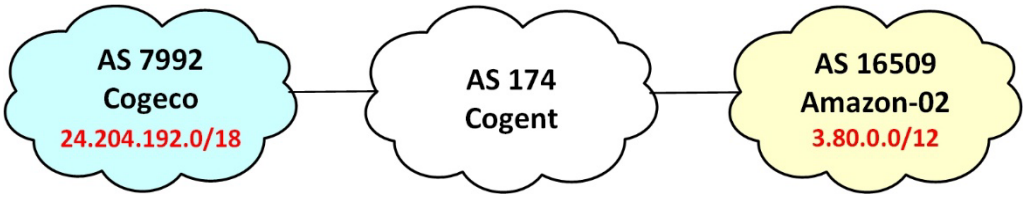
Now the webserver must reply to the IPv4 address of the client (142.55.66.77). The return process follows the same principle of encapsulation of data described before. A layer 2 protocol (Ethernet) contains the MAC address of the next hop device interface, an IPv4 packet contains the destination IPv4 address, and the upper layer protocols contain the data that carry on the end-to-end conversation.



Return message from server to client

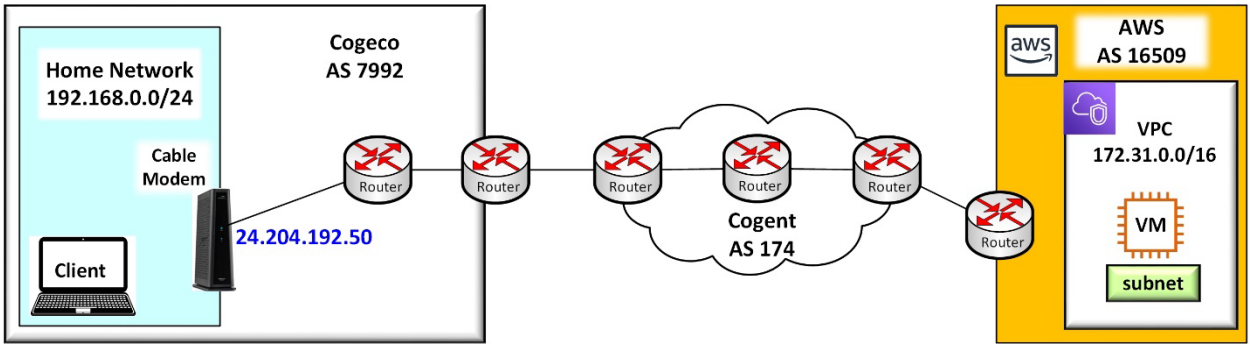
The frame is delivered to the default gateway router which inspects the IPv4 header to make a forwarding decision. Every router in the path follows the same procedure until the information arrives to the client. This is how the Internet works, the routers make decisions based on their tables of knowledge and such knowledge is acquired from the routing protocol. The routers look up the IPv4 routing tables to forward the traffic to the next hop router.

Let's switch the scenario now to showcase another concept. In this case, the client is no longer in Sheridan College, instead the client has gone home, literally. The Internet access of the home network is provided by Cogeco which assigns one of its IPv4 address (24.204.192.50) to it. Cogeco owns the AS 7992 [4] with many ranges of addresses. The general path between the customer and AWS would look like this:



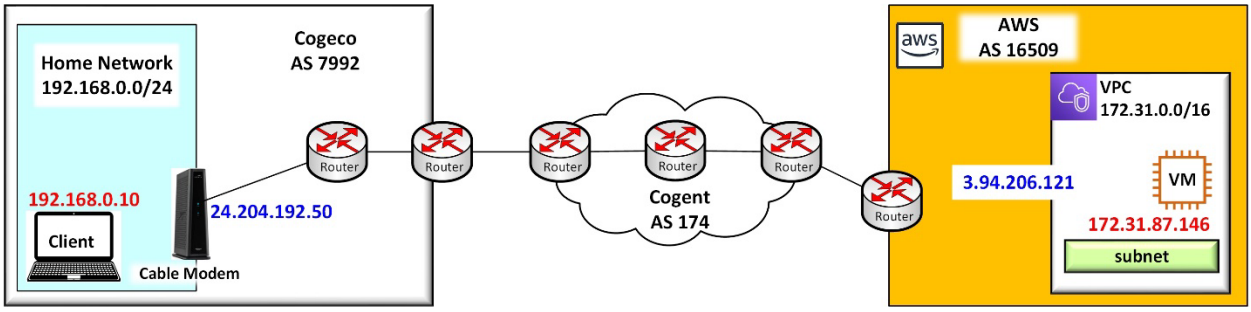
High-level view of the Internet path between a home network and AWS

In more detail, the home network appears to the Internet as one public address (24.204.192.50) that belongs to Cogeco [4]. Such address is assigned to the external interface of the home's cable modem. However, internally, the home network has the private IPv4 address range 192.168.0.0/24. The computer inside the home gets an IPv4 address from this range, for example 192.168.0.100. It is relevant to note that the same private block is reused by thousands of Cogeco home customers without any conflict.



Schematic view of an example home network with the path to AWS

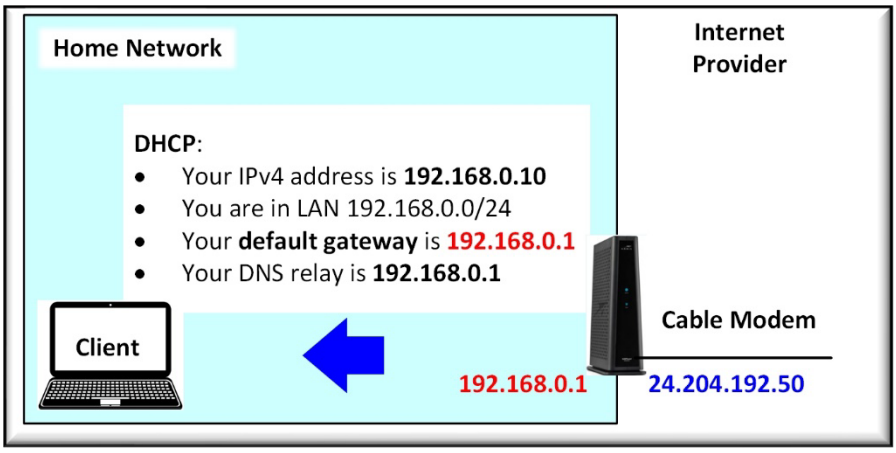
Notice, the parallel situation in the AWS side. The customers of AWS are also assigned a private IPv4 address block which is 172.31.0.0/16. The same private space is reused by AWS customers without any conflict. So, how does this work? Let's dissect the communication process from the client computer to the EC2 instance running in AWS. First, more details need to be added. The home computer has a private address 192.168.0.10 while the EC2 instance has a private address 172.31.87.146.



Public and private IPv4 addresses

The cable modem has multiple functions embedded. It is a Wi-Fi access point, a DHCP server, a DNS (name resolution) relay, a Wi-Fi access point, a basic firewall, and a default gateway router. All these functions are lump summed into one device to make it practical for the customers otherwise six devices would be required to do the same work.

When any host device connects to the home network, the DHCP server running in the cable modem provides all the networking information needed by the host. Beside assigning an IPv4 address to the host, the DHCP server also supplies the information about how to send traffic outside the Local Area Network. Notice that the cable modem has one address inside the LAN which is 192.168.0.1. If any home device needs to send traffic to an address on the Internet it must forward such traffic to the next hop with the address 192.168.0.1. In short, the modem is the **default gateway** router.

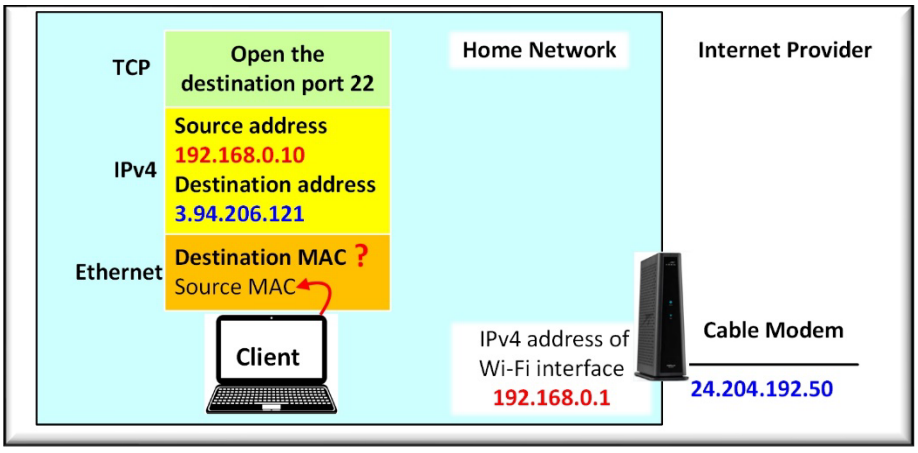


Example of a home network

The user decides to do administration work on the AWS EC2 instance from home. Thus, it opens an SSH client and types the destination IPv4 address like this:

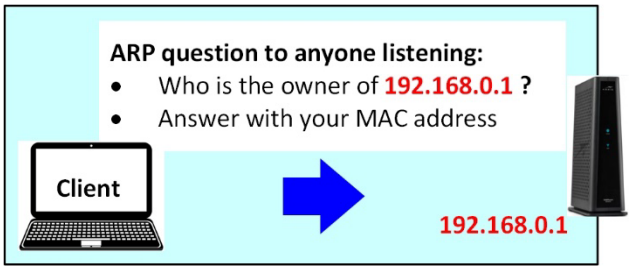
```
ssh ec2-user@3.94.206.121 – i  C:\Users\fc\ssh\key.pem
```

Which triggers the following:



Customer initiates an administration SSH session

The host computer has no clue where in the world the destination address 3.94.206.121 is located and it does not matter because the host has been informed by DHCP of the existence of a default gateway at 192.168.0.1. All that the host computer needs to do is to send an Ethernet frame with the destination MAC address of the antenna of the Wi-Fi access point (in the modem). A valid question now is how does the PC know such address? When the DHCP server first supplied the information about 192.168.0.1, the host sent an Address Resolution Protocol (ARP) request asking for the MAC address.

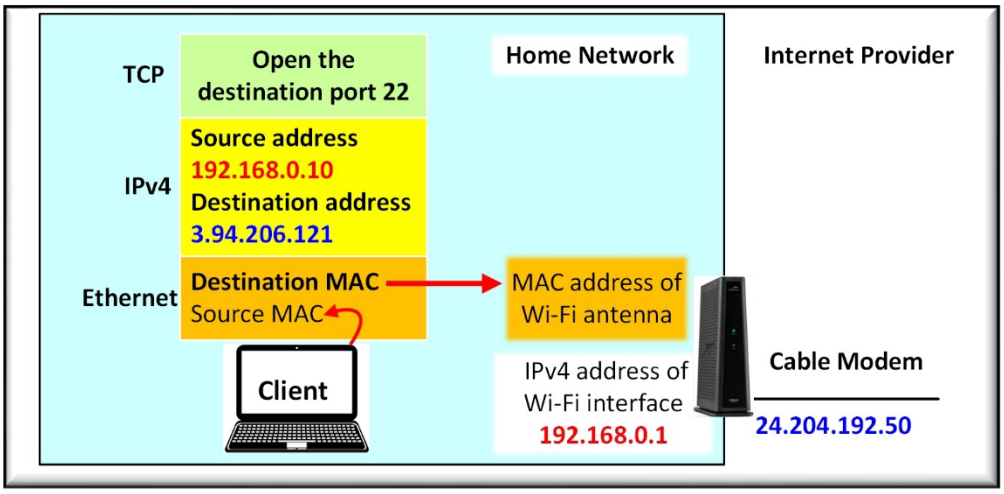


The client computer ARPs for the MAC address of the default gateway

The host computer receives an answer from 192.168.0.1 and it stores the MAC address in an ARP table. In consequence, the host PC does not need to repeat the question for a while (unless it is rebooted or shutdown). The contents of the ARP table are visible with the command:

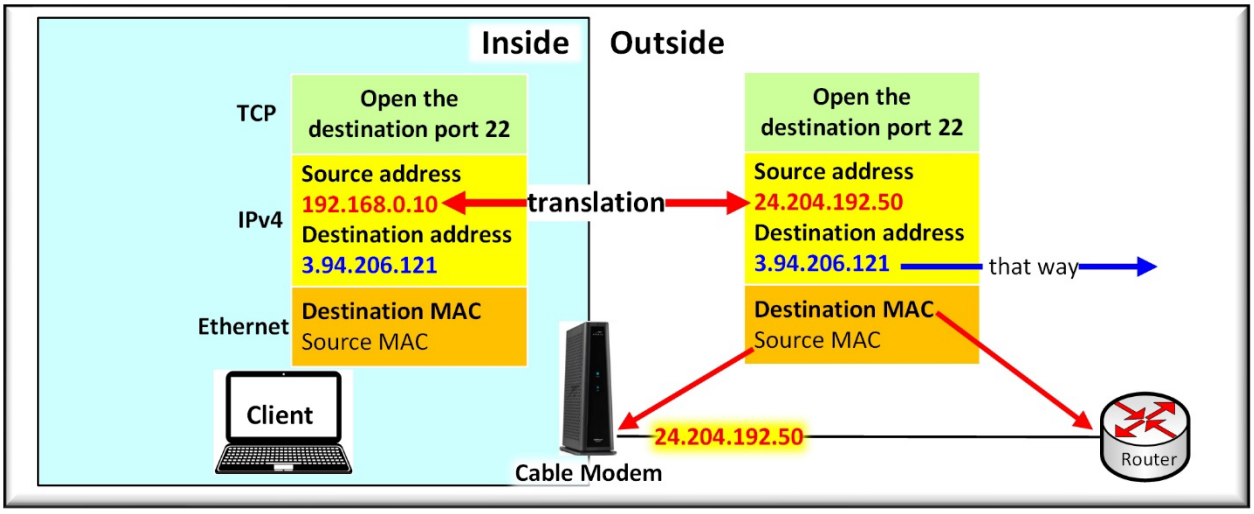
```
C:\Users\felix> arp –a
Interface: 192.168.0.10 --- 0xd
  Internet Address      Physical Address      Type
  192.168.0.1           c8-52-61-ba-68-2b     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
```

Now, the host computer can send the following encapsulated data:



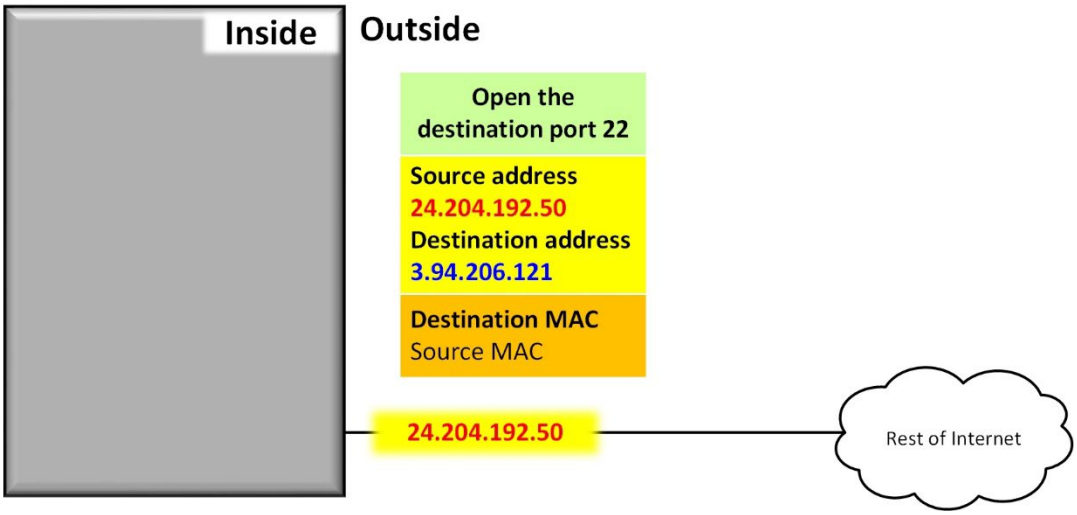
Client sends encapsulated data

Right now, the frame with the encapsulated message has arrived at the cable modem but there is a problem. The IPv4 packet has a source address 192.168.0.10 and that is a private address used by millions of computers everywhere. The IPv4 packet can not continue its journey toward the EC2 instance in AWS without a modification. The cable modem has yet one more function beside the six functions mentioned before. This device is also a **Network Address Translator (NAT)** that replaces any internal private address with the external public address 24.204.192.50.



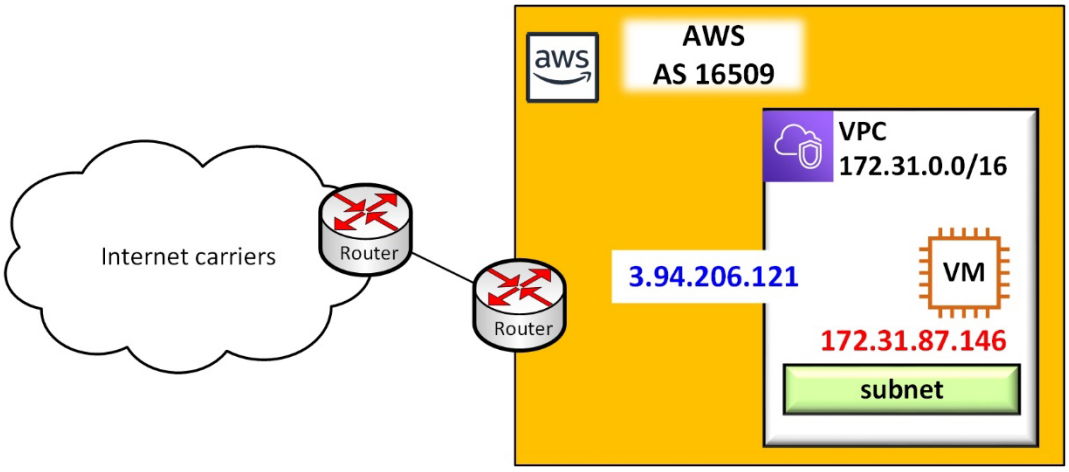
Translation between a private IPv4 address and public address

The NAT function keeps tracks of every session that is originated from within the home LAN going toward an Internet address by writing an identification record in a NAT table. That makes possible to handle multiple sessions. From the viewpoint of the destinations, all the sessions are sourced from the public address 24.204.192.50. The details of the home network are opaque to the public destinations.



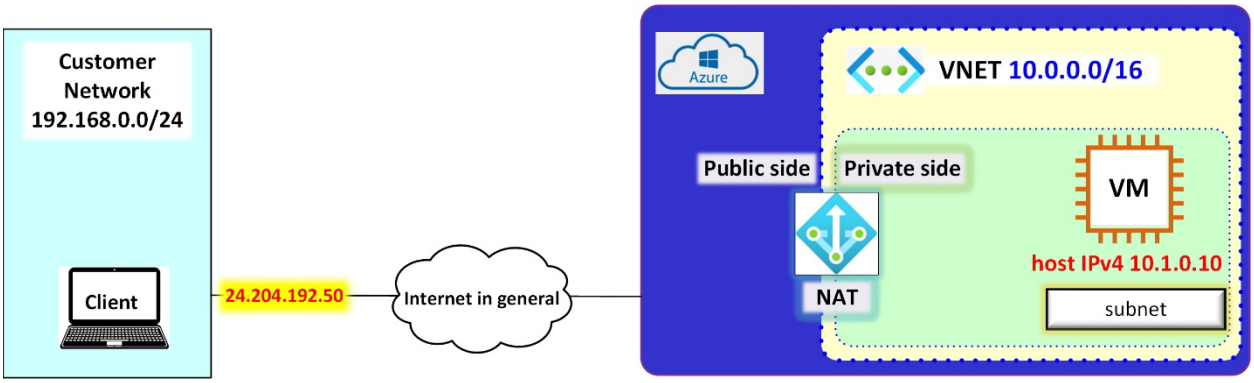
A home network appears as one public IPv4 address

Moving to the other side of the network connection, the logical topology of the AWS Virtual Private Cloud shows a similitude with the home network. Internally, the cloud provides the function of DHCP server, a DNS (name resolution), firewall, and gateway router. The virtual machines created in a default VPC always receive a private address from the space 172.31.0.0/16. Optionally, they can also get a public IPv4 address. The question is whether AWS performs Network Address Translation and if so, how. This is a technical detail that is not revealed for business reasons and for all that matters, the clients do not care much how that is done. An educated guess is that the network hypervisors must implement some mapping between the private addresses and the public addresses. So, in principle, a home network and the VPC implement similar concepts although at a different scale.



A home network appears as one public IPv4 address

To conclude this story, the same customer now decides to use another cloud provider, in this case Microsoft Azure. Notice again the conceptual similarity. Azure calls the private cloud space virtual network (VNET) instead of VPC but the idea is the same. This VNET has been created with the private address space 10.0.0.0/16 to be partitioned into subnets. A webserver is running in the virtual machine VM which has being assigned a private IPv4 address 10.1.0.10 by a DHCP service. This web service is reachable to customers on the Internet via a public IPv4 address assigned by Azure. A NAT function performs the network address translation between the public and private spaces. This comparison among the home network, AWS and Azure clouds exemplifies that the most fundamental principles of networking are the same regardless of the platform.



General Network Topology between a Client and a Server hosted in Azure

Chapter Summary

The fundamental principle of data communications is to deliver information from one place to another. This is achieved by the interaction of multiple devices and the operation of many protocols. When a company deploys complex scenarios that involve various cloud providers, a major consideration is the design of the networking infrastructure. Cloud or no cloud, IP packets need to travel from A to B. Those fundamentals have not changed with the irruption of the clouds. On the contrary, as organizations move their infrastructure to the clouds, the necessity for understanding the network operations is more important than ever. This chapter has offered a view of how data move from a customer to the cloud.

Chapter 4 coursework

This activity is intended to be done as a class seminar/discussion.

SSH into an EC2 instance.

Use the command **ip addr** in the command prompt.

```
ip addr
```

- What is the information that is obtained from this command?

Use the command **arp -a**.

```
arp -a
```

This should show the mapping of the MAC addresses to the corresponding IP address that the EC2 instance has learned so far.

- What are the addresses, both MAC and IPv4, that the EC2 instance has in the table?
- Why are they important for the functioning of the EC2 instance?

Use the command **route**.

```
route
```

This should show the basic routing table that the EC2 instance uses to forward data.

- What is the content of the table and why is it so basic? Should there be more information there?
- What is the address of the default router?
- What is the function of the default router in the AWS cloud?
- How does this information complement the information from the command arp -a?

Proceed to change the directory like this:

```
cd /var/lib/dhclient/
```

Display the information in the file dhclient--eth0.lease

```
cat dhclient--eth0.lease
```

This is the information that was provided by the DHCP server to the host (EC2 instance).

- Where is the IPv4 address of the DHCP server and where is it located?

Proceed to **ping** each of the following destinations:

```
ping 8.8.8.8
ping 142.55.47.60
ping www.sheridancollege.ca
ping www.google.ca
```

The utility ping is used to troubleshoot end to end connectivity. If the ping reply comes back, that means that the destination is reachable.

Observe that pinging to the names works as well as pinging to the IP addresses.

- How does the EC2 instance know the IP address that corresponds to the name?
- What is the network service inside AWS that solves this problem?

Now, proceed to **traceroute** as it follows:

```
traceroute 8.8.8.8
traceroute 142.55.47.60
traceroute www.sheridancollege.ca
traceroute www.google.ca
```

The utility traceroute shows the reached interfaces of the devices in the path to a destination. So using this utility is possible to infer a rough map of the networks that the message has crossed.

- Observe the number of hops that the traceroute messages have to cross to get to the destinations.

Use the commands **dig** and **nslookup.** These commands send queries to the local DNS server to ask for name resolution.

```
dig amazonaws.com
nslookup amazonaws.com
dig www.sheridancollege.com
nslookup www.sheridancollege.com
```

- What is the address of the local DNS server? Where is this server located?

The address **169.254.169.254** is a **Link Local** address. That means, it is only reachable by the resource that is connected to the link, in this case, the EC2 instance. This address is configured by AWS to respond to HTTP protocol queries requesting **metadata**.

Use the **curl** command to query the link local address for the metadata of the EC2 instance to find out:

What is the instance type?
What is the operating system id of the instance?
What is the microprocessor of the instance?
What is the identification string of the instance?
Where is the instance located regarding availability zone?
What is the availability zone id?
What is the security group applied to the instance?
What is the public IPv4 address of the instance?
What is the private IPv4 address of the instance?
What is the MAC address of the interface?
What is the hostname of the instance?
What is the private DNS name of the instance?
What is the public DNS name of the instance?

```
curl http://169.254.169.254/latest/meta-data/
curl http://169.254.169.254/latest/meta-data/instance-type
curl http://169.254.169.254/latest/meta-data/ami-id
curl http://169.254.169.254/latest/meta-data/system
curl http://169.254.169.254/latest/meta-data/instance-id
curl http://169.254.169.254/latest/meta-data/placement/availability-zone
curl http://169.254.169.254/latest/meta-data/placement/availability-zone-id
curl http://169.254.169.254/latest/meta-data/placement/security-group
curl http://169.254.169.254/latest/meta-data/security-groups
curl http://169.254.169.254/latest/meta-data/public-ipv4
curl http://169.254.169.254/latest/meta-data/local-ipv4
curl http://169.254.169.254/latest/meta-data/network/interfaces/macs
curl http://169.254.169.254/latest/meta-data/mac
curl http://169.254.169.254/latest/meta-data/hostname
curl http://169.254.169.254/latest/meta-data/local-hostname
curl http://169.254.169.254/latest/meta-data/public-hostname
```

References

[1] AS14618 Amazon.com. BGP View. Accessed. June 2021. [Online]. Available: https://bgpview.io/asn/14618#graph

[2] Frontier. Accessed. June 2021. [Online]. Available: https://frontier.com

[3] Cogent. Accessed. June 2021. [Online]. Available: https://www.cogentco.com/en

[4] 24.204.192.0/18 Cogeco Communications Inc. BGP View. Accessed. June 2021. [Online]. Available: https://bgpview.io/prefix/24.204.192.0/18#info

[5] AWS. AWS IP address ranges. Accessed. June 2021. [Online]. Available: https://docs.aws.amazon.com/general/latest/gr/aws-ip-ranges.html#aws-ip-download

Chapter 5

Cloud Access Control

Description

This main topic of this section is how to control the access to resources in the cloud. There are different tools and methods available to control the access to services deployed in a cloud. Nevertheless, in all of them there is a recurrent pattern which is to let the customer to choose the protocols and source addresses that can access a server. The clouds provide a configuration interface that interacts with the software-based control mechanisms such as security groups and network access control lists.

Learning Outcomes

- Analyze protocol encapsulation to implement secure access to resources.
- Apply the concept of port and address control to configure security groups.
- Configure and test network access control for elastic virtual computers.
- Test security groups to control access to virtual machines.

Main concepts

- Open system layer model and encapsulation of protocols.
- Protocol fields: ethertype, protocol, and ports.
- Frames, packets, segments, and messages.
- Security groups.
- Network access control lists.

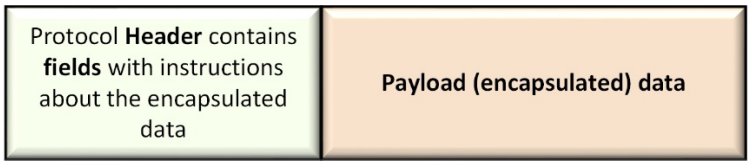
Learning Activities

- Create and Apply a NACL
- Configure and Apply Security Groups

Controlling the access to cloud resources

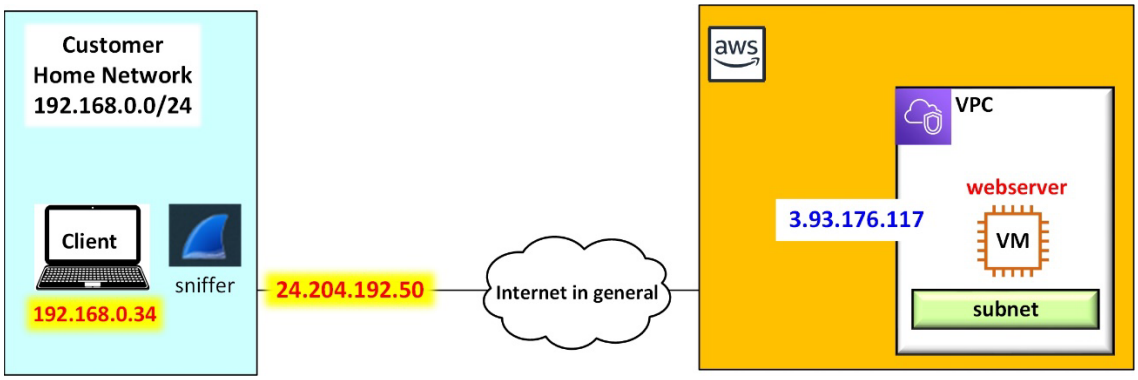
Cloud providers follow the same concept to control the access to the resources and service. The reason for this similarity is that access control is bound to the most fundamental concepts of networking. All data, that travels from one device to another in any network, follows the same universal rules. Hence, it is vital to understand the process of protocol encapsulation to be able to implement any type of secure control.

Communication across networks is achieved thanks to the organization of data according to the functional layers of a networking model. Every protocol has two main portions, a header with instructions and a payload to carry information. The header contains fields pertaining to the function of the protocol. For example, the header of the IPv4 protocol contains the sender's IP address and the destination IP address where the packet should go. Inside the payload, there might be another encapsulated protocol. For example, IPv4 may carry TCP segments, and TCP segments may carry web information.



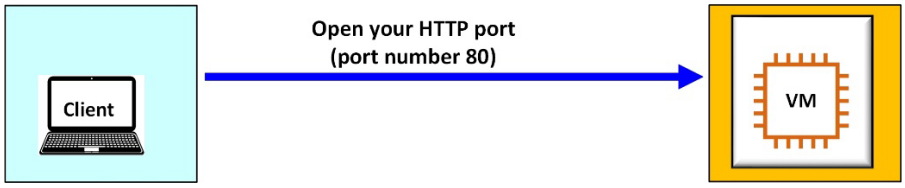
Protocol header and encapsulated data

To make more relevant this explanation, let's take the case of a client starting a conversation with a webserver located in a cloud. In the following topology, a network sniffer tool is used to capture the conversation and furthermore to analyze the structure of the data.



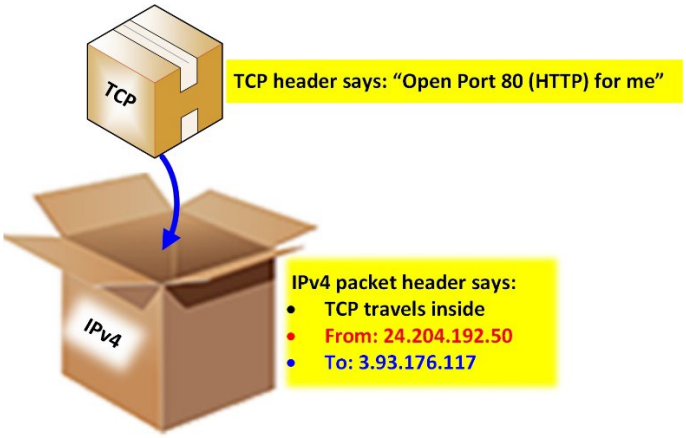
Topology to analyze a protocol conversation

The conversation starts when the client writes the public IPv4 address of the webserver in its web browser. The first message that the client sends is a request to open the server's HTTP. In simple terms, the client tells the server: "please open your HTTP service port so I want to talk with you".



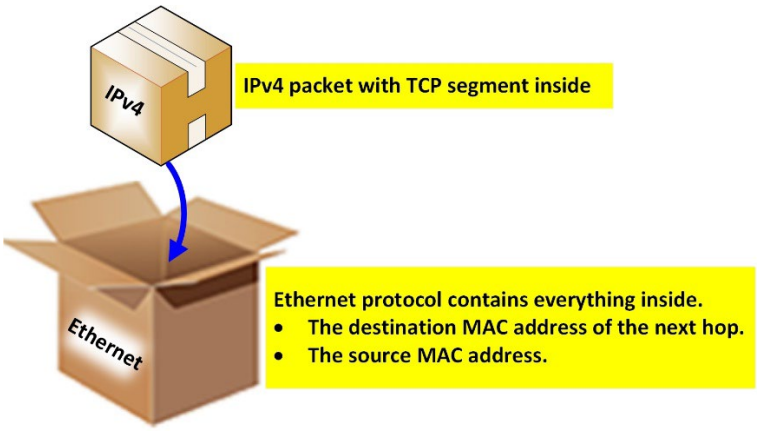
Request to start a conversation

The protocol that delivers the opening message of the conversation is the Transmission Control Protocol. TCP main function is to keep control of the end-to-end conversation, things like opening the session, arranging the data in order, retransmitting missing data, and closing the conversation. However, TCP can not reach the remote location by itself. It needs the help of another protocol that can carry packets of conversation from source to destination across the Internet and back. Hence, TCP needs the help of the Internet Protocol either IPv6 or, in this example, IPv4.



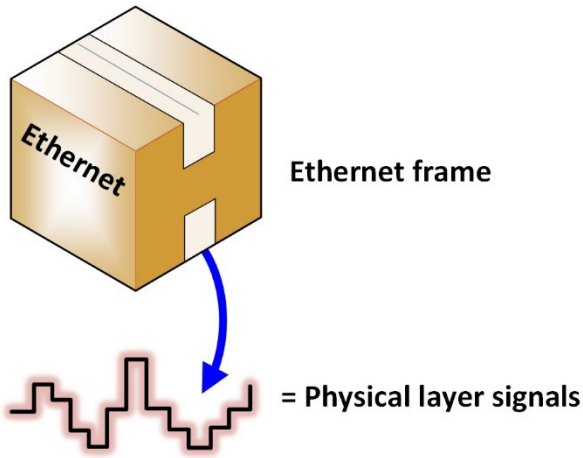
Protocol IPv4 encapsulating a TCP segment

The IPv4 header has several fields but there are three more relevant to this explanation: the protocol field, the source IPv4 address and the destination IPv4 address. First, the protocol field is a unique number that identifies the protocol (TCP in this case) being carried in the payload. Second, the source IPv4 address is used to reply to the other side of the conversation. Last, the destination IPv4 address is used by the routers in the path to determine where to send the packets. However, to move from router to router, the IPv4 protocol needs the help of a layer 2 protocol; for example, Ethernet which is the most commonly used layer 2 protocol. Thus, the IPv4 packets are encapsulated inside Ethernet frames that has the destination MAC address of the next physical hop.



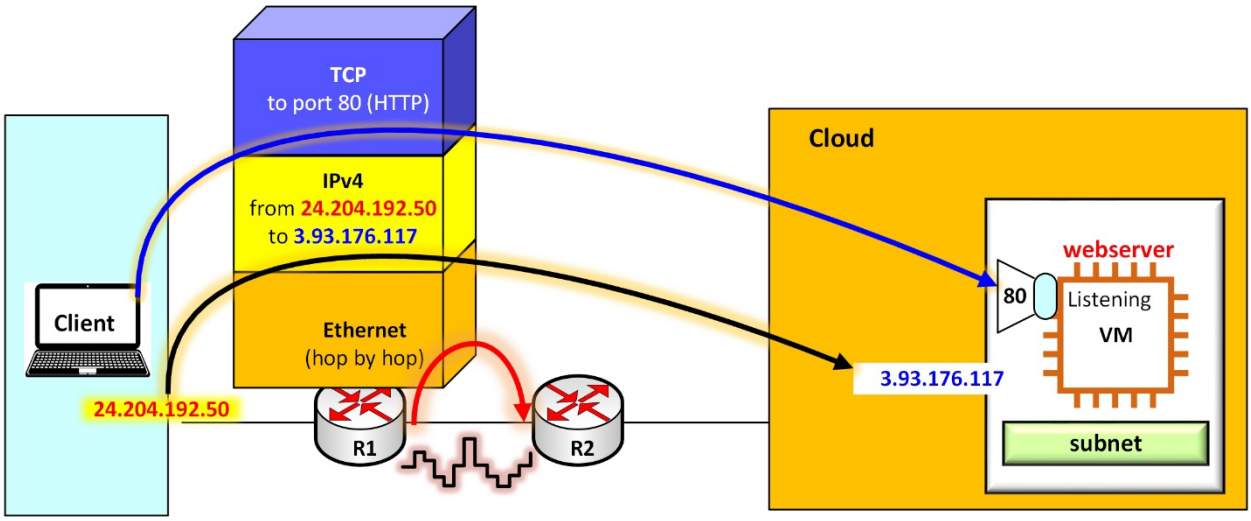
Protocol Ethernet encapsulating everything

The Ethernet frame containing everything from the upper layers is converted into electromagnetics signals by the physical layer and these signals travel over media that can be wired, optical fibre or radio.



The physical layer signals carry the information

An overall view of the process is depicted in the next diagram. The frame is forwarded to the next hop router R2. It contains an IPv4 packet from 24.204.192.50 going to 3.93.176.117. This packet contains a TCP header directed to port 80 to open an HTPP conversation. The physical layer signals on the wire transmit the information. Notice that each layer has a functional purpose, each one does a particular job.

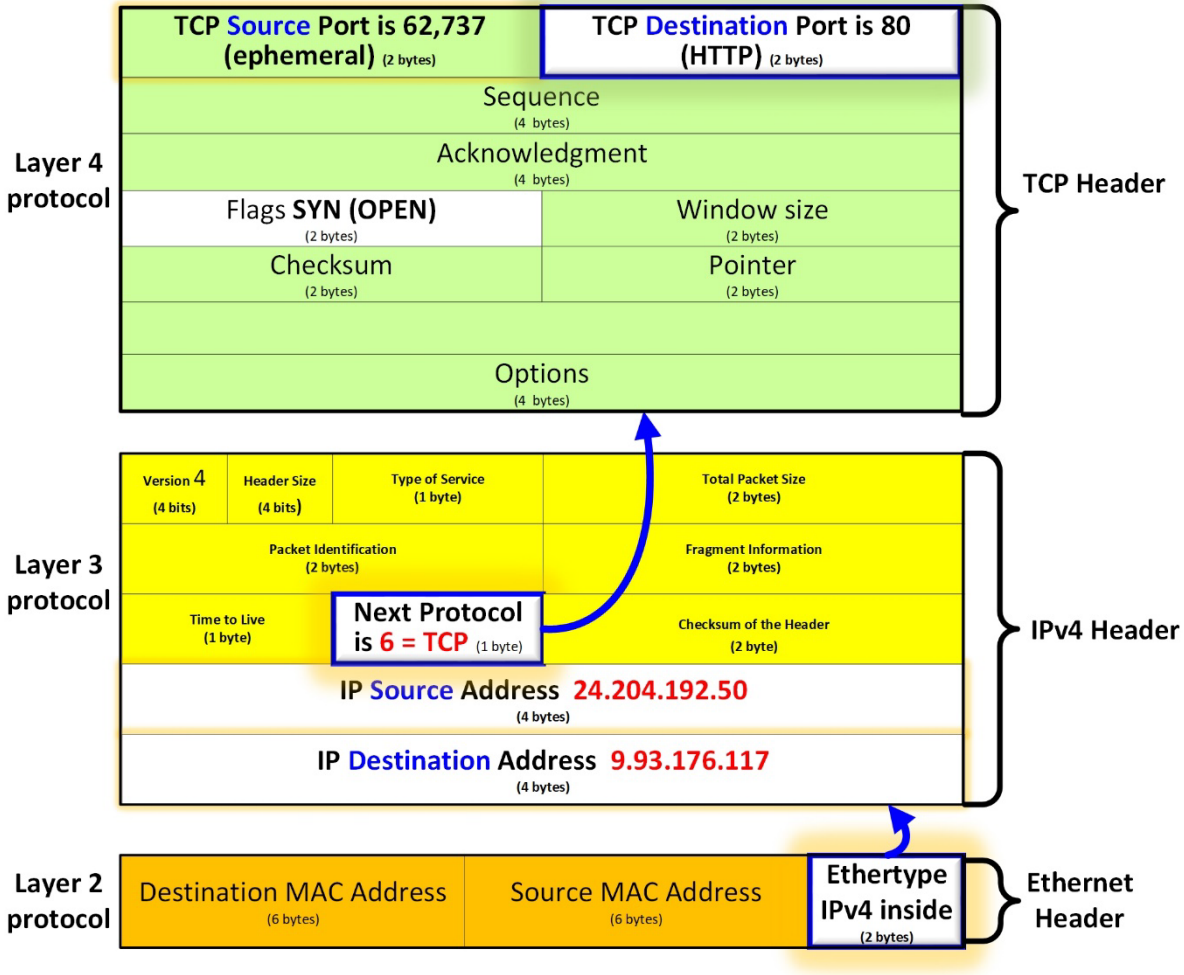


An overall view of the functions of the protocols

The information eventually arrives to the cloud hosting the webserver, but before it has to pass through security mechanisms that protect the server and this is the whole point of this explanation. To show how these security filters are intrinsically tied to the fields in the protocol headers. For the current example, there are three security access questions that must be answered:

- What are the IPv4 source addresses allowed inbound?
- What are the TCP port numbers allowed inbound?
- If traffic is allowed inbound, what is the traffic allowed outbound?

The following diagram shows the encapsulated data now in full detail. The most relevant fields of every protocol header are highlighted in the white boxes. The access security measures are applied precisely to those fields. If the access control is done at the network layer, the fields to control are in the IPv4 packet whereas if the protocol to control is at an upper layer, then there are additional fields to manage.



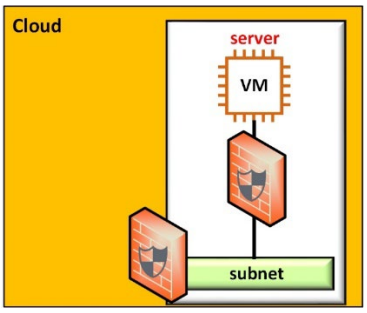
Detailed view of the encapsulated protocols

The principle of security access control is based on observing the fields in the protocols and taking actions based on their values. For instance, at the network layer, the most strict security filter would allow only IPv4 traffic originated from 24.204.192.50 to 9.93.176.117. However, that would make no sense for a webserver that is supposed to be open to the public. Thus, in such case, a more appropriate filter would allow traffic from every Internet address (0.0.0.0/0) to the destination 9.93.176.117 only.

Furthermore, at an upper layer, the server might have several ports open for different reasons. One of them is for secure administration access via SSH. The general public should not have access to that port, they should only have access to the webserver's port 80. Therefore, a second filter would allow access to SSH only from 24.204.192.50 while allowing access to everybody else to HTTP.

Clearly, anyone who wants to pursue a career in network and computer security needs to have a thorough understanding of the mechanics of the protocols involved.

Cloud providers implement security services at different places and levels; however, two basic models are recurrent: access control at the network (subnet) level and access control at the interface level. AWS implements **Network Access Control List (NACL)** to the **access control** at the **subnet level** and **Security Groups** for **access control** at the **interface level**. Microsoft Azure only implement Security Groups that can be applied either at the subnet level or at the interface level.

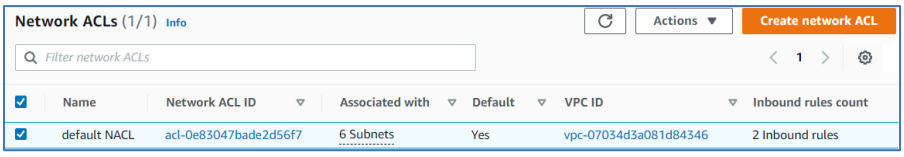


Access control at the subnet level and interface level

Network Access Control List (NACL)

A Network Access Control List (NACL) is a **stateless firewall** that is applied to the network (subnet) in the AWS cloud (Azure does not have NACLs). Stateless means that the firewall does not keep track of the traffic that has been allowed one way henceforth there must be specific rules to allow the traffic the other way around. If a NACL rule allows inbound traffic to pass toward a service, then the outbound NACL must have a specific rule to allow the response to pass thru also.

Thus, an AWS NACL is a firewall controlling both inbound and outbound traffic for associated subnets. NACLs are part of the VPC service and every subnet present in the virtual private cloud must be associated to one. The customer's default VPC (172.31.0.0/16) has six subnets if the base region is us-east-1. These subnets are associated by default to a pre-existing NACL. The following snippet shows that situation.

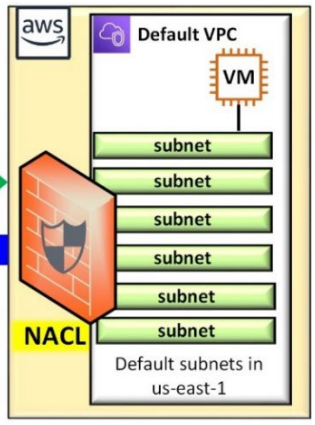


Rule number	Type	Protocol	Port range	Source	Allow/ Deny
100	All traffic	All	All	0.0.0.0/0	Allow
*	All traffic	All	All	0.0.0.0/0	Deny

Inbound

Outbound

Rule number	Type	Protocol	Port range	Source	Allow/ Deny
100	All traffic	All	All	0.0.0.0/0	Allow
*	All traffic	All	All	0.0.0.0/0	Deny

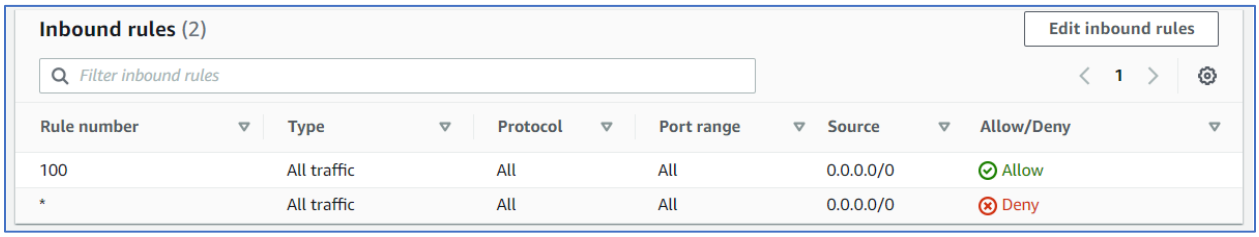


The default NACL of the default VPC

The pre-existing NACL has two parts, one with the inbound rules and another with the outbound rules and this has to do with the stateless nature of this security filter. A NACL is an ordered list of rules that it works by processing the rules from the lowest rule number to the highest. Once that a match is found, the respective action (allow or deny) is taken and the processing of the rules stops at that point. Latter rules are not considered after a rule match is found. If no match exist, the processing continues until the end where there is always a default deny all.

The following inbound NACL rules read like this:

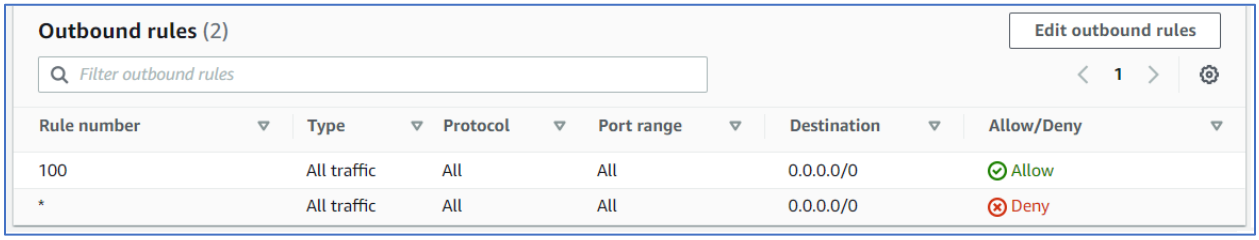
- Rule 100 for all traffic, all protocols and all port coming from everywhere, allow.
- Default implicit rule, deny else.



The Inbound Rules of the default NACL

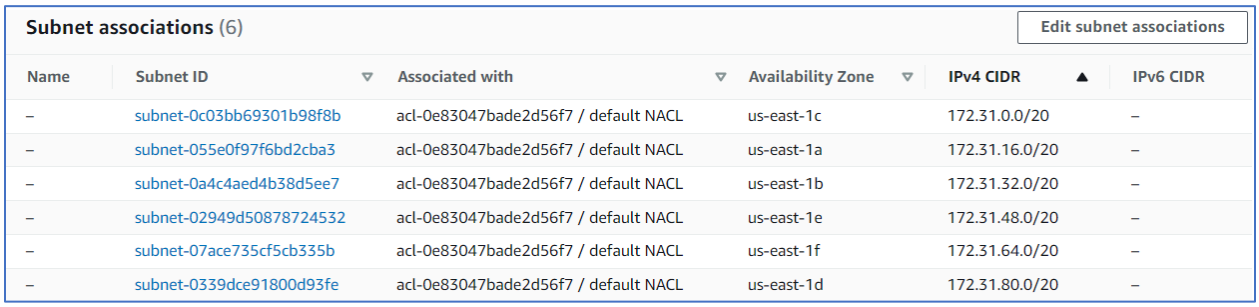
The following outbound NACL rules read like this:

- Rule 100 for all traffic, all protocols and all port going everywhere, allow.
- Default implicit rule, deny else.



The Outbound Rules of the default NACL

The default NACL is associated to the six subnets existing in the default VPC in AWS region us-east-1. This means that the access to any resource attached to any of these subnets is controlled by the NACL. However, this association can be changed to newly created NACLs with different rules.



Subnet association of the default NACL

The NACL is structured with six columns which are the rule number, the traffic type, the protocol evaluated, port range, the IP source of the traffic and the decision allow/deny. Let's see each field in detail.

Rule Numbers

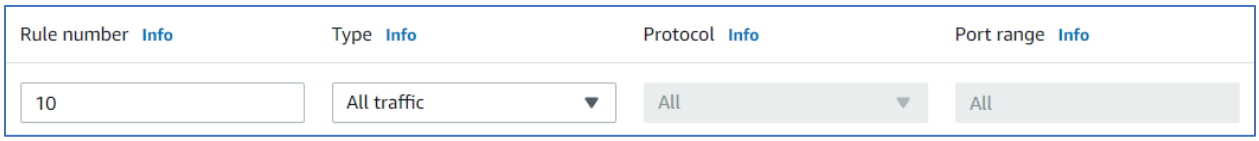
The rule numbers must be assigned incrementally from lowest to highest and it is recommended that they have steps to allow inserting future rules. For example, starting with rule 5, then rule 10, rule 15, etc. This practice leaves space to insert future rules. Otherwise, if there is no room and a NACL needs to be modified, the NACL must be completely rebuilt. The highest possible line number in a NACL list is 32,766 but AWS has quotas for the total number of rules

Type

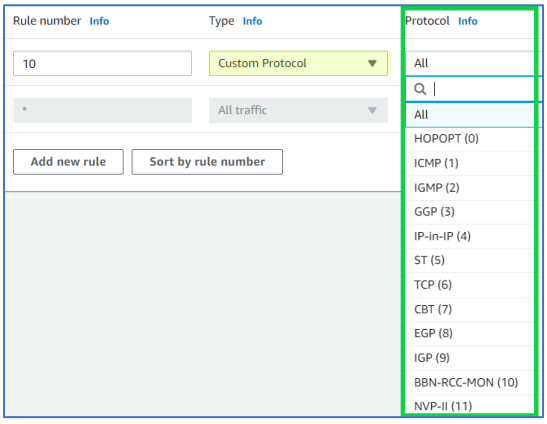
This column is used to specify the type of traffic. It provides a list of the most commonly used protocols to choose from. The list is comprehensive and it is very likely that the protocol to match will be found there.

All traffic	All TCP	SMTP (25)	IMAP (143)	POP3S (995)	PostgreSQL (5432)
Custom protocol	All UDP	Nameserver (42)	LDAP (389)	MS SQL (1433)	Redshift (5439)
Custom TCP	All ICMP - IPv4	DNS (TCP) (53)	HTTPS (443)	Oracle (1521)	WinRM-HTTP (5985)
Custom UDP	All ICMP - IPv6	DNS (UDP) (53)	SMB (445)	NFS (2049)	WinRM-HTTPS (5986)
Custom ICMP – IPv4	SSH (22)	HTTP (80)	SMTPS (465)	MySQL/Aurora (3306)	HTTP* (8080)
Custom ICMP – IPv6	telnet (23)	POP3 (110)	IMAPS (993)	RDP (3389)	HTTPS* (8443)

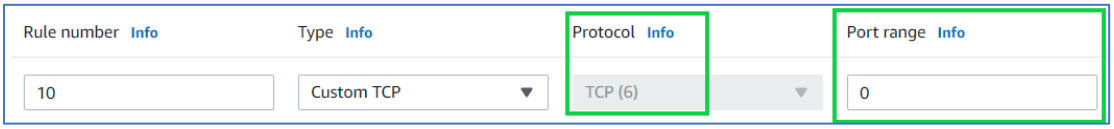
- **All traffic** includes everything as the name indicates. When this option is chosen, all the protocols and all the ports are included. Obviously, this is a lax rule option, so it must be used in combination with the source/destination column to make it more granular.



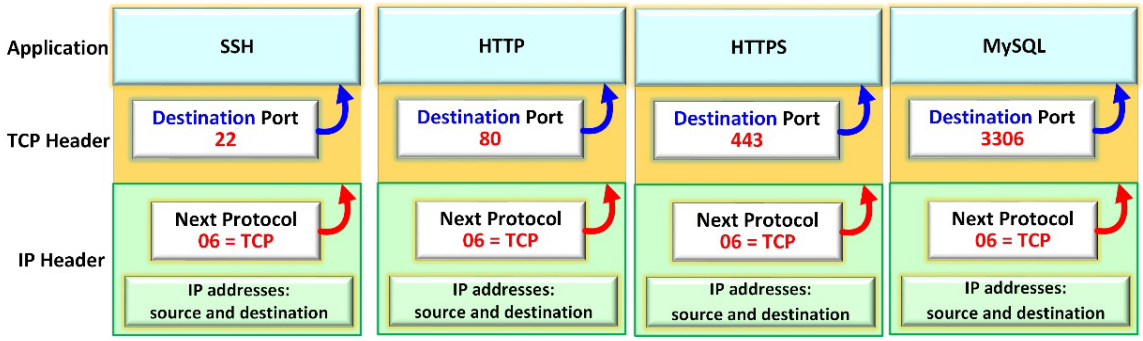
- **Custom protocol** must be used in conjunction with the column protocol where an even larger list of protocols is available. These protocols are for specialized situations instead of regular traffic.



- **Custom TCP** is the option that would match most services and applications. This type pre-sets the layer 4 protocol or transport layer to the unique identifier 6 which means TCP. Then any application layer protocol can be selected as long as the port number is specified.

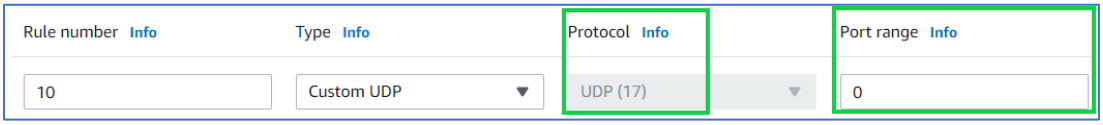


- For example, to allow secure access SSH protocol the port number must be set to 22, to allow web HTTP the port must be set to 80, to allow secure web HTTPS the port must be set to 443, and to allow MySQL the port number must be set to 3306. All of these protocols are transported inside TCP segments as the following figure illustrates.

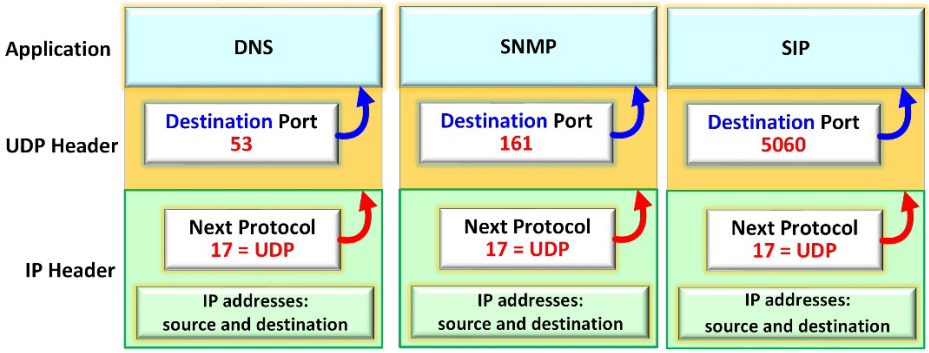


Four frequently used applications encapsulated by TCP

- **Custom UDP** has the protocol field in the IPv4 packet set to number 17 which is the unique identifier for the User Datagram Protocol (UDP).

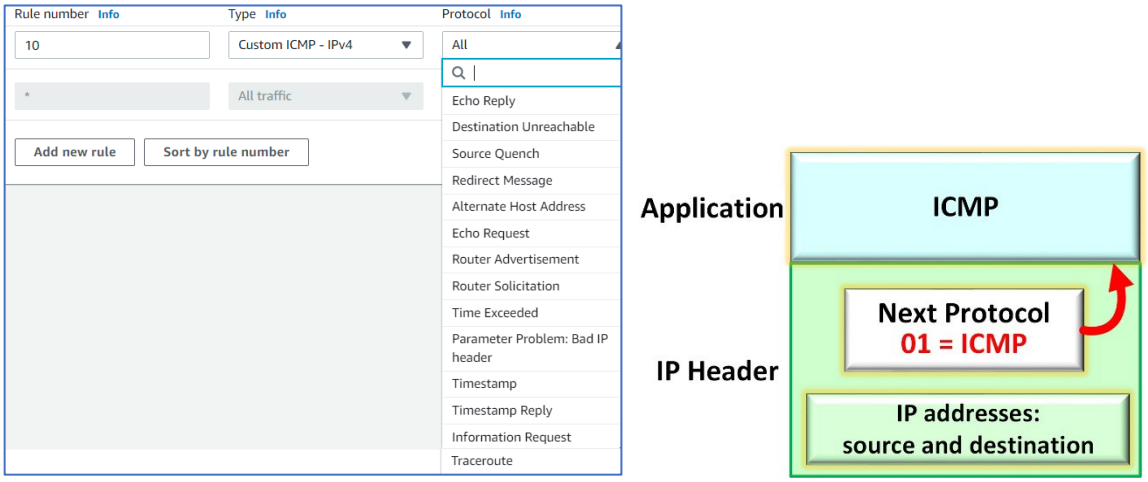


- Some applications are transported by UDP because it is a bare bone, faster, although unreliable transport layer protocol. Applications that do not require the reliability provided by TCP might be ported on UDP like DNS requests to resolve web names to IP addresses, SNMP queries used for monitoring applications, SIP for signalling voice over IP and video game applications.



Three applications encapsulated by UDP

- **Custom ICMP** is used for troubleshooting purposes either for IPv4 or IPv6. The Internet Control Message Protocol messages are delivered directly inside IP packets. ICMP is identified with the protocol number one. There is no transport layer involved in ICMP as the diagram below indicates. The NACL column "protocol" actually shows the options messages of ICMP (not protocols).



The ICMP protocol encapsulation and options in the NACL

- ICMP is used by the useful tools **ping** and **traceroute**. The tool ping uses the options echo request and echo reply to troubleshoot reachability to an IP address. The tool traceroute lists the IP network addresses in the path to a destination.

Other protocol options present in the NACL configuration are:

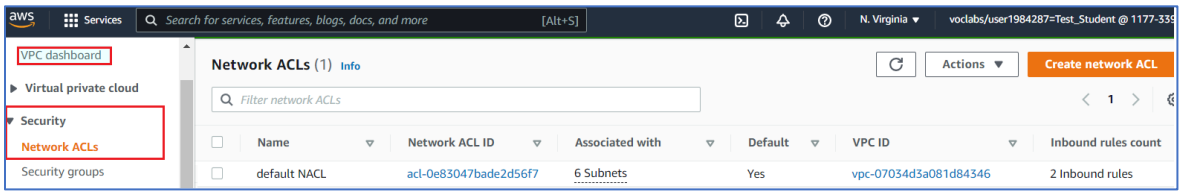
- SSH (22) Secure Shell Protocol to securely access a resource such as a server.
- telnet (23) is the equivalent of SSH but completely clear text (insecure). This should not be used.
- SMTP (25) Simple Message Transport Protocol is for email delivery.
- Nameserver (42) is a legacy name resolution protocol.
- DNS (UDP) (53) Domain Name System is the protocol that resolves names such as www.sheridancollege.ca to the IPv4 address 142.55.7.60. When a client types a web name in the browser, a DNS name resolution request is sent over an UDP segment.
- DNS (TCP) (53) is used by the DNS servers to exchange name records. A primary server hosts the name database which is replicated over to secondary DNS servers. In this case, DNS is transported on TCP segments.
- HTTP is the Hypertext Protocol that carries over the websites information. It is clear text.
- HTTPS is the same HTTP but ciphered.
- Post Office POP3 (110) and Internet Message Access Protocol IMAP (143) are used by email servers.
- LDAP (389) Lightweight Directory Access Protocol is a directory system to store organizational records such as a company structure with departments and users.
- SMB (445) is a file sharing protocol.
- MS SQL (1433), Oracle (1521), MySQL/Aurora (3306), and PostgreSQL (5432) are Structure Query Language used in relational databases.
- Redshift (5439) is an AWS application to warehouse large data and to perform analytics on it.

The NACL configuration tool makes possible to control any type of traffic that exists. It is time to put this knowledge into a practical use.

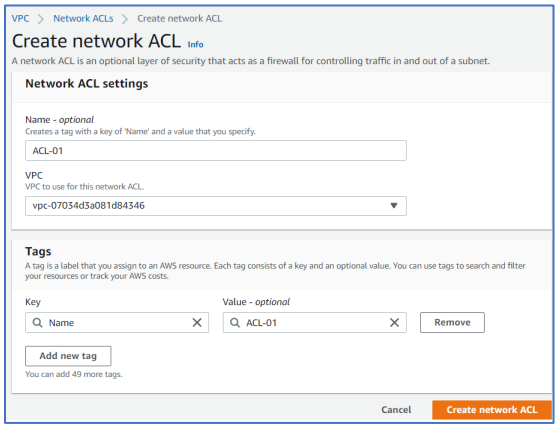
Learning Activity

Create and Apply a NACL

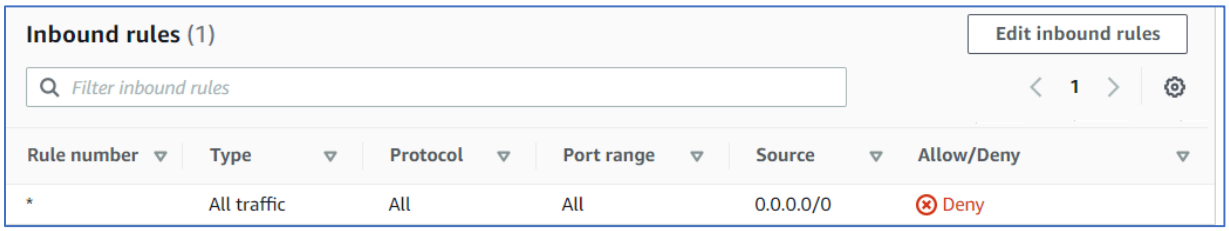
- Go to VPC services in the AWS Academy account.
- Find Network ACLs under Security.
- Notice that there is a default NACL associated to 6 subnets.



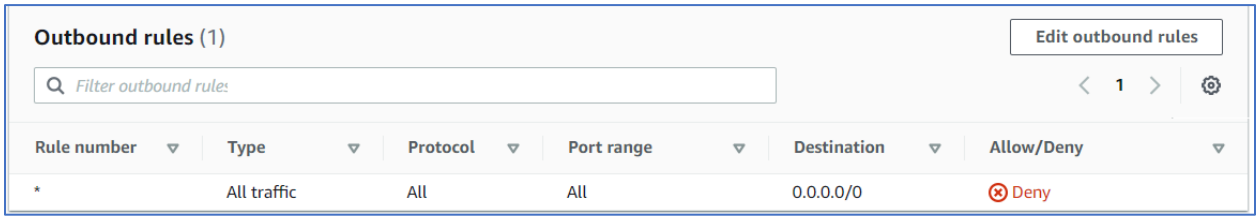
Create a new network ACL.



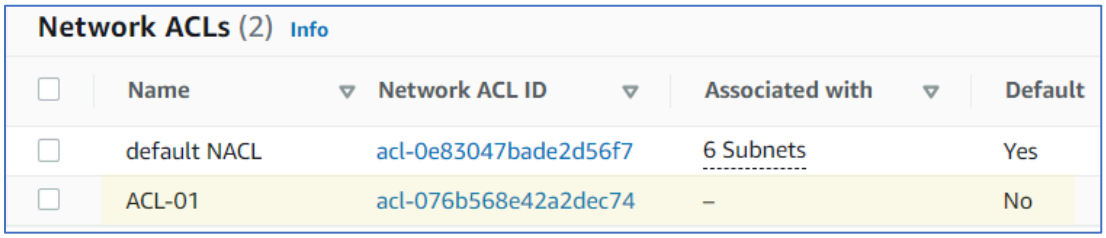
- Once that the NACL is created, look into the inbound and outbound rules.
- All the inbound traffic is denied by default in a new NACL.



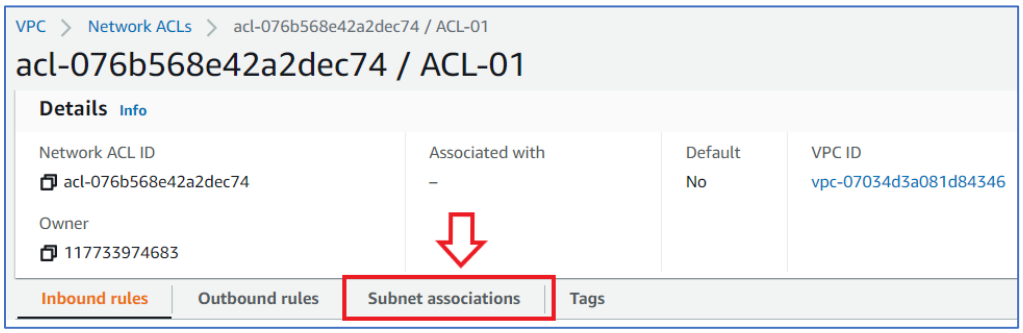
- All the outbound traffic is denied by default in a new NACL.



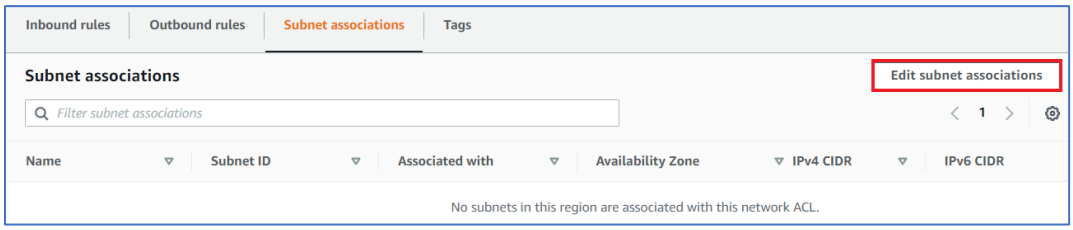
- Leave the rules as they are by default. They will be modified later.
- Notice that the new NACL is not associated with any subnet (so it is not being used).



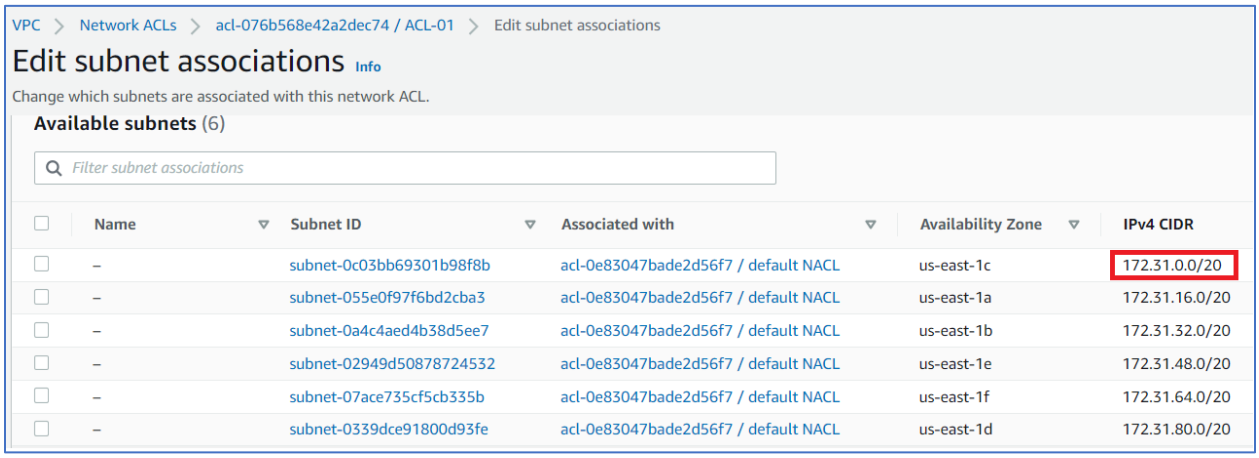
- Click on the new NACL.



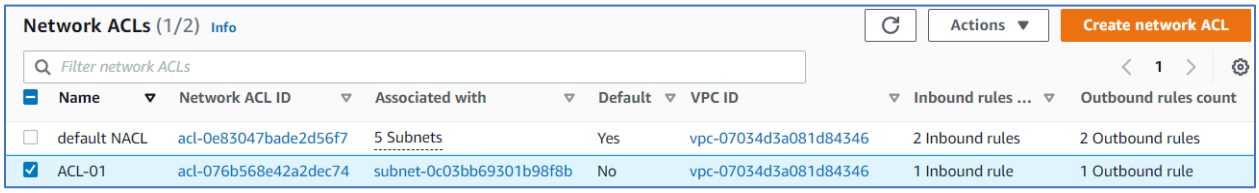
- Edit subnet associations



- Choose the subnet 172.31.0.0/20.

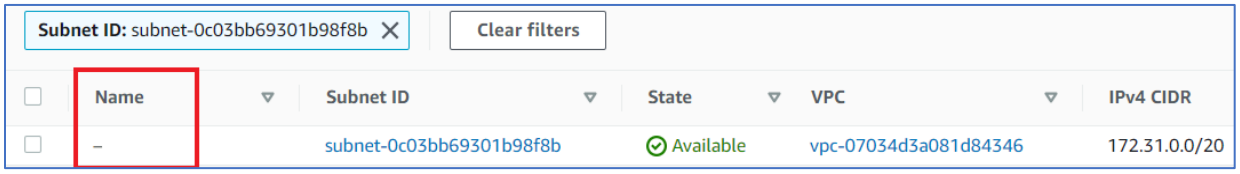


- Now the subnet is associated with the new NACL (but no longer with the default NACL).

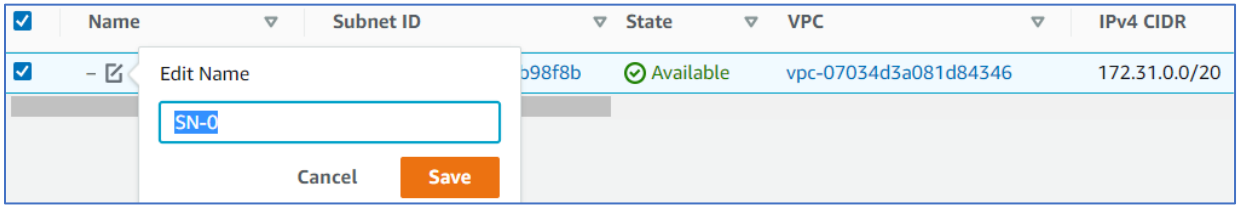


- Click on the subnet unique identifier.

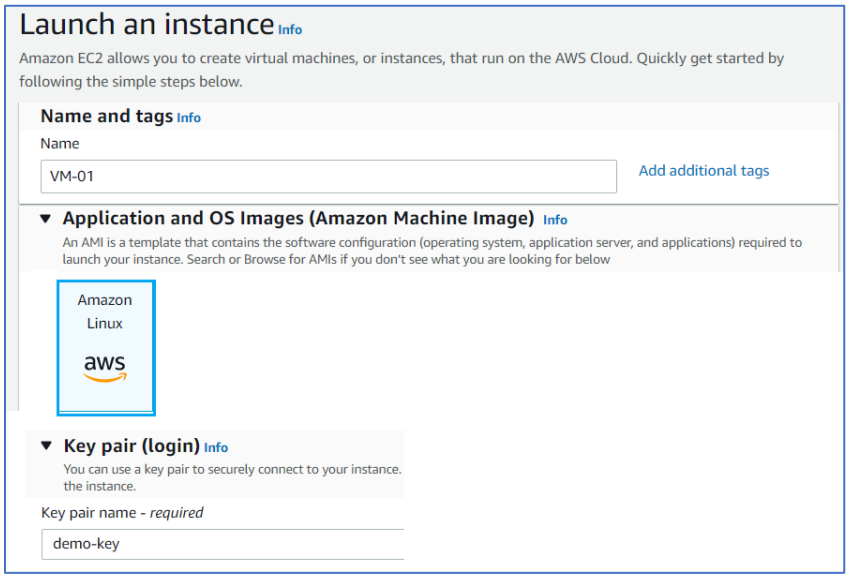
- Assign a name to the subnet so it will be easier to identify.



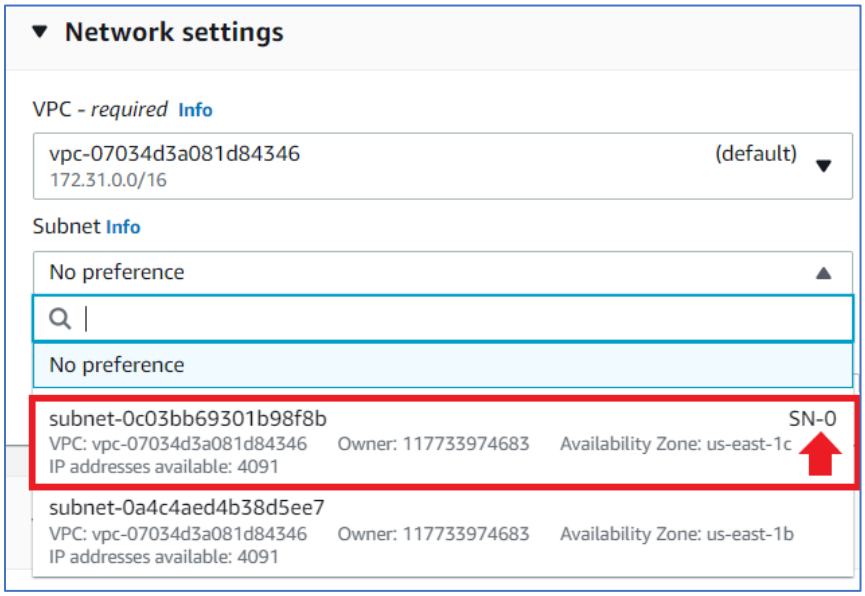
- The name will be needed later.



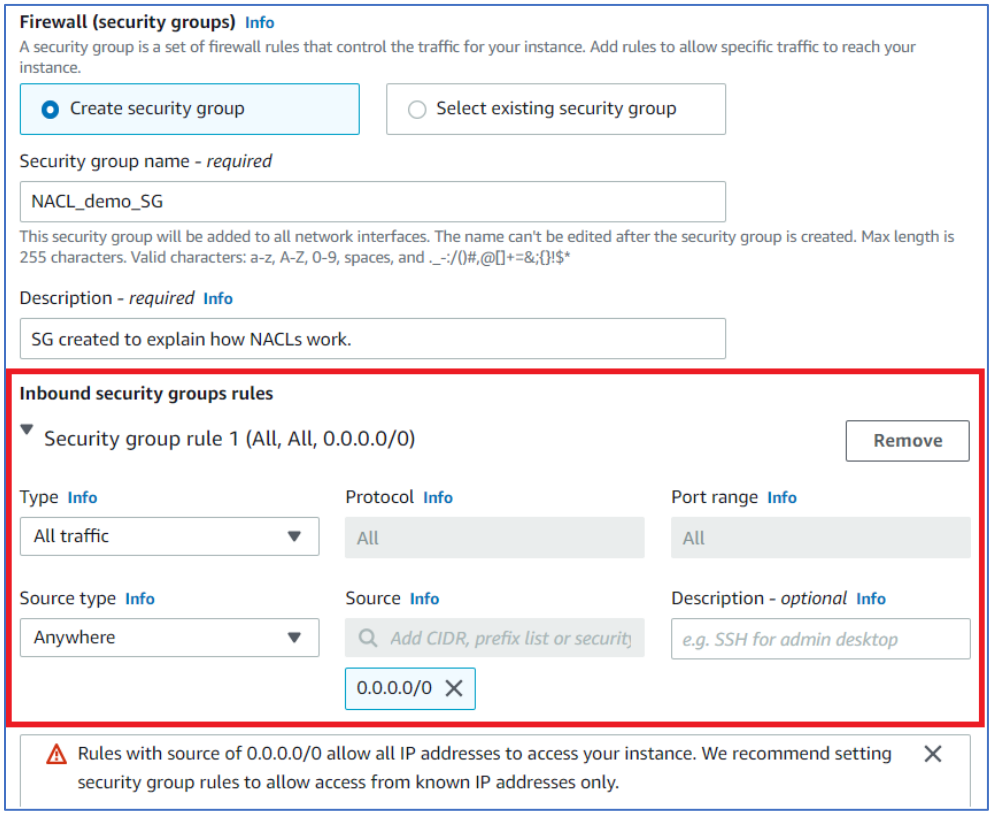
- A new NACL has been created and associated with subnet SN-0 with the IPv4 172.31.0.0/20.
- Now go to the EC2 service.
- Create a new EC2 instance.



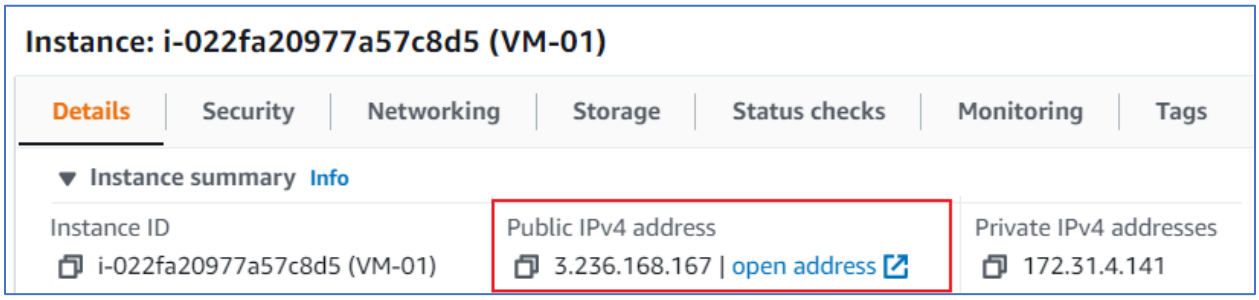
- Under network settings, select the subnet SN-0 which was associated with the NACL.



- Create a new security group that allows all traffic.
- A security group is another level of access security that will be explained later in this chapter. Right now, the concern is to avoid interference with the explanation of how the NACLs work.
- The new security group allows all traffic so this level of access control will not influence the NACL experimentation to follow; however, in reality, a firewall should never be left open like this.



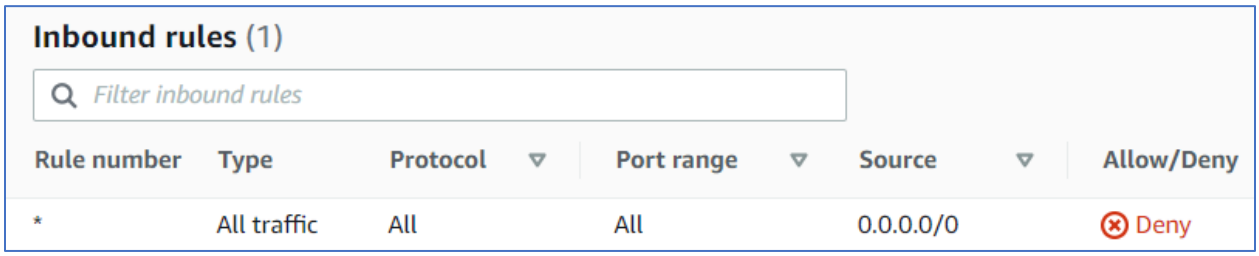
- Create the new EC2 instance.
- Grab the public IPv4 address.

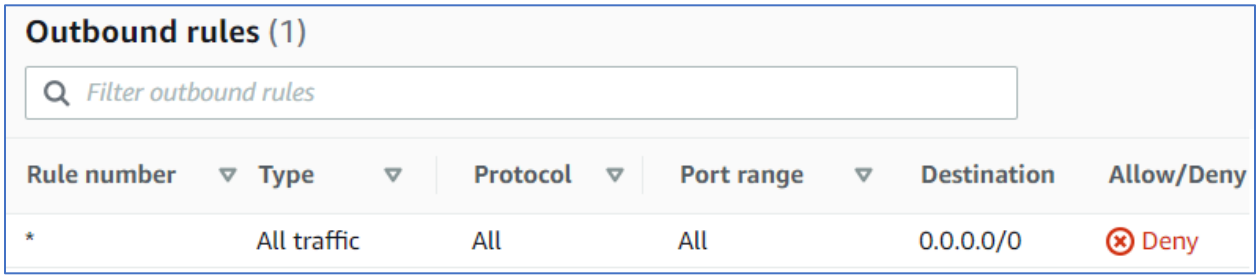


- Test the access.

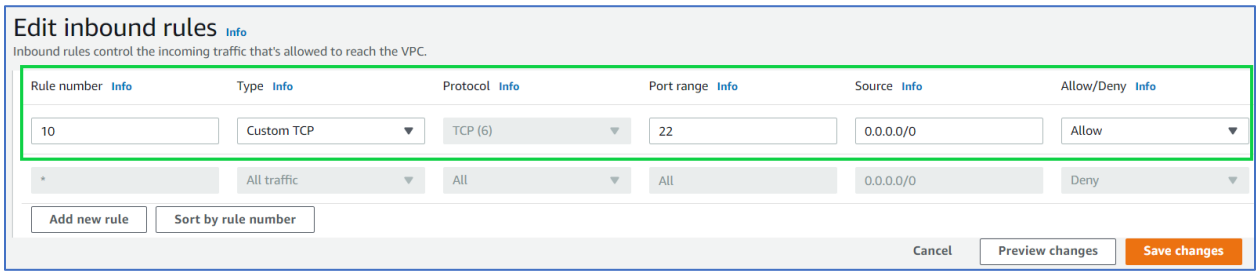
```
C:\Users\fc>ssh ec2-user@3.236.168.167 -i C:\Users\fc\Downloads\demo-key.pem
ssh: connect to host 3.236.168.167 port 22: Connection timed out
```

- The VM is unreachable as it should because the NACL rules are set to deny all.



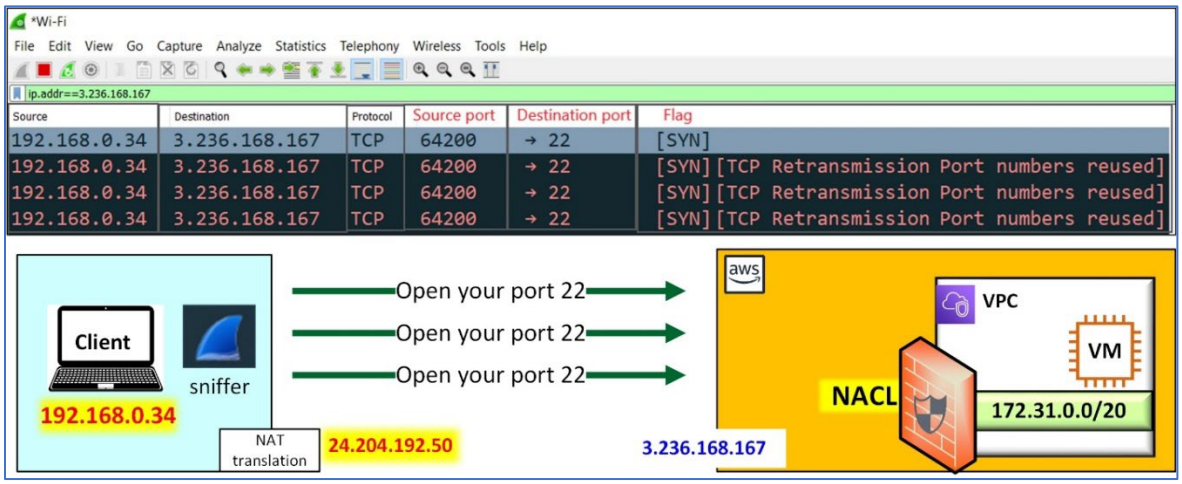


- Go to the NACL to modify the inbound rules.
- Add a rule to allow HTTP (TCP port 22) inbound.



- Test it. It is still unreachable. Why?

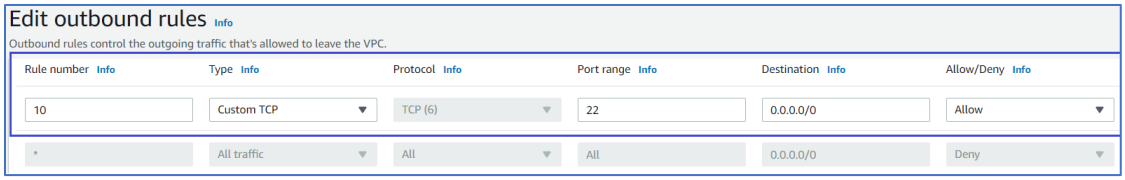
- Let's find out by sniffing the traffic in the client's interface.
- The filter expression **ip.addr== 3.236.168.167** was used to show only the interested traffic (otherwise, it shows all unrelated sessions making the analysis too difficult).



Attempt to start a conversation using SSH between the client and the server

According to the sniffed traffic, the client repeatedly asks the server to open the TCP port 22, but there is no answer to the request. This has to do with the stateless nature of the NACLs. That the traffic passed inbound does not imply that the response will be allowed. Hence the NACL outbound rules must be modified also.

- Add a rule to allow TCP port 22 outbound to all destinations.



- Test it and it still does not work, why? Let's see the sniff to look for a clue.

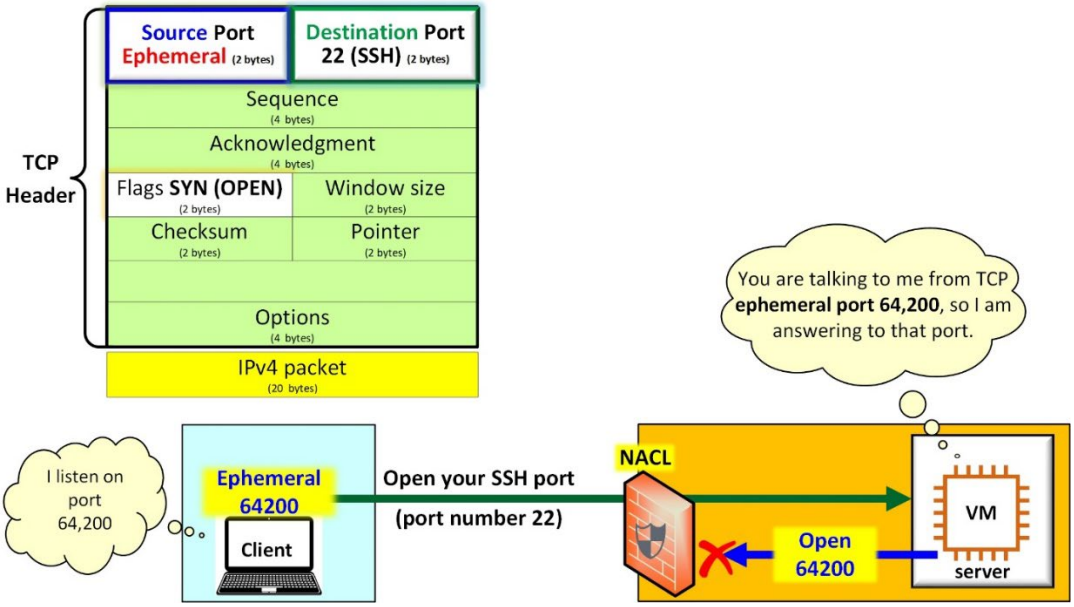
Source IPv4	Destination IPv4	Protocol	Source Port	Destination Port	Flag
192.168.0.34	3.236.168.167	TCP	64200	→ 22	[SYN]
192.168.0.34	3.236.168.167	TCP	64200	→ 22	[SYN]
192.168.0.34	3.236.168.167	TCP	64200	→ 22	[SYN]
192.168.0.34	3.236.168.167	TCP	64200	→ 22	[SYN]
192.168.0.34	3.236.168.167	TCP	64200	→ 22	[SYN]
192.168.0.34	3.236.168.167	TCP	64214	→ 22	[SYN]
192.168.0.34	3.236.168.167	TCP	64214	→ 22	[SYN]
192.168.0.34	3.236.168.167	TCP	64214	→ 22	[SYN]
192.168.0.34	3.236.168.167	TCP	64214	→ 22	[SYN]
192.168.0.34	3.236.168.167	TCP	64214	→ 22	[SYN]
192.168.0.34	3.236.168.167	TCP	52167	→ 22	[SYN]
192.168.0.34	3.236.168.167	TCP	52167	→ 22	[SYN]
192.168.0.34	3.236.168.167	TCP	52167	→ 22	[SYN]
192.168.0.34	3.236.168.167	TCP	52167	→ 22	[SYN]
192.168.0.34	3.236.168.167	TCP	52167	→ 22	[SYN]

Sniffed attempt to SSH into the EC2 instance

The sniff reveals something interesting that has to do with the mechanics of a TCP conversation in general. The client sends opening TCP (SYN) segments to port 22 (SSH) repeatedly, but no answer appears in the sniffed data. Notice that the client, initially, opens with a TCP source port 64,200, then it tries another source port 64,214, and yet another 52,167, and then it gives up.

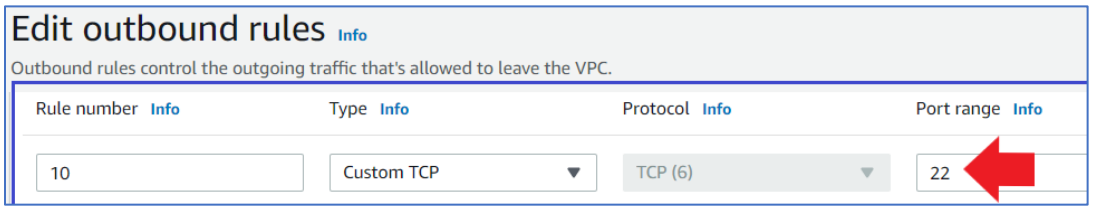
This is the opportunity to explain some fundamentals of the TCP protocol. Every application that exists listens on a **well-known port**. That means a unique number that the organization **Internet Assigned Numbers Authority (IANA)** has reserved for the exclusive use of the service; for example, 22 for SSH and 80 for HTTP. The **server side** listens and talks using the corresponding well-known ports.

On the other hand, the **clients** do not have reserved numbers so they take on any port number between 1024 through 65,535. Typically, a high number is chosen such as the ones seen in the sniffed transactions. These ports are chosen randomly by the client and used on a temporal manner to be released once that the session is over. For this reason, they are called **ephemeral** which means lasting for a short time. The **clients always open** TCP sessions from ephemeral ports and they listen also on the same ephemeral port.

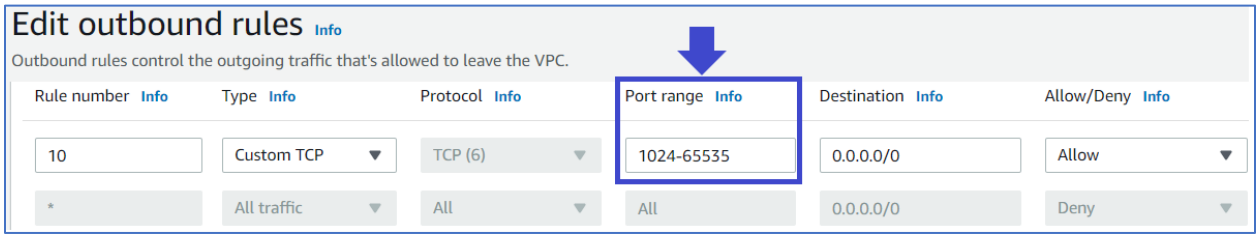


Client opening message from ephemeral port to the well-known SSH port on the server

Provided with this knowledge, the revision of the configuration of the outbound rule shows a mistake, the return port can not be 22 because the client is not listening on that port. In principle, the NACL outbound rule can be fixed now. But there is still another problem, the previous sniff of the traffic showed that the client attempted to open from several ephemeral ports, so which one is it going to be?



The answer is that all the ephemeral ports must be allowed outbound so that the replies can pass through the security filter. The port range 1024-65535 covers all the ephemeral ports.



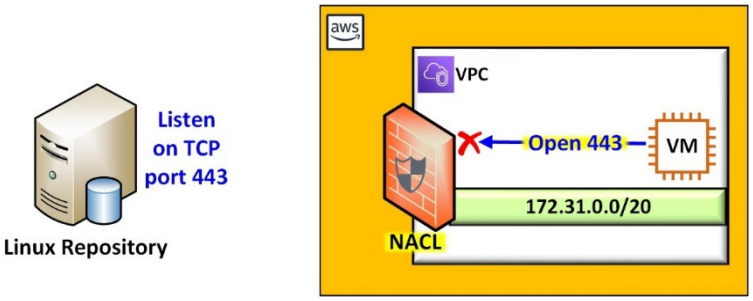
Now, the SSH into the EC2 instance works!

```
C:\Users\fc>ssh ec2-user@3.236.168.167 –i  C:\Users\fc\Downloads\demo-key.pem

Last login: Fri Jun 10 18:38:45 2022 from d226-76-239.home.cgocable.net

     __|  __|_  )

     _|  (     /   Amazon Linux 2 AMI

     ___|\___|___|

https://aws.amazon.com/amazon-linux-2/

16 package(s) needed for security, out of 26 available

Run "sudo yum update" to apply all updates.

[ec2-user@ip-172-31-4-141 ~]$
```

- Notice that the EC2 welcome message recommends to run an update of the packages installed in the EC2. Let's try such update.

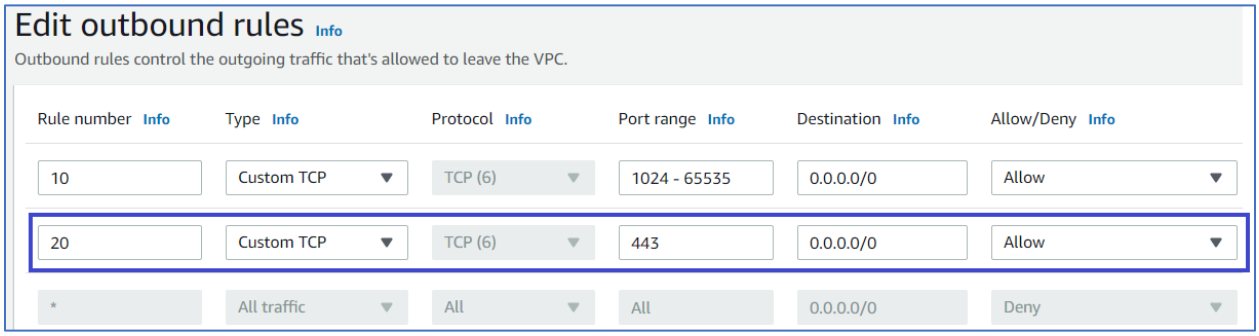
```
[ec2-user@ip-172-31-1-184 ~]$ sudo yum install httpd -y

Loaded plugins: extras_suggestions, langpacks, priorities, update-motd

Could not retrieve mirrorlist https://amazonlinux-2-repos-us-east-1.s3.dualstack.us-east-1.amazonaws.com/2/core/latest/x86_64/mirror.list error was 12: Timeout on https://amazonlinux-2-repos-us-east-1.s3.dualstack.us-east-1.amazonaws.com/2/core/latest/x86_64/mirror.list: (28, 'Failed to connect to amazonlinux-2-repos-us-east-1.s3.dualstack.us-east-1.amazonaws.com port 443 after 2702 ms: Connection timed out')
```

It tried, but then it failed. Let's troubleshoot using the information provided by the EC2 instance. The error description indicates that the repositories could not be reached. Linux operating systems are updated from well-known, trusted repositories located on the Internet. The interpretation of the message description above infers that the AWS AMI repositories are in servers located in amazonaws.com. The traffic must leave the customer's VPC to get to the repositories even if they are in the same Amazon domain. The error message offers an additional clue as to why the access to the repository failed. The secure HTTPS protocol (port 443) is used to reach the servers. Let's reason this problem with a diagram.



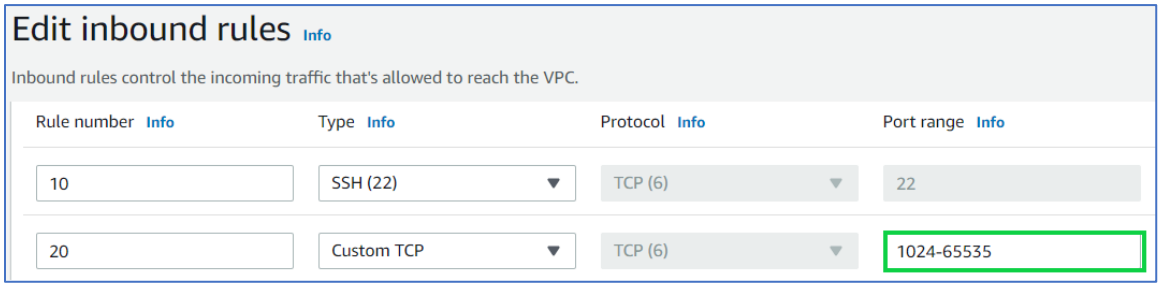
Linux updates are obtained from secure repositories using secure HTTPS

The NACL does not have an outbound rule to allow HTTPS to pass through. The VM tries to reach outbound to the destination port 443, but it is stopped by the NACL. Let's fix this problem by adding a rule 20 to allows outbound traffic to HTTPS .



The traffic is evaluated in order, first the traffic outbound is evaluated against rule 10 but there is no match yet. Then the evaluation continues to the next rule (20) and that is a match for the port 443, so the traffic is allowed outbound.

Regarding the **client-server relationship**, the VM is the client in this case and the repository is the server. That implies that the VM opens the conversation from an ephemeral port. Therefore, the response must be directed to a port within the range 1024-65535. That rule is already in place in the inbound filter.



Now, it works. The traffic found the way from the VM outbound to the repository and the responses were allowed back. The packages have been downloaded using HTTPs and the EC2 instances has been updated.

```
Packages Updated:
  curl.x86_64 0:7.79.1-2.amzn2.0.1  iproute.x86_64 0:5.10.0-2.amzn2.0.2   kernel-tools.x86_64 0:5.10.118-111.515.amzn2
  python.x86_64 0:2.7.18-1.amzn2.0.5   etc, etc...
Complete!
```

Currently, this “learning activity” network access control list allows the following:

- The access inbound to port 22 from anywhere on the Internet.
- The access inbound to all the ephemeral ports from anywhere on the Internet.
- The access outbound to port 443 to anywhere on the Internet.
- The access outbound to all the ephemeral ports from anywhere on the Internet.

It is a common practice that resources behind the protection of a firewall are allowed to open sessions to the exterior, but not the opposite. For example, a user sitting at home computer should be able to open session to anywhere on the Internet. However, computers on the Internet should not be opening sessions inbound to a home computer (a hacker would love to do that). So, firewalls are permissive going outbound, but highly restrictive going inbound. Applying this concept to the “learning activity” demo NACL, the outbound rules should be simplified to allow all traffic. The restrictions should be one the inbound direction mostly.

Clean up

This completes this investigation on the functioning of NACLs, now proceed to clean up the VPC to avoid interference with another activities.

- Terminate the EC2 instance.
- Disassociate the subnet 172.31.0.0/20 from the demo NACL. Reassociate it with the default NACL.
- Remove the demo NACL.
- The six default subnets should be associated with the default NACL.
- Remove the security group (do not leave a full open security group behind).
- Reminder: the default NACL allows all traffic in either direction.

Rule number	Type	Protocol	Port range	Source	Allow/Deny
100	All traffic	All	All	0.0.0.0/0	Allow
*	All traffic	All	All	0.0.0.0/0	Deny

NACL Summary

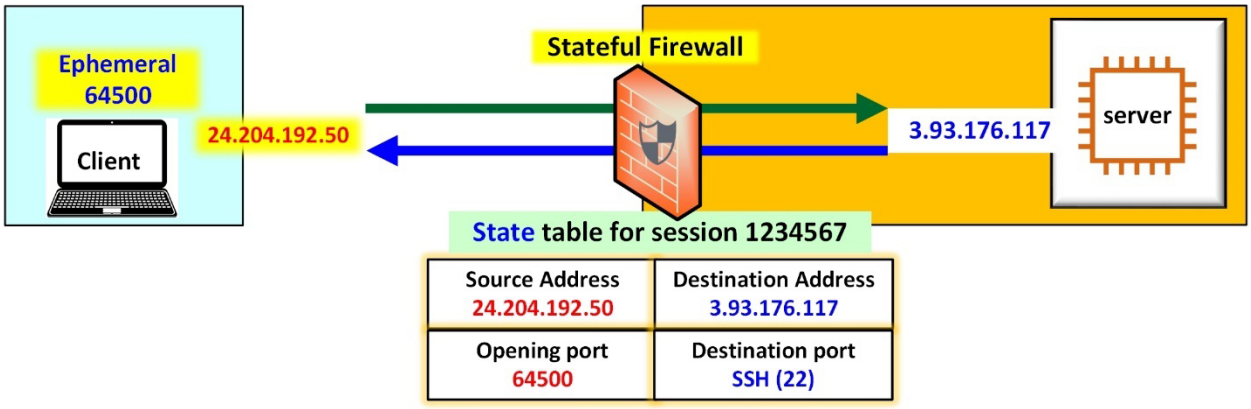
There are three main lessons from this exploration of the Network Access Control Lists principles and mechanics. The first one is that to correctly deploy precise NACLs a thorough understanding of the encapsulation and structure of protocols is required. More generally, anyone who pursues a career in network and computer security must have such knowledge.

The second learning point is about the NACL settings. that NACLs control the access through subnets. Any virtual machine attached to a particular subnet will be subjected to the rules of the NACL associated with that subnet. The VPC has a default NACL which applies to all the default subnets and this NACL is completely open to all traffic. On the other hand, new NACLs are created with the deny all traffic rule.

The third takeaway is regarding the stateless nature of NACLs. If the traffic passes one way, it is not implicit that the response will be allowed. For stateless firewalls, the rules must be explicit to allow traffic in either direction, inbound or outbound so that makes them harder to configure than the other type of firewall, the stateful firewall which is the next topic in this exploration.

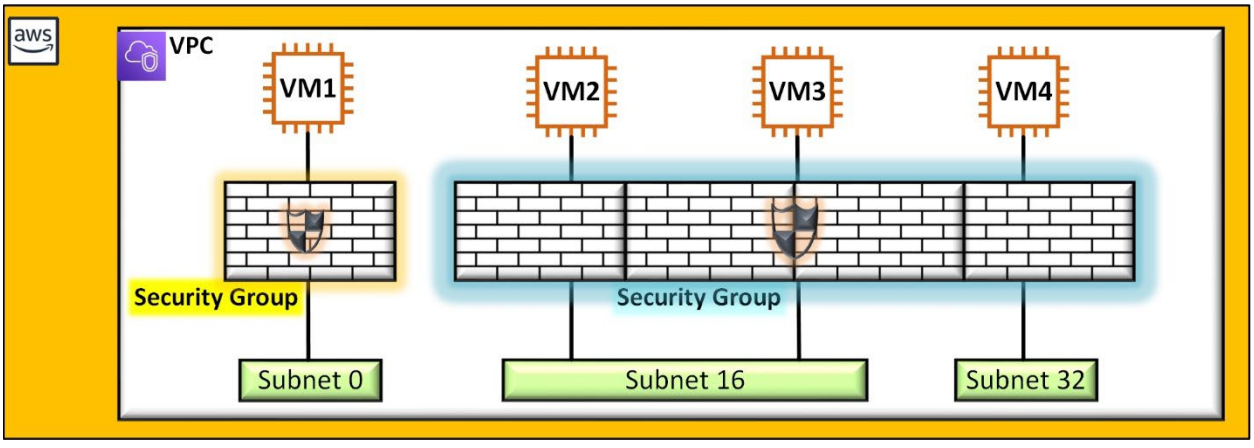
The Security Group

A **security group** is a **stateful firewall**. This means that the firewall keeps track of the traffic allowed to pass one way with the implicit condition that the response traffic is granted passage. The stateful firewall is harder to design internally because it must keep a state table for many sessions. For that reason, it requires more processing capacity than a stateless firewall. In compensation, the stateful firewall is easier to configure because the administrator only needs to consider the traffic one way.



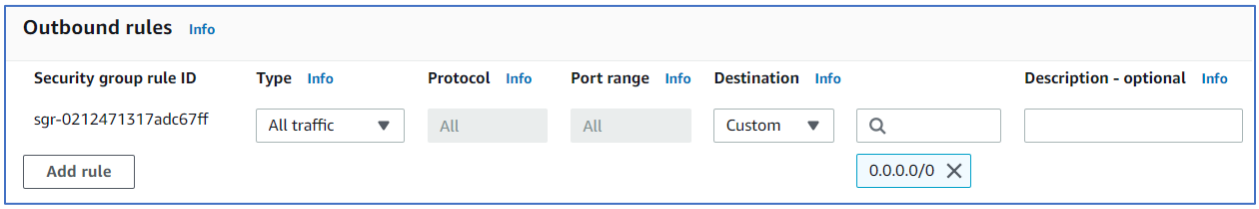
Stateful firewalls keeps state tables of the communications sessions

The security group can be applied to different points of control. In contrast, a NACL can only be applied to control access to whole subnets. The most common use case for security groups is to control the access to a particular cloud resource or to a group of resources. In the following figure two security groups are presented as protective walls applied to the network interfaces of four virtual machines. A security group (in yellow) controls the access to the VM1 instance while another security group (in blue) controls the access to a group of EC2 instances deployed across different subnets. The same rules apply to the traffic flowing up or down the interfaces of instances VM2, VM3, and VM4.

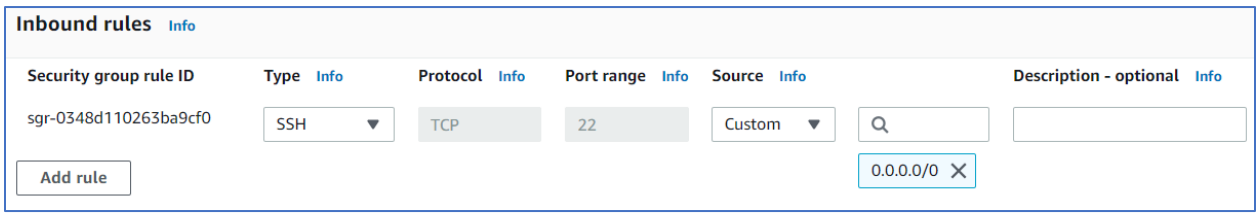


Two security groups controlling the access to virtual machines

The most fundamental principle of secure access establishes that hosts behind a firewall should be able to initiate sessions to the exterior whereas traffic from the opposite direction should be restricted. The outbound rules of a new security groups follow such principle as seen below. The outbound rules of a sample security group shows that protected VMs can send any kind of traffic to the exterior.



However, in this example security group, the only traffic allowed toward a protected VM is SSH.



The following figure illustrates the functioning of this security group. The only action existing in security groups is to allow. Anything else is a deny by default. The VM can be accessed securely via SSH from anywhere on the Internet. (A better rule would be to restrict the source to a range of addresses, but this is an initial example). No other protocol will reach the VM if the session is started from outside the VPC.

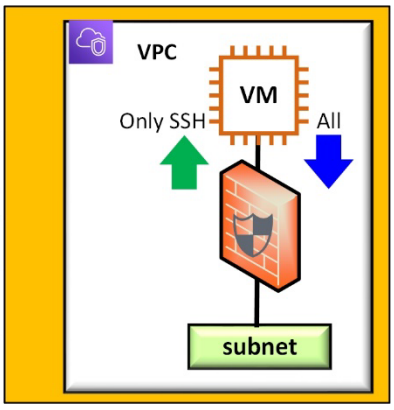
On the other hand, the VM can initiate any kind of protocol, to any port, to anywhere regardless of the inbound rule that only allows SSH. This is due to the stateful nature of the security group. If the traffic is allowed one way, the response is granted no matter what the port number might be. Hence, the VM can initiate any session to reach repositories to update its operating system, or to install packages and applications or for any other reason.

Inbound

Rule id	Type	Protocol	Port range	Source	Action
sgr-0b12	SSH	TCP	22	0.0.0.0/0	Allow

Outbound

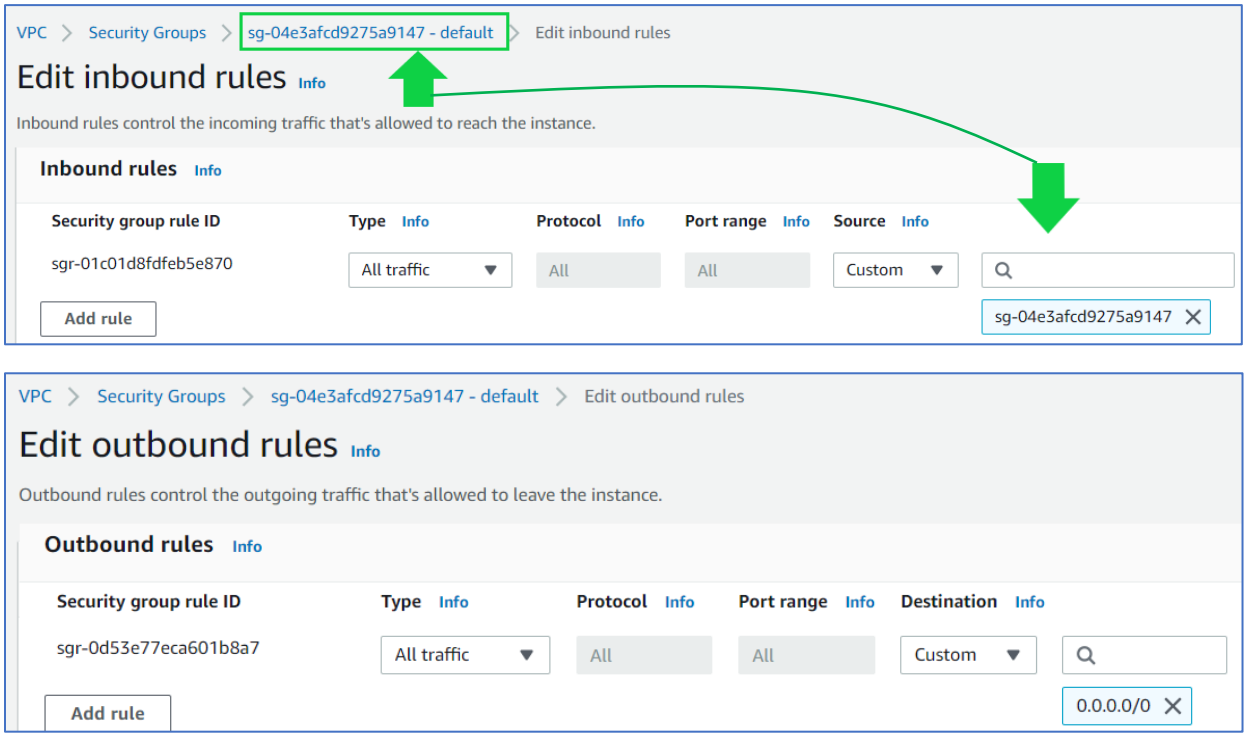
Rule id	Type	Protocol	Port range	Source	Action
sgr-67ff	All traffic	All	All	0.0.0.0/0	Allow



Security group that allows SSH traffic inbound from any source IPv4 address

The Default Security Group

When a new resource is created; for instance, an EC2 instance, it is optional to assign a security group immediately. In the meantime, the resource can not be left without protection and that is the role of the default security group. This security filter allows all traffic outbound and the corresponding responses. Regarding the inbound rule, the default security group accepts traffic from "itself" which means all the internal traffic from within the VPC which is the minimum access that can be granted.



The rules of the VPC default security group

Structure of the Security Group

Security groups have four configurable columns and an optional description column. The security groups service assigns an identification string automatically to every rule that is created.

Security Group **rule ID**	**Type** of traffic	**Protocol**	**Port range**	**Source** of traffic	**Description** optional

Most protocols are included under the **type** column. If the protocol is not in the following table, then the option Custom protocol is available. In such case, the user needs to provide the particular **protocol** and **port** if that is needed.

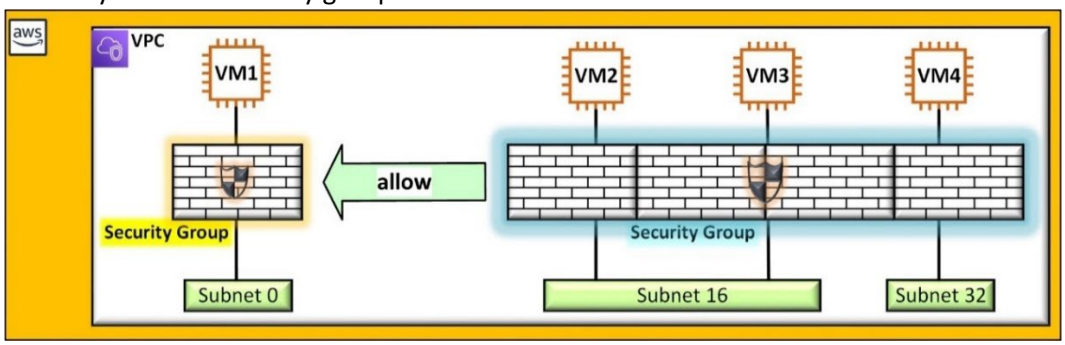
Custom protocol	All ICMP - IPv4	DNS (TCP) (53)	LDAP (389)	MS SQL (1433)	PostgreSQL (5432)
Custom TCP	All ICMP - IPv6	DNS (UDP) (53)	HTTPS (443)	Oracle-RDS (1521)	Redshift (5439)
Custom UDP	Custom protocol	HTTP (80)	SMB (445)	LDAP (389)	WinRM-HTTP (5985)
Custom ICMP - IPv4	SSH (22)	POP3 (110)	SMTPS (465)	NFS (2049)	WinRM-HTTPS (5986)
All TCP	telnet (23)	IMAP (143)	IMAPS (993)	MySQL/Aurora (3306)	Elastic Graphics
All UDP	SMTP (25)	SSH (22)	POP3S (995)	RDP (3389)	NFS (2049)

Type of traffic options for security group configuration

The **source** field pertains to the origin of the IP traffic, these are the most commonly used possibilities:

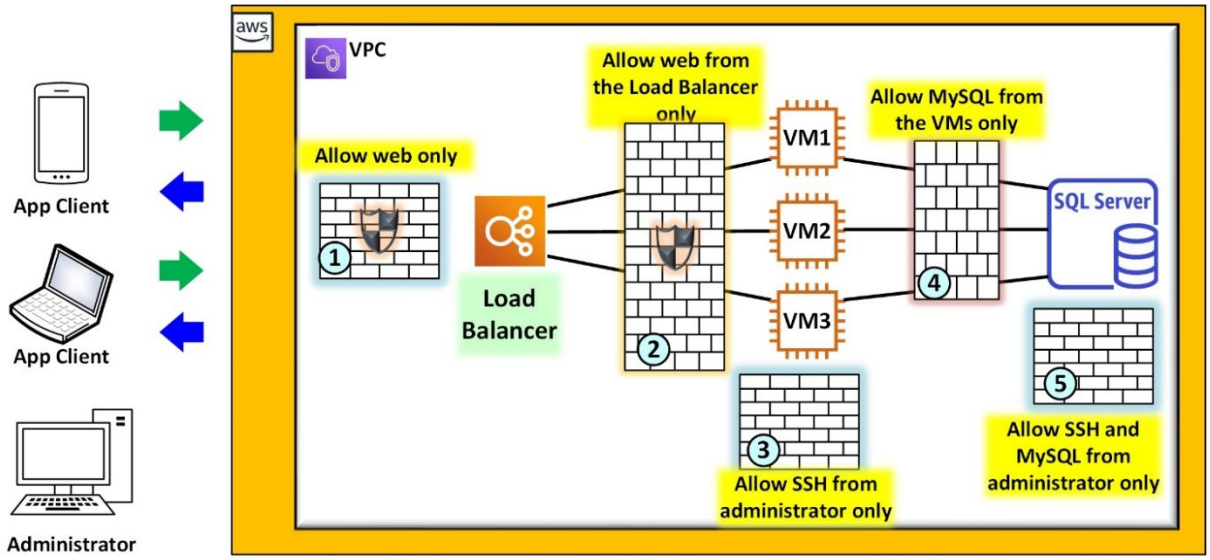
IPv4 addresses	0.0.0.0/0	This means **all** the IPv4 addresses that exists.
IPv4 addresses	0.0.0.0/32	**Exactly** the route 0.0.0.0 (default gateway route)
IPv6 addresses	::/0	This means **all** the IPv6 addresses that exists.
Other security groups *	SG identification string	See explanation below.
My IP address	A public source IP address	The IP address being used by the user's computer.
Custom made	A public IP range	For example, 142.55.0.0/16 covers all Sheridan College.
Prefix Lists	Lists of allowed IP addresses	AWS public addresses for intercloud connections.

The security groups in AWS can be set to allow traffic from another security group. In the following diagram, VM1 can receive traffic from VM2, VM3, VM4 and any other VM that might be associated to the blue security group. This is a helpful property because it simplifies the security policies. In the figure, instead of creating one rule per every VM, the security group yellow accepts any traffic sourced from any VM protected by the blue security group.



Security group yellow allows traffic coming from security group blue

The following diagram represents a more complex scenario with five security groups in place. This figure represents a web application running in a cluster of virtual servers. A MySQL database stores the data pertaining to the application. The access to the app is done via a load balancer that distributes the workload among the three servers.



Security groups protecting a robust platform of a web application with access to a database

This is the description of the roles of the different security groups:

- The first security group only allows HTTPS traffic from any of app clients from the Internet through the load balancer.
- The second security group allows HTTPS from the load balancer to the group of VM servers.
- The VM servers require maintenance so the third security group allows SSH access.
- The fourth security group allows traffic from the VMs to the SQL server using destination port 1433 which is MySQL's TCP port.
- The fifth security group allows the administrator to work on the database resource by providing access to SSH and to a MySQL client-server application.

The security group number 4 makes a good case for further explanation. The webservers in the cluster are deployed using cloud elasticity. That is, depending on the traffic conditions, more VM can be added or removed automatically. In such circumstance, having a hard coded security group accepting traffic from the IPv4 addresses of the VMs will not work. Instead, the MySQL security group is configured to accept inbound traffic from the security group of the VM cluster. Hence, it will always work regardless of whether VMs are being added or removed. The same situation occurs between the load balancer and the webserver cluster. That a security group can accept traffic from another security group is a very helpful property that simplifies access control.

Multiple security groups can be assigned to an instance or a group of instances. In such case, all the security groups are evaluated completely before a decision is made. Fundamentally, the rules of the different security groups get aggregated. In the case of multiple security groups there is no contradiction because security groups can only allow traffic and they are stateful. The following figure shows EC2 instances with various security groups.

Security Group **allow-ssh**

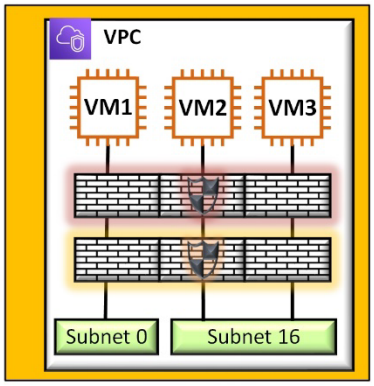
					Inbound
Rule id	Type	Protocol	Port range	Source	Action
sgr-c0b5	SSH	TCP	22	24.204.192.0/24	Allow

Security Group **allow-https**

					Inbound
Rule id	Type	Protocol	Port range	Source	Action
sgr-0b12	HTTPS	TCP	443	0.0.0.0/0	Allow

Same rules outbound

					Outbound
Rule id	Type	Protocol	Port range	Source	Action
sgr-67ff	All traffic	All	All	0.0.0.0/0	Allow



Two security groups applied to the same group of VMs

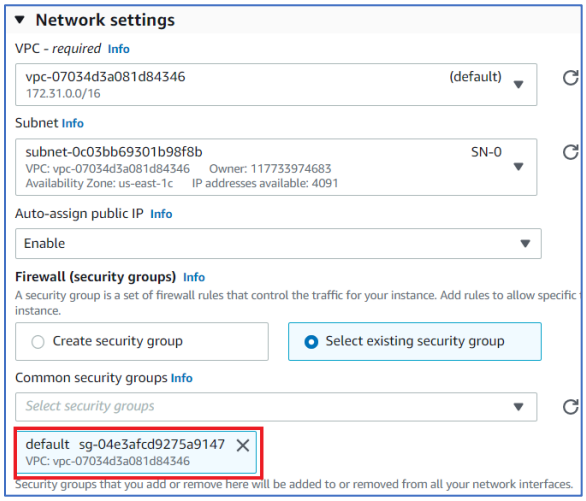
The VMs are secure webservers and consequently they are open for public access using the protocol HTTPS. The security group allow-https does exactly that and nothing more. However, the administrators require secure access to the VMs using SSH from hosts located in the network 24.204.192.0/24. So, the security group allow-ssh provides that access policy. The rules in both security groups are evaluated for incoming traffic. If the traffic is HTTPS, coming from anywhere, then the rule sgr-0b12 is a match. But if the traffic is SSH, coming from 24.204.192.0/24, then the rule sgr-c0b5 is a match. Obviously, these two rules could be merged into just one security group, but this is just to demonstrate the flexibility of this security tool.

Learning Activity

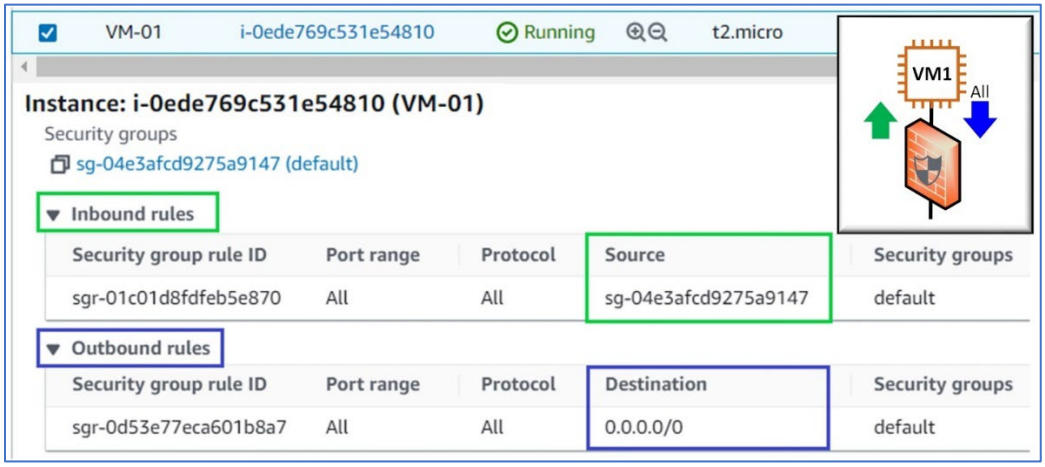
Configure and Apply Security Groups

Go to the AWS EC2 service and create an EC2 instance with the AWS AMI image on a t2.micro type.

- Choose the **default** security group under **Network settings**.



- The default security group allows all traffic outbound while it limits the inbound traffic to locally sourced only.

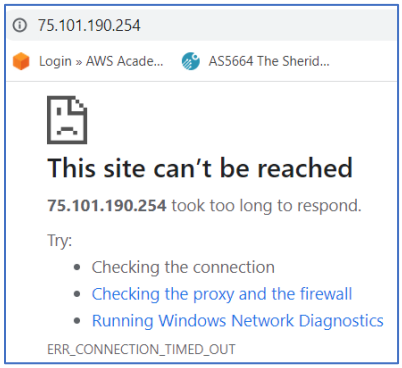


- Use the following for the user data to bootstrap a webserver.

```
#!/bin/bash
yum install httpd -y
cd /var/www/html
echo "<html<body><h1> Hi from $(hostname -f) </html></body></h1>" > index.html
sudo systemctl restart httpd
sudo systemctl enable httpd
```

The point being tested here is the firewall stateful condition. The EC2 instance will need to download all the packages to install HTTPD from the external repositories. The outbound rule allows that traffic to pass though. Since the firewall is stateful, the responses should be allowed regardless of the inbound rules saying that only local traffic is allowed.

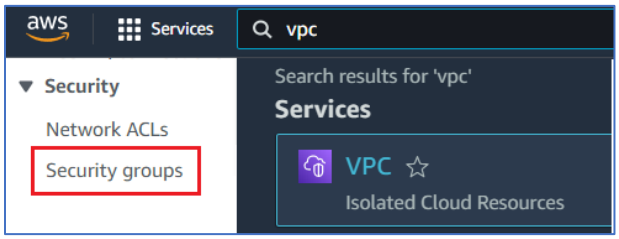
If the webserver was installed, it should be working, right?

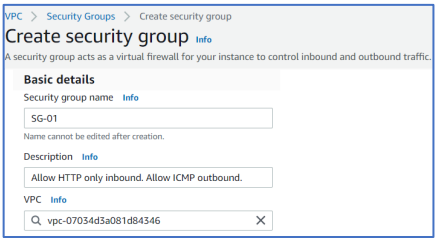


But it is not reachable and it should not be because in this case, the traffic is initiated from outside, so the inbound rules are preventing that traffic as they are intended. To access the webserver, a new security group can be added.

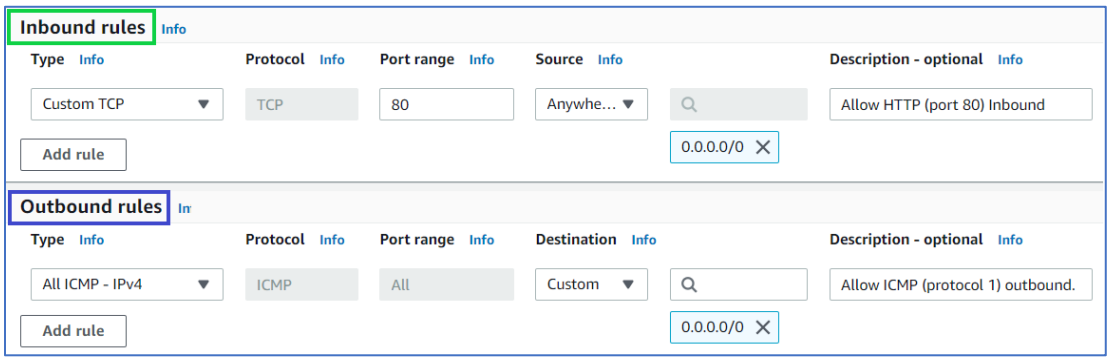
NOTE: Do not modify the default security group, that is a bad practice. The default security group has its own function and it should never be modified. Make another security group instead.

- Go to the AWS VPC services.
- Security groups.
- Create a new security group with the name SG-01.

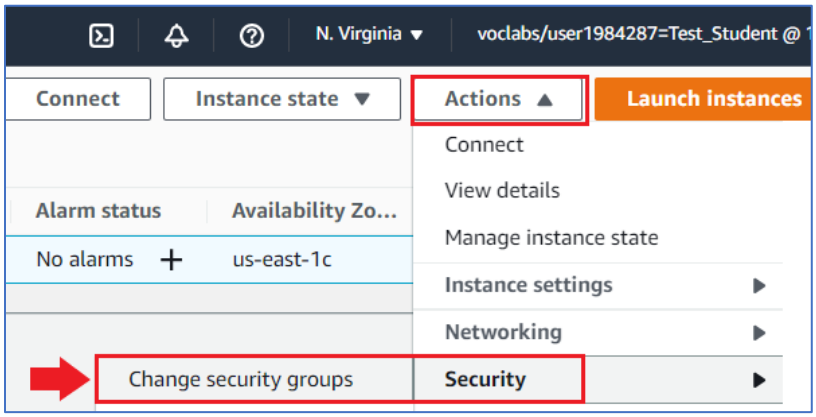




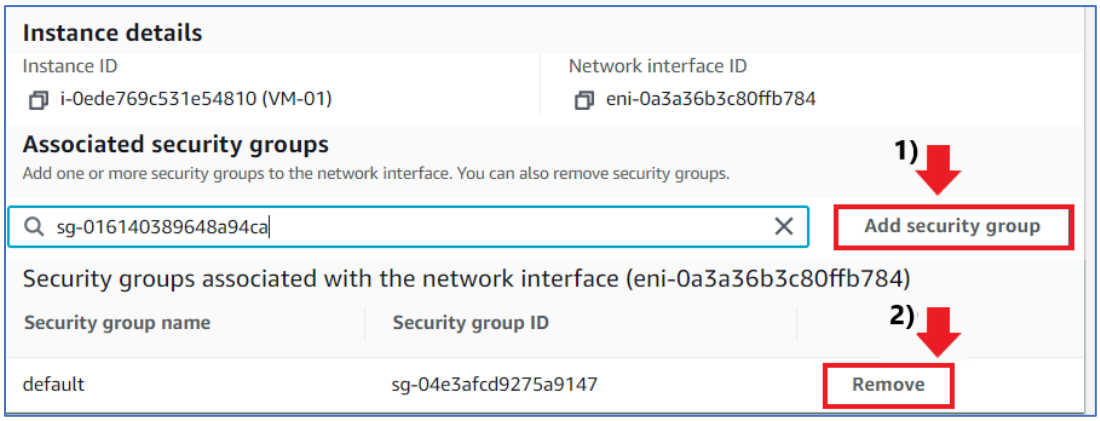
- Add an inbound rule to allow TCP port 80 (ssh) from everywhere (0.0.0.0/0).
- Add an outbound rule to allow ICMP to everywhere (this is to keep testing concepts).



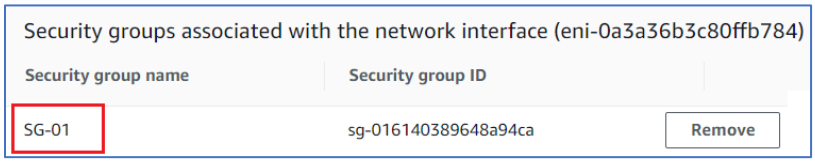
- Go back to the EC2 service.
- Under Actions, change the security group to the recently created SG-01.



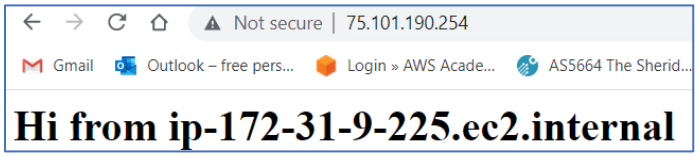
- Proceed to associate the new SG-01 to the EC2 and remove the association with the default security group.



- Notice that the security group is associated with the Elastic Network Interface (ENI) which is the virtual network interface of the EC2 instance.



- A new try proves not only that the webserver works but also it further proves the stateful nature of the security group. Even though the outbound rules are set to allow only ICMP traffic, the response to HTTP are allowed outbound.



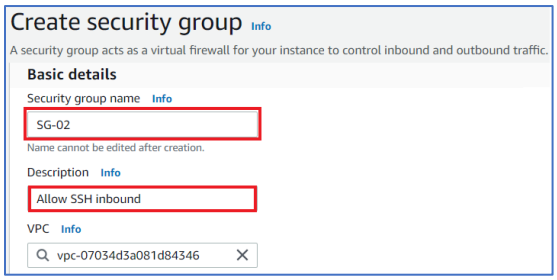
- Attempt to SSH into the EC2 instance. It should fail and it does. Why?

```
C:\Users\fc>ssh ec2-user@75.101.190.254 -i C:\Users\fc\keys\demo-key.pem
ssh: connect to host 75.101.190.254 port 22: Connection timed out
```

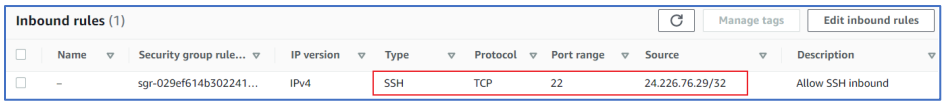
- Because SSH is not being allowed inbound.

Let's test the concept of aggregation of security rules. Multiple security groups can be applied to the interface of an EC2 instance. So, let's create another security group that allows SSH and associate that to the EC2's ENI too.

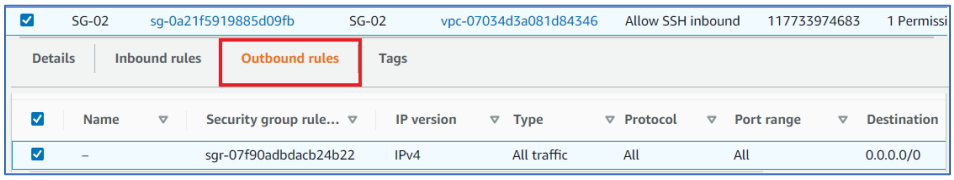
- Create a security group SG-02.



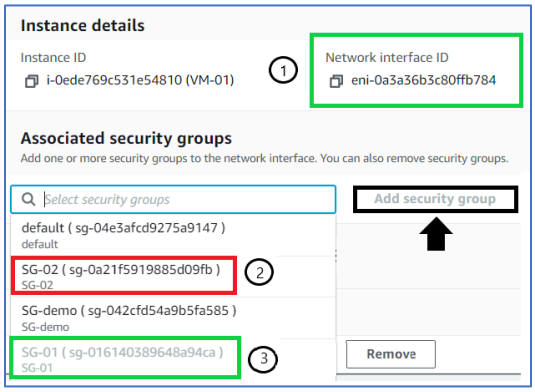
- Allow SSH from the source option "My IP". This will pick the public host address that AWS see as the source. In the example, the source is a host address 24.226.76.29/32 that belongs to the service provider Cogeco.



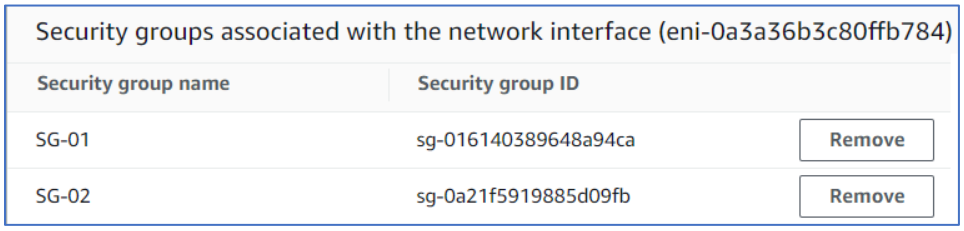
- The outbound rules are set to allow all traffic.

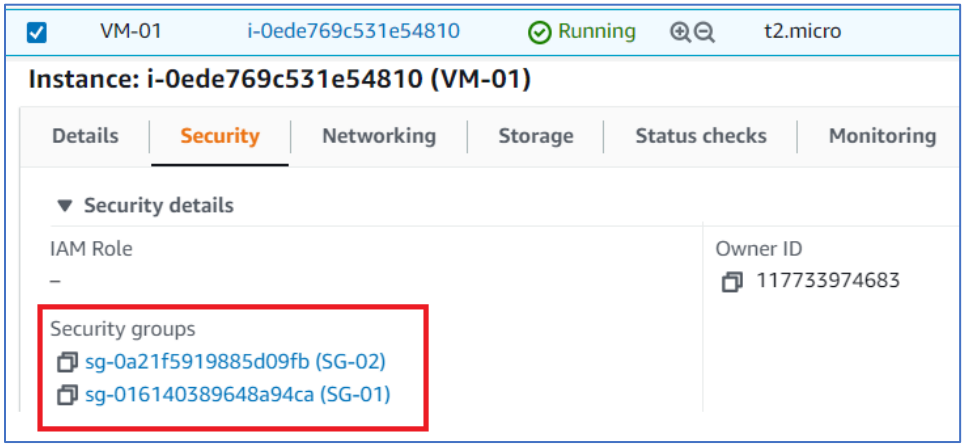


- Add the security group SG-02, but do not remove SG-01.



- Now, there are two security groups associated to the network interface.





- Test it, and it works!

```
C:\Users\fc>ssh ec2-user@75.101.190.254 -i C:\Users\fc\Downloads\demo-key.pem
The authenticity of host '75.101.190.254 (75.101.190.254)' can't be established.
ECDSA key fingerprint is SHA256:fKbM683BVj6MMT7WRpTunlaZLaSFAS+281NW8BfMIls.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '75.101.190.254' (ECDSA) to the list of known hosts.
       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|
https://aws.amazon.com/amazon-linux-2/
16 package(s) needed for security, out of 26 available
Run "sudo yum update" to apply all updates.
```

- From within the EC2, ping the name of Sheridan website and then ping the IPv4 address of the webserver.

```
[ec2-user@ip-172-31-9-225 ~]$ ping www.sheridancollege.ca
64 bytes from www-miss.sheridanc.on.ca (142.55.47.60): icmp_seq=1 ttl=42 time=23.5 ms
64 bytes from www-miss.sheridanc.on.ca (142.55.47.60): icmp_seq=2 ttl=42 time=23.4 ms
64 bytes from www-miss.sheridanc.on.ca (142.55.47.60): icmp_seq=3 ttl=42 time=23.4 ms

[ec2-user@ip-172-31-9-225 ~]$ ping 142.55.47.60
64 bytes from 142.55.47.60: icmp_seq=1 ttl=42 time=23.5 ms
64 bytes from 142.55.47.60: icmp_seq=2 ttl=42 time=23.7 ms
64 bytes from 142.55.47.60: icmp_seq=3 ttl=42 time=23.4 ms
```

- Now, do the opposite. Ping from your home computer to the public IPv4 address of the EC2 instance. That should not be reachable.

```
Microsoft Windows [Version 10.0.19044.1706]
(c) Microsoft Corporation. All rights reserved.

C:\Users\fc>ping 75.101.190.254
Pinging 75.101.190.254 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
```

The previous tests proved several points. The first one is that the EC2 instance can ping an external public IPv4 address because SG-01 allows ICMP outbound and SG-02 allows all traffic, so either one will allow ICMP going from the EC2 to any Internet address. The responses to ICMP queries are allowed back because of the stateful nature of the security group. Second, pinging from the home address to the EC2 instance is blocked because there is no inbound rule allowing such traffic. However, access to the webserver port 80 and port 22 is allowed.

Security Group Summary

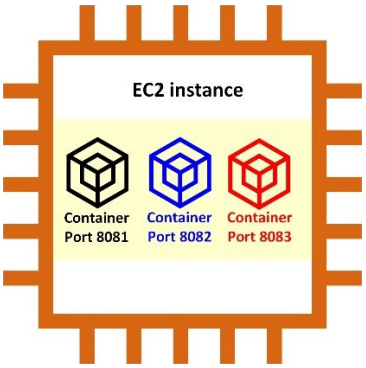
The security group is the most versatile generic access control tool in the cloud. The stateful nature of the security group simplifies the work of the cloud administrator because the configuration focuses on the type of traffic that is allowed inbound. The responses to the allowed traffic are automatically granted. A new security group does not allow any access. It is like a blank slate on which holes must be poked. It is the role of the administrator to decide what protocols and ports will be allowed inbound. For that reason, the cloud firewall administrator must have a strong knowledge of network protocols.

Chapter 5 Coursework

Security Group for three Docker containers

- Create a new EC2 instance, AWS AMI, t2.micro.
- Use this User Data to bootstrap the EC2 instance.

```
#!/bin/bash
yum update -y
yum install docker -y
systemctl start docker
docker run -p 8081:80 -d nginx
docker run -p 8082:80 -d nginx
docker run -p 8083:80 -d nginx
```



- This user data installs the Docker application and it creates three Docker containers on the EC2 instance.
- The container image used is NGINX webserver.
- Each container is a virtual server running inside the virtual EC2 instance. (virtual on virtual).
- Containers only do one thing, in this case, the three containers are webservers.
- Each container runs on a different port, see user data and the figure above.
- Create a new security group that allows HTTP access to the three containers in just one rule.
- Test the functioning of every container on a web browser with the address and port expressed in this format: the_ipv4_address:port_number for example 3.236.168.167:8081.

Security Group and the functioning of the VPC

- Create a security group with these rules (replace the source with your own source address)

Inbound rules (1/1)

Name	Security group rule...	IP version	Type	Protocol	Port range	Source
–	sgr-0cd222f1a72840633	IPv4	SSH	TCP	22	24.226.76.29/32

Outbound rules (2)

Name	Security group rule...	IP version	Type	Protocol	Port range	Destination
–	sgr-07d071dc2ffb4f126	IPv4	Custom TCP	TCP	1024 - 65535	0.0.0.0/0
–	sgr-0a7b30e56dc87a0...	IPv4	All ICMP - IPv4	ICMP	All	0.0.0.0/0

- An EC2 instance is created and the previous security group is associated to its network interface.
- The EC2 instance is accessed using SSH and then a ping to a name is successfully tested.

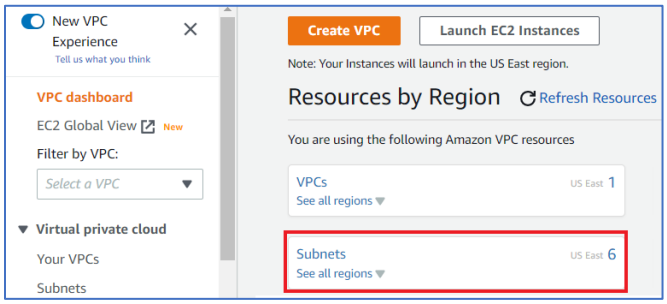
```
[ec2-user@ip-172-31-95-149 ~]$ ping www.sheridancollege.ca
64 bytes from www-traf.sheridanc.on.ca (142.55.7.60): icmp_seq=1 ttl=41 time=22.9 ms
64 bytes from www-traf.sheridanc.on.ca (142.55.7.60): icmp_seq=2 ttl=41 time=22.8 ms
```

Discussion:

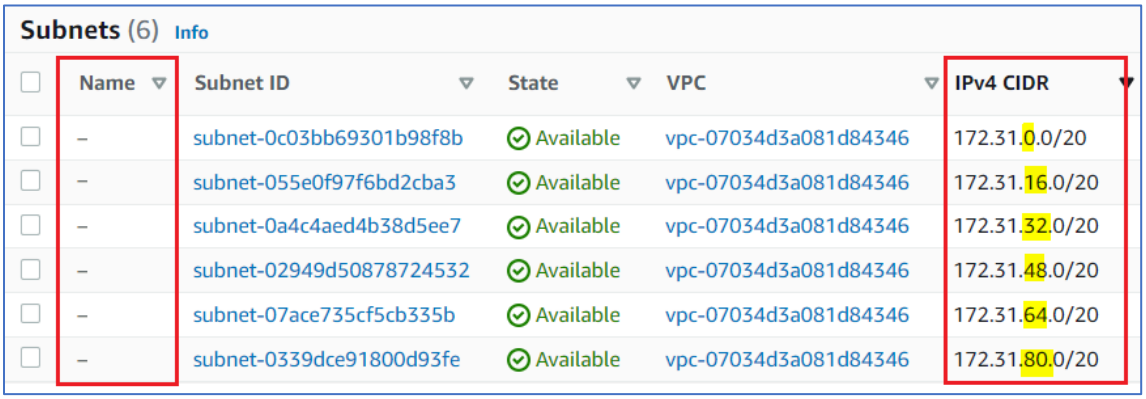
- What is the protocol that resolves the name www.sheridancollege.ca?
- Investigate how this protocol works. (The basics).
- What is the transport protocol?
- What is the port number?
- Is the security group allowing such protocol and port?
- How can the EC2 instance ping to the name?

Lab Access Control, NACLs and Security Groups

- Go to the AWS VPC Service
- Subnets



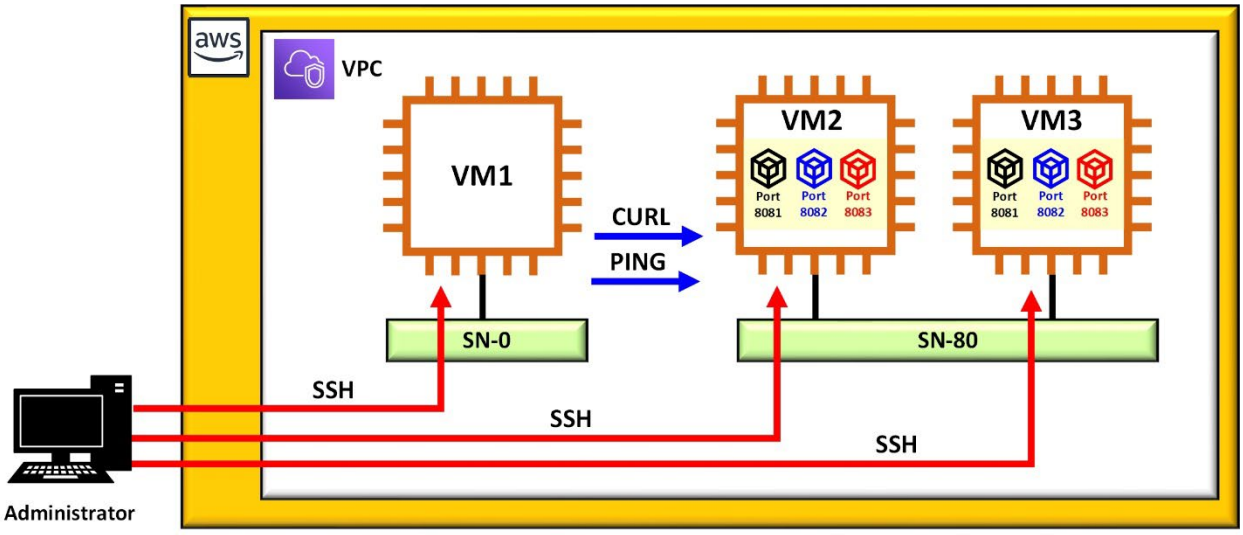
- Notice that there are six (6) subnets by default.



- Name the subnets so they are easier to identify when they are needed later.

Name	Subnet ID	State	VPC	IPv4 CIDR
SN-0	subnet-0c03bb69301b98f8b	Available	vpc-07034d3a081d84346	172.31.0.0/20
SN-16	subnet-055e0f97f6bd2cba3	Available	vpc-07034d3a081d84346	172.31.16.0/20
SN-32	subnet-0a4c4aed4b38d5ee7	Available	vpc-07034d3a081d84346	172.31.32.0/20
SN-48	subnet-02949d50878724532	Available	vpc-07034d3a081d84346	172.31.48.0/20
SN-64	subnet-07ace735cf5cb335b	Available	vpc-07034d3a081d84346	172.31.64.0/20
SN-80	subnet-0339dce91800d93fe	Available	vpc-07034d3a081d84346	172.31.80.0/20

- Deploy the following topology:



- VM1 is an AWS AMI located in subnet SN-0 (172.31.0.0/20). This is for testing purposes.
- VM2 is an AWS AMI located in subnet SN-80 (172.31.80.0/20). It contains three Docker containers running NGINX webservers.
- VM3 is an AWS AMI located in subnet SN-80 (172.31.80.0/20). It contains three Docker containers running NGINX webservers.
- The user data to bootstrap VM2 and VM3 is the following:

```
#!/bin/bash
yum update -y
yum install docker -y
systemctl start docker
docker run -p 8081:80 -d nginx
docker run -p 8082:80 -d nginx
docker run -p 8083:80 -d nginx
```

- Use a temporal security group to allow all traffic inbound and outbound (make sure that you delete this security group afterwards).

NACLs

- Configure NACLs to control the access to the VMs according to the following:
- Allow the administrator to access all the VMs via SSH.
- Allow **pinging** from VM1 only to either VM2 or VM3.
- Allow VM1 to **curl** the webservers in the containers.
- The command curl is a tool to test the functioning of webservers since the VM do not have a graphic web browser interface. The answer to a curl is the website landing page in text.

Submit

- Submit clear snippets of all the NACLs and a brief explanation of their functioning.
- Submit the proofs of the successful ping to VM2 and VM3 from VM1.
- Submit the proofs of the successful curl to VM2 and VM3 from VM1.

Security Groups

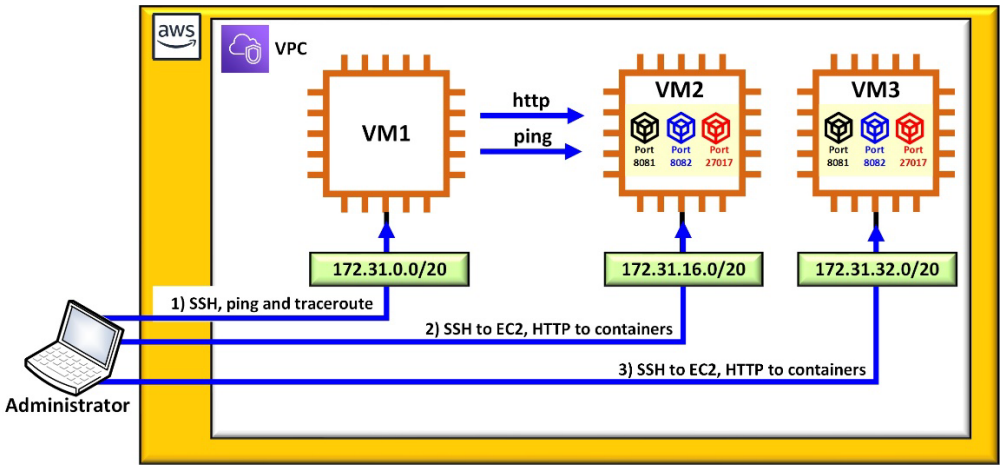
- Remove the association of the previous NACLs to avoid interference.
- Associate the subnets back to the default NACL.
- Remove the temporal security group from the previous step to avoid interference.
- Configure Security Groups to control the access to the VMs according to the following:
- Allow the administrator to access all the VMs via SSH.
- Allow **pinging** from VM1 only to either VM2 or VM3.
- Allow VM1 to **curl** the webservers in the containers.
- The command curl is a tool to test the functioning of webservers since the VM do not have a graphic web browser interface. The answer to a curl is the website landing page in text.

Submit

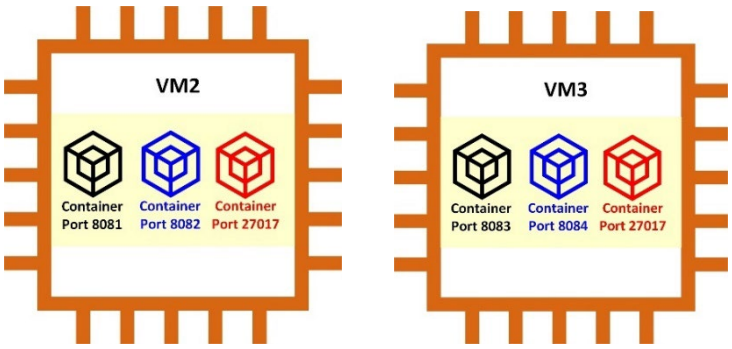
- Submit clear snippets of all the Security Groups and a brief explanation of their functioning.
- Submit the proof of successfully pinging VM2 and VM3 from VM1.
- Submit the proofs of the successful curl to VM2 and VM3 from VM1.

Lab Access Control with Security Groups and TCP analysis

- Deploy three EC2 instances.
- VM1 is just a plain EC2 instance.
- VM2 and VM3 will be running the Docker application to support three containers. On each VM, two containers are NGINX webservers and one is a MongoDB (see below for details).
- Create security groups that accomplish the following (see diagram and table):



General diagram



Detail of the EC2 instances VM2 and VM3

Sample user data for VM2. Modify accordingly to create VM3.

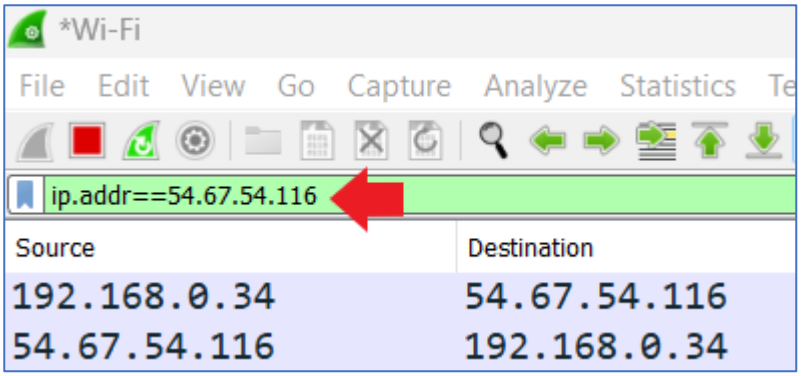
```
#!/bin/bash
yum update -y
yum install docker -y
systemctl start docker
docker run -p 8081:80 -d nginx
docker run -p 8082:80 -d nginx
docker run --name mongodb1 -d -p 27017:27017 mongo
```

This sample user data installs the application Docker and then it deploys two NGINX webserver containers and one MongoDB container. The application ports exposed are 8081, 8082, and 27017 (use the same sample data to create the VM3, just adjust the port numbers according to the instruction).

- Your task is to create security groups that enforce the conditions listed in the table below:

Source of the traffic	**Destination**
Admin's PC host address	SSH to VM1's public address
Admin's PC host address	Ping and traceroute to VM1 public address
Admin's PC host address	SSH to VM2's public address
Admin's PC host address	SSH to VM2's public address
Admin's PC host address	HTTP to VM2's container 8081
Admin's PC host address	HTTP to VM2's container 8082
Admin's PC host address	HTTP to VM2's container 27017
Admin's PC host address	HTTP to VM3's container 8083
Admin's PC host address	HTTP to VM3's container 8084
Admin's PC host address	HTTP to VM3's container 27017
From VM1	HTTP (curl) to VM2's container 8081
From VM1	HTTP (curl) to VM2's container 8082
From VM1	HTTP (curl) to VM2's container 27017
From VM1	HTTP (curl) to VM3's container 8083
From VM1	HTTP (curl) to VM3's container 8084
From VM1	HTTP(curl) to VM3's container 27017
From VM1	Ping private address of VM2
From VM1	Ping private address of VM3

Use the traffic analyzer Wireshark to capture the conversation between your PC and the VM2's MongoDB container. In the Wireshark tab, set a filter ip.addr==the destination address to capture only the conversation between your PC and the VM2's address, for example:



Otherwise there will be too much clutter making the analysis harder. Follow these instructions:

- Browse to the MongoDB container.
- Your should get this message:

```
It looks like you are trying to access MongoDB over HTTP on the native driver port.
```

- Close the web browser, then stop the sniffer capture and answer the following questions:

For all these questions use snippets of the Wireshark sniffer to complement your explanation.

- **How does your PC start the conversation?** What are the source and destination IPv4 addresses? What are the source and destination TCP ports? What are the TCP messages involved?
- **When does the HTTP protocol appear in the conversation?** What is the HTTP message sent by the PC and what is the reply? Is the conversation clear-text?
- **How does the conversation end?** What are the messages exchanged to close the session?

Submit

In a formal report, the proof of functioning of every one of the security group conditions. Use clear screen-shots of:

- The security groups with all the listed rules.
- The proof of functioning.

The explanation of the conversation between your PC and the MongoDB container complemented with the snippets from the sniffer. Answer the questions in your explanation.

References

[1] Mozilla at al. 2022. An overview of HTTP. Accessed. June 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[2] Amazon EC2 key pairs and Linux instances. Accessed. June 2021. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html?icmpid=docs_ec2_console

[3] AWS. Security Groups. Accessed. June 2021. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-security-groups.html

[4] AWS. Security group connection tracking. Accessed. June 2021. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/security-group-connection-tracking.html

Chapter 6

Virtual Private Cloud

Description

The main topic of this section is the virtual private computing environment. This is the most fundamental service that all cloud providers offer.

Learning Outcomes

- Design and deploy virtual private clouds.
- Plan IPv4 and IPv6 address allocation for virtual private clouds.
- Configure IPv4 address subnetting in a cloud environment.
- Enable controlled public access to network resources and services.
- Configure private networks in a cloud.
- Configure virtual gateways to access resources in the cloud.

Main concepts

- The Virtual Private Cloud (VPC).
- The components of the VPC.
- The Internet Gateway.
- The main routing table.
- The default route.
- Subnetting IPv4 address space in the cloud.
- Public and private subnets.
- Allocation of IPv6 addresses to the VPC.

Activities

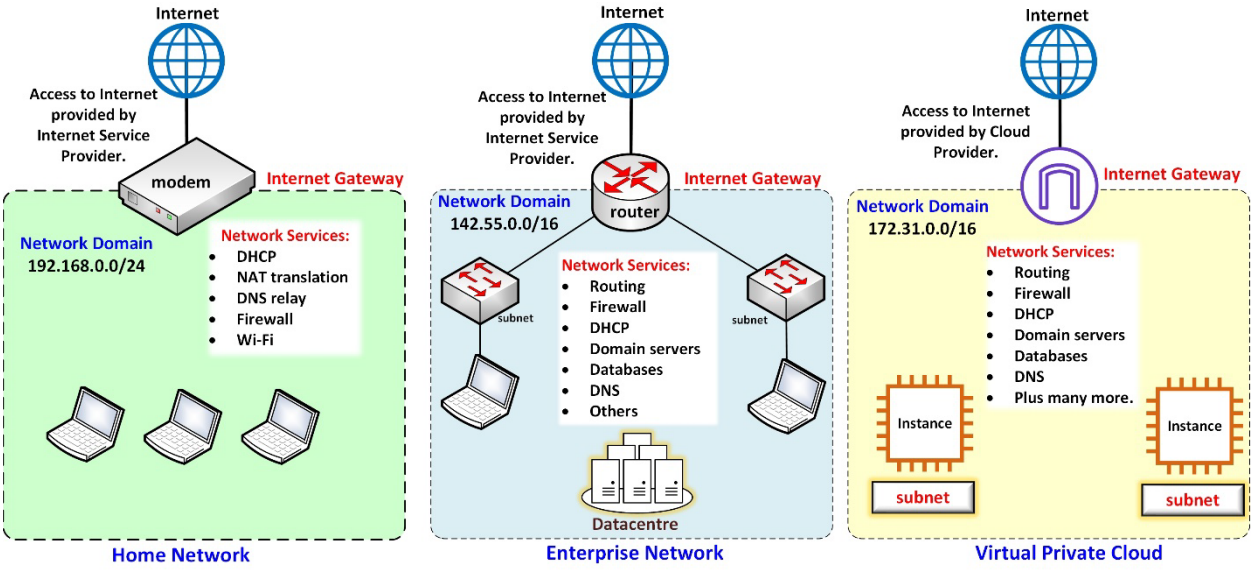
- Creation of subnets in the default VPC
- Creating a subnet with AWS CLI
- Using Boto3 Python to obtain information regarding the subnets.
- Obtaining IPv6 Address Space from AWS.
- Deployment of an EC2 instance with dual stack IPv4 and IPv6.

Definition of Virtual Private Cloud

A **Virtual Private Cloud (VPC)** is a **network domain** virtualized inside a cloud provider. A network domain is an environment under one administrative control with all the components required to implement any kind of service that depends on IP networking. In that sense, every cloud customer account has one VPC pre-defined by default. The customer-administrator can change the configuration of such VPC or create new VPCs from scratch. In short, the VPC service provides the customer with a complete control of its virtual networking environment in the cloud [1]. All the cloud providers offer this same service concept but with different names. For instance, AWS, Google cloud, IBM cloud, Digital Ocean, and Alibabacloud call it VPC while Microsoft Azure calls it Virtual Network VNET. Oracle cloud calls it Virtual Cloud Network (VCN).

A Virtual Private Cloud is, fundamentally, a logically isolated section in the cloud that mimics the structure of a private network domain. At the very core, the same principles that make possible for people getting home Internet access apply to accessing resources deployed in a cloud. The following diagram offers an analogy among a small office/home network (soho), an organization with its own on-premises network domain and an organization using a cloud provider. It is to notice that there are common factors among all of them.

To gain access to the Internet, a **gateway** is required for all cases. In the home network, the cable modem performs this function whereas in the enterprise, a router does the same. In the cloud, a virtualized Internet gateway (IGW) performs exactly this role. The main difference is that the IGW is mostly a software object while the others are physical devices.



High level comparison among a home network, an enterprise network and a virtual private cloud.

Internally, the network domains require IP addresses. In the diagram, the home and the cloud share one feature, they both are using **private** IP address space. On the other hand, the enterprise has an address space 142.55.0.0/16 which is **global** or **public**. The difference between this two is that the Internet's main routing tables contain the public addresses but not the private. This is possible because the public addresses are blocks of addresses which are owned by organizations and nobody else can use them

whereas the private can be used and reused by everybody since they are not reachable across the Internet. Hence, to connect a network domain that has private addresses internally, there must be a translator sitting at the edge of the network doing the mapping between the private addresses and a public address that must exist in the outside. In the figure, the cable modem receives one public IPv4 address from the Internet Service Provider. In the case of the cloud, the very cloud provider has public addresses that assigns to its customers. In both cases, there is **Network Address Translation (NAT).** The cable modem has the NAT function embedded on the device. In the cloud, the process is done with code.

A home network contains most of the basic components that a network domain needs to operate. Thus, that makes it a good reference case. Normally, the users get access via a Wi-Fi access point. This service is also typically embedded inside the cable modem. After connecting to Wi-Fi, the home devices get IP address information from a **DHCP** server which is also embedded inside the cable modem. The DHCP service offers critical information that the hosts require to gain access to the Internet. That includes, a host IPv4 address, the location of a DNS server or a DNS relay, and the information about how to get to the Internet. To get to the Internet, the hosts must send the traffic via the **default gateway**.

The enterprise implements the same concept but since the scale is larger, it is more complicated. There are more networks inside and each requires its own DHCP server. There are also many routers which perform as default gateways. The administrators in the enterprise must deploy and manage all these resources.

In the cloud, the DHCP service still exists, it has to. Without this service, the host computers, real or virtual, could not get IP connectivity. The main difference though, with the other two cases, is that the DHCP service is fully managed by the cloud provider. When a subnet is created, the service is automatically provisioned by the cloud. This simplifies the life of the network administrators.

A universal method to access resources on the Internet is to call them by names. This service is provided by the **Domain Name System** servers that hold databases with names and corresponding IP addresses. These DNS servers answer to name resolution names queries coming from hosts and applications. In the home network, the DNS servers are in the Internet Service Provider. In the enterprise network, the DNS server is inside the on-premises datacentre. In the cloud, the DNS services are managed by the cloud provider by default. Every resource created in the cloud can receive a name that is publicly accessible. Beside this default mode, the cloud also offers naming services to migrate the on-premises DNS server to the cloud.

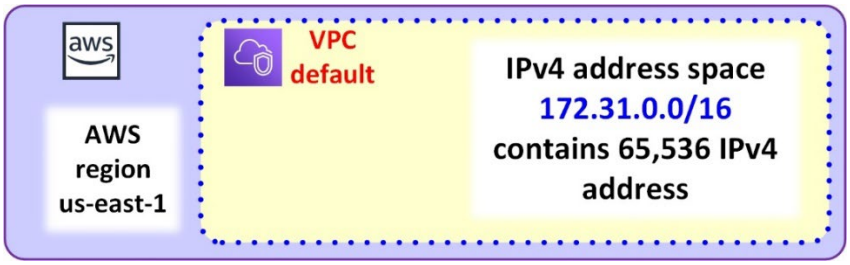
Every network domain must be protected at the boundaries. Firewall services only allow traffic to flow in either direction according to forwarding rules. A basic firewall function is embedded in the home cable modem (everything is embedded in the modem). On the hand, an enterprise must configure a very stringent system of firewalls and access control to protect its assets. The clouds already have these functions implanted with access control lists, security groups and additional firewall services.

To summarize, for every networking service and device that have traditionally existed for any network domain, there is an equivalent service in the cloud. The aspect that takes the cloud apart is that these services are dynamically managed by the cloud or they are configured on demand by the customers. Let’s turn the attention to the Virtual Private Cloud specifically.

AWS Virtual Private Cloud (VPC)

AWS's Virtual Private Cloud is a logical, isolated space in the AWS datacentres. Essentially, a VPC is a virtual datacentre inside the cloud. Every customer account receives one pre-made base VPC by default. Nevertheless, more VPCs can be created by the account's administrator. To create VPCs, it is a requirement to have a detailed knowledge of all components that make up the VPC. The configuration of VPC is part of the cloud architecture specialization career.

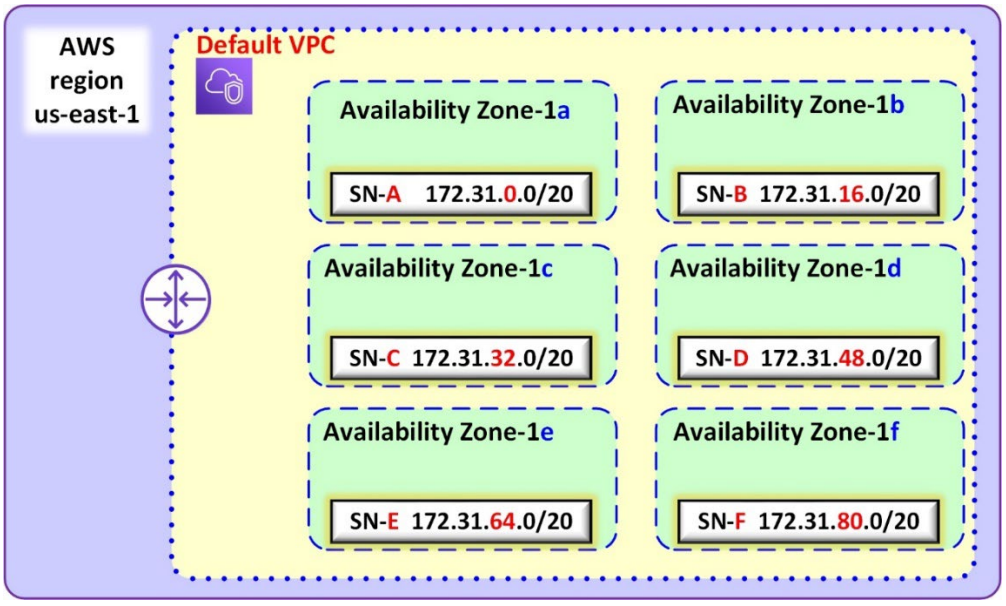
The default AWS VPC always contains the IPv4 address space 172.31.0.0/16 regardless of the region location. This is a private address space; however, there is no overlapping conflict among the customers because every customer space is logically contained.



The default AWS VPC

Subnets in the cloud

The IPv4 address space 172.31.0.0/16 is divided into smaller, more manageable portions and assigned to subnets. A default VPC in the us-east-1 region has six (6) default subnets. Other regions have less availability zones, so it depends on the region where the VPC resides. Furthermore, each of the subnets is located in a different availability zone. In the following diagram, the availability zones are neatly organized from a to f because it is just an example; but in reality, the zones allocation varies per customer.



A default VPC with six subnets in six availability zones in the us-east-1 region.

IPv4 Address System

Before proceeding with the creation of subnets, it is necessary to understand the IPv4 address allocation system. An IPv4 address is a number represented in base 256. For example, the address 172.31.16.1 is:

$$172.31.16.1 = \mathbf{172} \times 256^3 + \mathbf{31} \times 256^2 + \mathbf{16} \times 256^1 + \mathbf{1} \times 256^0 = 2{,}887{,}716{,}865$$

Thus, 172.31.16.1 is the quantity 2 billion, 887 million, 716 thousand, and 865. It is obviously more convenient to represent this large number in a larger base system such as 256. Every dot in the IPv4 address replaces the base and the exponent that multiples the coefficients. The IPv4 address system is fundamentally a nomenclature to represent large quantities in a compact, simple way.

Also, an IPv4 address always have four coefficients, each one can have a value in the range from 0 to 255. The number 255 in binary base is $(1111\ 1111)_2$ and since 8 bits is one byte, an IPv4 address has a total of **four bytes** or **32 bits**.

The IPv4 address space begins with the first IPv4 address 0.0.0.0 and ends with 255.255.255.255 which converted back to decimal is:

$$255.255.255.255 = \mathbf{255} \times 256^3 + \mathbf{255} \times 256^2 + \mathbf{255} \times 256^1 + \mathbf{255} \times 256^0 = 4{,}294{,}967{,}295$$

Consequently, there are 4 billion, 294 million, 967 thousand and 296 IPv4 addresses. Since the beginning of the Internet, this large number space was sliced into smaller blocks to be assigned to the organizations that make up the Internet. For example, Sheridan College owns the IPv4 address space from **142.55**.0.0 to **142.55**.255.255. No one else can use such space because it is globally assigned to Sheridan College.

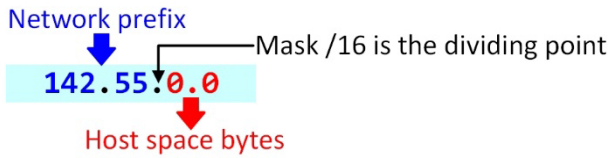
142.55.0.0
142.55.0.1
142.55.0.2
142.55.0.3
142.55.0.4
142.55.0.5
...etc
all the way to
142.55.255.254
142.55.255.255

Sheridan College IPv4 address space

Stating all the range of addresses from beginning to end is obviously not practical. Therefore, there must be a way to represent the whole block of addresses that belong to the organization without specifying all these numbers. Notice that all the addresses in the range begin with the same value (142.55). In this case, the first two bytes or 16 bits of the address block are **prefixed** to the value 142.55. All that is needed is a way to indicate that these bits belong to the **network prefix**. The **mask** does exactly that. A **mask /16** means that the first 16 bits of the address belongs to the network prefix which is the part of the address that does not varies. Thus, Sheridan College 142.55.0.0**/16** means that all the addresses that begin with such prefix are located inside the College's network domain.

Whatever remains after the "masked" portion of the address belongs to the **host space** which is the combination of all the numbers from 0 to 255 for the remaining bytes. The host space is used to allocate

the addresses to the computers, servers, and mobile devices. Since the host space also has 16 bits in this example, the total number of addresses is 2^{16} = 65,536. There are 65,526 addresses counting all the combinations from 142.55.0.0 to 142.55.255.255. In summary:



The structure of an IPv4 network address

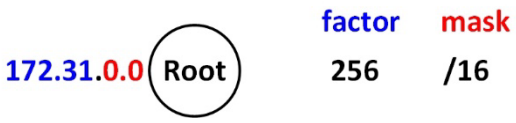
The main concept to keep is that the network mask indicates the number of binary bits that belong to the network prefix. Once that these fundamentals are understood, the next task is to learn how to divide IPv4 space into subnets.

Subnetting the IPv4 address space of the default VPC with a binary tree method.

In this new example, the address space of a default AWS VPC is going to be subneted to come up with the six contiguous /20 default subnets. The default VPC address space is a block of addresses that begin in **172.31**.0.0 and ends in **172.31**.255.255. Hence, the space is summarized by the prefix **172.31.0.0/16**.

A binary tree method is a simple way to do subnetting. This is the method, step by step:

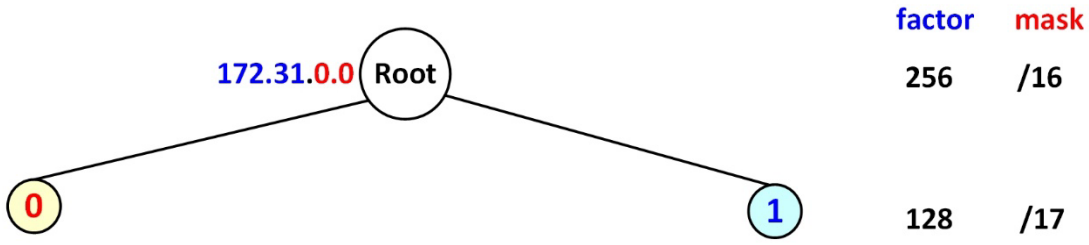
- First, set up a root which is the very first address of the block or network prefix like this:



The root is set to the network prefix of the block of addresses; in this case, 172.31. The mask /16 is set at the left and a factor of 256.

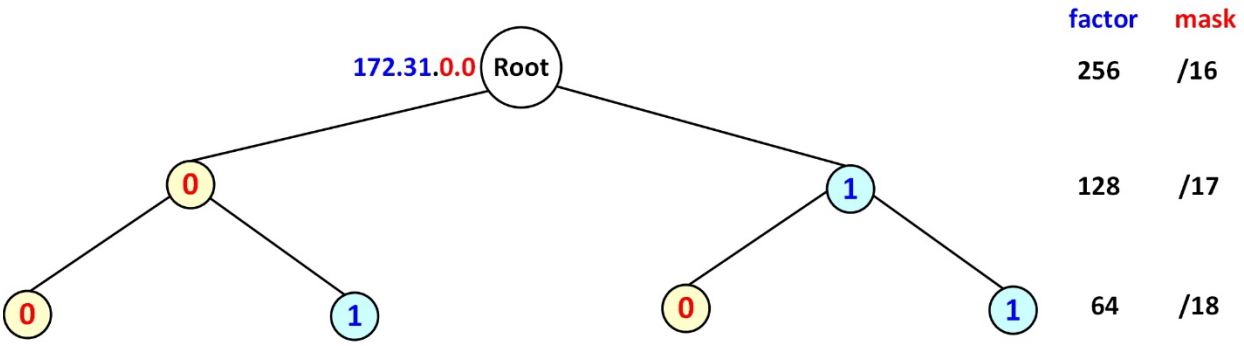
Note: this method works for masks /8, /16, and /24.

- The next step is to divide the space into two equal size portions. The factor 256 decreases by half (to 128) because each node contains one half of the addresses. The mask increases by one to /17. The node to the left has a bit set to 0 and the node to the right has a bit set to 1.

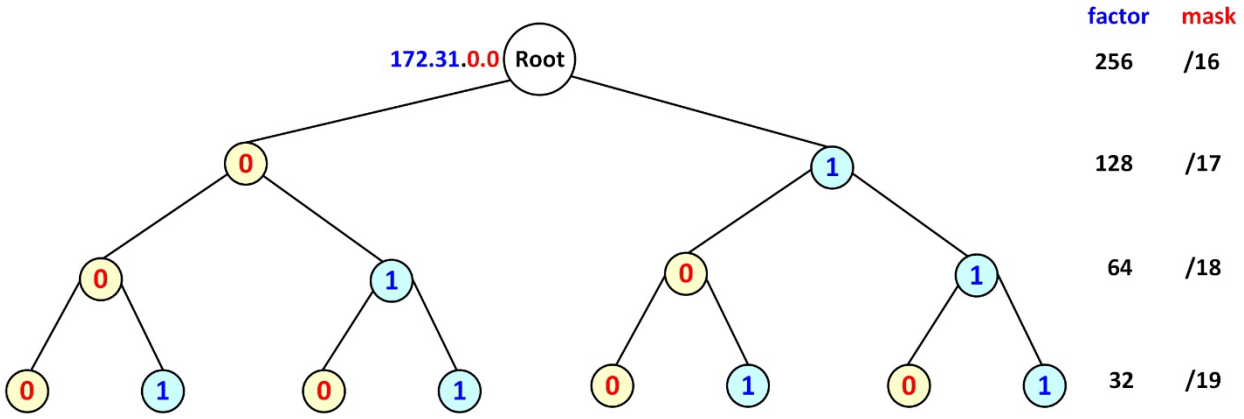


- The next step is an iteration of the previous. The spaces are divided into halves and the mask is increased by one to /18. For each branch, the nodes to the left are always zero and the nodes to

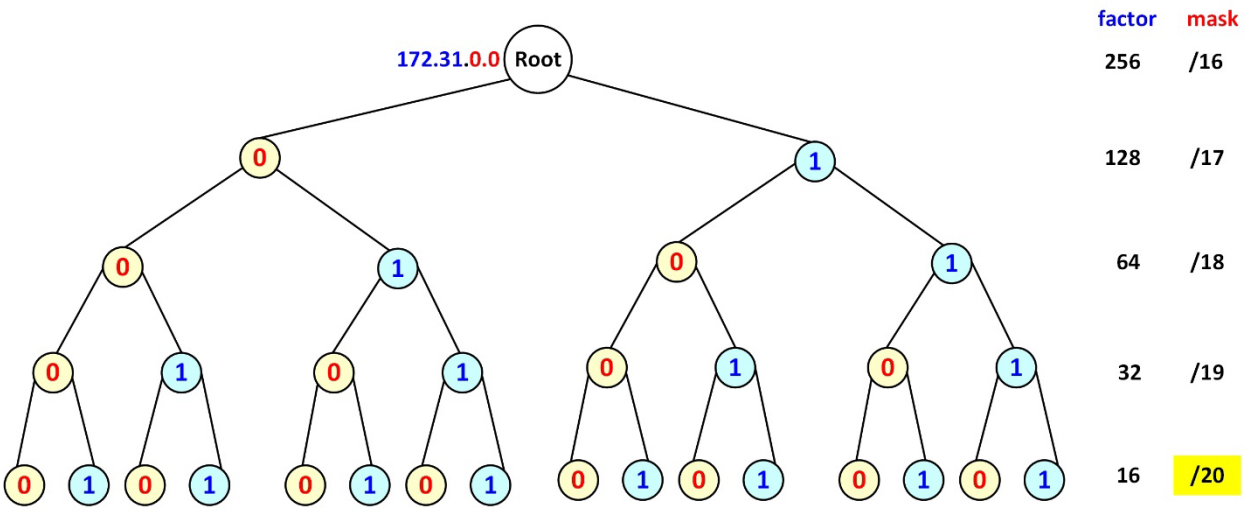
the right are always 1. Notice that the IPv4 address space is the same, it is just that it has been divided into four quarters.



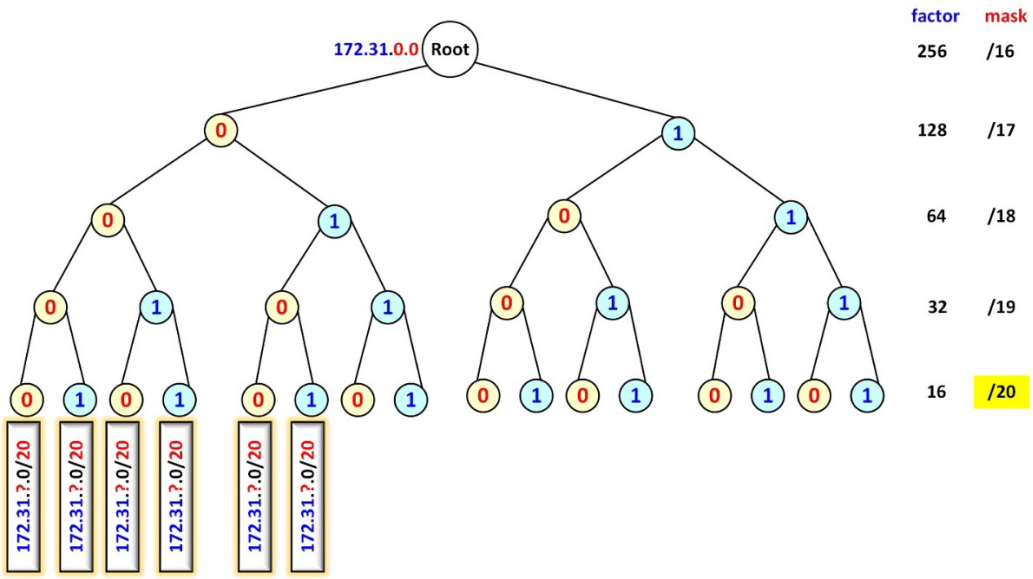
- Repeat the same process. Divide 64 by half, increase the mask by one, and the new left nodes are set to binary 0 while the right nodes are set to binary 1. (Hence the binary tree).



- Divide each node space one more time to arrive to the desired mask /20.

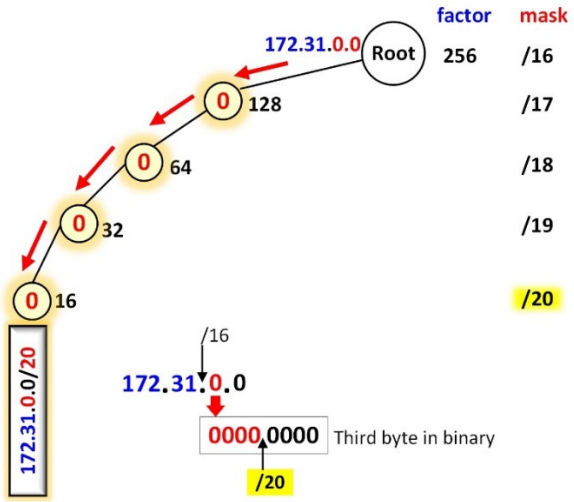


- Choose six contiguous nodes (subnets) of address space /20 starting from the leftmost. (It is always easier to start from the left because the numbers for the prefixes are easier to add).



Notice that the third byte of each subnet has a question mark. These values must be found in the next steps.

- Navigate or walk from the top root to each node, collecting the factor values of the 1s nodes.
- This is the rule: zero does not count, but a one is equal to the factor value to the right. For example, if a one is at level 64, then 64 must be carried over.
- The first subnet has four consecutive zeroes from the top to the bottom. So, that adds to zero.

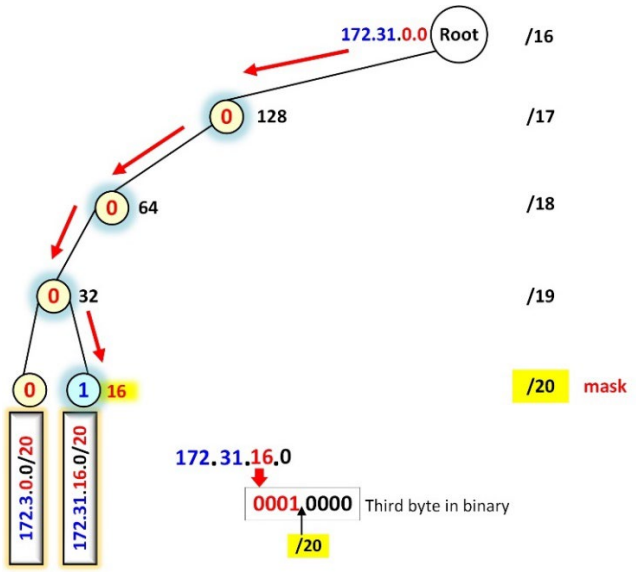


The root supplies the value 172.31 for the first two bytes of the address while the binary tree supplies the first four bits of the third byte (0001). That yields:

$$\mathbf{0} \times 2^7 + \mathbf{0} \times 2^6 + \mathbf{0} \times 2^5 + \mathbf{0} \times 2^4 + \mathbf{0} \times 2^3 + \mathbf{0} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{0} \times 2^0 = 0$$

Thus, the first subnet is 172.31.0.0/20.

- Find the next subnet following the trail of bits from the root to the bottom node.

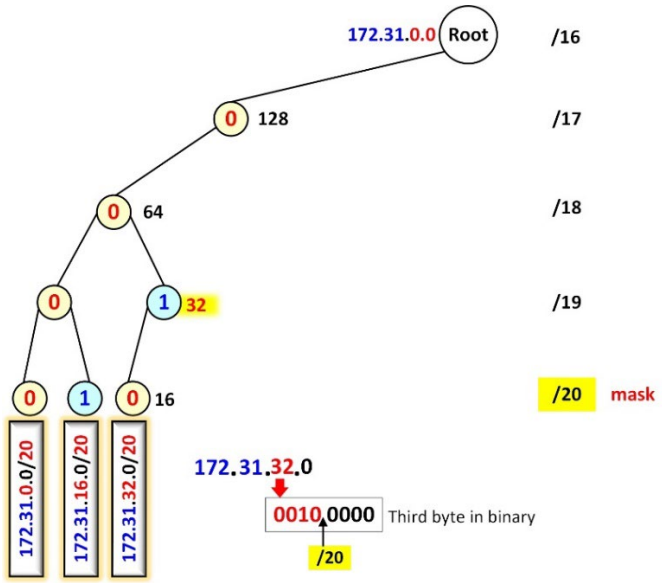


The root supplies the value 172.31 for the first two bytes of the address while the binary tree supplies the first four bits of the third byte (0001). That yields:

$$0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 16$$

Thus, the second subnet is 172.31.16.0/20.

- The same procedure is used to find the third subnet.

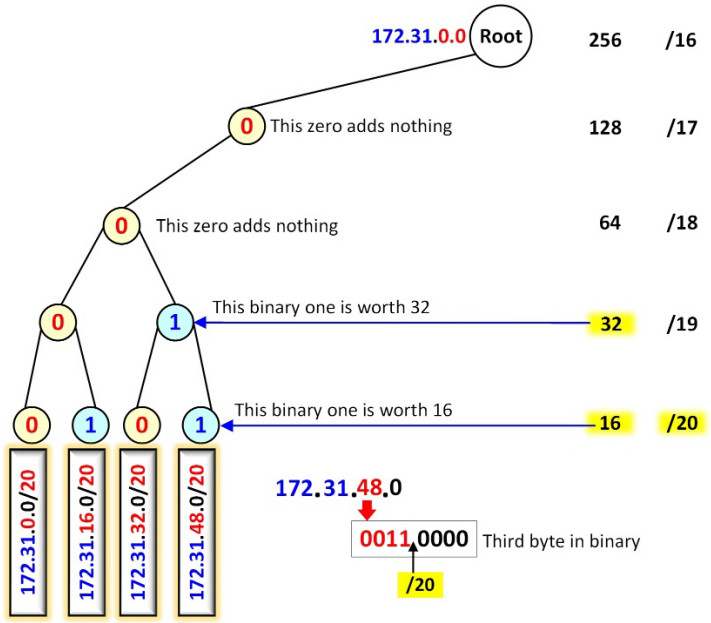


The third byte yields:

$$0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 32$$

Thus, the third subnet is 172.31.32.0/20.

- Find the fourth subnet using the same method.

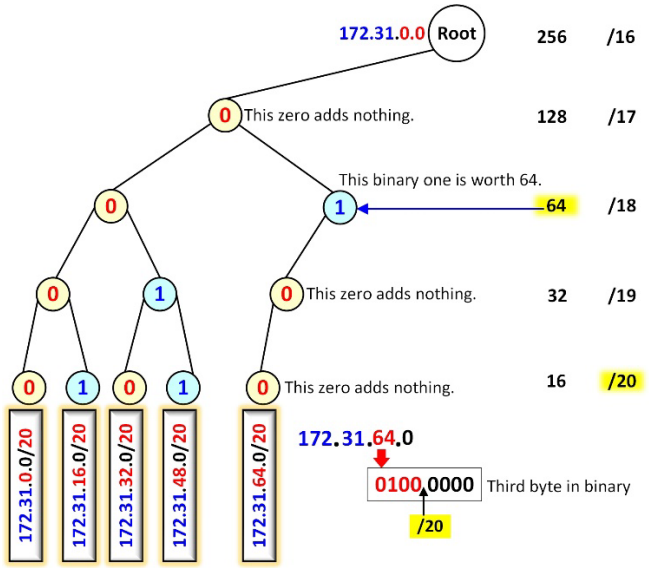


The third byte yields:

$$0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 48$$

Thus, the fourth subnet is 172.31.48.0/20.

- Find the fifth subnet using the same method.

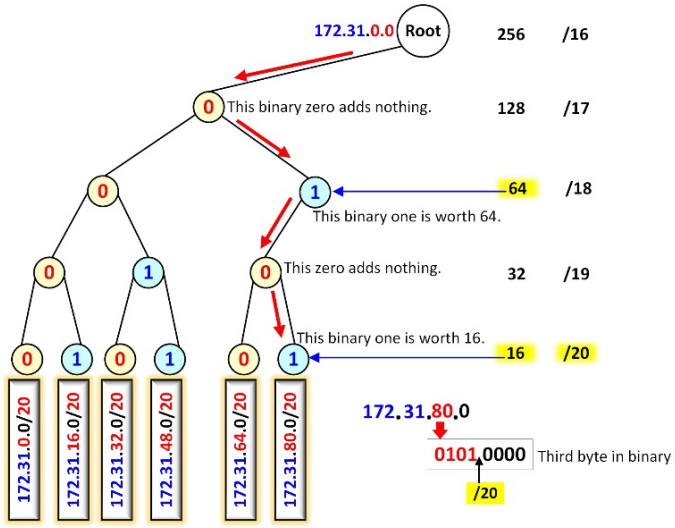


The third byte yields:

$$0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 64$$

Thus, the fifth subnet is 172.31.64.0/20.

- Find the sixth subnet using the same method.



The third byte yields:

$$\mathbf{0} \times 2^7 + \mathbf{1} \times 2^6 + \mathbf{0} \times 2^5 + \mathbf{1} \times 2^4 + \mathbf{0} \times 2^3 + \mathbf{0} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{0} \times 2^0 = 80$$

Thus, the sixth subnet is 172.31.80.0/20.

This concludes the operation of finding the first six default subnets of the VPC. A good question to ask now is how many IPv4 addresses are in each of the subnets. All the subnets have the same mask (/20) consequently, they have the same size. All IPv4 addresses are made up of 32 binary bits. Since the first 20 bits of the addresses belong to the subnet prefix, whatever is left belongs to the host space. Therefore:

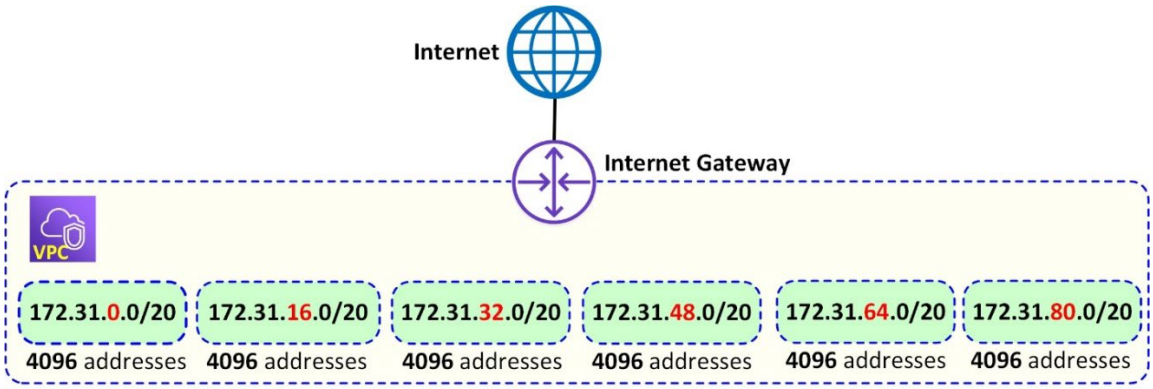
$$Host\ space = 32 - /mask$$

$$Host\ space = 32 - 20 = 12\ bits\ in\ the\ host\ space$$

The binary combination of 12 bits yields 4,096 addresses.

$$total\ addresses = 2^{12} = 4{,}096$$

Each default subnet contains 4096 IPv4 addresses.



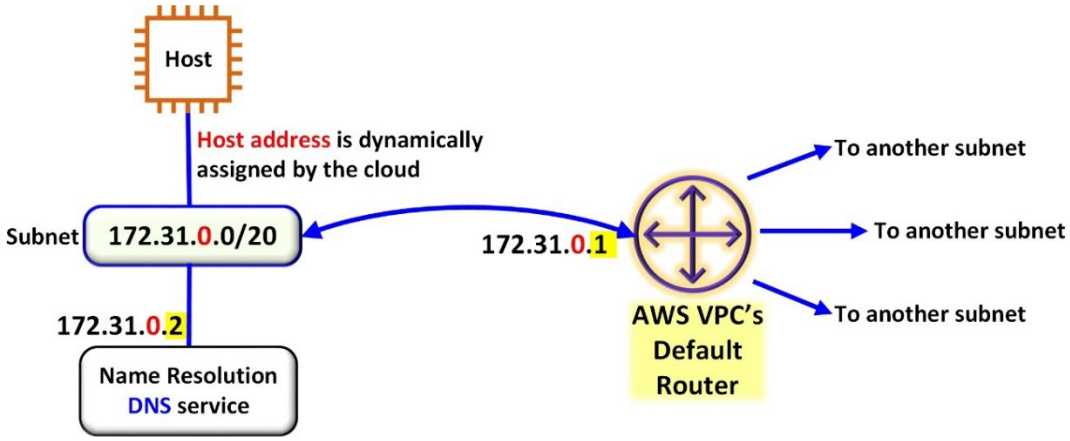
IPv4 address space of the six default subnets in the default VPC.

The following diagram shows the six subnets hosts combinations. There are too many to list them all, but each block represents 4096 addresses. Notice that the first subnet begins in 172.31.0.0 and it ends in 172.31.15.255. This is the result of the prefix 0000 and all the binary combinations of the host bits. Also notice, that the last address of the first block is contiguous to the first address of the second subnet 172.31.16.0. Basically, the end of every subnet is the boundary where the next subnet begins because they were chosen to be contiguous.

Third byte 0000 0000 ...etc, all the way to 0000 1111	Third byte 0001 0000 ...etc, all the way to 0001 1111	Third byte 0010 0000 ...etc, all the way to 0010 1111	Third byte 0011 0000 ...etc, all the way to 0011 1111	Third byte 0100 0000 ...etc, all the way to 0100 1111	Third byte 0101 0000 ...etc, all the way to 0101 1111
172.31.0.0	172.31.16.0	172.31.32.0	172.31.48.0	172.31.64.0	172.31.80.0
172.31.0.1	172.31.16.1	172.31.32.1	172.31.48.1	172.31.64.1	172.31.80.1
172.31.0.2	172.31.16.2	172.31.32.2	172.31.48.2	172.31.64.2	172.31.80.2
172.31.0.3	172.31.16.3	172.31.32.3	172.31.48.3	172.31.64.3	172.31.80.3
172.31.0.4	172.31.16.4	172.31.32.4	172.31.48.4	172.31.64.4	172.31.80.4
...etc, all the way to	...etc, all the way to	...etc, all the way to	...etc, all the way to	...etc, all the way to	...etc, all the way to
172.31.15.253	172.31.31.253	172.31.47.253	172.31.63.253	172.31.79.253	172.31.95.253
172.31.15.254	172.31.31.254	172.31.47.254	172.31.63.254	172.31.79.254	172.31.95.254
172.31.15.255	172.31.31.255	172.31.47.255	172.31.63.255	172.31.79.255	172.31.95.255

Blocks of IPv4 addresses of the default subnets in AWS us-east-1 region

There are universal rules [2] for the use of IPv4 addresses. The subnet prefix, for example 172.31.0.0/20 can not be used by any device because it is precisely the address of the subnet. The last address of every subnet can not be used by any device either because it has a special purpose which is broadcast (although AWS hypervisor does not implement broadcast in the subnets). Beside those rules, AWS also has its own rules [3] for the usage of the address space. The first host address (172.31.0.1 in the example) is assigned to the **default gateway router** of the subnet. The second host address (172.31.0.2 in the example) is assigned to the Domain Name System (DNS) service that resolves names to IPv4 addresses. The third host address (172.31.0.3) is reserved for special VPC purposes.



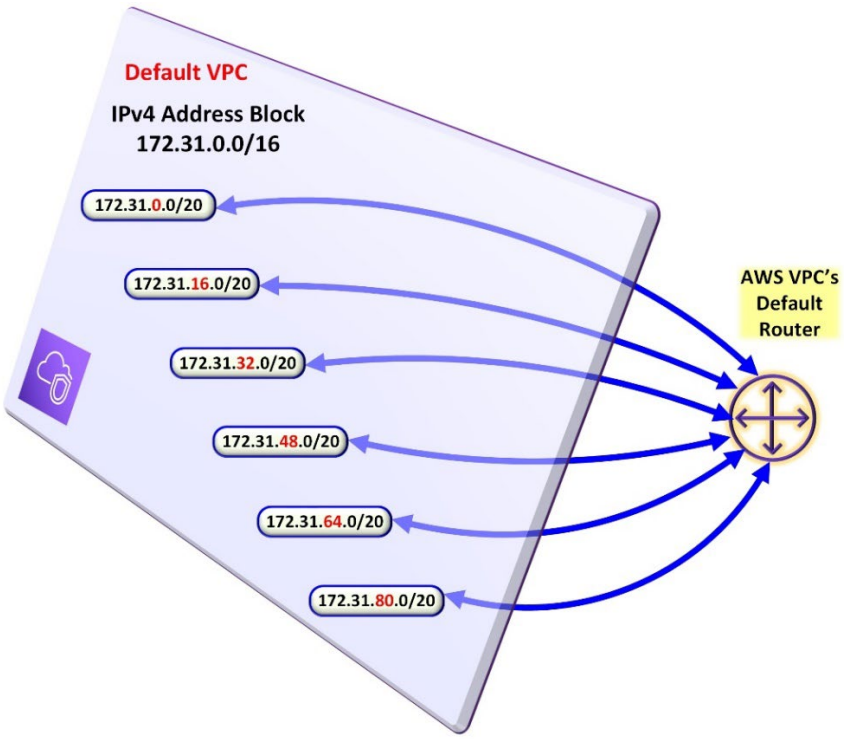
Representation of a subnet with a host EC2 instance, the default router and the DNS service.

Since there are five addresses in each subnet reserved for the reason presented above, a host EC2 instance deployed in the subnet will get a private IPv4 address within the range from 172.31.0.4 to 172.31.15.254 inclusive. This allocation is done automatically by the VPC's services. When the host resource is created, it is configured with the location of the DNS server and the default gateway so it knows how to resolve names to IPv4 addresses and how to send traffic to other networks via the default router. The following snippet taken from within an EC2 instance deployed in the 172.31.0.0/20 subnet shows how to gather this information with Linux commands.

```
[ec2-user@ip-172-31-0-25]$ ip route
default via 172.31.0.1 dev eth0
[ec2-user@ip-172-31-0-25]$ cat /etc/resolv.conf
search ec2.internal
nameserver 172.31.0.2
```

The VPC's default router.

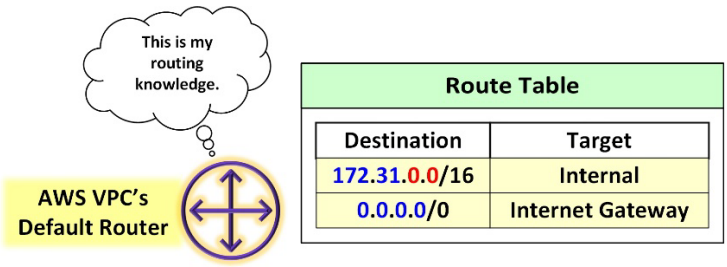
A router is a device that makes the IP traffic forwarding decisions. The VPC's default router is a virtualized router that interconnects the subnets in a VPC. This default router makes possible that the traffic sourced from a host to reach another instance in another subnet in the VPC (provided that the access lists and security groups in the path are not blocking such traffic). The VPC's default router is a software object that always exists in every VPC. It can not be eliminated or modified; in fact, the customer has no access to its properties. This is to prevent the customer from blackholing a subnet by mistake. The following diagram is an interpretation of the role of the virtual router. It is shown as an object, running behind the scene, interconnecting the subnets.



Representation of the VPC default router.

VPC Route Table

A **routing table** is the **table of IP knowledge**. For the routers to make the forwarding decisions, they must know where the destination subnets are located. Simply put, no knowledge, no routing. Every subnet must be associated with a routing table. The VPC's default router has a simple route table with two entries.



The default router and its basic routing table.

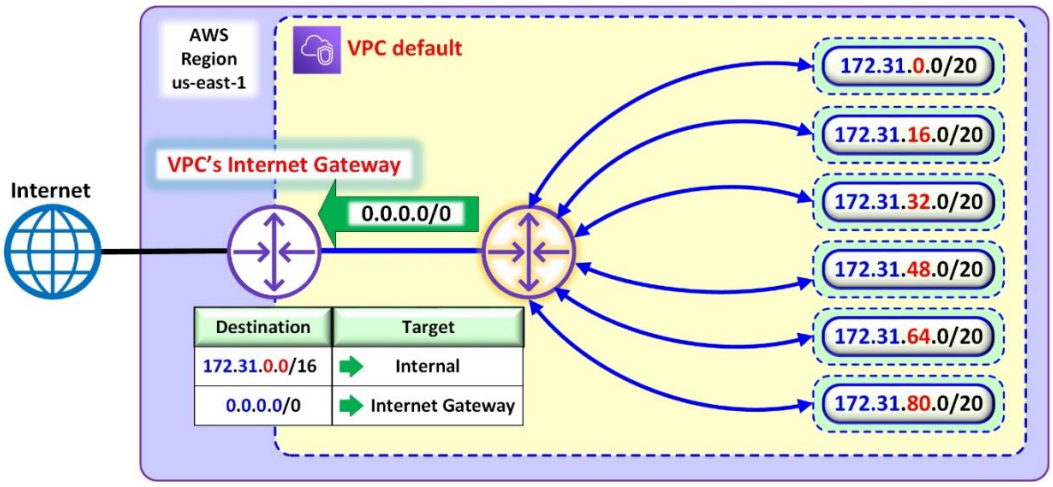
The first entry means: "any address from the IPv4 space 172.31.0.0/16 is found internally in this VPC". The second entry means: "for everywhere else, send the traffic to the Internet Gateway".

The route **0.0.0.0/0** has a universal meaning [2] which is **all the IPv4 addresses**. Basically, what the routing table is telling the router is "if the destination address begins with 172.31. then it is inside this VPC. Otherwise, it must be on the Internet; in such case, just send the traffic to the Internet gateway".

The default route table can be modified to add more information. This is typically done when an organization adds new VPCs or when it interconnects an on-premise datacentre with the AWS VPC via a Virtual Private Network (VPN).

The Internet Gateway

The Internet Gateway provides the access to the Internet. It is also a virtual router object that manages the IP traffic between the Internet and the VPC. Fundamentally, this is a **gateway of last resource**. The VPC's route table points all the traffic that is not local toward the Internet Gateway.



Interpretation of the routing logic of the VPC.

The Internet Gateway is not the same as the VPC's virtual router because a fully private VPC can exist without an Internet Gateway.

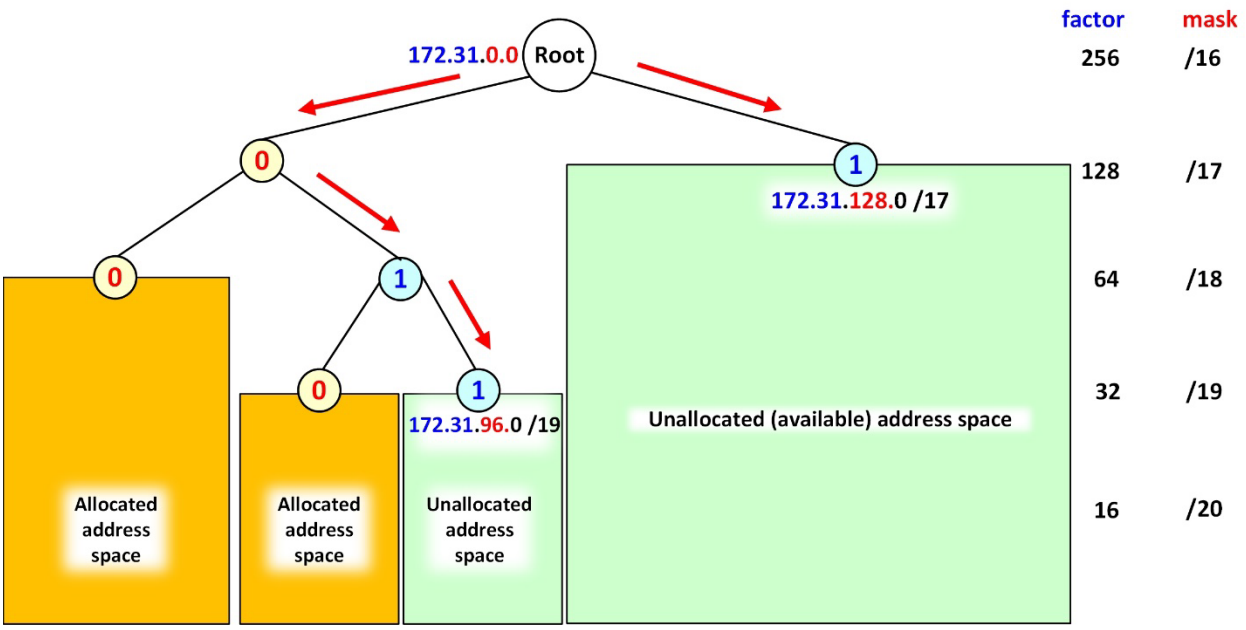
Public IPv4 addresses for resources in the cloud.

Compute resources such as EC2 instances receives two IPv4 addresses, one is from the private subnet from the space 172.31.0.0/16 and another from a pool of public [4] IPv4 addresses that belongs to AWS. Allocating a public IPv4 address to an EC2 instance is optional and it depends on the customer needs. An EC2 instance with a public IPv4 address has implicit access to the Internet via the Internet Gateway provided that the network access control list and the security groups allow it. Public addresses are valuable assets because they are scarce so when an EC2 instance is stopped, the address is returned to the pool of addresses. Consequently, the EC2 instance gets a different public address when it is re-started.

By now, the basic components of the VPC has been described. An Internet Gateway provides connectivity to the Internet, a default router object provides internal routing, the route table provides routing knowledge, and virtual subnets provide connectivity to resources such as virtual machines.

Configuring additional subnets.

Six subnets come pre-configured within the default VPC in the us-east-1 region from the space 172.31.0.0/16, but they do not take all the space. Therefore, a good question to ask is what happens to the rest of the 172.31.0.0/16 IPv4 space? Again, the binary tree offers a way to visualize the space. The answer is that the rest of the space is not allocated yet. It is available for add more subnets.



A view of the allocated and unallocated IPv4 space 172.31.0.0/16.

There are two large blocks of addresses available for subnetting. The binary tree shows that 172.31.96.0/19 and 172.31.128.0/17 are available.

- The block 172.31.96.0/19 contains: $32 - 19 = 13;\ \ then\ 2^{13} = 8192\ addresses$
- The block 172.31.128.0/17 contains: $32 - 17 = 15;\ \ then\ 2^{15} = 32{,}768\ addresses$

As a manner of demonstration, let's take the smaller space 172.31.96.0/19 to create two new subnets.

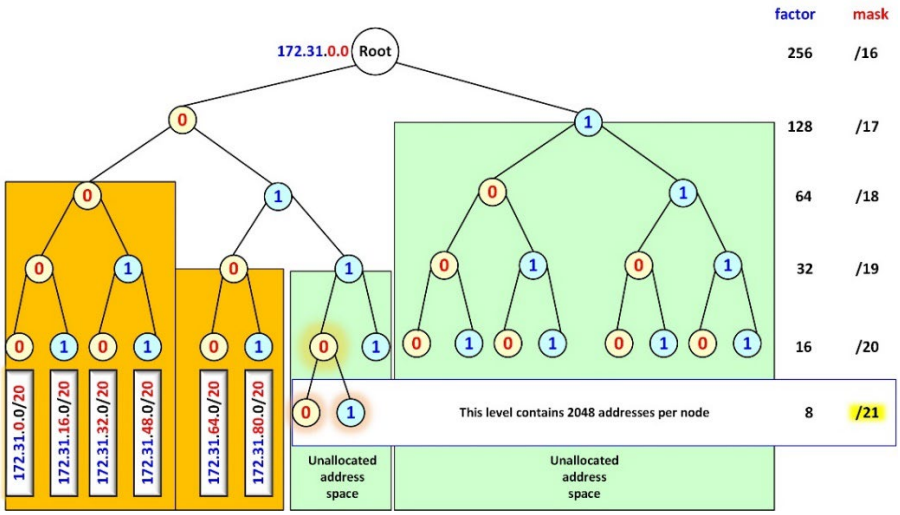
Learning Activity

Creation of subnets in the default VPC.

Each subnet will have 2048 IPv4 addresses. A mask of /21 is required to have blocks of 2048 addresses.

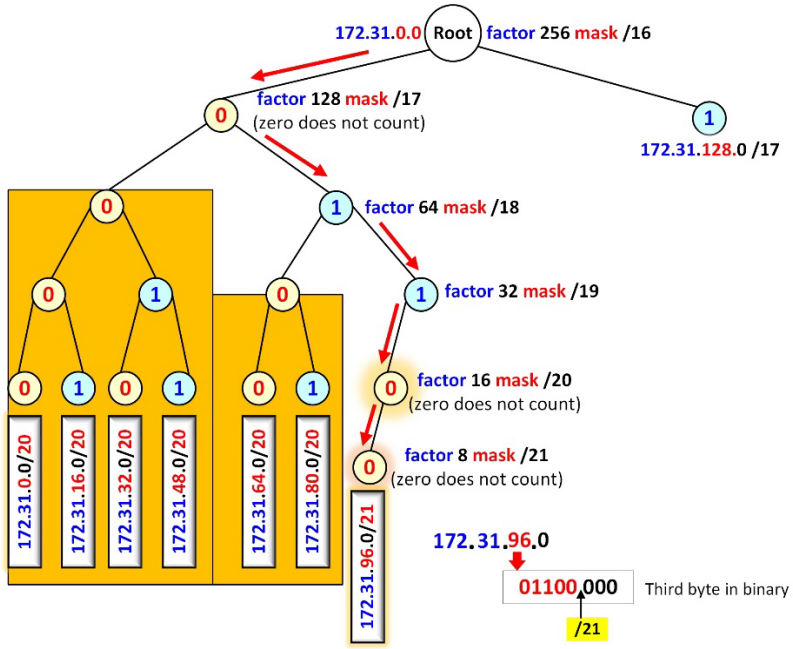
$$32 - 21 = 11;\ \ then\ 2^{11} = 2{,}048\ addresses$$

The next step is to find a node that contains such address space. All the available nodes /20 contain two /21 nodes. In principle, anyone will do, but the best practice is to move in order from left to right.



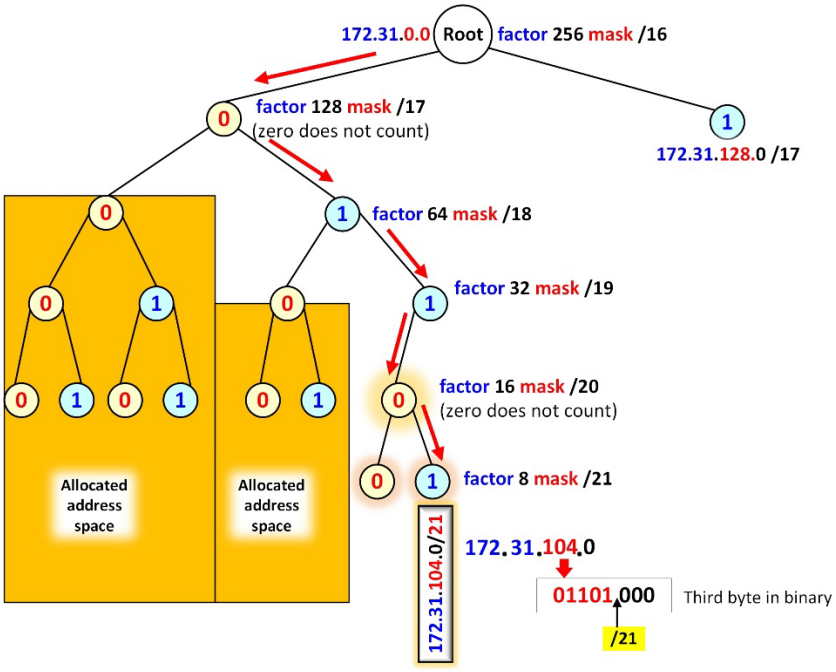
Choosing two subnets of 2,048 addresses each.

- Find the first subnet.



Subnet 172.31.96.0/21 contains 2048 addresses.

- Find the next subnet.



Subnet 172.31.104.0/21 contains 2048 addresses.

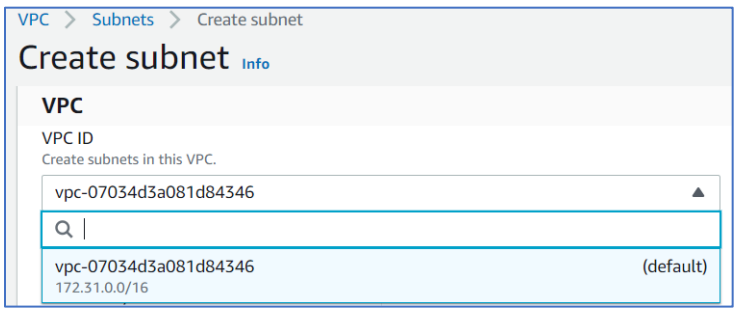
Now, let's proceed to the AWS console to deploy these two subnets 172.31.96.0/21 and 172.31.104.0/21.

Subnets (6) Info

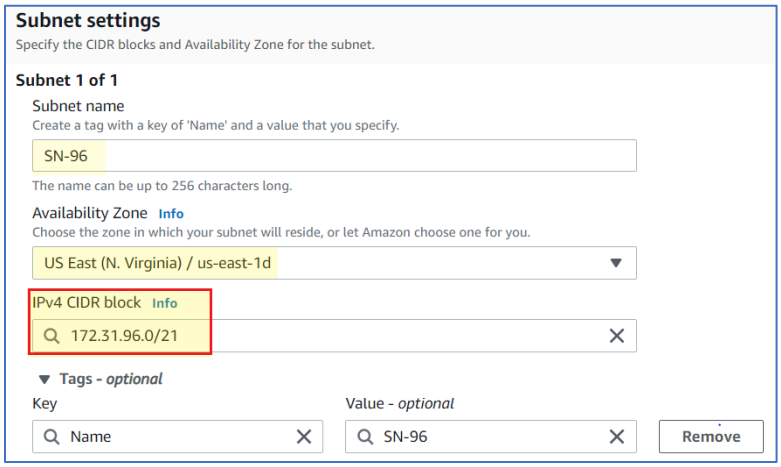
Name	Subnet ID	State	VPC	IPv4 CIDR
SN-0	subnet-0c03bb69301b98f8b	Available	vpc-07034d3a081d84346	172.31.0.0/20
SN-16	subnet-055e0f97f6bd2cba3	Available	vpc-07034d3a081d84346	172.31.16.0/20
SN-32	subnet-0a4c4aed4b38d5ee7	Available	vpc-07034d3a081d84346	172.31.32.0/20
SN-48	subnet-02949d50878724532	Available	vpc-07034d3a081d84346	172.31.48.0/20
SN-64	subnet-07ace735cf5cb335b	Available	vpc-07034d3a081d84346	172.31.64.0/20
SN-80	subnet-0339dce91800d93fe	Available	vpc-07034d3a081d84346	172.31.80.0/20

The VPC service subnets shows the six default subnets.

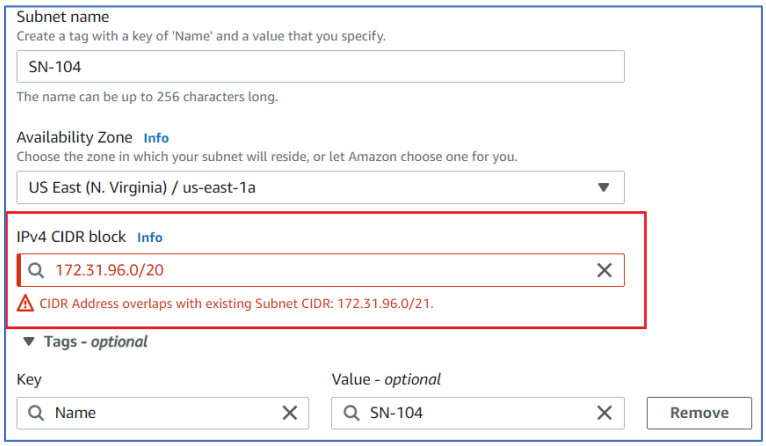
- Select the default VPC.



- Assign a name, an availability zone, and the subnet address prefix and mask.
- Create the subnet.

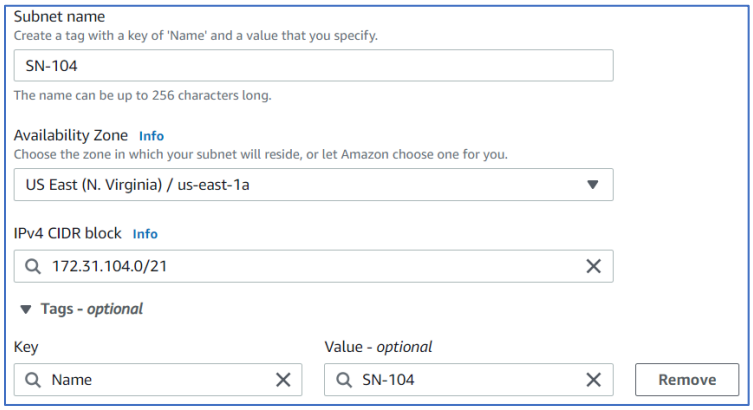


- Create the next subnet, but first make a mistake, try to assign the prefix 172.31.96.0/20.

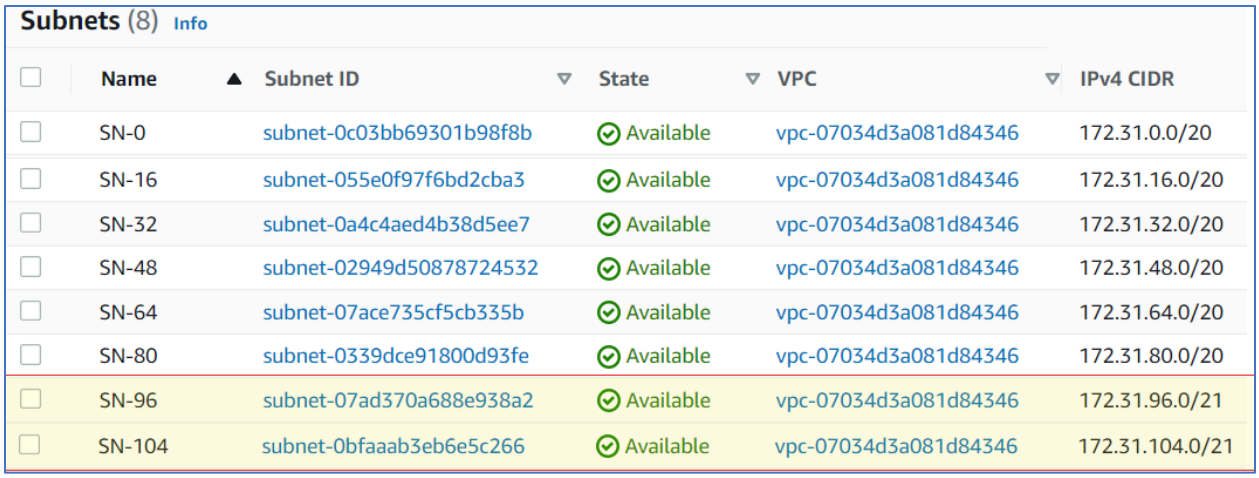


The configuration GUI rejects the attempt because 172.31.96.0/20 contains the space of 172.31.96.0/21, this is called an **overlap** which means that the same space can not be allocated twice, it is a conflict.

- Let's try with the correct allocation is 172.31.104.0/21.



The two subnets have been created.

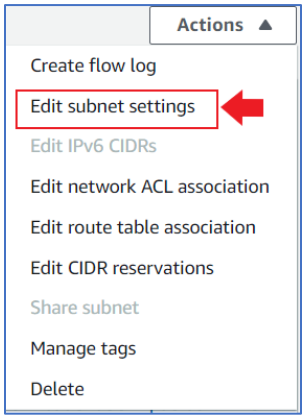


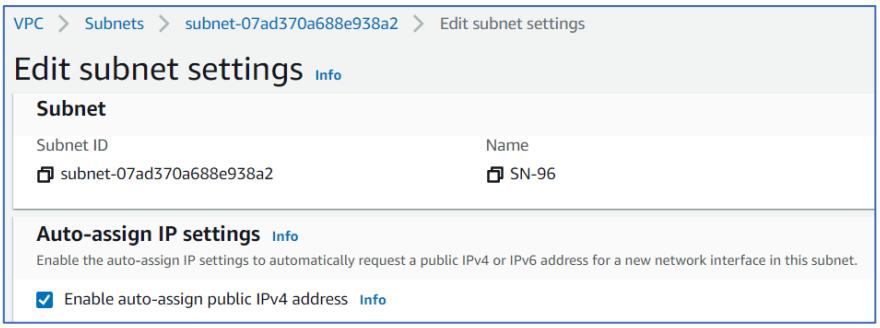
Subnets (8) Info

Name	Subnet ID	State	VPC	IPv4 CIDR
SN-0	subnet-0c03bb69301b98f8b	Available	vpc-07034d3a081d84346	172.31.0.0/20
SN-16	subnet-055e0f97f6bd2cba3	Available	vpc-07034d3a081d84346	172.31.16.0/20
SN-32	subnet-0a4c4aed4b38d5ee7	Available	vpc-07034d3a081d84346	172.31.32.0/20
SN-48	subnet-02949d50878724532	Available	vpc-07034d3a081d84346	172.31.48.0/20
SN-64	subnet-07ace735cf5cb335b	Available	vpc-07034d3a081d84346	172.31.64.0/20
SN-80	subnet-0339dce91800d93fe	Available	vpc-07034d3a081d84346	172.31.80.0/20
SN-96	subnet-07ad370a688e938a2	Available	vpc-07034d3a081d84346	172.31.96.0/21
SN-104	subnet-0bfaaab3eb6e5c266	Available	vpc-07034d3a081d84346	172.31.104.0/21

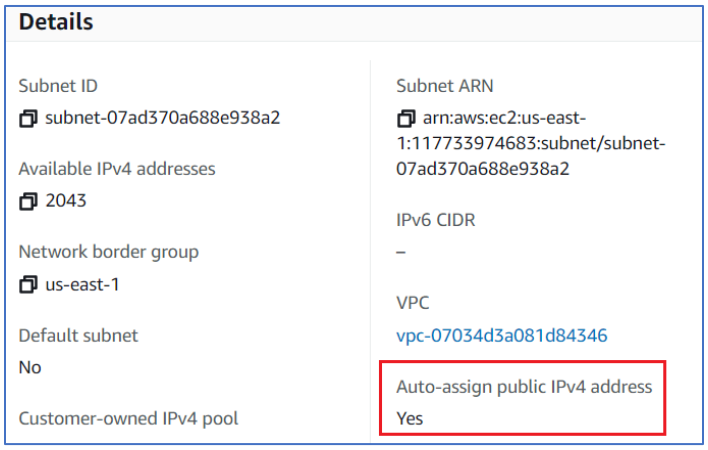
Let's examine the details of subnet named SN-96.

- There are 2,043 IPv4 addresses available because of the five addresses reserved for special uses.
- The subnet is already associated (by default) to the VPC's main routing table.
- The default NACL (Network Access Control List) is also applied to the subnet.
- But the Auto-assign public IPv4 address is set to No. New EC2 instances connected to this subnet will not get a public IPv4 address automatically. That can be fixed using **Actions**.





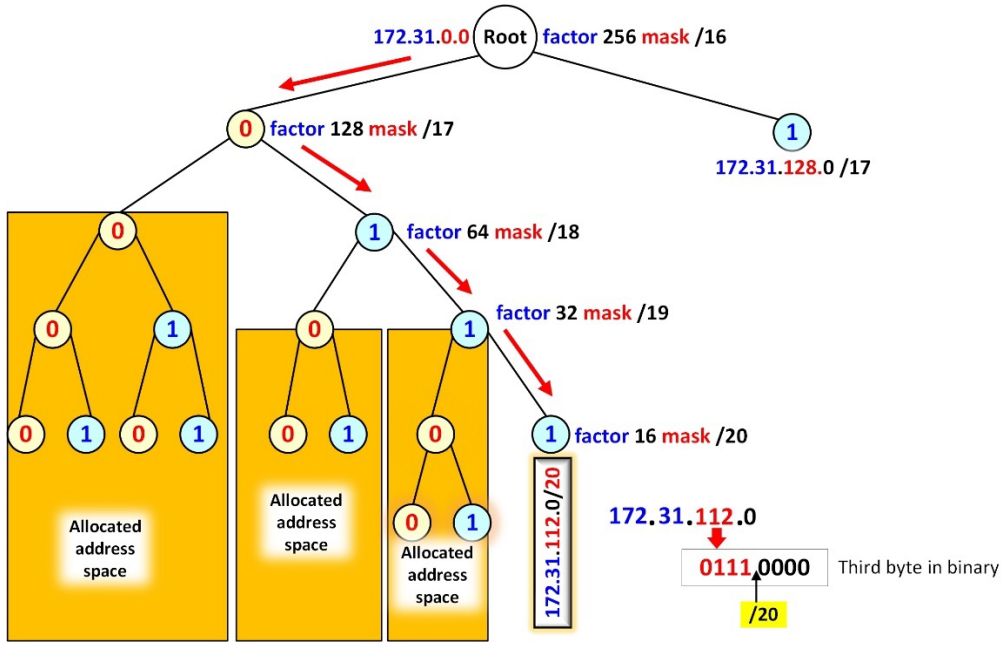
The Auto-assign public IPv4 address is set to yes now. (Repeat action for subnet SN-104)



Learning Activity

Creating a subnet with AWS CLI

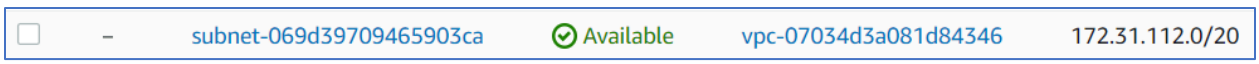
This activity is to learn another way to create a subnet using the command line tool AWS CLI. First, let's calculate the subnet.



- Issue the AWS CLI command on the AWS Academy console. The VPC id is required.

```
aws ec2 create-subnet --vpc-id vpc-07034d3a081d84346 --cidr-block 172.31.112.0/20
```

- The subnet is deployed now:



Learning Activity

Using Boto3 Python to obtain information regarding the subnets.

This Boto3 Python program prints the subnet IPv4 address, the availability zone and the number of available addresses in each subnet.

```
import boto3
sn = int(input('Enter the number of subnets: '))
ec2 = boto3.client('ec2')
response = ec2.describe_subnets()
for n in range(sn):
    cidr = response.get('Subnets', [{}])[n].get('CidrBlock', '')
    az   = response.get('Subnets', [{}])[n].get('AvailabilityZone', '')
    ip   = response.get('Subnets', [{}])[n].get('AvailableIpAddressCount', '')
    print('Subnet', cidr, 'in Availability zone', az, 'has', ip, 'addresses')
```

- Save it with the text editor nano or vi in the AWS Academy console. Run the program.

```
aws_academy_console:$  python get_subnet_info.py
Enter the number of subnets: 9
Subnet 172.31.0.0/20 in Availability zone us-east-1c has 4091 addresses
Subnet 172.31.16.0/20 in Availability zone us-east-1a has 4090 addresses
Subnet 172.31.32.0/20 in Availability zone us-east-1b has 4091 addresses
Subnet 172.31.48.0/20 in Availability zone us-east-1e has 4091 addresses
Subnet 172.31.64.0/20 in Availability zone us-east-1f has 4091 addresses
Subnet 172.31.80.0/20 in Availability zone us-east-1d has 4089 addresses
Subnet 172.31.96.0/21 in Availability zone us-east-1d has 2043 addresses
Subnet 172.31.104.0/21 in Availability zone us-east-1a has 2043 addresses
Subnet 172.31.112.0/20 in Availability zone us-east-1c has 4091 addresses
```

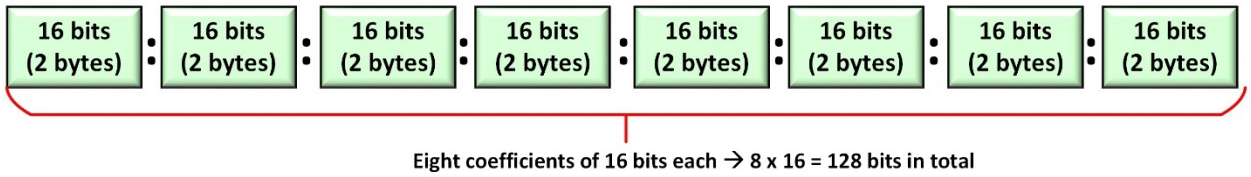
This concludes the exploration of the IPv4 address space of the default VPC. Knowing how to allocate IPv4 address space to subnets is one of the skills required for architecting virtual private clouds. The cloud architect must also know really well all the components of the VPC such as the default router, the internet gateway, and the routing tables. Additionally, the cloud environment designer must know how to handle IPv6 addresses. That is the next topic of this chapter.

Allocating IPv6 address space to the Virtual Private Cloud.

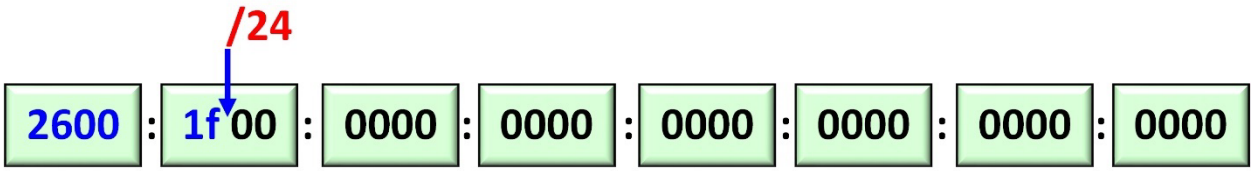
The Internet Protocol version 6 was designed [5] as a potential replacement for the IPv4 protocol in the 1990s. The main concern at that time was that the IPv4 address space of more than 4 billion addresses would be depleted, so a new system with more addresses would be needed to keep up with the continuous growth of the Internet. Therefore, the IPv6 protocol has a gigantic address space of 128 bits.

$$2^{128} = 3.402823669209384634633746074317 7 \times 10^{38}$$

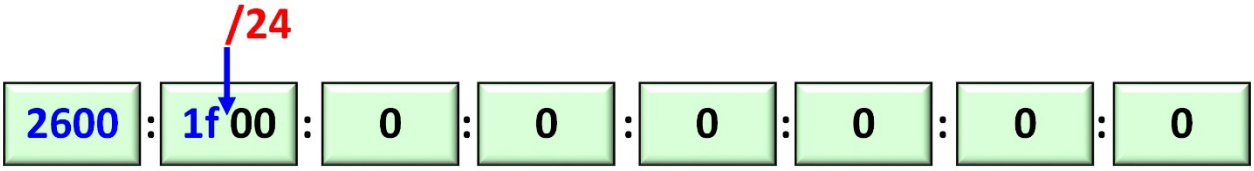
This address space is in the order of 340 undecillion so it is too large to even comprehend it. Because the numbers are so incredibly large, they are converted to base 65,536 and they are represented in a hexadecimal format of eight (8) coefficients to make them compact. Each coefficient is made up of two bytes or 16 bits and it is separated by a colon sign from the next coefficient.



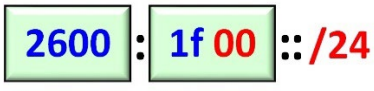
The following is an IPv6 address:



The network prefix is conformed by the first 24 bits of the address hence the mask is /24. Furthermore, IPv6 has rules to simplify the use of the address in some cases. One the rules is that the coefficients with consecutive zeroes can be simplified to this:



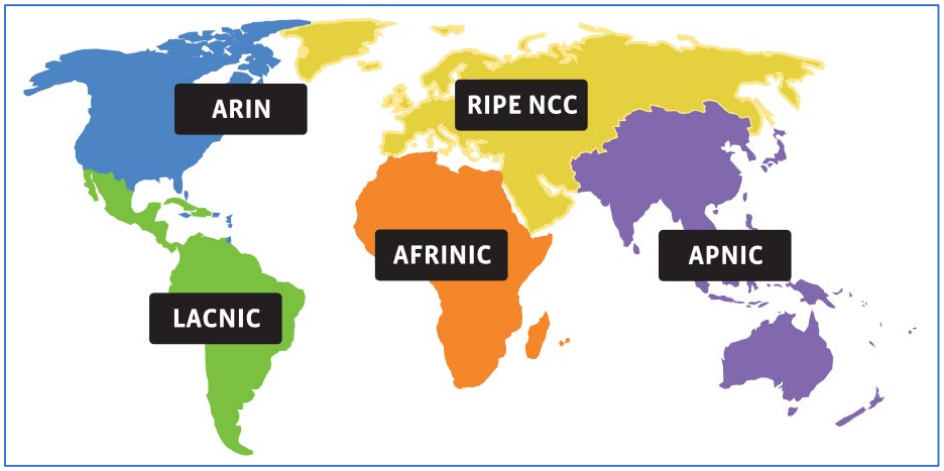
Another rule says that consecutive coefficients' of 0000 or 0 can be replaced with one double colon per address (this is allowed only once per address). The long IPv6 address becomes:



This is a public IPv6 block of addresses that has been allocated to Amazon Web Services (AWS). This block contains:

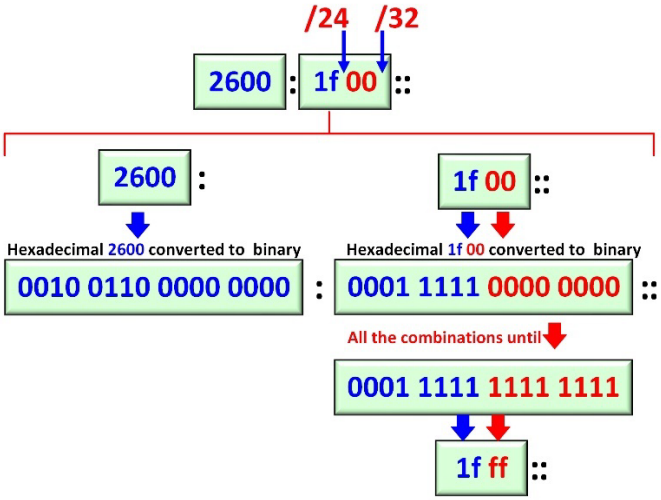
$$128 - 24 = 104\ bits \rightarrow\ 2^{104} =\ 20{,}282{,}409{,}603{,}651{,}670{,}423{,}947{,}251{,}286{,}016\ addresses$$

The deployment of the IPv6 addresses is done in a hierarchical manner. There are five continental organizations [6] responsible for the allocation of the address blocks. Each continent has several blocks assigned so the addresses can be identified geographically. The continental organizations provide address space to network carriers which in turn, allocate blocks of addresses to smaller Internet Service Providers and finally to the end customers. Because there are so many addresses, the distribution of the blocks must be done in an organized manner to keep the routing tables of the Internet to a manageable size.



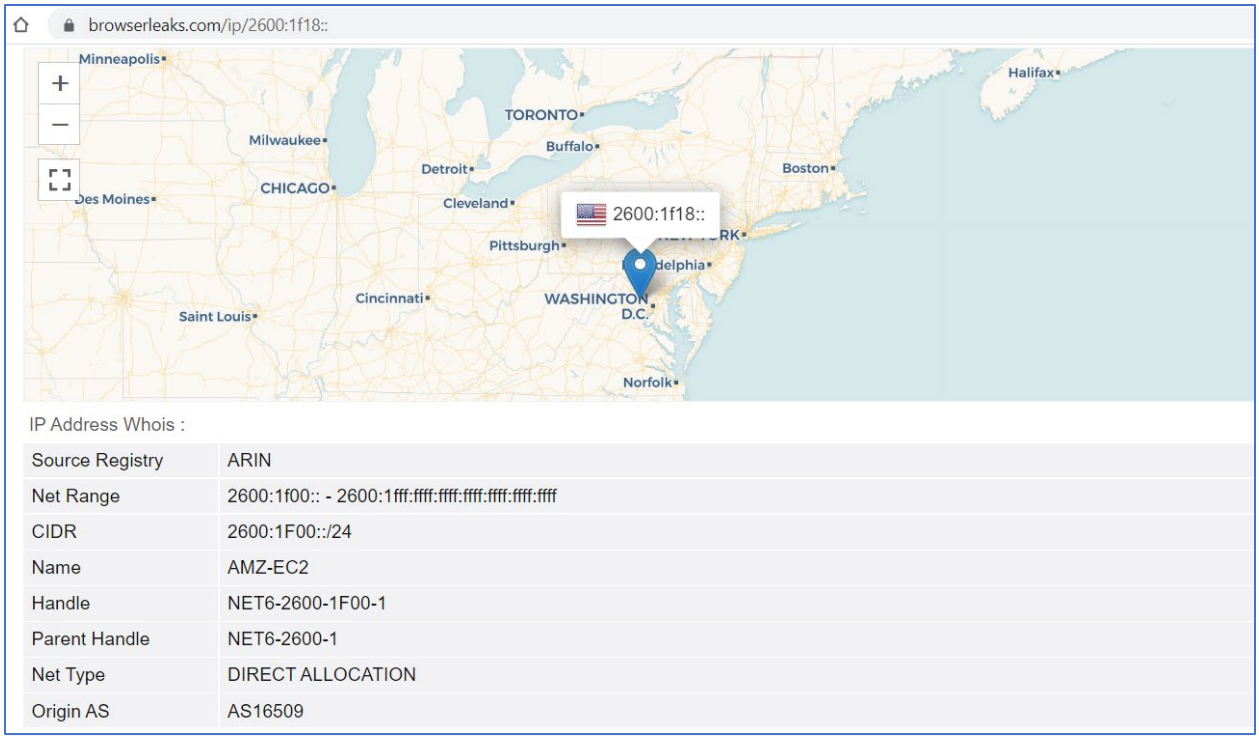
World Organizations [7] responsible for IPv6 allocation.

The American Registry of Internet Numbers (ARIN) is in charge of assigning the blocks of addresses in the U.S and Canada. Consequently, AWS got the prefix **2600:1f**00::/24 from ARIN. Using such massive block, AWS carves out other blocks, subnets literally, from mask /24 to mask /32.



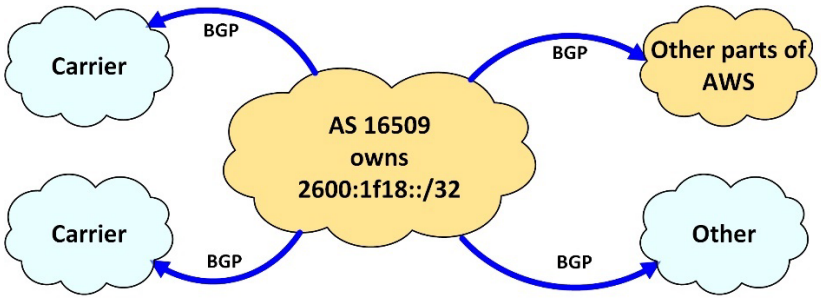
Subnetting IPv6 2600:1f00 from /24 to /32.

Because there are 8 bits from /24 to /32, AWS obtains 2^8 = 256 subnets beginning with 2600:1f00::/32 and ending with 2600:1fff::/32. One of those subnets is 2600:1f18::/32. AWS has given this prefix to its us-east-1 region as shown in the next picture [8].



IPv6 Address Space assigned [8] to AWS US-east-1 Region.

AWS owns the networking domain AS 16509 for the us-east-1 region. AS stands for Autonomous System. Every organization that connects to the Internet owns a unique, public AS number. (For instance, Sheridan College owns AS 5664 with the address space IPv4 142.55.0.0/16).



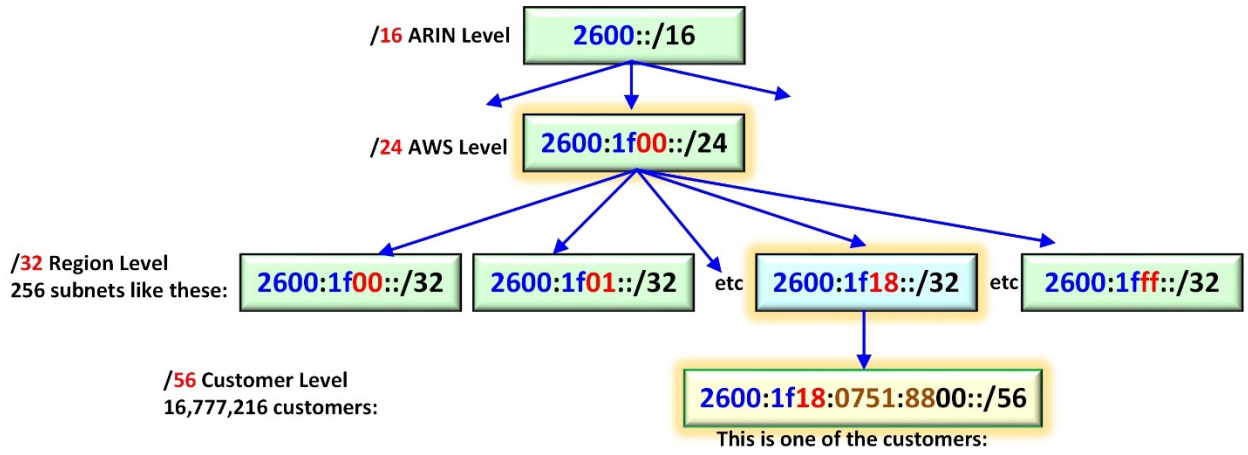
A very simplified view of the operation of the Internet.

AWS AS 16509 advertises the IPv6 prefix **2600:1f18::/32** address space to other Internet organizations such as major carriers [9] Cogent, GTT, Lumen (Level3), NTT, TATA, Telstra and others. These organizations talk to their neighbours using the routing protocol BGP. The Internet is a conglomerate of interconnected autonomous systems. The clouds AWS, Azure, GPC, etc are all connected to the Internet in the same way. Each AS advertises the address space inside. In short, if an IPv6 address begins with the prefix 2600:1f18 then it is inside AS 16509. That greatly simplifies the routing tables because one prefix represents a gigantic address space.

Inside the AWS cloud us-east-1a region, the prefix 2600:1f18::/32 is further subneted from /32 to /56. These are the subnets given to the customers that request them. AWS leases IPv6 address space to its customers. This subnetting yields:

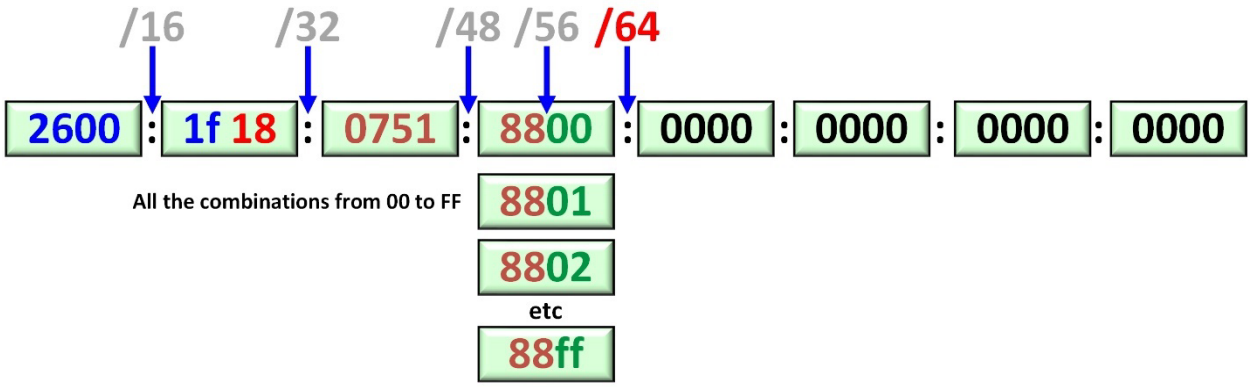
$$56 - 32 = 24\ bits \rightarrow 2^{24} = \ 16{,}777{,}216\ customer\ subnets$$

More than 16 million subnets are drawn from the prefix 2600:1f18::/32 to provide the customers with /56 size subnets. But the subnetting does not ends here. Let's suppose than a customer's VPC receives the subnet **2600:1f18:0751:8800::/56**.



Hierarchy of IPv6 subnetting from top to customer space /56.

In the example, the customer has all the addresses that begin with 2600:1f18:0751:88. AWS then allows the customer to draw subnets from /56 to /64 (8 bits of space). Consequently, the customer can obtain 256 subnets each with mask /64. These are the final prefixes to be assigned to the VPC's subnets.



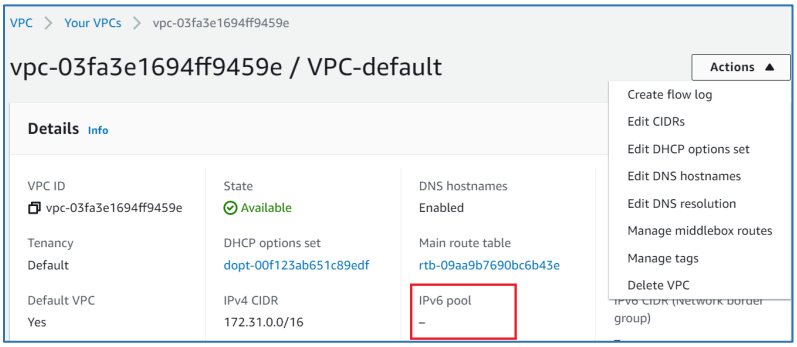
The customer has available all the subnets from /56 to /64.

In principle, it appears to be a difficult process to perform all these subnetting. In practice, the cloud service provides a user interface that makes trivially easy to obtain and use IPv6 address space as the next Learning Activity will show.

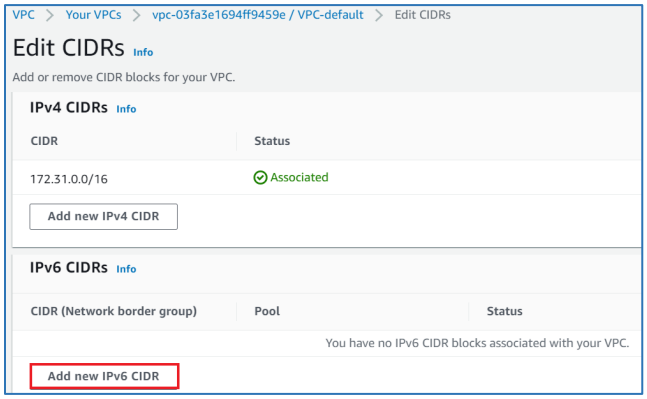
Learning Activity

Obtaining IPv6 Address Space from AWS

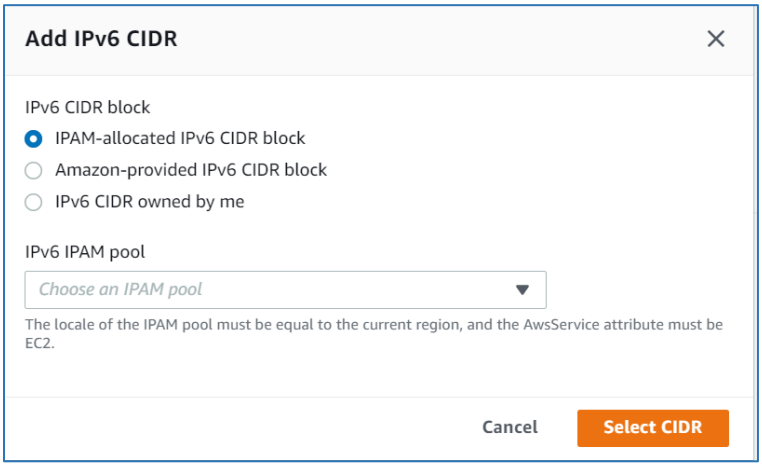
In this activity and IPv4 address space will be requested from AWS to be allocated to subnets in the VPC. Initially, the VPC does not have any IPv6 as seen in the following snippet.



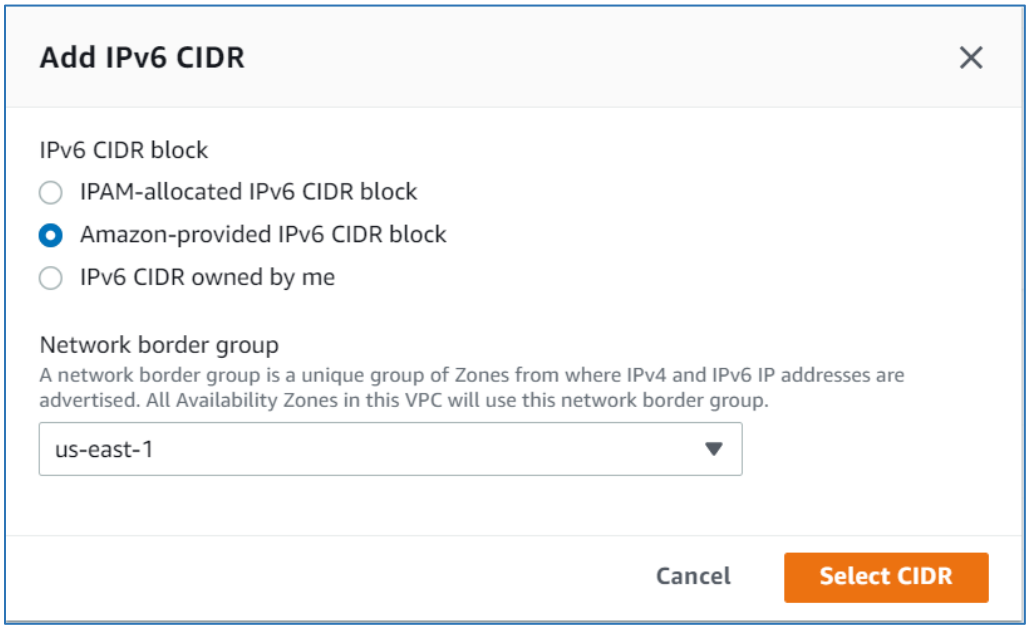
- Proceed to **Edit CIDRs**. Then **Add new IPv6 CIDR**.



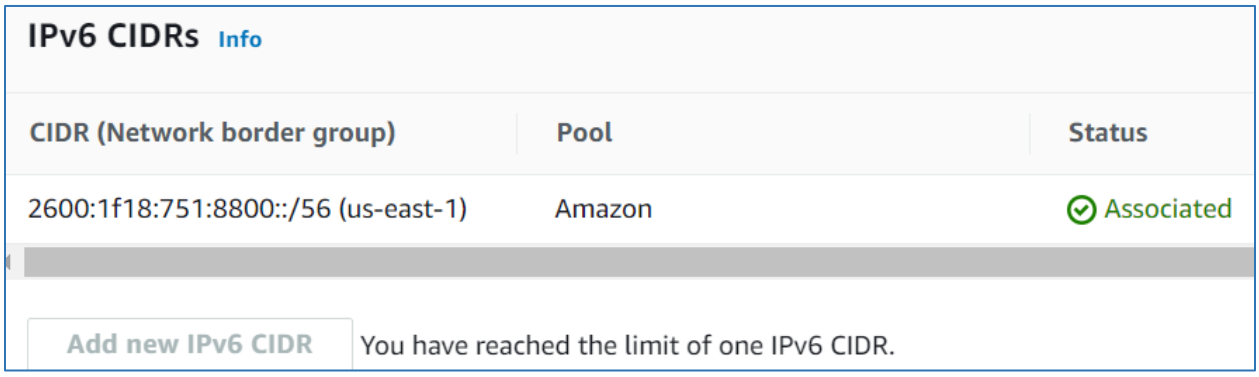
Three options are presented to the customer. The two options **IPAM (IP address management system)** and **IPv6 CIDR owned by me** allow a customer to bring its already allocated IPv6 address space into AWS with the proviso that the addresses must be regionally compatible. For example, an address block from North America (ARIN) can not be placed in the AWS Asia Pacific Region (APNIC).



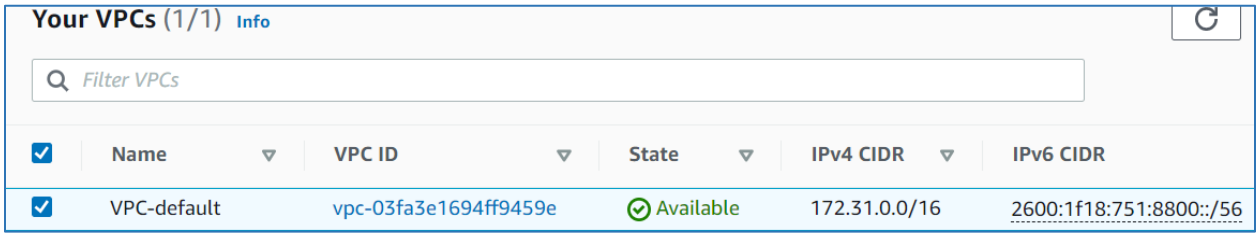
- Choose the option Amazon-provided IPv6 CIDR block.



When the region is indicated, AWS identifies the Network Border Group that matches the location. Automatically, a /56 chuck of addresses is selected from the available subnets. In this example, AWS offered the prefix **2600:1f18:0751:8800**:0000:0000:0000:0000/56.

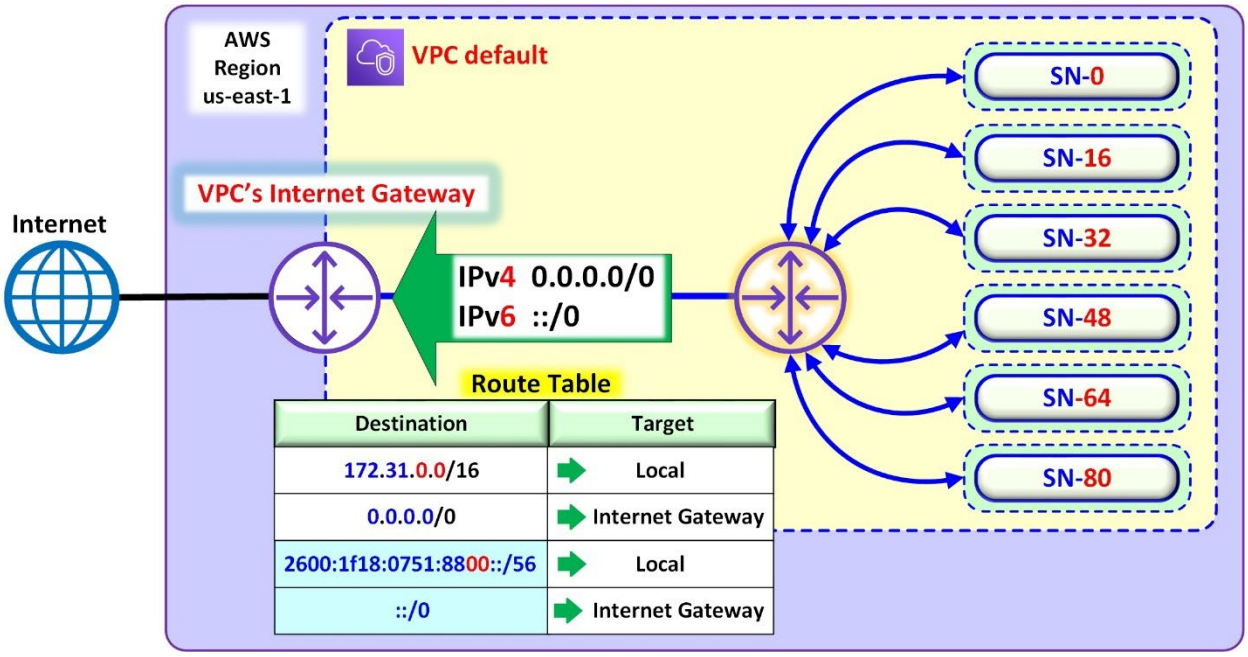


Finally, the customer is in possession of an IPv6 address block. Out of this block mask /64 subnets will be drawn to be assigned to the logical networks.



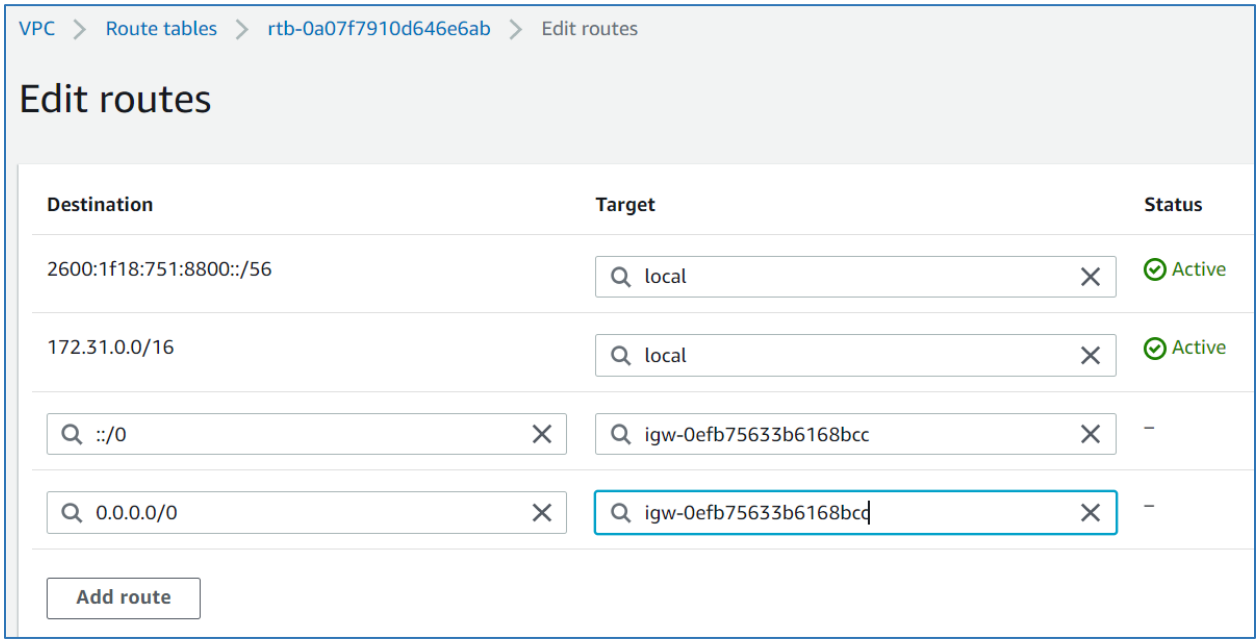
The next step is to adjust the VPC's route table to add the default route toward the Internet Gateway. All the addresses from **2600:1f18:0751:8800::/56** are located internally in this VPC. The route table got that entry by default when the address space was provided. However, the information regarding the access to

the Internet depends on the decision of the network administrator, so it has to be manually added. The IPv6 default gateway address is 0000:0000:0000:0000:0000:0000:0000:0000/0 mercifully simplified to just **::/0** (by protocol convention, all the contiguous zeroes got replaced by the double colon).

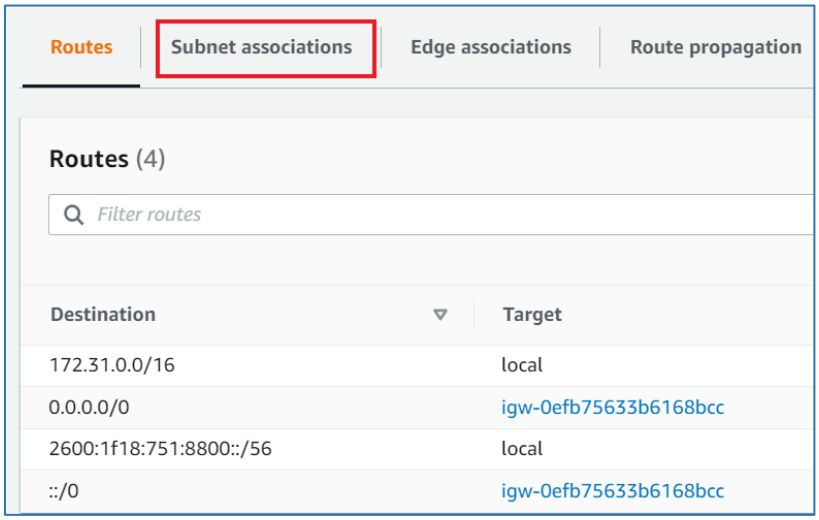


Representation of the default VPC with its Route Table for both IPv4 and IPv6.

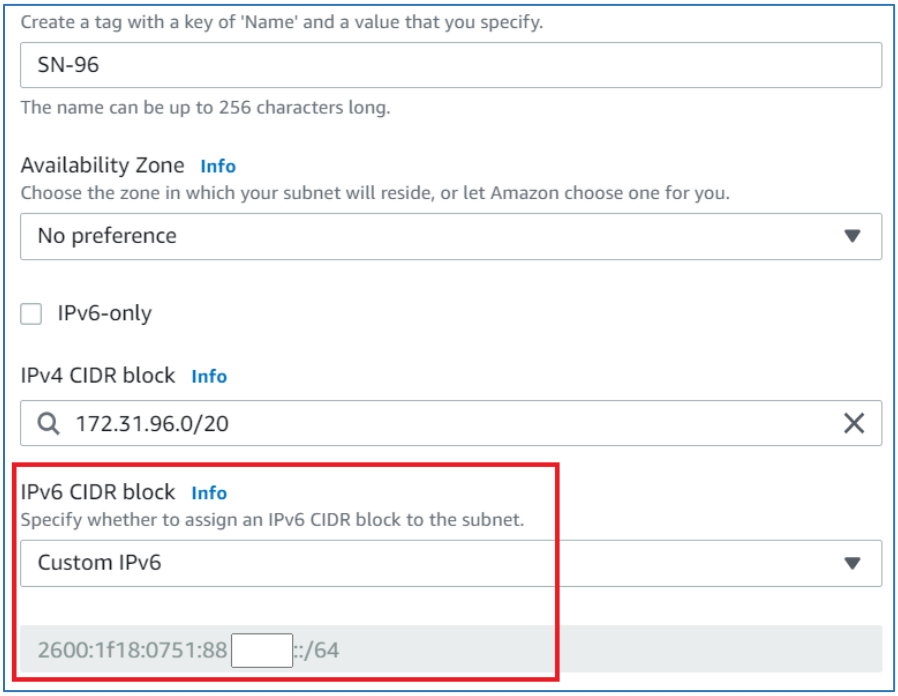
- Under VPC, Route Tables, choose the routing table and edit it.
- Add a route **::/0** and point it toward the Internet Gateway (igw).



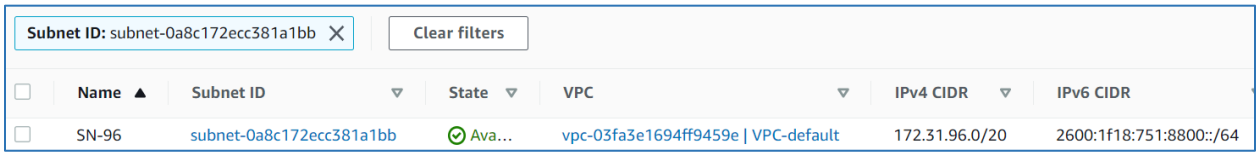
- Verify that the Subnet associations reflects the changes to the route table.



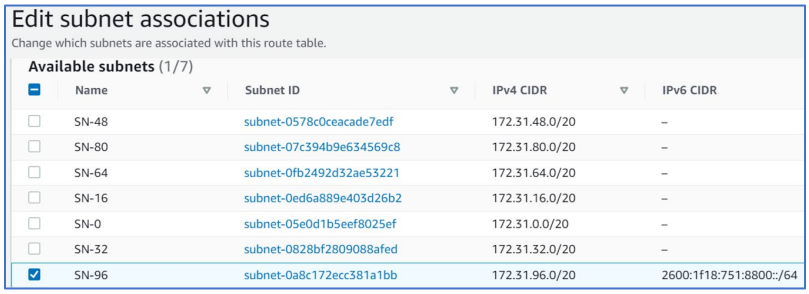
- Select an existing subnet to assign IPv6 address space. (A new subnet 172.31.96.0/20 was created for this example).



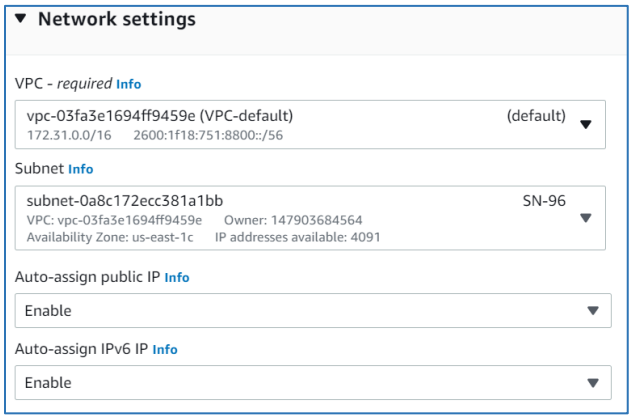
Notice that AWS does all the work, the customer only needs to supply a hexadecimal number between 00 and FF. The number 00 was selected since this is the first virtual network to receive an IPv6 prefix.



- Observe that SN-96 is the only subnet with an IPv6 block.



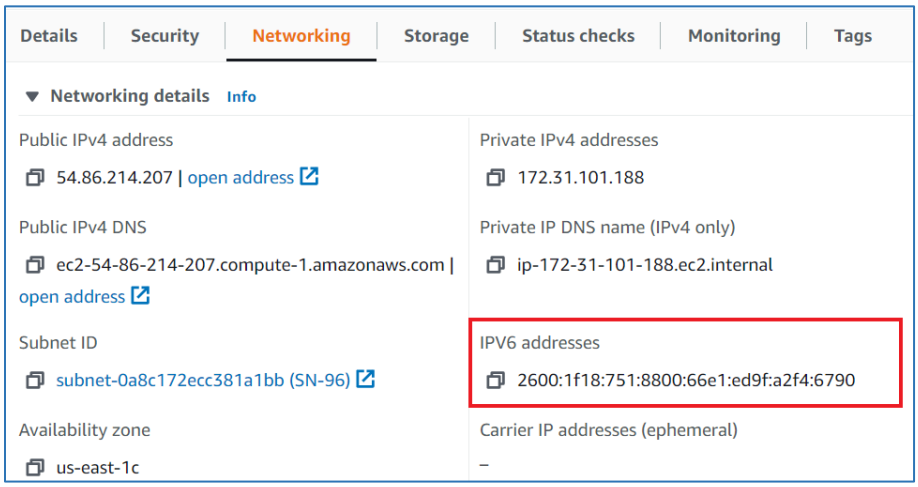
- Edit the Network Settings of the subnet to enable Auto-assign IPv6 so that the EC2 instances obtain a host IPv6 address automatically.



Learning Activity

Deployment of an EC2 instance with dual stack IPv4 and IPv6.

- Create an EC2 instance attached to the subnet that has IPv6 enabled.



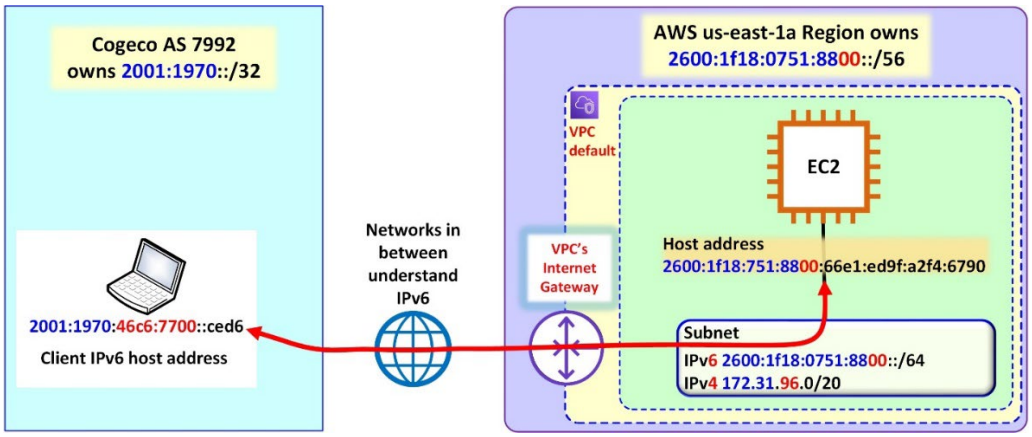
The new EC2 instance obtained three IP addresses, one private IPv4 172.31.101.188/20, another IPv4 public 54.86.214.207, and the IPv6 2600:1f18:0751:8800:66e1:ed9f:a2f4:6790. The final four coefficients are the host specific part assigned by the Auto-Assign IPv6 DHCP service in AWS.

The address is reachable from a computer with an IPv6 address also. (This test was done from a Cogeco address since Sheridan College does not have IPv6 addresses).

```
C:\Users\fc>ssh ec2-user@2600:1f18:751:8800:66e1:ed9f:a2f4:6790 -i C:\Users\fc\demo-key.pem
```

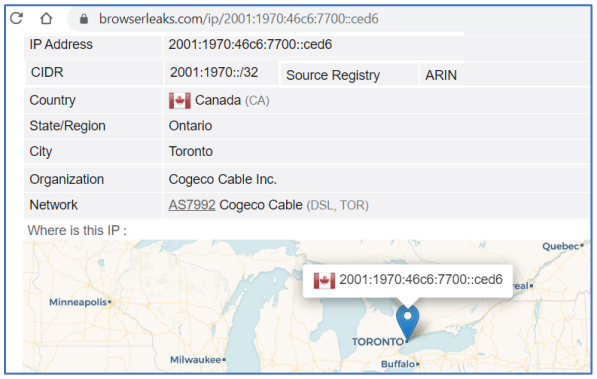
```
The authenticity of host '2600:1f18:751:8800:66e1:ed9f:a2f4:6790 (2600:1f18:751:8800:66e1:ed9f:a2f4:6790)
n't be established.
ECDSA key fingerprint is SHA256:CRaL1+PewF24MVRTzVbgBPq3dw24kLCVZlcDBs/1Wrc.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '2600:1f18:751:8800:66e1:ed9f:a2f4:6790' (ECDSA) to the list of known hosts.

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-101-188 ~]$
```

The following diagram represents the connectivity situation between a client PC served by the Internet Service Provider Cogeco and the EC2 in AWS us-east-1a. Both ends have IPv6 host addresses, so they can talk to each other using IPv6 packets.



Customer PC talks with AWS EC2 instance using IPv6.

The Internet service provider Cogeco has the Autonomous System number 7992 [8] that contains all the IPv6 addresses that begin with 2001:1970. This mask /32 block was allocated to Cogeco by ARIN. From such space, Cogeco does subnetting to allocate to its multiple cable networks. The customers get host addresses from this abundant range.



Cogeco's AS7992 [8] owns 2001:1970::/32

Chapter Conclusion

The Virtual Private Cloud is the canvas where the Cloud Architect deploys the virtual resources. It is a simplified platform with all the basic networking components that support virtual computers. A virtual Internet gateway router controls the access of the IP traffic between the VPC and the Internet. Internally, a default router object defines the interconnection of the local subnets. A route table contains the IP knowledge that enables the forwarding decisions of the IP packets. The default VPC only has the private IPv4 address space 172.31.0.0/16 assigned. Six subnets with mask /20 are obtained from such space in the case of the us-east-1a region. Beside the private IPv4 space, AWS allocates public IPv4 address from its address pool to the instances if that is selected by the customer.

AWS owns IPv6 address blocks that divides to allocate to its regions. From these blocks, AWS provides IPv6 address space to its customers. Each customer can receive spaces with /56 mask to make 256 IPv6 subnets. The virtual computers receive host IPv6 addresses from such subnets.

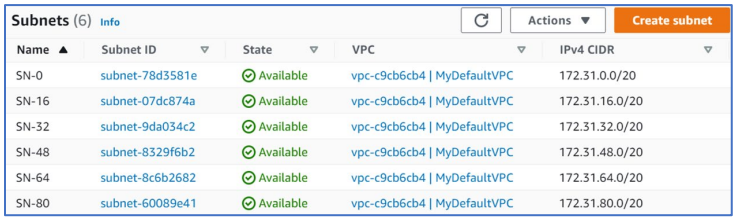
Chapter 6 Coursework

Lab AWS Creation of VPC Subnets

The objective of this lab is to create new subnets, route tables, network access control lists (NACL) and to programmatically retrieve information about these resources.

Description

- The default VPC has these subnets already in the address space 172.31.0.0/16:



Subnets (6) Info — Actions — Create subnet

Name	Subnet ID	State	VPC	IPv4 CIDR
SN-0	subnet-78d3581e	Available	vpc-c9cb6cb4 | MyDefaultVPC	172.31.0.0/20
SN-16	subnet-07dc874a	Available	vpc-c9cb6cb4 | MyDefaultVPC	172.31.16.0/20
SN-32	subnet-9da034c2	Available	vpc-c9cb6cb4 | MyDefaultVPC	172.31.32.0/20
SN-48	subnet-8329f6b2	Available	vpc-c9cb6cb4 | MyDefaultVPC	172.31.48.0/20
SN-64	subnet-8c6b2682	Available	vpc-c9cb6cb4 | MyDefaultVPC	172.31.64.0/20
SN-80	subnet-60089e41	Available	vpc-c9cb6cb4 | MyDefaultVPC	172.31.80.0/20

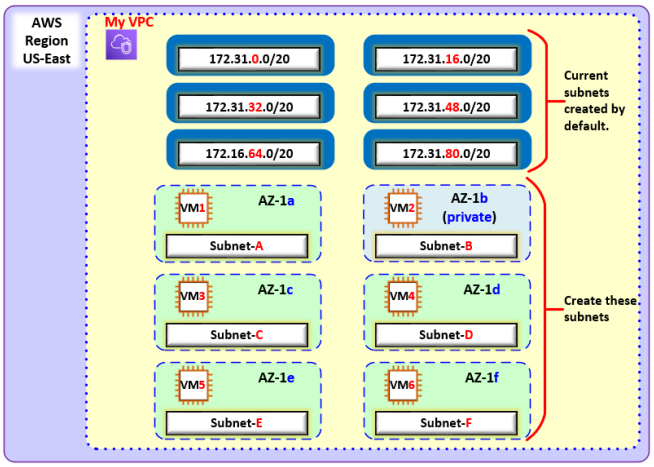
- The address space 172.31.0.0/16 still have room to create more subnets.

TASK: Create six new subnets

- Create **six subnets within the remaining space of 172.31.0.0/16** as specified in this table:

Subnet name	Number of IPv4 addresses	Availability Zone
SN-A	2048	us-east-1a
SN-B (private subnet)	2048	us-east-1b
SN-C	4096	us-east-1c
SN-D	8192	us-east-1d
SN-E	8192	us-east-1e
SN-F	16,384	us-east-1f

- For example, subnet A (SN-A) will have 2,048 addresses in total including the subnet address, the broadcast address, and the reserved addresses and it will be created in the us-east region availability zone 1a. Follow the table instructions to create your subnets.
- This must be the VPC's topology at the end of this lab:



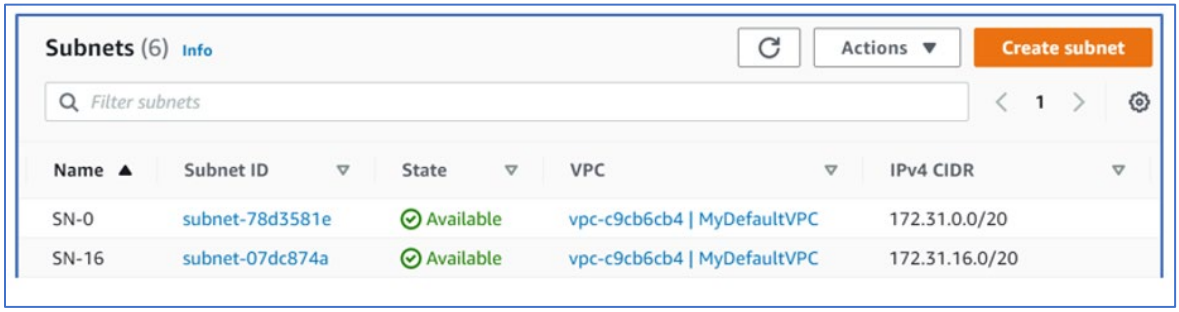
- The **subnets** must be deployed in different **availability zones**.
- Subnet SN-B is a private subnet. That means it does not have access to the Internet. Solve this.

SUBMIT:

- The snippets (follow each checkmark) from the AWS Subnet tab clearly showing:

Activity	Marks	
1		The **VPC** unique **id**entification.
2		Each **subnet.**
3		The **subnet Names**.
4		The **IPV4 CIDR** of each subnet.
5		The **number of addresses** available of each subnet.
6		The **availability zone** of each subnet.
7		**Auto-assign public IPv4** address condition of each subnet.

- For example, submit snippets like the one shown below (this is only part of the snippet, make sure to make enough snippets to include all the requirements above)



- Write a Boto3 Python script that lists the subnets ids, the az and the cidr of each subnet.
- Submit the result of running the Python script.

References

[1] AWS. Amazon Virtual Private Cloud (Amazon VPC). August 2022. [Online]. Available. https://aws.amazon.com/vpc/?vpc-blogs.sort-by=item.additionalFields.createdDate&vpc-blogs.sort-order=desc

[2] M. Cotton. L. Vegota. RFC-5735. Special Use IPv4 Addresses. 2010. [Online]. Available. https://www.rfc-editor.org/rfc/rfc5735

[3] AWS Documentation. Amazon Virtual Private Cloud. August 2022. [Online]. Available. https://docs.aws.amazon.com/vpc/latest/userguide/configure-subnets.html

[4] AWS IP address ranges. August 2022. [Online]. Available. https://ip-ranges.amazonaws.com/ip-ranges.json

[5] S. Deering. R. Hinden. RFC-2460. Internet Protocol, version 6 (IPv6). 1998. [Online]. Available. https://www.rfc-editor.org/rfc/rfc2460

[6] American Registry for Internet Numbers. August 2022. [Online]. Available. https://www.arin.net

[7] RIPE Network Coordination Centre. IPv6 Address Allocation and Assignment Policy. August 2022. [Online]. Available. https://www.ripe.net/publications/docs/ripe-738

[8] IPv6 Look up. August 2022. [Online]. Available. https://browserleaks.com/ip/2600:1f18::

[9] The CIDR Report for AS16509. August 2022. [Online]. Available. https://www.cidr-report.org/cgi-bin/as-report?as=AS16509&view=2.0

Chapter 7

VPC Architecture

Description

This section focuses on the architecting and deployment of virtual private clouds. Through practical learning activities, VPCs are created using the AWS GUI console and programmatically with Python boto3. The main components of VPCs are configured including allocating the IPv4 address space, creating public and private subnets, and configuring gateways for Internet access. Access across different VPC is enabled with VPC peering.

Learning Outcomes

- Design and deploy virtual private clouds.
- Allocate IPv4 address space to create subnets.
- Configure public Internet access.
- Configure private subnet with NAT gateway access.
- Interconnected VPCs via peering.

Main concepts

- Virtual Private Cloud (VPC)
- Main Route Table.
- Internet Gateway.
- Public subnet route table.
- Private subnet route table.
- Access from private subnet to the Internet.
- The NAT gateway.
- VPC peering.

Learning Activities

- Learning Activity
- Creation of a virtual private cloud
- Configuring external access for the private subnet via NAT gateway.
- Creating a VPC programmatically with AWS python boto3.
- Configuration of two VPCs with VPC peering.

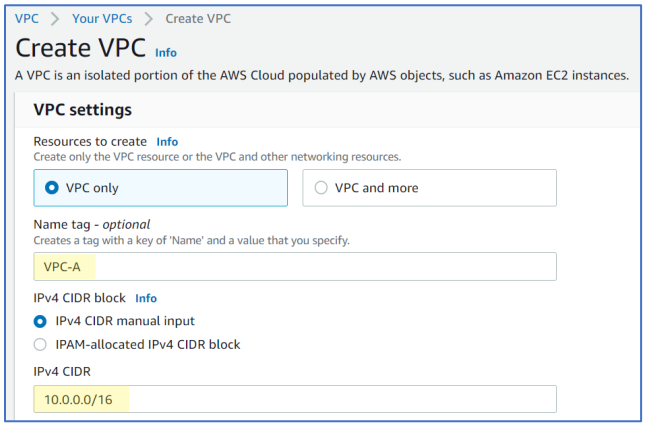
Designing and deploying VPCs

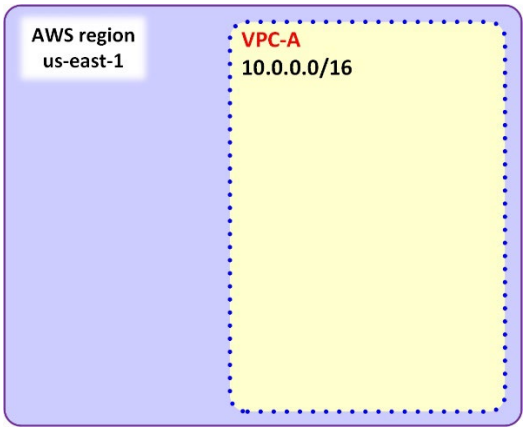
Cloud providers offer a basic Virtual Private Cloud infrastructure by default. This VPC has all the components needed to deploy workload resources in an environment controlled by the user. However, in many use cases, the default VPC is not enough for specific use cases. For example, an organization would like to keep a VPC completely private just for the sake of designing applications. In another case, an organization could design a VPC that offers services only to clients inside the same cloud provider. Frequently, organizations have multi-cloud interconnections with a mix of On-Premises data centre and different cloud providers. In all these situations, the customer organization would like to deploy VPCs with a particular configuration or purpose. That is the whole point of Cloud Architecture, to design VPC according to the needs of the customers.

Learning Activity

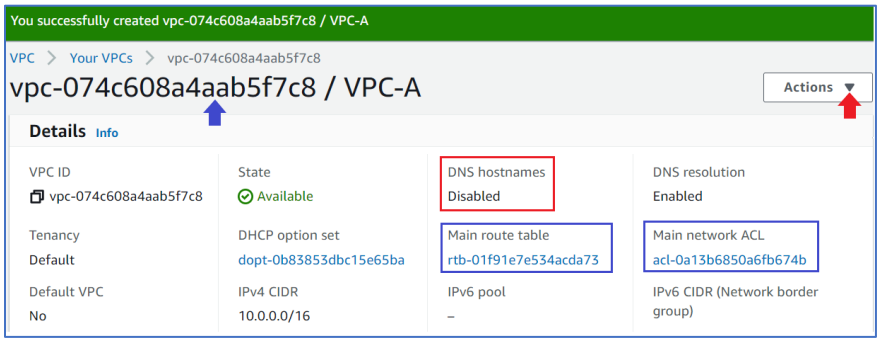
Creation of a Virtual Private Cloud (VPC)

In the following example, a new VPC called VPC-A with the address space 10.0.0.0/16 is being created. The range of IPv4 addresses is typically derived from the private address spaces 10.0.0.0/8, 100.64.0.0/16, or 172.16.0.0/12 (excepting 172.31.0.0/16 which is already allocated to the default VPC). However, the IPv4 address space can also be public which is the case of a customer that already owns the range and wishes to move it into the cloud.

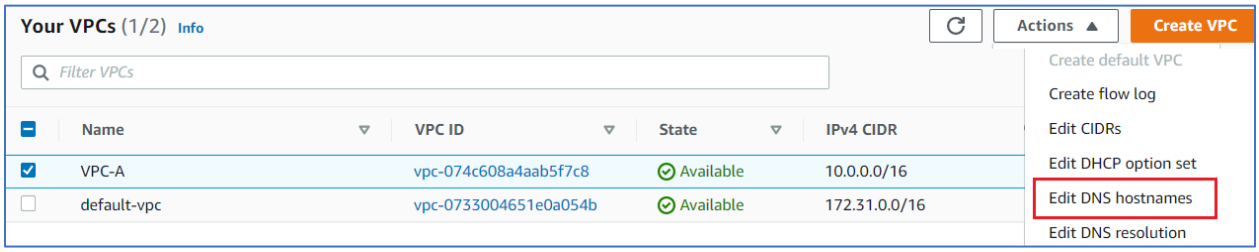




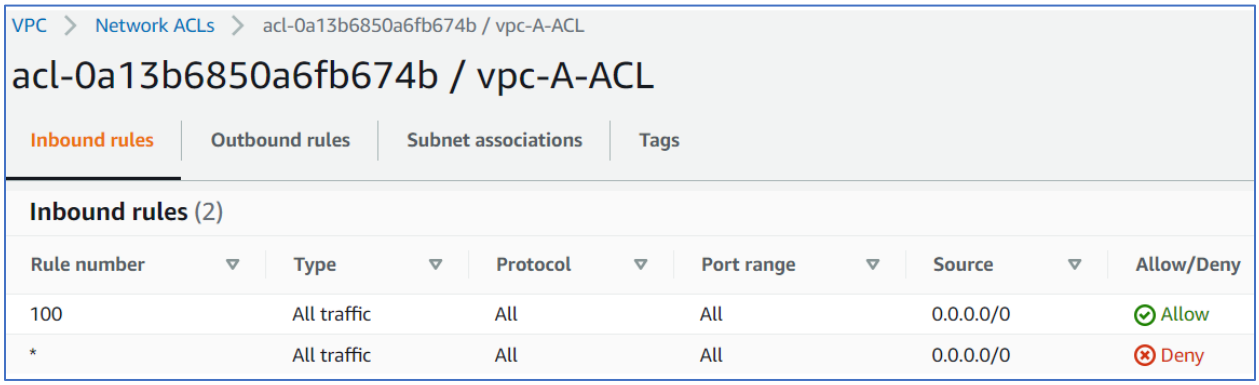
The VPC is created and a unique resource id is automatically issued by AWS. The VPC is associated with a new main routing table and a main network access control list. The DNS service to provide names to the hosts must be enabled manually.

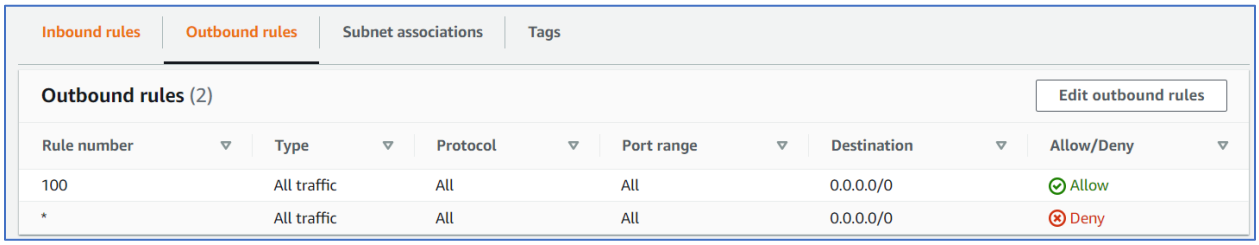


Using the configuration **Actions**, the **DNS hostnames** is activated. When an EC2 instance is created, a DNS A record will be created associating the resource to it.

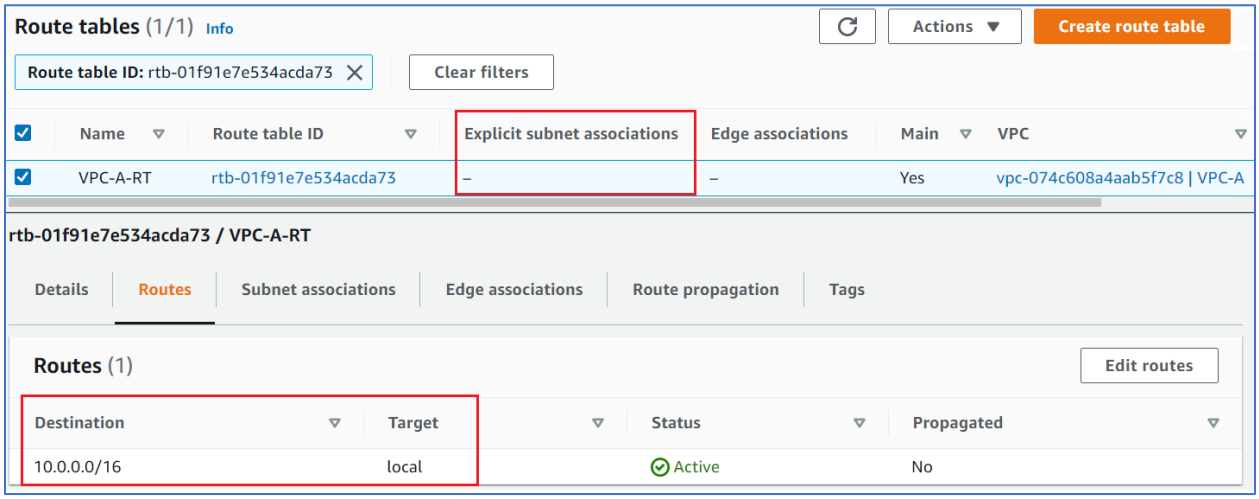


The default Access Control List is open to all traffic in both directions, inbound to the VPC and outbound from it. The next step is then verify whether this VPC has the access to the Internet enabled.

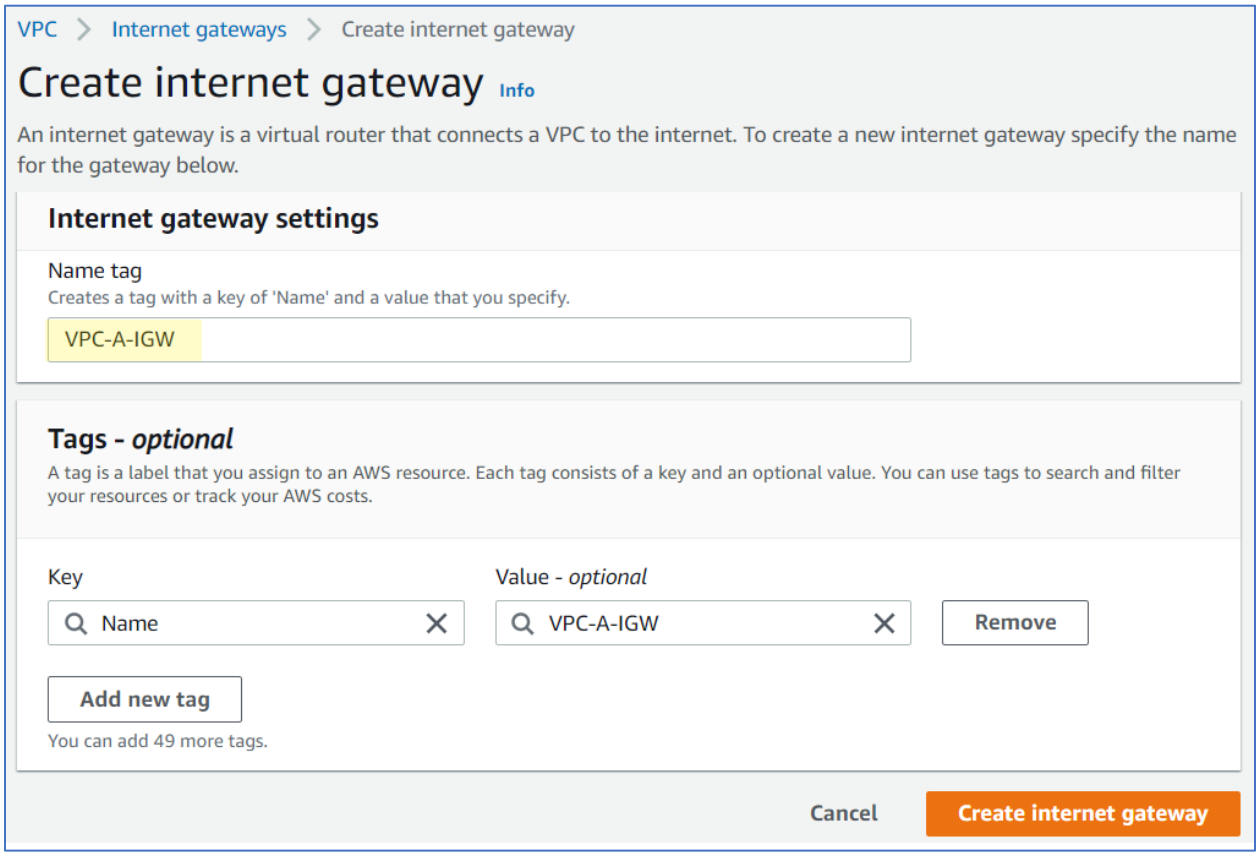




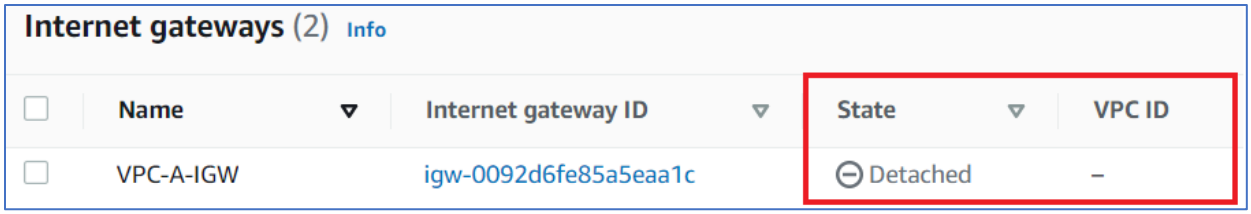
A VPC can not exists without a routing table hence the act of creation includes the setup of a default main route table with only one destination route which is the local 10.0.0.0/16 space. In other words, the only routes which are reachable from within this VPC are in the address space 10.0.0.0/16.



Consequently, to provide Internet access the routing table must have an entry pointing to a default router or Internet Gateway (igw) which does not exist yet. So, the next step is to create such IGW and to associate it to the new VPC.

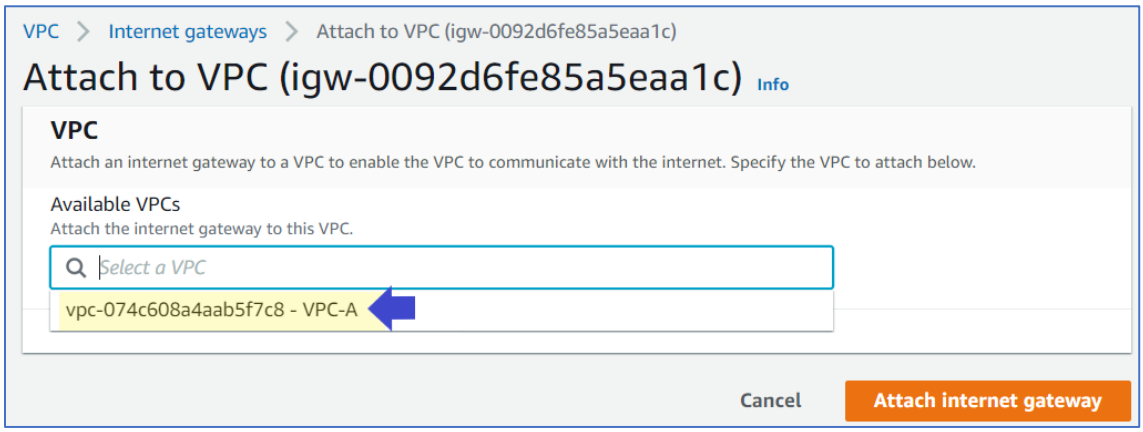


The Internet Gateway has been created, nevertheless it is not attached to any VPC yet.

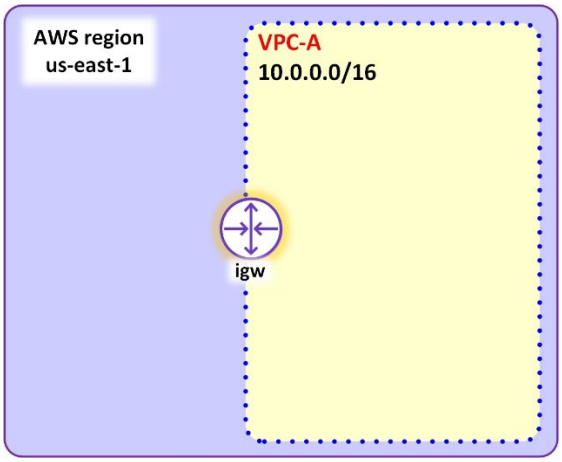


Let's proceed to attach it to the newly created VPC-A.

The IGW is associated with the unique identification of the VPC.

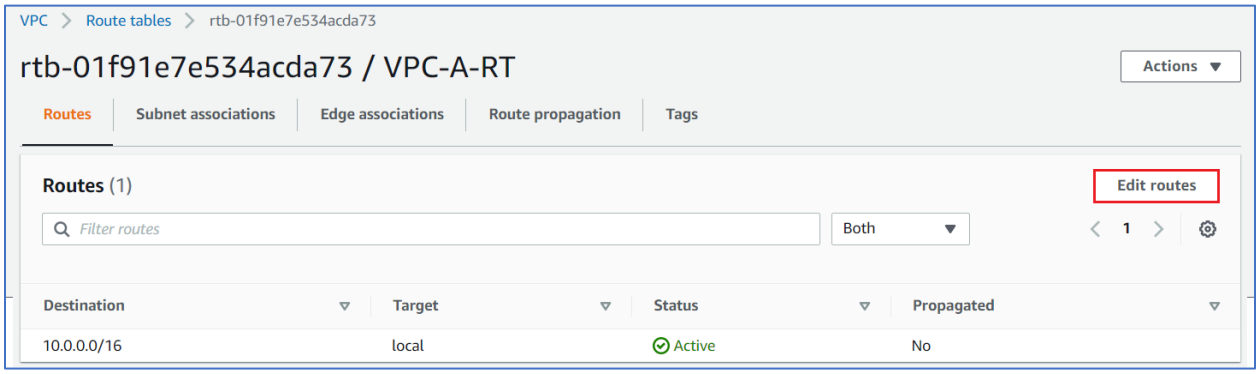


This is the current situation. The VPC has a default router that could provide access to the Internet.

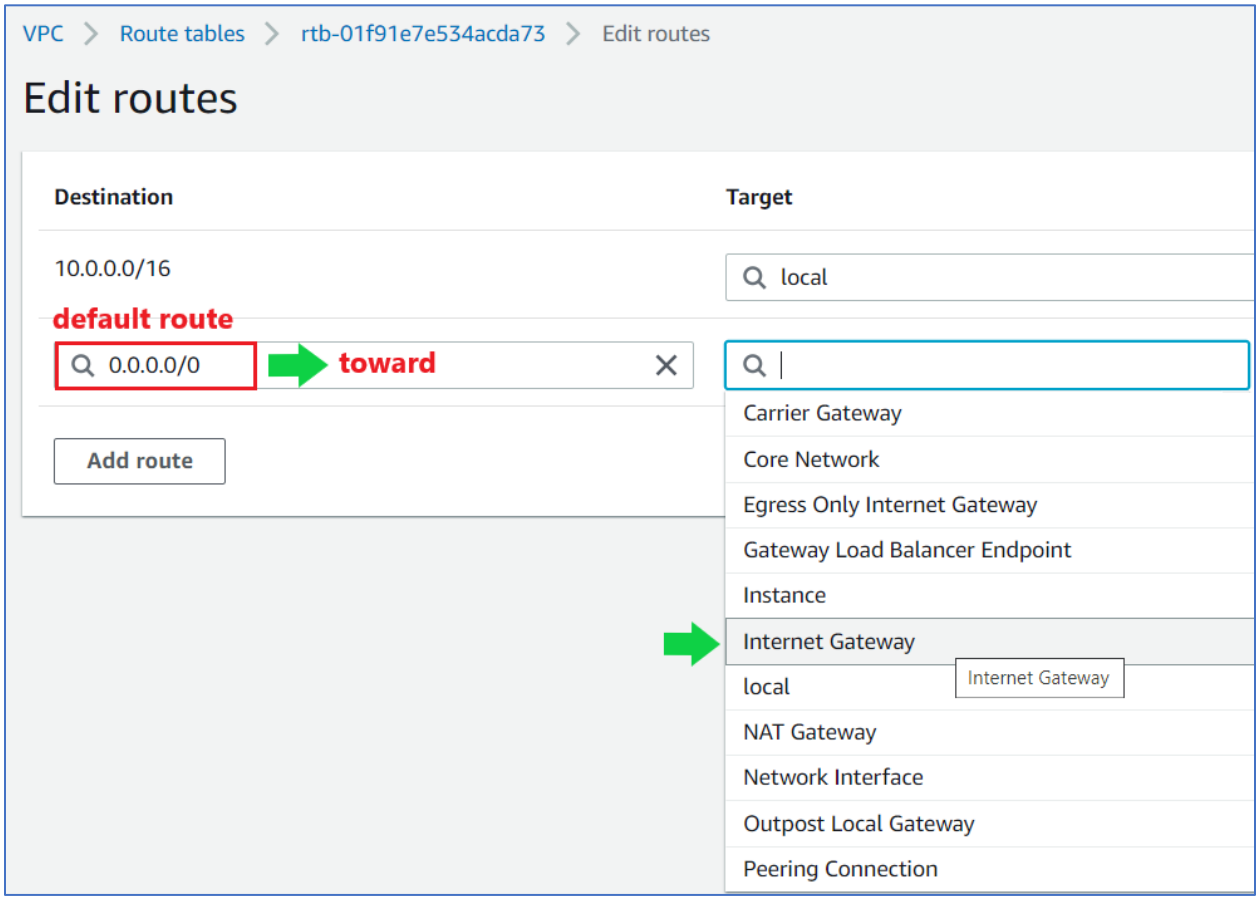


Schematic of the VPC with the Internet Gateway attached.

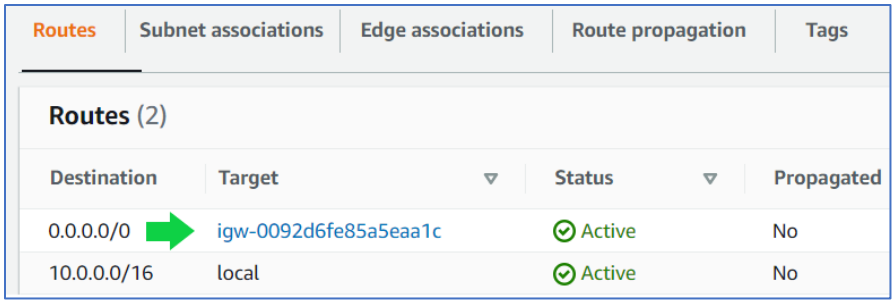
The routing table must be informed of the availability of the Internet Gateway. This is done by editing the Route table of the VPC-A and adding a default route entry pointing to the IGW.



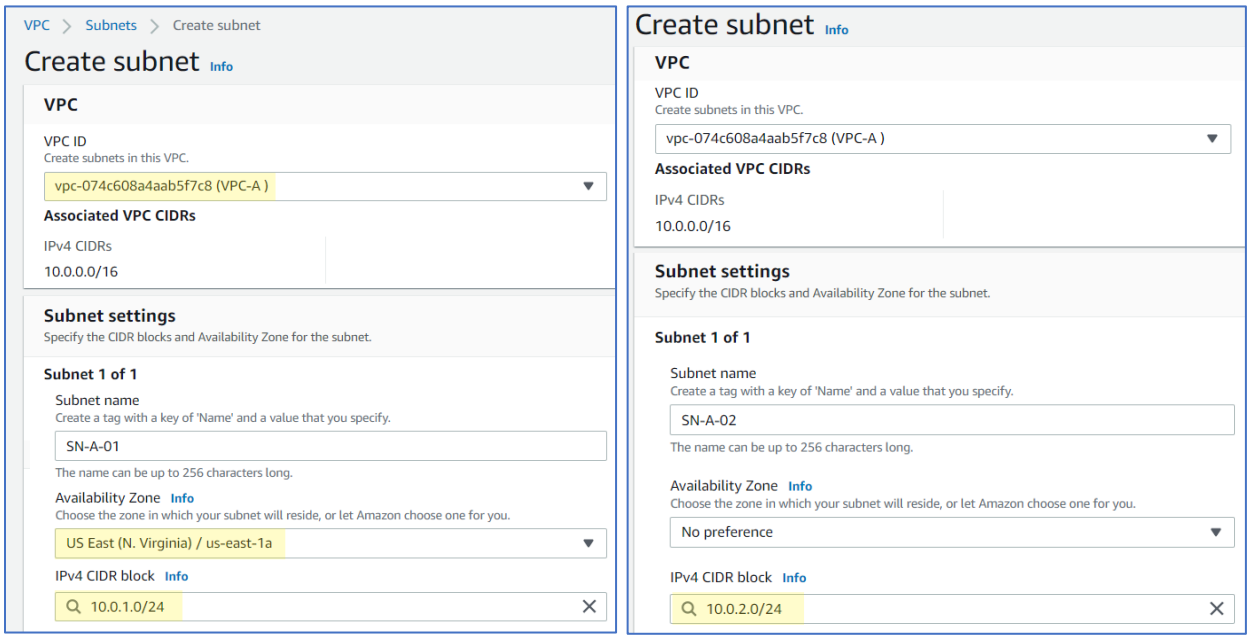
The routing table must contain the knowledge of where to send IP packets. By default, it only has the local route to reach any 10.0.0.0/16 address. Hence, a route to reach the Internet must be added. The route 0.0.0.0/0 is the universal default route. If it is pointed toward the Internet Gateway, all traffic that does not have a destination locally will be sent toward the IGW. The IGW has the knowledge on how to reach the public Internet destinations (although this part is opaque to the cloud customer).

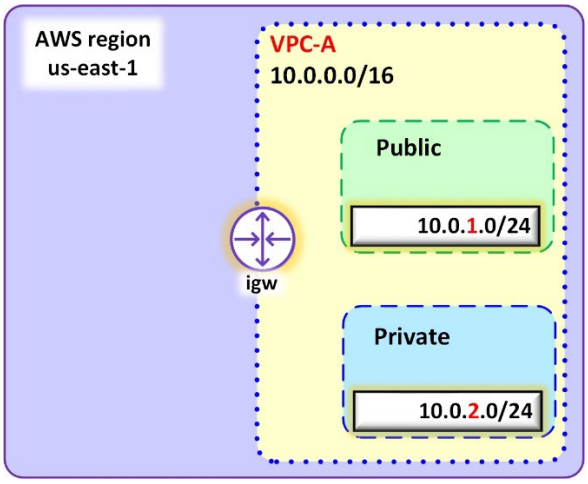


Finally, the VPC has the public access enabled through the Internet Gateway.



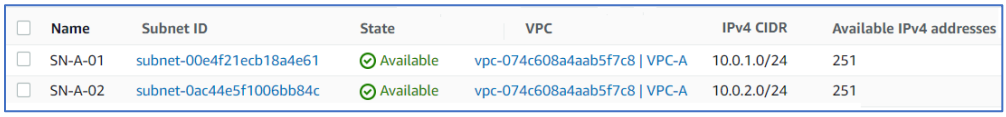
The next step is to architect the internal configuration of the VPC. Two subnets are to be created, one with Internet access and the other without. The public subnet (SN-A-01) is assigned the IPv4 address block 10.0.1.0/24 while the private subnet (SN-A-02) is assigned the IPv4 address block 10.0.2.0/24.





VPC with two subnets, one public and one private.

Each subnet has 254 host addresses available since the mask is /24. However, the first four addresses and the last address are reserved for special purposes (0 network address, 1 virtual default gateway, 2 DNS server, 3 reserved and 255 broadcast). That leaves 251 addresses available for 251 virtual resources.

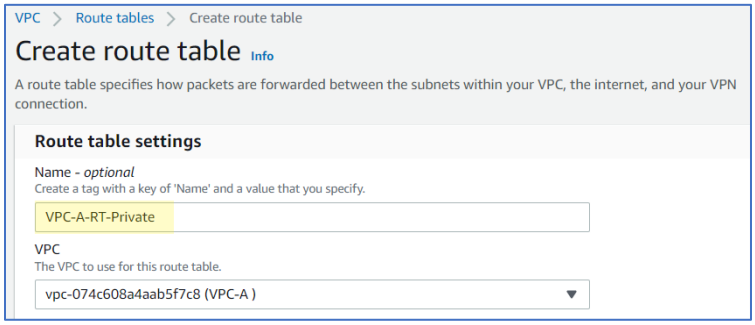


Name	Subnet ID	State	VPC	IPv4 CIDR	Available IPv4 addresses
SN-A-01	subnet-00e4f21ecb18a4e61	Available	vpc-074c608a4aab5f7c8 | VPC-A	10.0.1.0/24	251
SN-A-02	subnet-0ac44e5f1006bb84c	Available	vpc-074c608a4aab5f7c8 | VPC-A	10.0.2.0/24	251

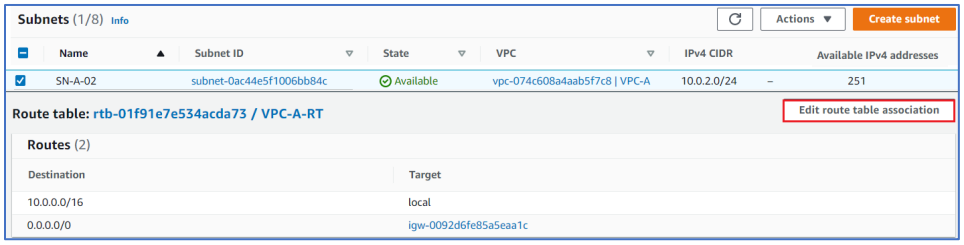
Once that the subnets are created, their routing tables must match the design requirements. When they were created, both subnets were automatically associated to the VPC's default table as shown below.

Name	Subnet ID	Route table
SN-A-01	subnet-00e4f21ecb18a4e61	rtb-01f91e7e534acda73 | VPC-A-RT
SN-A-02	subnet-0ac44e5f1006bb84c	rtb-01f91e7e534acda73 | VPC-A-RT

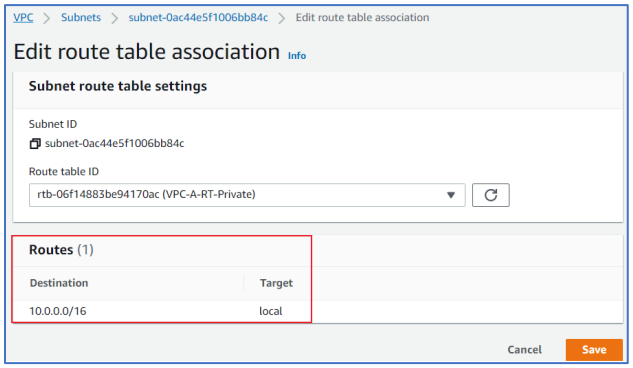
This is not the right match for the private subnet because this table has Internet access via the IGW. Hence, another route table must be created without public access.



Next, the subnet SN-A-02 (private) must be associated to the new table.



The new route table must have a route to the local networks from 10.0.0.0/16.

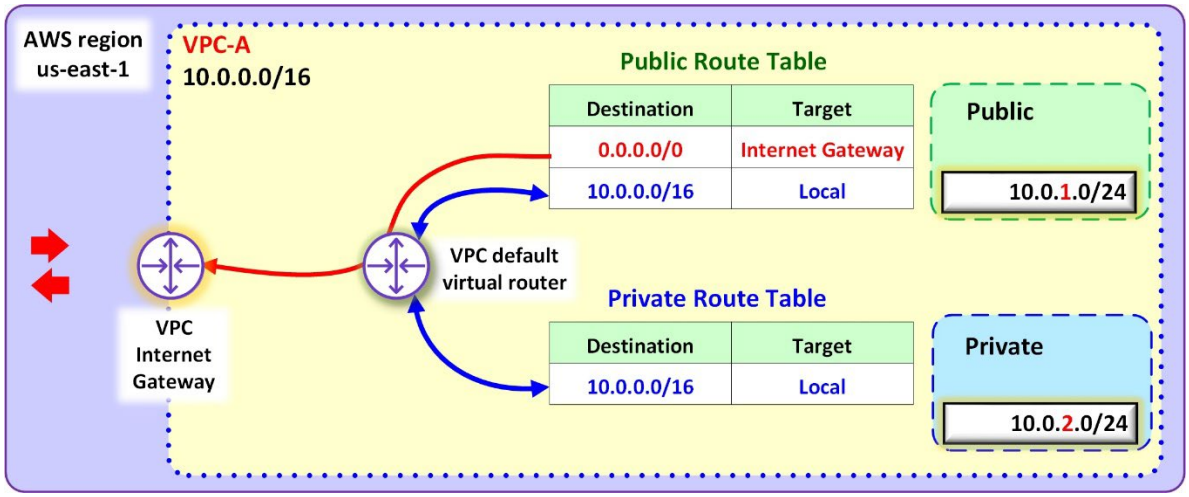


The two subnets are associated now with the corresponding routing tables.

Subnets (8) Info

Name	Subnet ID	VPC	IPv4 CIDR	Route table
SN-A-01	subnet-00e4f21ecb18a4e61	vpc-074c608a4aab5f7c8 | VPC-A	10.0.1.0/24	rtb-01f91e7e534acda73 | VPC-A-RT-Public
SN-A-02	subnet-0ac44e5f1006bb84c	vpc-074c608a4aab5f7c8 | VPC-A	10.0.2.0/24	rtb-06f14883be94170ac | VPC-A-RT-Private

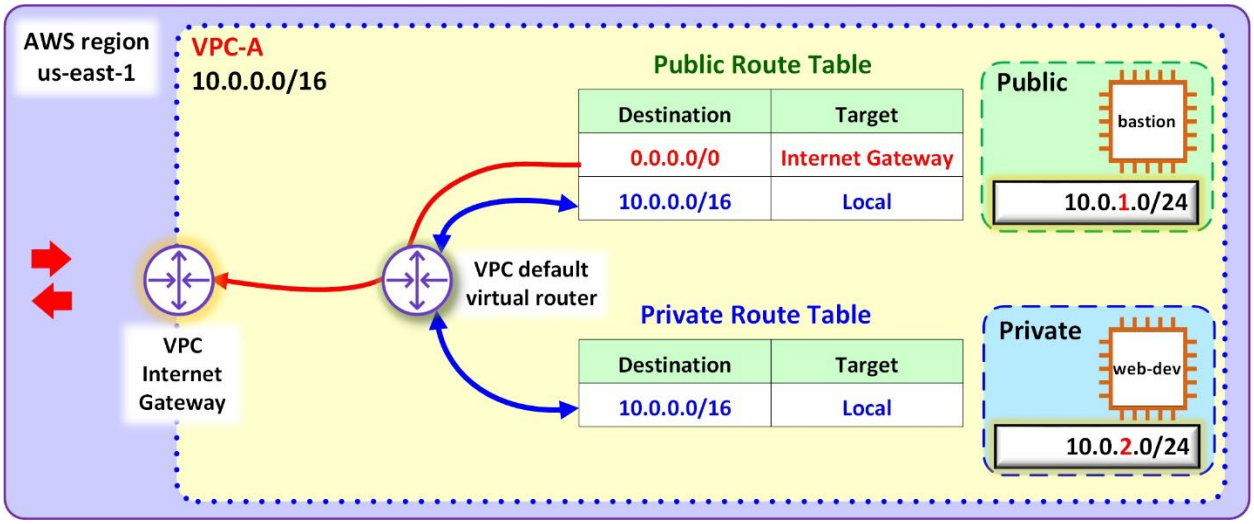
The resources deployed in the public and private subnets can intercommunicate through the VPC's default virtual router. The resources deployed in the public subnet will have Internet access via the VPC's IGW.



Representation of the routing interest of the public and private subnets.

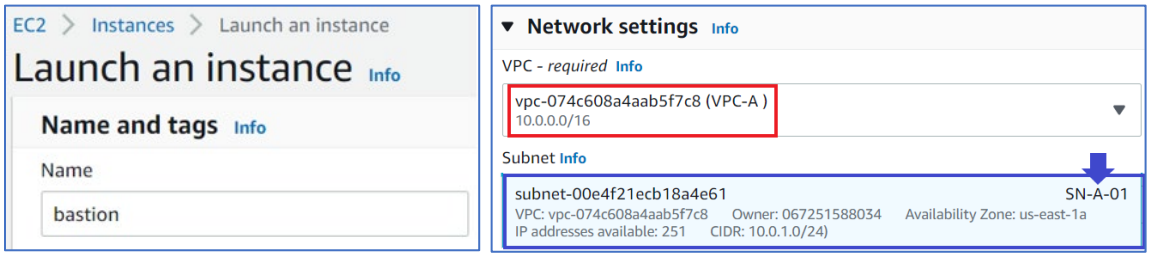
Deployment of EC2 instances in the public and private subnets.

Once that the VPC has been deployed and two subnets has been configured, the next step is to deploy some EC2 resources on each subnet. An EC2 instance named "bastion" is to be deployed in the public subnet and another EC2 instance named "web-dev" is to be launched in the private subnet.

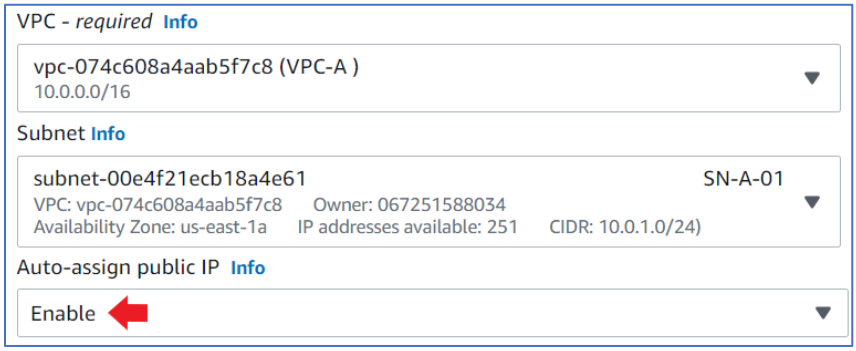


Public and private subnets with EC2 resources.

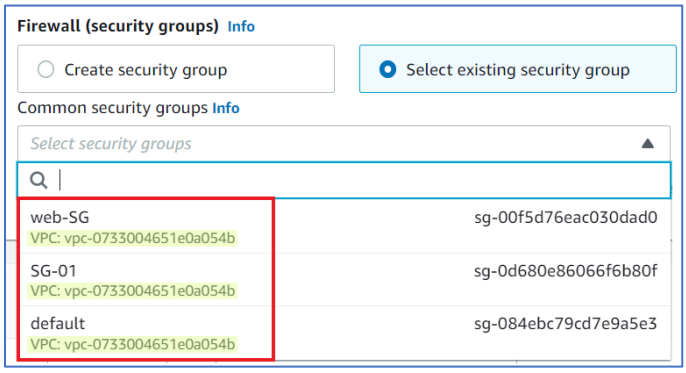
First, create the EC2 instance called bastion in the public subnet to do so, select the VPC-A in the network settings and then choose the public subnet SN-A-1.



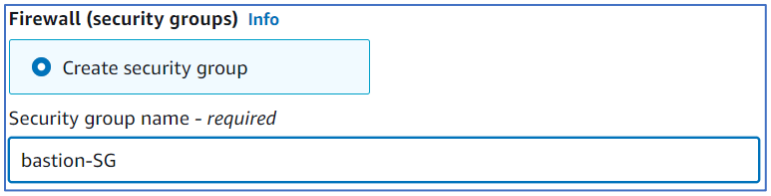
Verify that the Auto-assign public IP address is enabled so that the EC2 instance will get a public IP address.



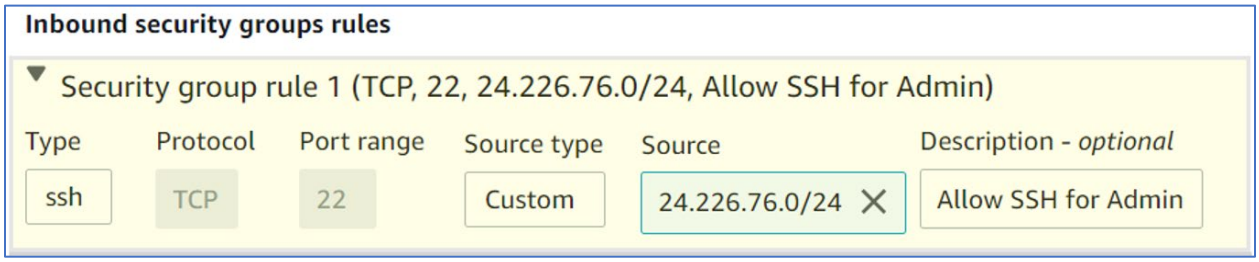
Security groups are VPC specific. Notice that all the existing security groups that appear in the Firewall tab are associated to the default VPC.



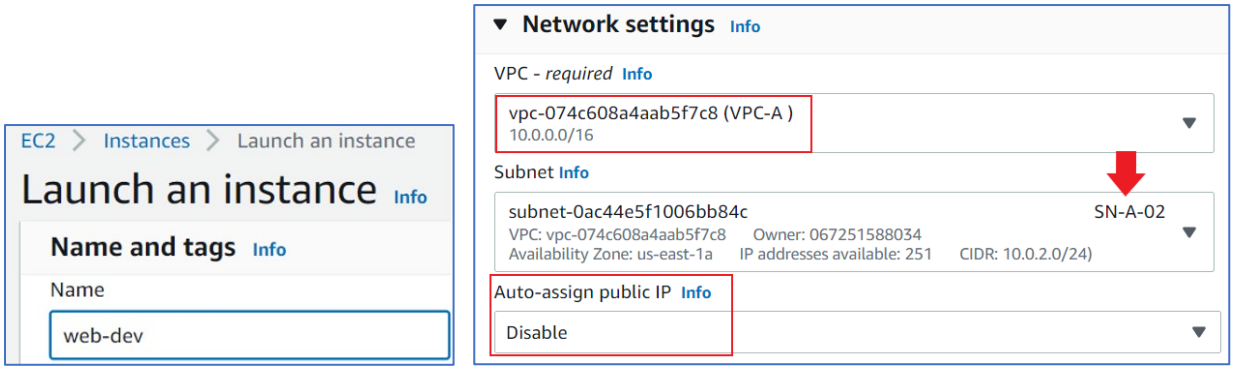
Consequently, a new security group needs to be created in the VPC-A to allow access to the bastion EC2 instance. This security group is meant to be strict by allowing SSH to the bastion instance only from a particular range of addresses belonging to the local Internet Provider.



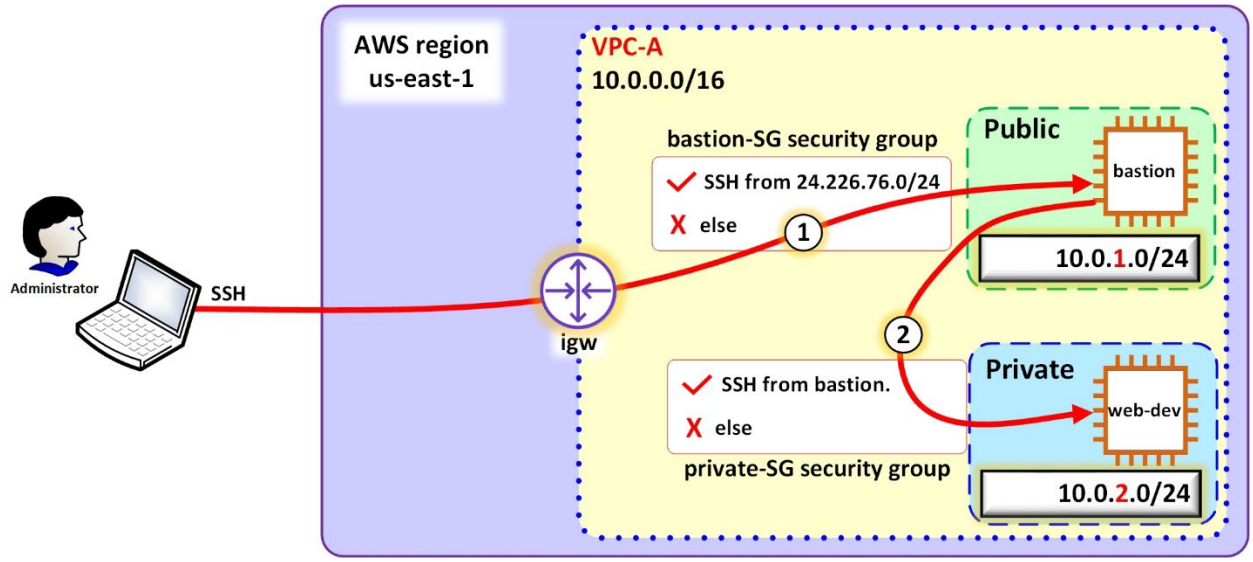
In this example, the address range is 24.226.76.0/24 (belonging to the Internet Provider Cogeco). Thus, the only way to access the EC2 instance will be if the source address is in such range.



After this, the EC2 instance named bastion is completed and launched. The next step is to launch the EC2 instance named "web-dev" in the private subnet SN-A-2. Notice that this EC2 instance will not have a public IPv4 address so the **Auto-assign public IP** option is left in disable mode.

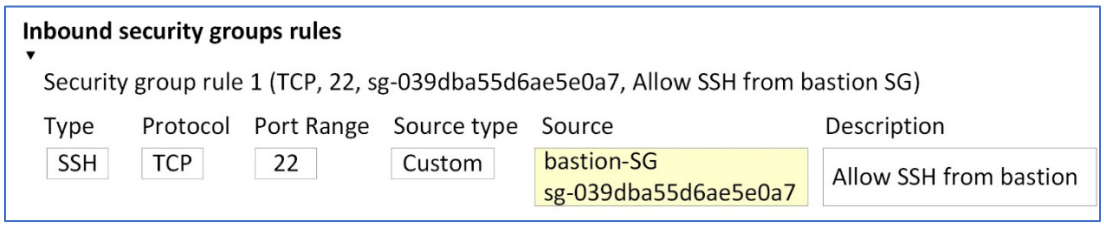


This EC2 instance web-dev also needs to be protected with a firewall, so a new security group is created because it requires a particular setting. The EC2 web-dev in the private subnet should be accessible only from the bastion EC2 instance in the public subnet.

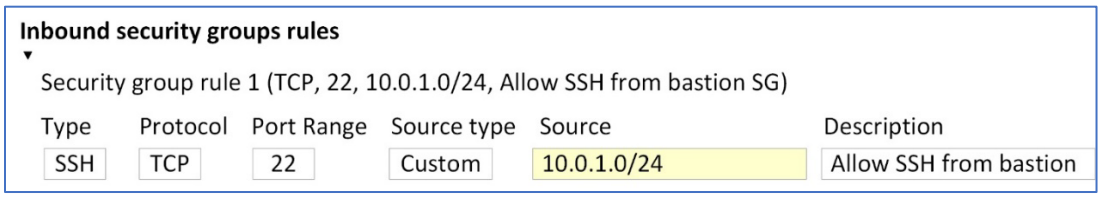


The cloud administrator SSH into the bastion instance. From there, it pivots to the web-dev instance.

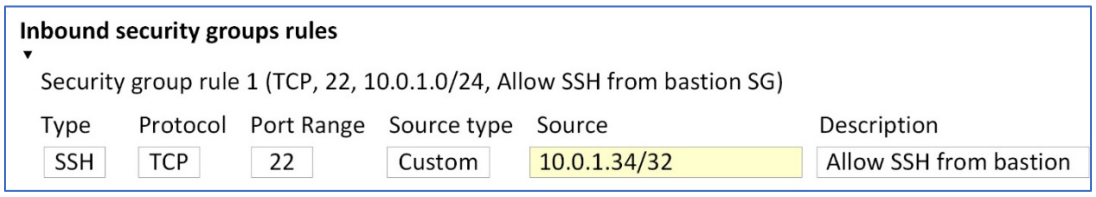
There are three ways to accomplish that, all of them are show next. The first one is to allow the SSH traffic coming from the security group (bastion-SG) that is protecting the bastion instance. Fundamentally, it establishes a trust to whatever is protected by such security group. This option would include any other instance that is associated to the same security group regardless of the subnet. So, maybe that is not the option that the administrator would want. It depends on how strict the rule is desired.



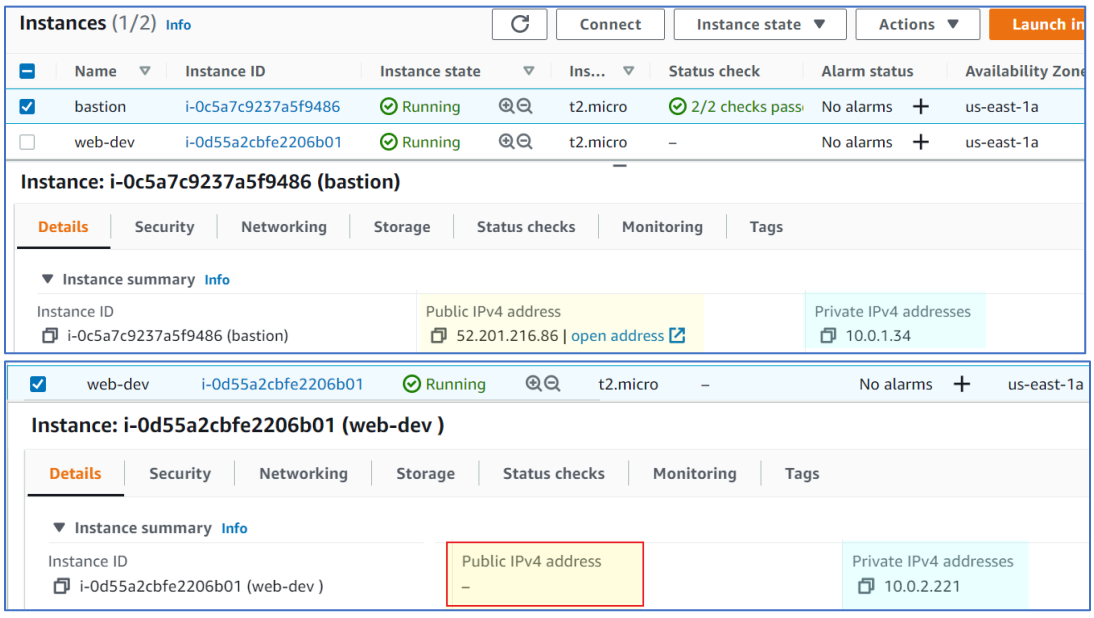
Another option is to allow SSH from any host address located in the public subnet 10.0.1.0/24.



Finally, the most strict option is to allow SSH only from the private IPv4 address of the EC2 bastion. This private IPv4 address was obtained from the EC2 instances service.



To recap, the bastion EC2 has a public address while the web-dev does not as these snippets show.



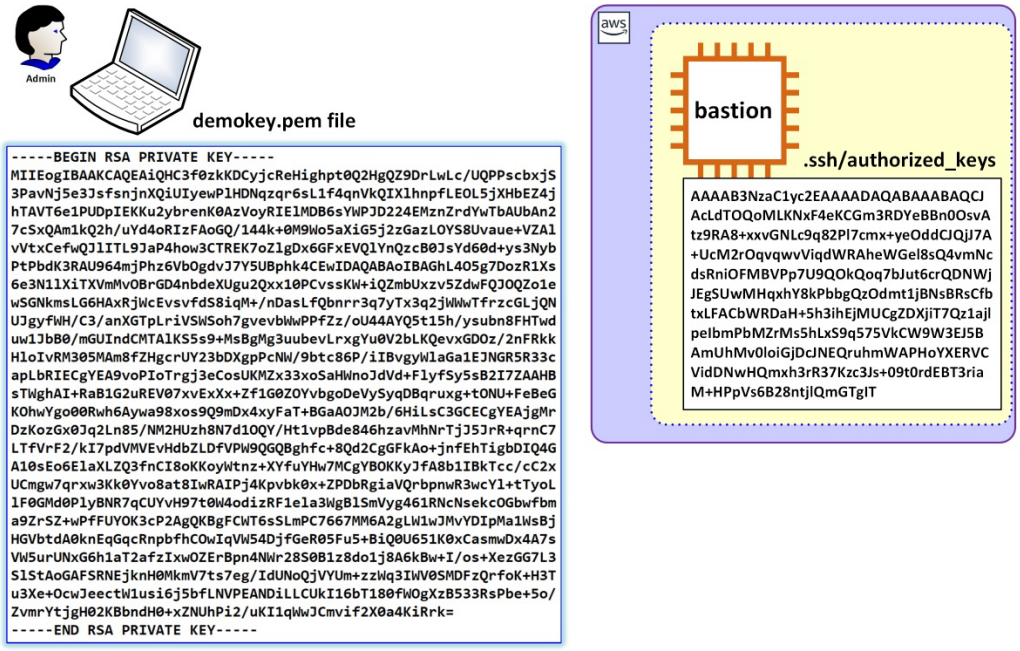
Since the infrastructure is ready, it is time to test the access. First, let's SSH into the bastion EC2 using its public IPv4 address, and from there; SSH to the private address of the web-dev EC2.

```
[ec2-user@ip-10-0-1-34 ~]$ ssh ec2-user@10.0.2.221
The authenticity of host '10.0.2.221 (10.0.2.221)' can't be established.
ECDSA key fingerprint is SHA256:eoDr0BcxrBbAYv8Lh9T594NANxyXigLmvAJU/qW8hTA.
ECDSA key fingerprint is MD5:3d:80:0a:fe:11:bd:6f:83:48:5f:8c:7a:c0:7a:fc:e1.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.221' (ECDSA) to the list of known hosts.
Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
```

The SSH into the web-dev instance failed! The message indicates that the permission is denied. So, what is happening here? A look into the .ssh directory shows the authorized_keys file demokey.

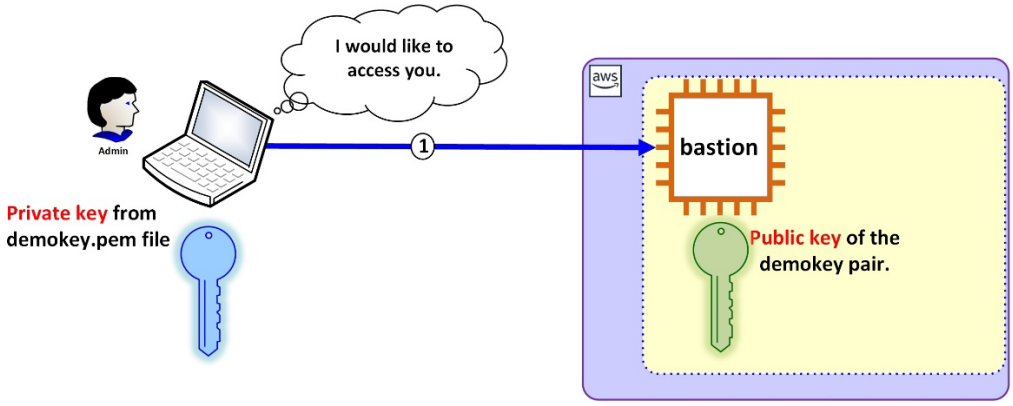
```
[ec2-user@ip-10-0-1-34 ~]$ cd .ssh
[ec2-user@ip-10-0-1-34 .ssh]$ ls
authorized_keys  known_hosts
[ec2-user@ip-10-0-1-34 .ssh]$ cat authorized_keys
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCJAcLd/TOQoMLKNxF4eKCGm3RDYeBBn0OsvAtz9RA8+xxvGNLc9q82Pl7cmx
+yeOddCJQjJ7A+UcM2rOqvqwvV/iqdWRAheWGel8sQ4vmNcdsRniOFMBVPp7U9QOkgQoq7bJut6crQDNWjJEgSUwMH
qxhY8kPbbgQzOdmt1jBNsBRsCfbtxLFACbWRDaH+5h3ihEjMUCgZD/XjiT7Qz1ajlpeIbmPbMZrMs5hLxS9q575VkC
W9W3EJ5/BAmUhMv0lo/iGjDcJNEQruhmWAPHoYXERVCVidDNwHQmxh3rR37Kzc3Js+09t0rdEBT3riaM+HPpVs6B28
ntjlQGmGTgIT demokey
```

A look into the PEM file in the customer laptop shows that these files are different. The demokey PEM file in the laptop contains the private key (plus all the key calculation components to generate both public and private key). On the other hand, the key in the EC2 bastion instance is just the public key.



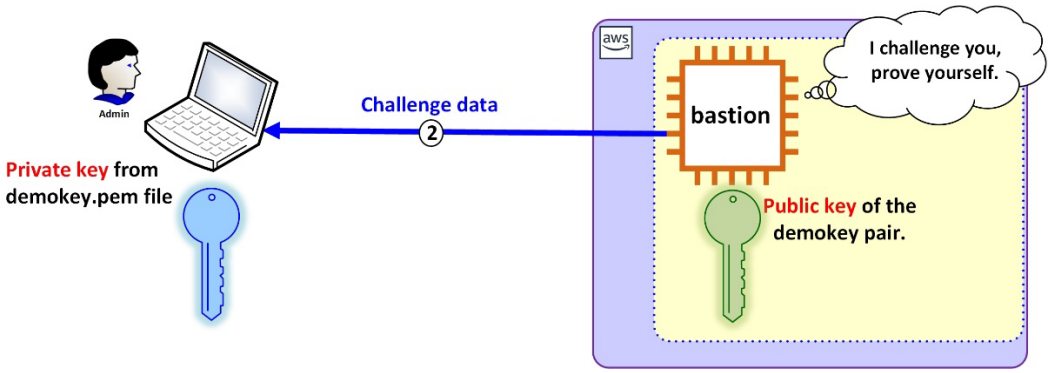
The key PEM file in the administrator laptop versus the public key in the EC2 bastion instance.

A complex conversation ensues when the client tries to gain access via SSH. This dialog has been reduced to a simplified view just to understand the purpose and usage of the public and private key. During the exchange, the SSH client requests access to the EC2 instance (which the SSH server side in this case).



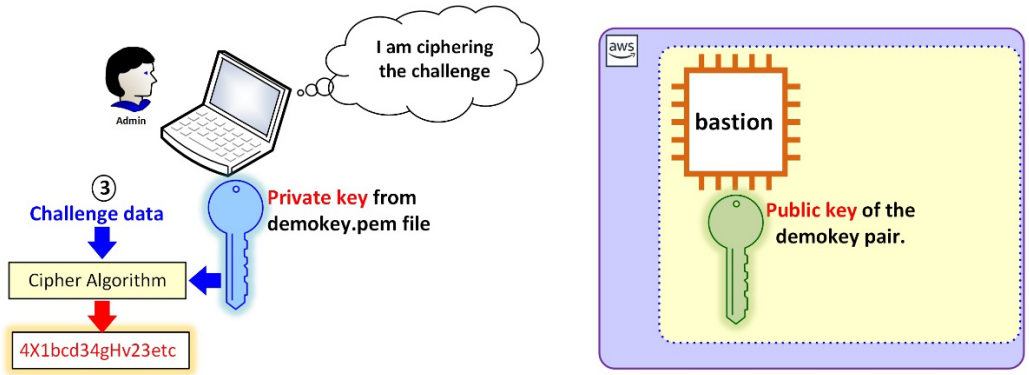
Client requests access to the server side.

But the EC2 just does not agree to that. Instead, it challenges the client to prove its **authenticity** by sending a string of data which is denominated challenge data in the diagram.

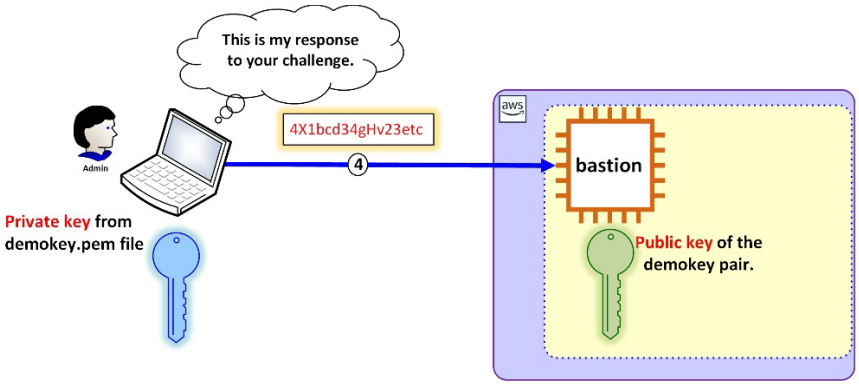


Server challenges the client to demonstrate its authenticity.

The client ciphers the challenge data with its private key obtained from the PEM file. Only the public key of the pair can be used to decipher the ciphertext. Now, who has that public key?

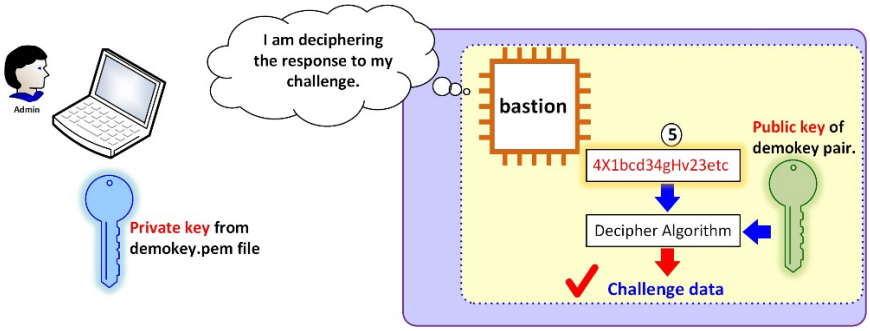


The client ciphers the challenge data with its private key.



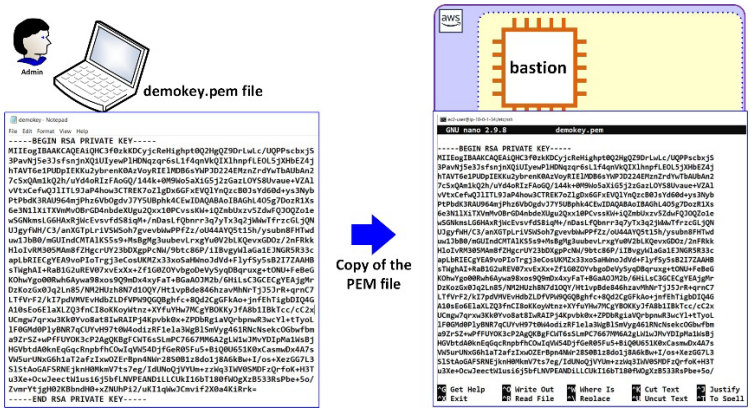
The client returns the ciphertext of the challenge data to the server side.

Finally, the server bastion takes the ciphered response and it passes it through a deciphering algorithm together with the public key. The deciphered data is the challenge data. If that is exactly the same data that was sent originally, then the SSH server concludes that the client must be authentic for only the legit client would be in possession of the private key of the pair of keys. The possession of the private key is assumed as a **proof of authenticity**.



The SSH server side obtains the challenge data back from the ciphertext.

There is more into this process, but this is enough of an overview to gather the general idea. This explains why it was not possible to SSH from the bastion EC2 into the web-dev EC2. The bastion EC2 does not have the PEM file that contains the private key, so the authentication process fails. To solve this problem, the PEM file in the administrator's laptop must be also present in the EC2 bastion.



The PEM file must be also in the bastion EC2 instance.

The PEM can be exported using the SCP tool or simply by copying and pasting onto a text editor (nano or vi) in the destination EC2 instance. In this example, the PEM file was saved in the /etc/ssh directory which is the SSH protocol default directory.

```
[ec2-user@ip-10-0-1-34 ssh]$ pwd
/etc/ssh
[ec2-user@ip-10-0-1-34 ssh]$ ls
demokey.pem  ssh_config   ssh_host_ecdsa_key  ssh_host_ed25519_key
ssh_host_rsa_key   moduli sshd_config         ssh_host_ecdsa_key.pub
ssh_host_ed25519_key.pub  ssh_host_rsa_key.pub
```

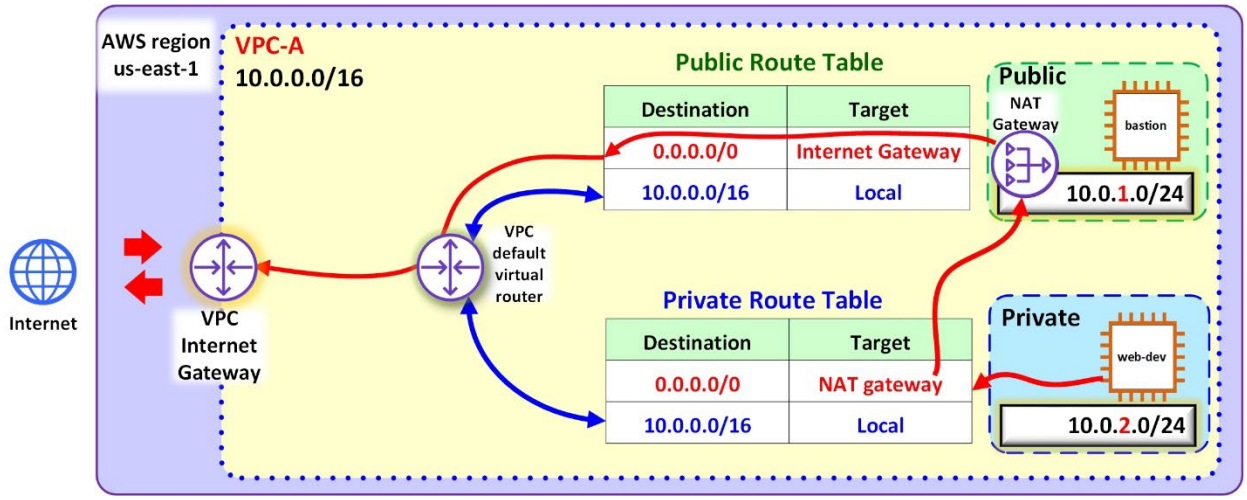
Once this is completed, the bastion machine can SSH into the web-dev EC2.

```
[ec2-user@ip-10-0-1-34 ssh]$ # This is the bastion EC2 instance.
[ec2-user@ip-10-0-1-34 ssh]$ ssh ec2-user@10.0.2.221 -i demokey.pem
       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|
https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-10-0-2-221 ~]$ # This is the web-dev EC2 instance.
```

Learning Activity

Configuring external access for the private subnet via NAT gateway

Up to this point, the web-dev virtual machine has no access to the Internet whatsoever just as intended. However, the necessity arises that the web-dev machine needs to access the repositories of the operating system to update and patch its software. Thus, the question is how to allow access from the web-dev EC2 to the OS repositories located on the Internet without changing the private character of the subnet. The solution is a **NAT Gateway**. The following schematic shows the logic of a NAT gateway configuration.



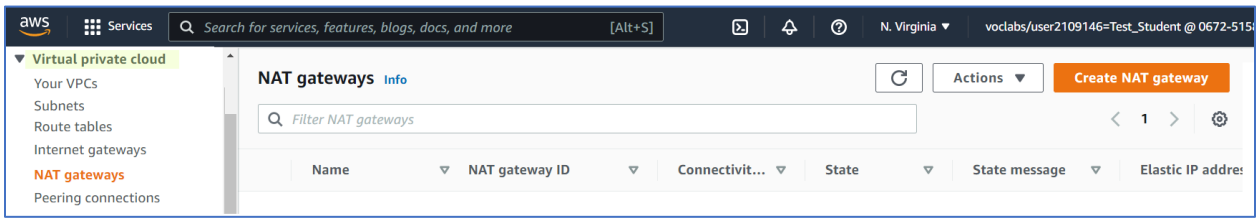
NAT Gateway deployment in a public subnet allowing traffic to the Internet from the private subnet.

A NAT gateway resource allows the IPv4 traffic originated from the private subnet to a destination on the Internet and the responses to that traffic as well. That means that traffic originated from the Internet can

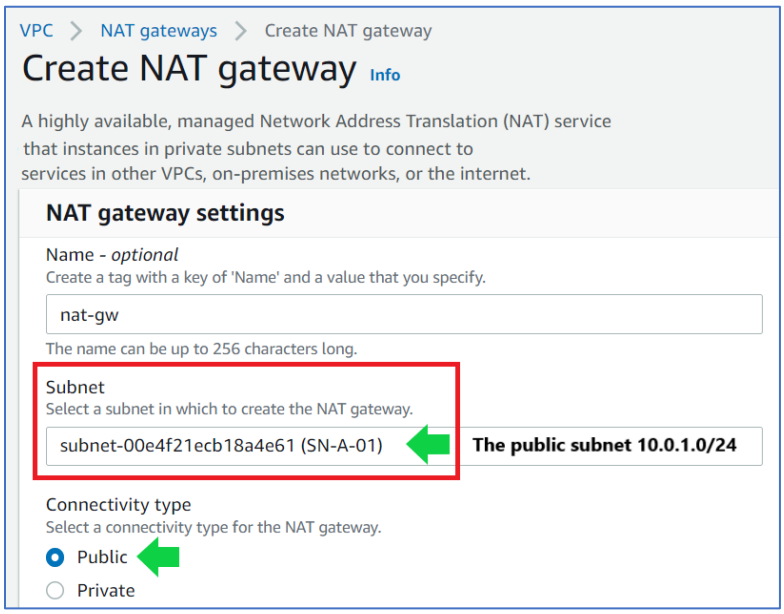
not access the web-dev EC2 instance unless it SSH coming through the bastion EC2 instance. Therefore, the private character of web-dev instance is preserved while allowing software updates.

Create a NAT Gateway.

Proceed to create a NAT gateway in the VPC service.

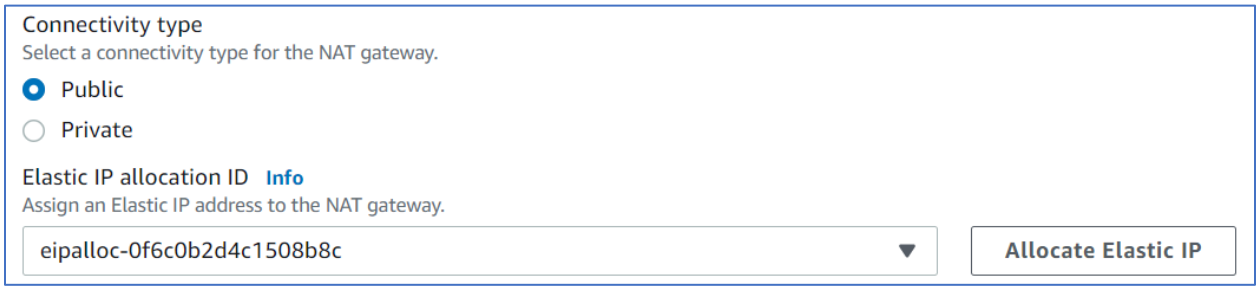


Bind the NAT gateway to the public subnet SN-A-01 as shown below.

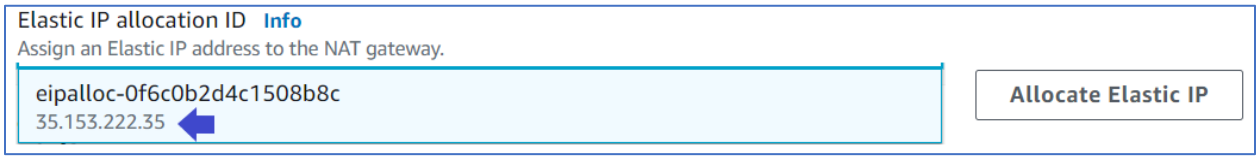


The NAT gateway requires a public IPv4 address to gain access to the Internet. This public IPv4 address comes from the pool of addresses that belong to AWS and it must be allocated permanently to the NAT gateway as long as it exists. If the NAT gateway is deleted, the address must be returned to the AWS common pool of public addresses.

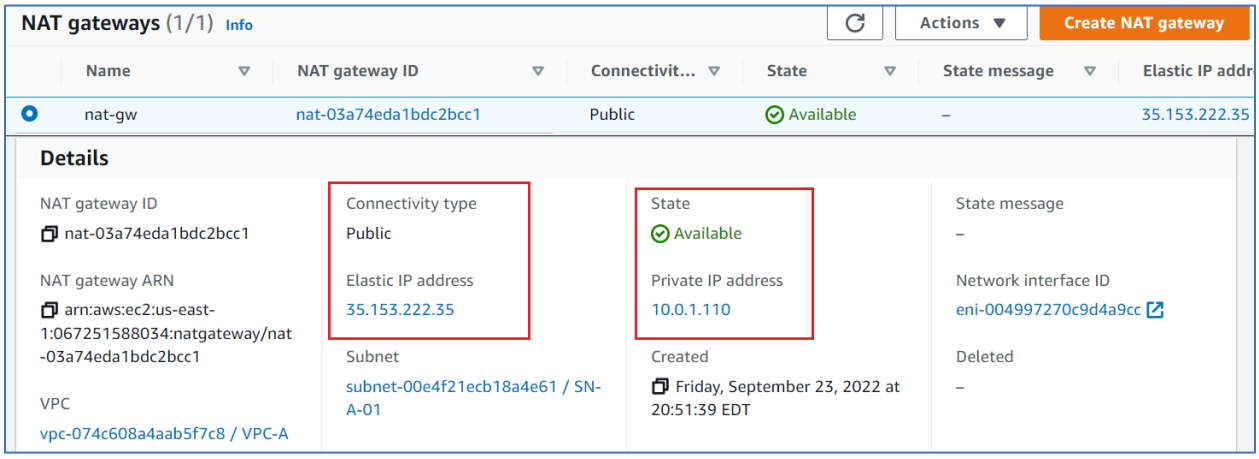
An **Elastic IP Address (EIP)** is a permanently allocated public address. AWS charges a fee for using an EIP since IPv4 public addresses are scarce assets.



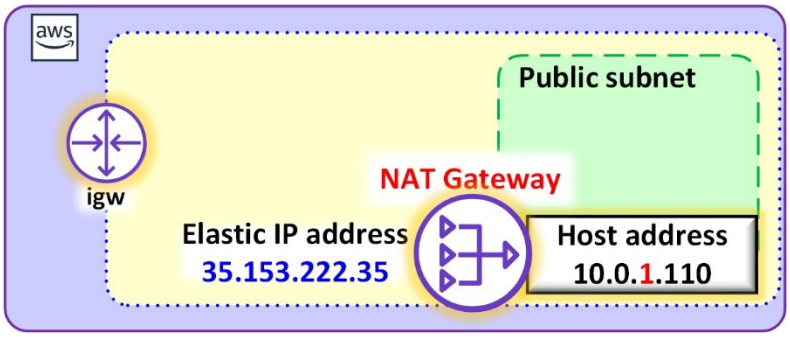
In this example, the public address 35.153.222.35 was the EIP assigned.



Finally, the NAT gateway gets deployed and it picks two addresses, the public address exposed to the outside and the local IPv4 address from the public subnet SN-A-01 where it is attached.

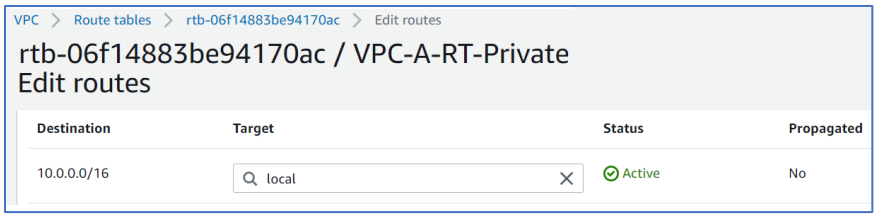


The following diagram represents the previous information.

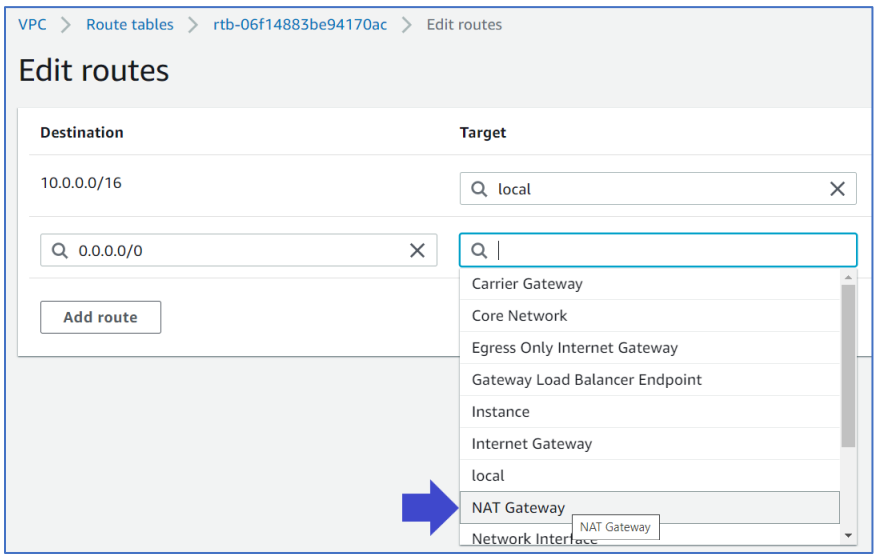


The NAT gateway has one external interface and one internal interface.

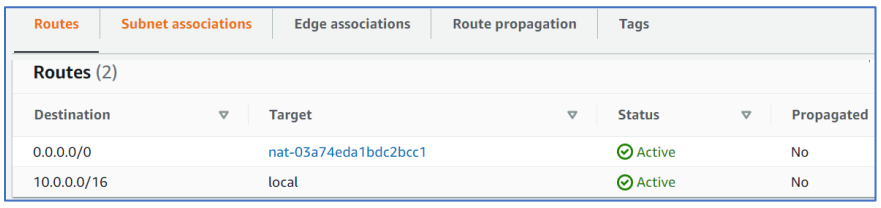
Currently, the private subnet routing table (VPC-A-RT-Private) has no knowledge about the NAT gateway.

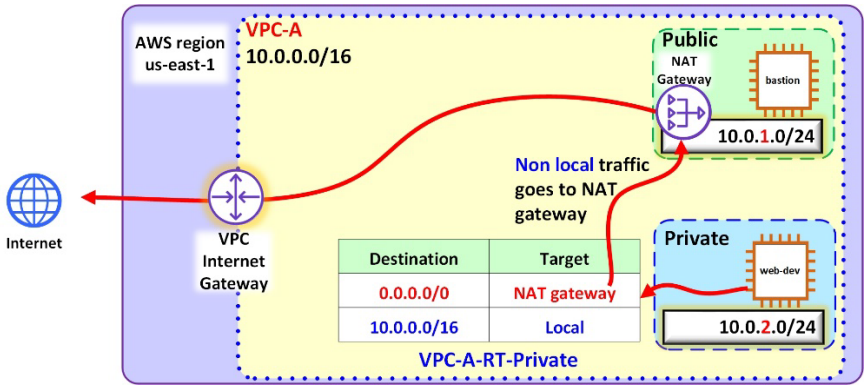


Hence, a default route 0.0.0.0/0 must be added to the route table pointing to the NAT Gateway.



Resulting in this routing table.





The VPC-A-RT route table has a default route pointing toward the NAT gateway.

All that is left is to test the access to the Internet by pinging Google's primary DNS server address 8.8.8.8. from the web-dev EC2 instance.

```
# If this ping gets a reply back, it means that there is Internet access.
[ec2-user@ip-10-0-2-221 ~]$ # This is the web-dev EC2 instance.
[ec2-user@ip-10-0-2-221 ~]$ ping 8.8.8.8
64 bytes from 8.8.8.8: icmp_seq=1 ttl=106 time=2.23 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=106 time=0.874 ms
```

It works! Next, let's try to download and install an httpd webserver.

```
[ec2-user@ip-10-0-2-221 html]$ sudo su
[ec2-user@ip-10-0-2-221 html]$ yum install httpd -y
[root@ip-10-0-2-221 html]# pwd
/var/www/html
[root@ip-10-0-2-221 html]# echo "<html><body><h1> Hello from $(hostname -f)
</html></body></h1>" > index.html
[root@ip-10-0-2-221 html]# sudo systemctl restart httpd
[root@ip-10-0-2-221 html]# systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service)
   Active: active (running) since Sat 2022-09-24 01:20:12 UTC; 8s ago
[root@ip-10-0-2-221 ~]# curl 127.0.0.1
<html><body><h1> Hello from ip-10-0-2-221.ec2.internal </html></body></h1>
[root@ip-10-0-2-221 ~]# exit
```

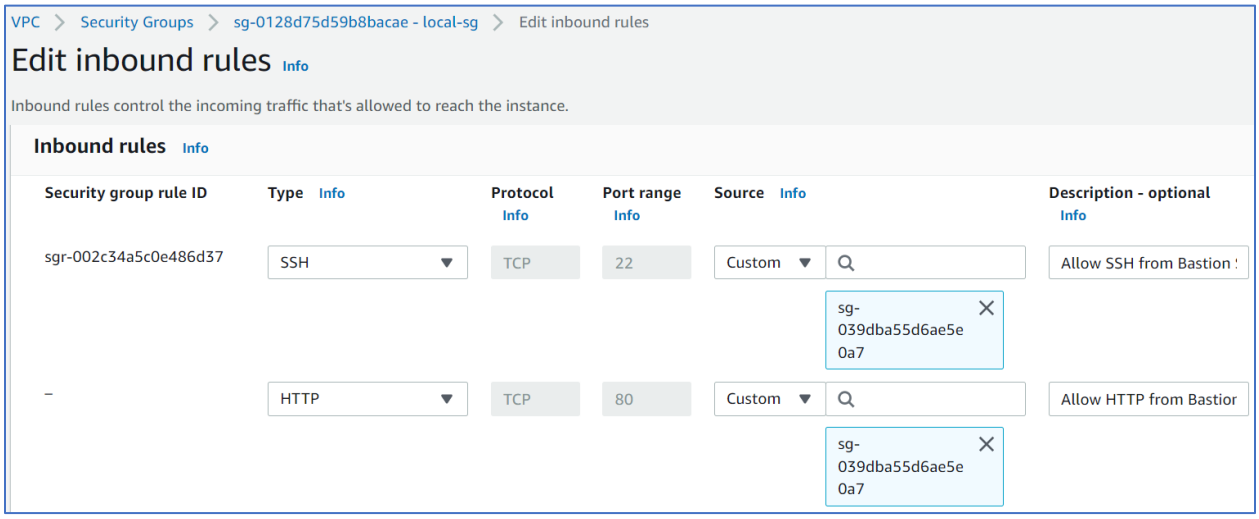
The webserver web-dev is running now. Let's try to browse it from the bastion EC2 instance.

```
[ec2-user@ip-10-0-1-34 ssh]$ hostname
ip-10-0-1-34.ec2.internal # This is the bastion EC2 instance.
[ec2-user@ip-10-0-1-34 ssh]$ curl 10.0.2.221
curl: (28) Failed to connect to 10.0.2.221 port 80 after 130051 ms:
Connection timed out
[ec2-user@ip-10-0-1-34 ssh]$ curl 10.0.2.221
<html><body><h1> Hello from ip-10-0-2-221.ec2.internal </html></body></h1>
[ec2-user@ip-10-0-1-34 ssh]$
```

No reply came back. The reason must be found in the security group.

In effect, there is only one rule that allows SSH only from the security group of the bastion EC2.

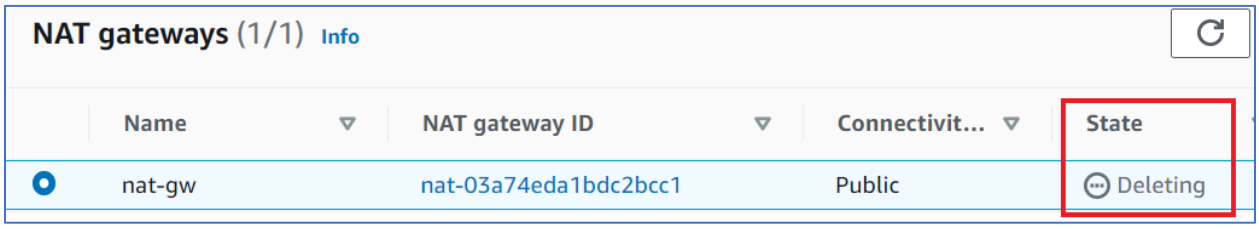
The security group that protects the web-dev EC2 is modified to allows SSH and HTTP coming from the security group that protects the bastion EC2.



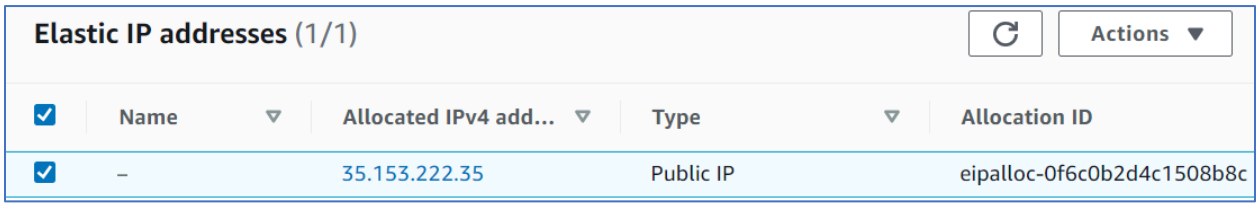
A new test proves that the access is successful now.

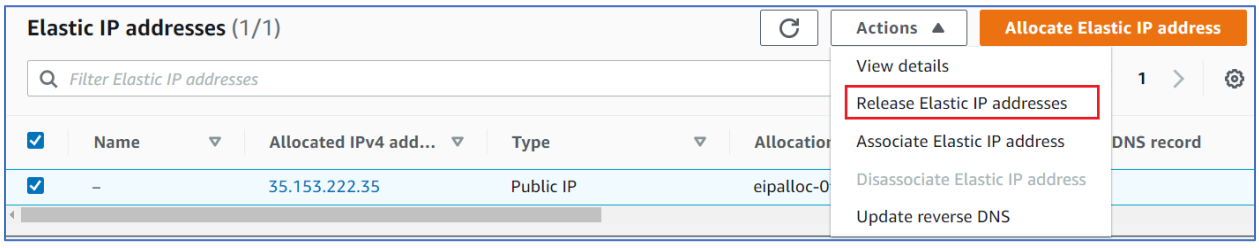
```
[ec2-user@ip-10-0-1-34 ssh]$ hostname
ip-10-0-1-34.ec2.internal # This is the bastion EC2 instance.
[ec2-user@ip-10-0-1-34 ssh]$ curl 10.0.2.221
<html><body><h1> Hello from ip-10-0-2-221.ec2.internal </html></body></h1>
[ec2-user@ip-10-0-1-34 ssh]$
```

Note: after completing this activity, delete the NAT gateway and release the Elastic IP Address.



Deleting the NAT Gateway does not release the EIP. This must be done specifically as shown below.





Learning Activity

Creating a VPC programmatically with AWS python boto3.

This activity shows a different way to create another VPC, this time using the AWS SDK for python. The following python script uses the module resource to deploy a VPC by asking two input values from the administrator, the IPv4 address space and the name of the VPC. These variable values are passed onto the CreateVpc function which configures the new VPC. This function returns a vpc object which is passed onto a second function that creates and attach an Internet gateway. This base script can be expanded to further adding functions to create subnets and routing tables.

```
# This python program creates a VPC and its Internet gateway.
import boto3

object = boto3.resource('ec2')

ipv4_space_var = input('Enter the IPv4 address space: ')
vpc_name_var = input('Enter the name of the VPC: ')

# Create VPC; assign IPv4 address space and name tag.
def CreateVpc():
    vpc = object.create_vpc(CidrBlock=ipv4_space_var)
    vpc.create_tags(Tags=[{'Key': 'Name', 'Value': vpc_name_var}])
    vpc.wait_until_available()
    print(vpc.id)
    return(vpc)

# Create the Internet Gateway and attach it to the VPC.
def CreateIgw(vpc):
    igw = object.create_internet_gateway()
    vpc.attach_internet_gateway(InternetGatewayId=igw.id)
    print(igw.id)

vpc = CreateVpc()
CreateIgw(vpc)
```

Once that the script runs, a VPC-B with the address space 100.64.0.0/16 appears in the VPC service.

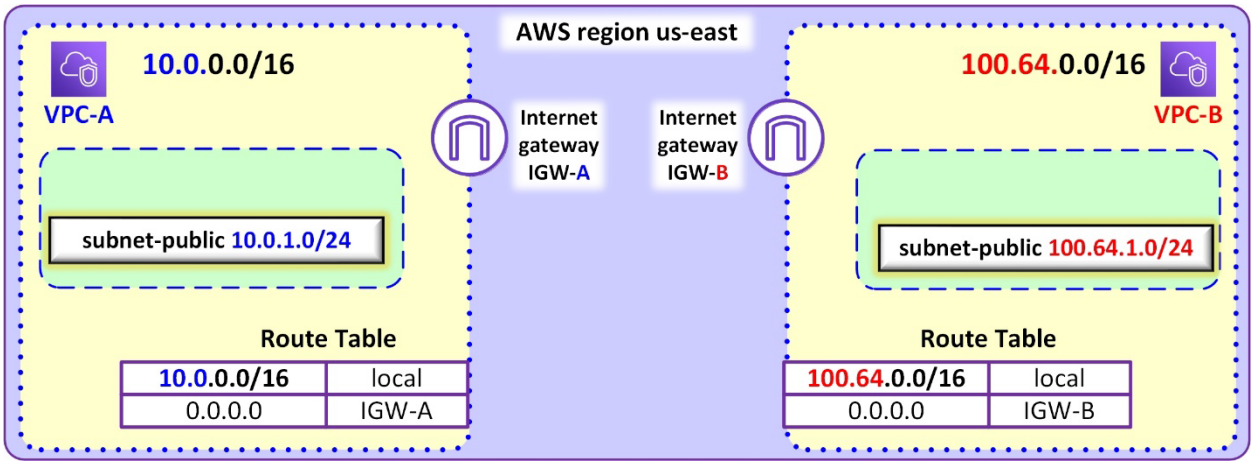
Your VPCs (3) Info

Name	VPC ID	State	IPv4 CIDR
VPC-A	vpc-074c608a4aab5f7c8	Available	10.0.0.0/16
VPC-B	vpc-05ff8ea4d36a4dfc5	Available	100.64.0.0/16

Learning Activity

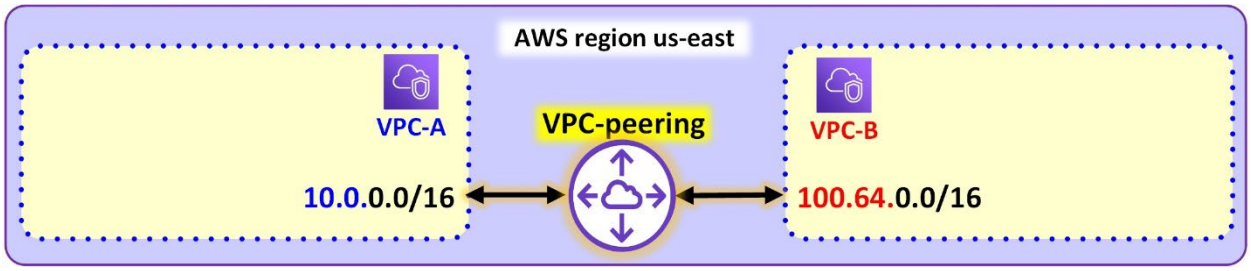
Configuration of two VPCs with VPC peering

This activity will reuse the VPCs A and B previously created. Currently, this is the organization's topology:



Same organization, two VPCs that do not communicate yet.

The organization needs the two separate VPCs for administrative and cost control reasons. However, the two VPCs need to share the access to some resources and this communication should be kept inside the AWS infrastructure. The solution to this requirement is to have the two VPCs **peering** with each other as shown in the next schematic:



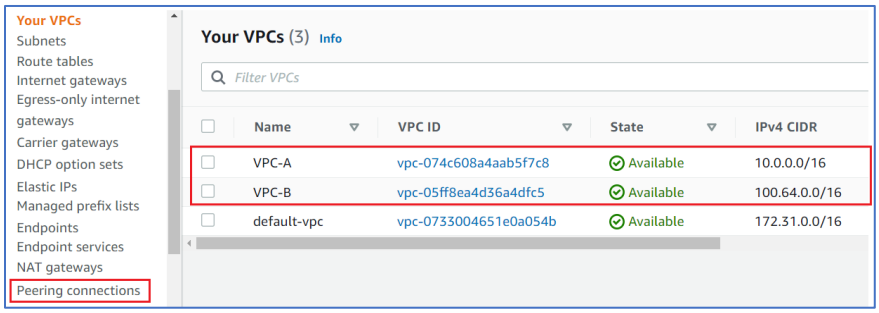
VPC peering between VPC-A and VPC-B.

VPC peering uses AWS cloud infrastructure to deploy a virtual interconnection between VPC in the same organization or in different organizations. VPC peering can be enabled across different accounts and regions too. The access to the resources can be fine tuned using the access control resources such as NACLs, permission policies and security groups.

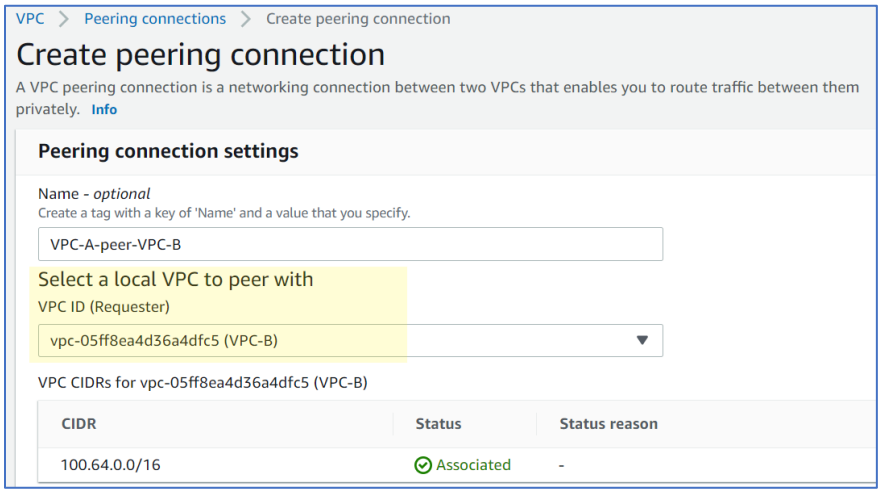
The following tasks are required to deploy such VPC peering in this example case:

- Create two **VPC**s (already done).
- Create **VPC peering connections**.
- Create or modify the corresponding **Route Tables**.
- Create **subnets** (if needed).
- Deploy **EC2 instances** for testing.

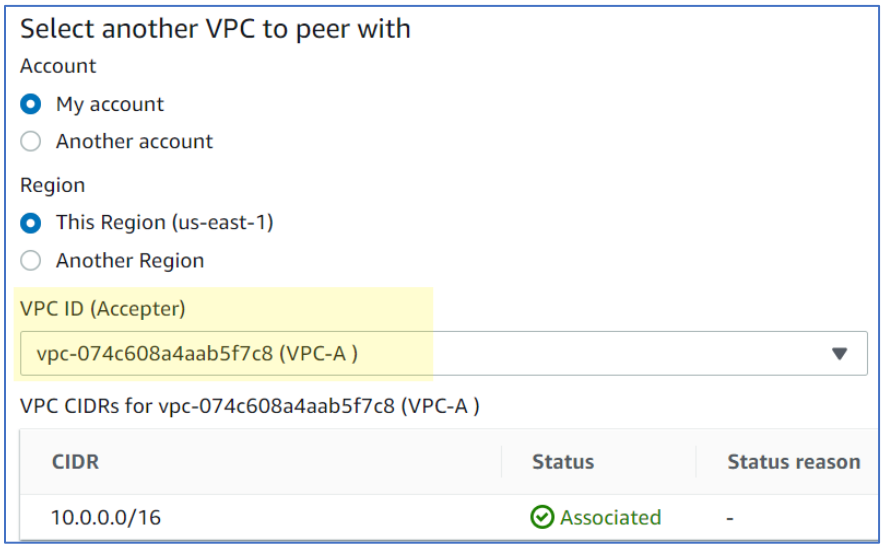
First, proceed to create a **Peering Connection.**



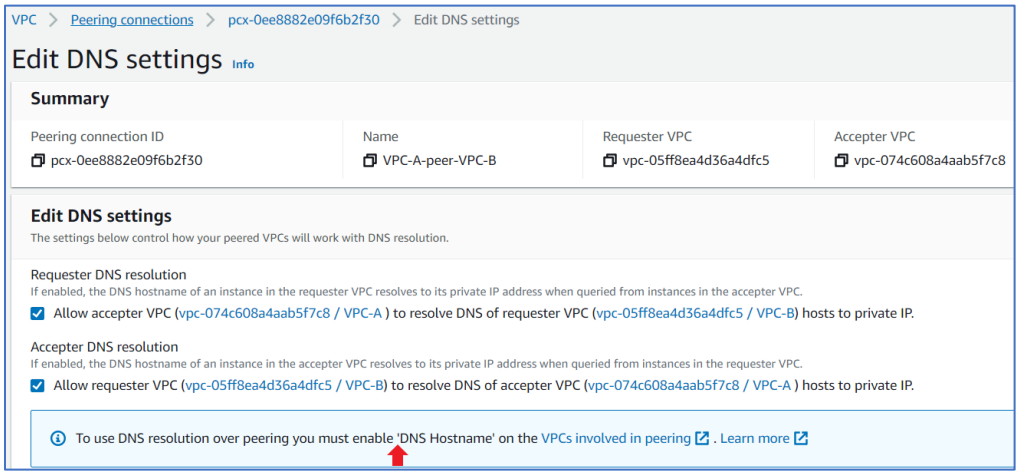
One of the VPCs initiates the connection. In this case, both VPC are in the same account, so either one will do. VPC-B is selected as the **requester** VPC.



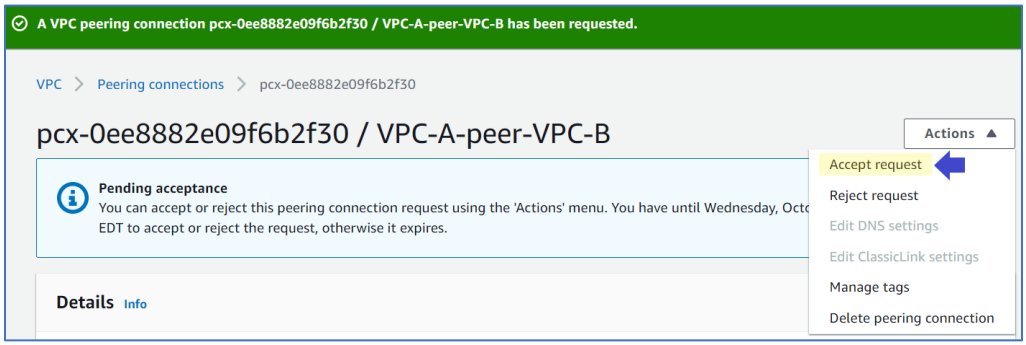
VPC-A will be the **accepter** on the other end of the VPC peering.



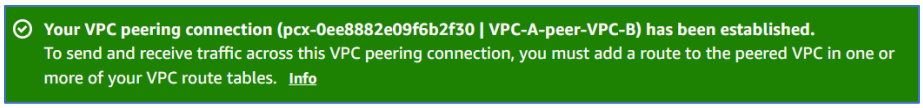
Optionally, the DNS settings might be modified so that either side can resolve the DNS names of the resources in the other VPC.

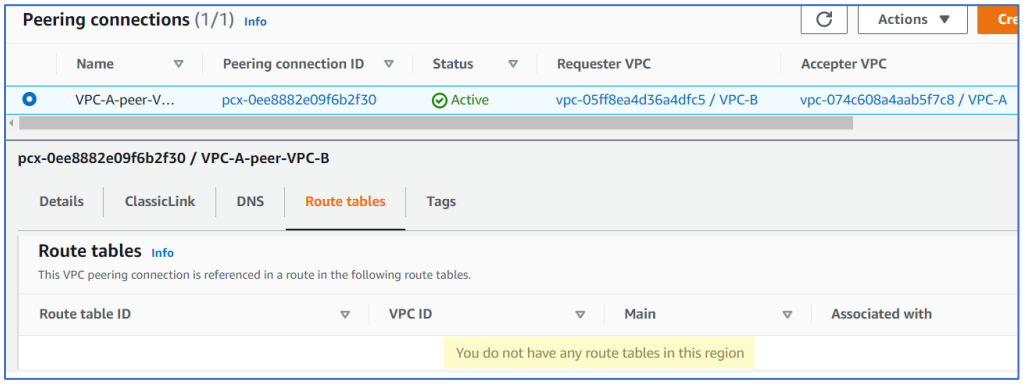


Once that the VPC peering is created, accept the request.

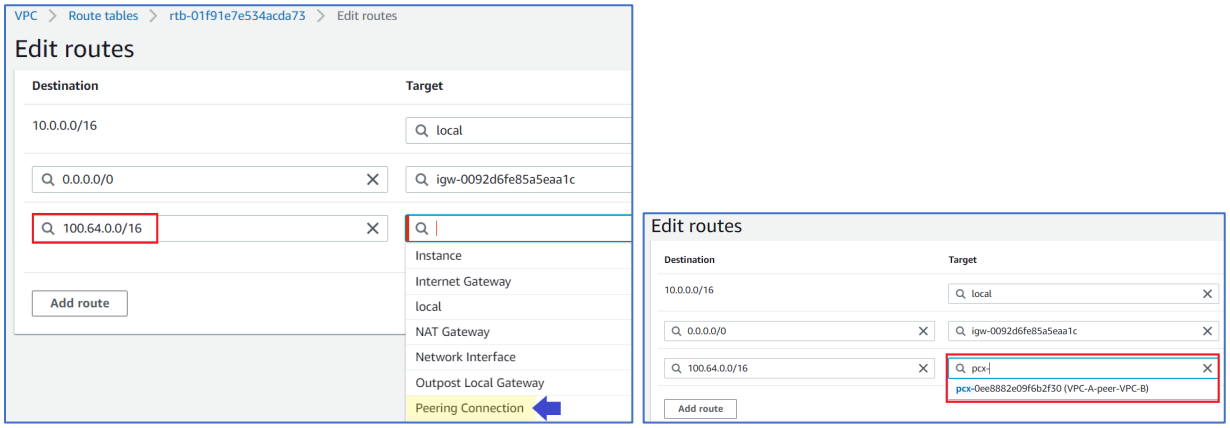


The VPC peering has been established. However, the traffic flow is not automatically configured because the routing tables must be either created or the existing ones modified to add the destination routes of the respective VPC IPv4 address spaces.

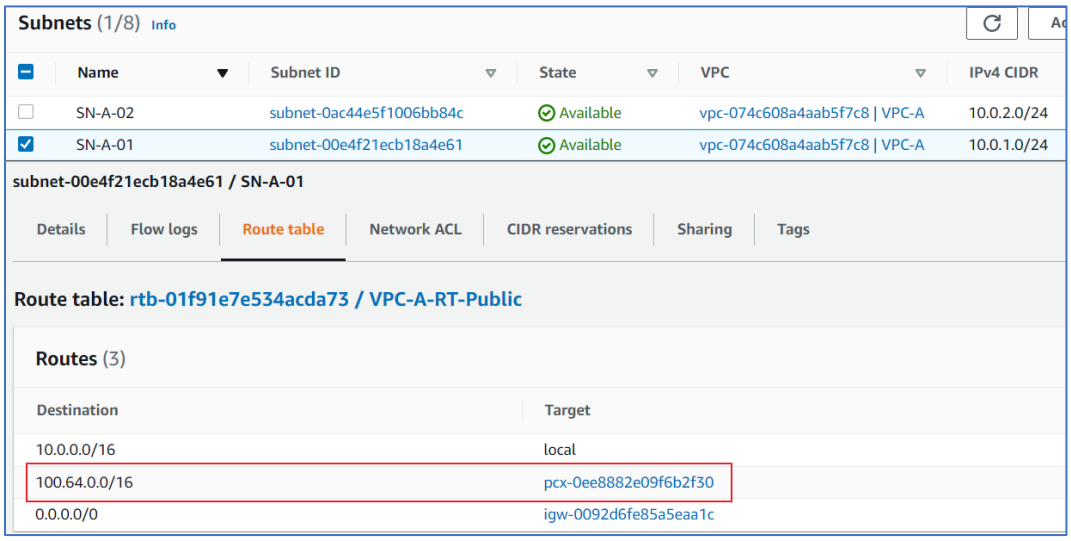




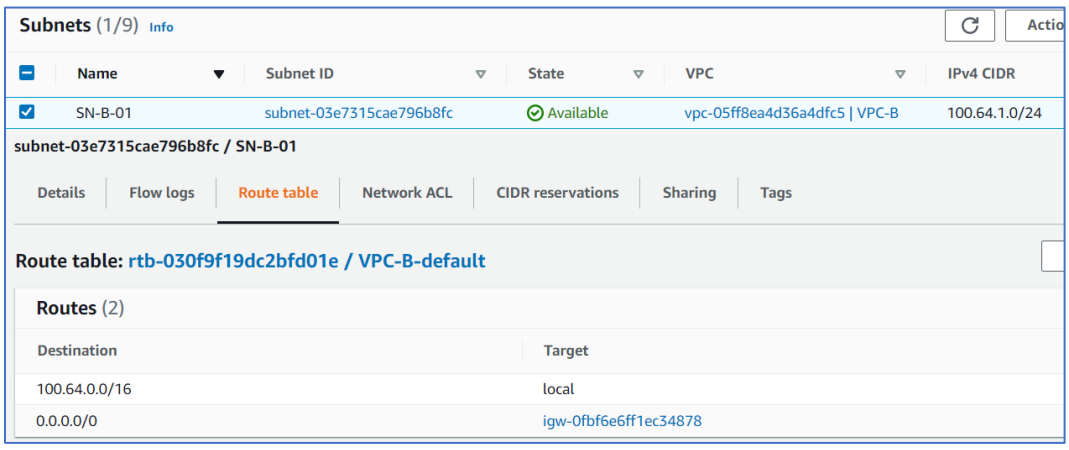
In this example case, the existing routing table for VPC-A is modified to add the destination address space 100.64.0.0/16 via the peering connection. It is just a manner of locating the peering connection pcx between the two VPCs.



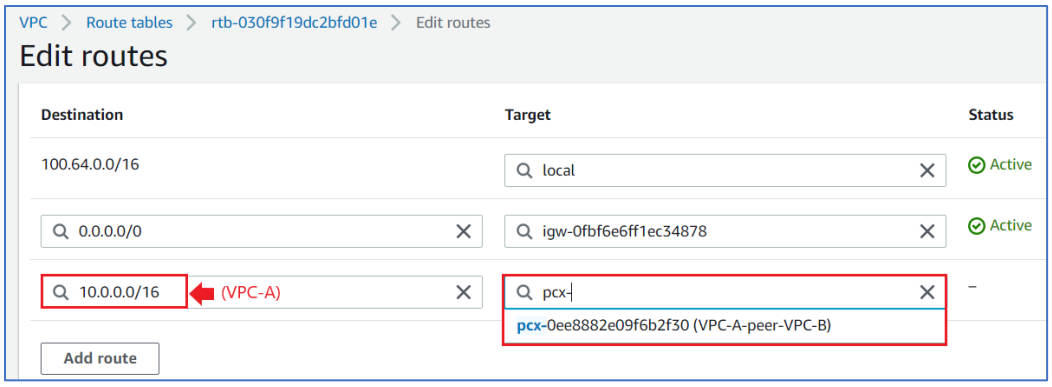
Resulting in the following routing table. Now, VPC-A "knows" how to get to 100.64.0.0/16.



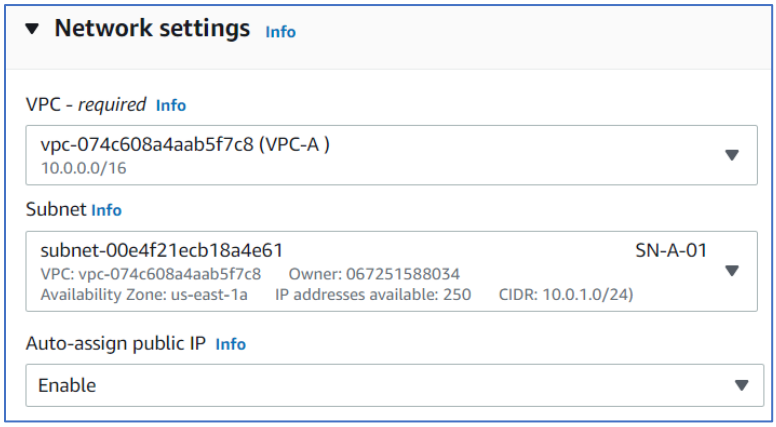
The route table of the other side must be modified as well. This is the current route table of VPC-B.



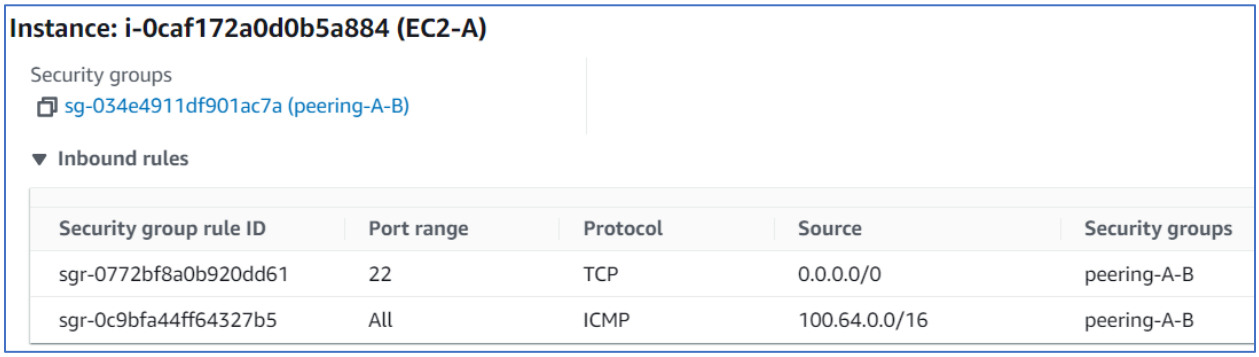
Let's proceed to add a route to the destination 10.0.0.0/16 in VPC-A going via the peering connection.



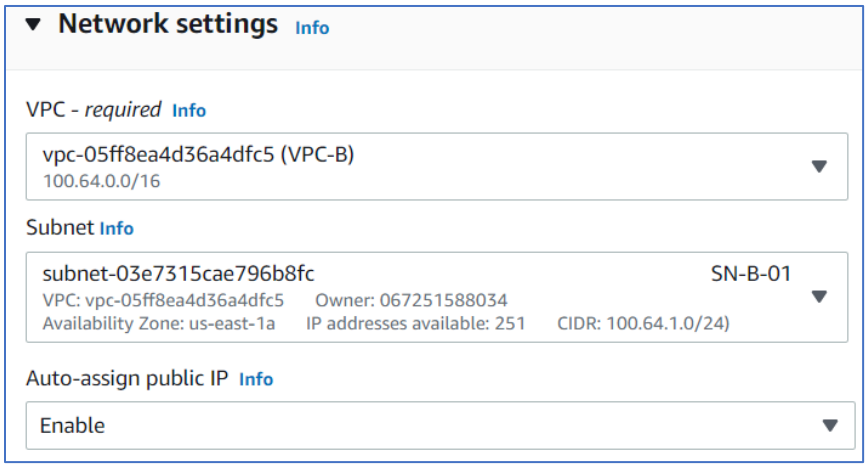
That completes the peering connection between the two VPCs. The next step is to deploy a couple of EC2 instances to test that the connectivity works as intended. First, deploy an EC2 instance (named EC2-A) in the existing subnet SN-A-01 in VPC-A. The network settings are used to place the EC2 instance in the right subnet at the moment of launching it.



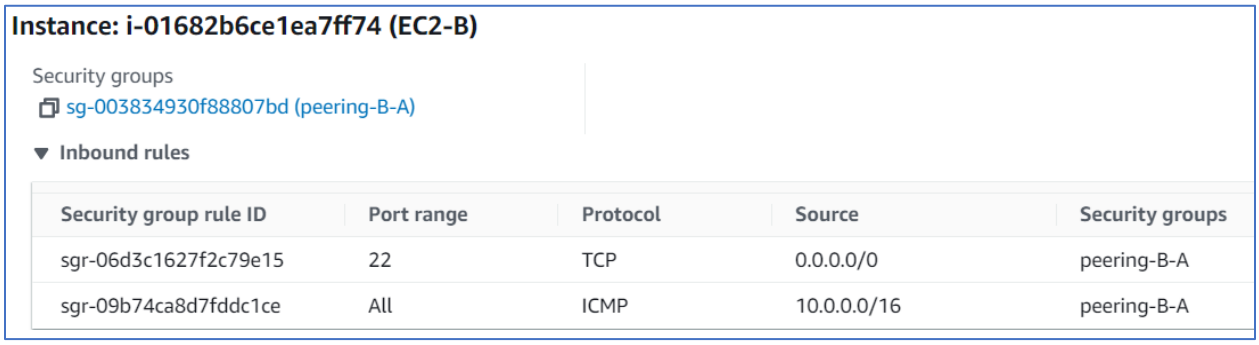
A new security group (named peering A-B) is configured to proceed with the testing. There are two rules, one allows SSH from the Internet so that the EC2 can be accessed. The second rule allows the ICMP protocol (used by the tool ping) but only if the source address is in VPC-B (100.64.0.0/16).



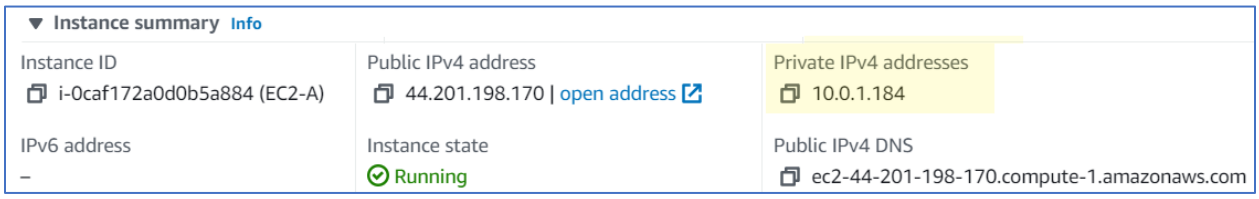
Similarly, an EC2 instance (named EC2-B) is created in a public subnet of the VPC-B.

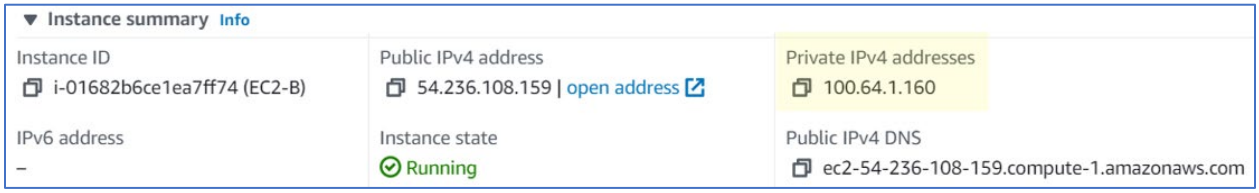


This EC2-B is associated to a new security group that allows SSH from everywhere and pinging only from the VPC-A (10.0.0.0/16). Therefore, the two EC2s can ping each other.



The following snippets show the instance summaries with the public and private addresses required to perform the testing of the access. The top snippet corresponds to EC2-A and the bottom to EC2-B.





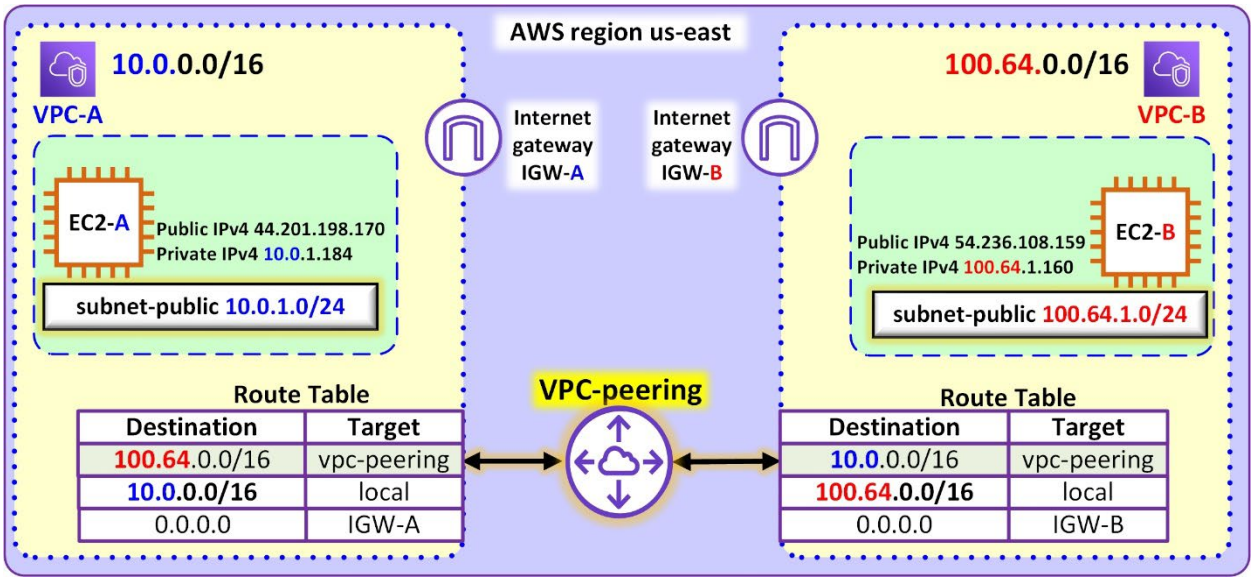
The EC2 instance in VPC-A is accessed via SSH and from there the private IPv4 of the EC2 in VPC-B is pinged.

```
C:\Users\felix>ssh ec2-user@44.201.198.170 -i C:\Users\felix\demokey.pem
       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

[ec2-user@ip-10-0-1-184 ~]$ ping 100.64.1.160
64 bytes from 100.64.1.160: icmp_seq=1 ttl=255 time=0.862 ms
64 bytes from 100.64.1.160: icmp_seq=2 ttl=255 time=0.596 ms
```

The same test is done the other way around. Both EC2 instances can reach each other.

```
C:\Users\felix>ssh ec2-user@54.236.108.159 C:\Users\felix\demokey.pem

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

[ec2-user@ip-100-64-1-160 ~]$ ping 10.0.1.184
64 bytes from 10.0.1.184: icmp_seq=1 ttl=255 time=0.554 ms
64 bytes from 10.0.1.184: icmp_seq=2 ttl=255 time=0.590 ms
```

This is the organization's topology once that everything is completed.



Conclusion

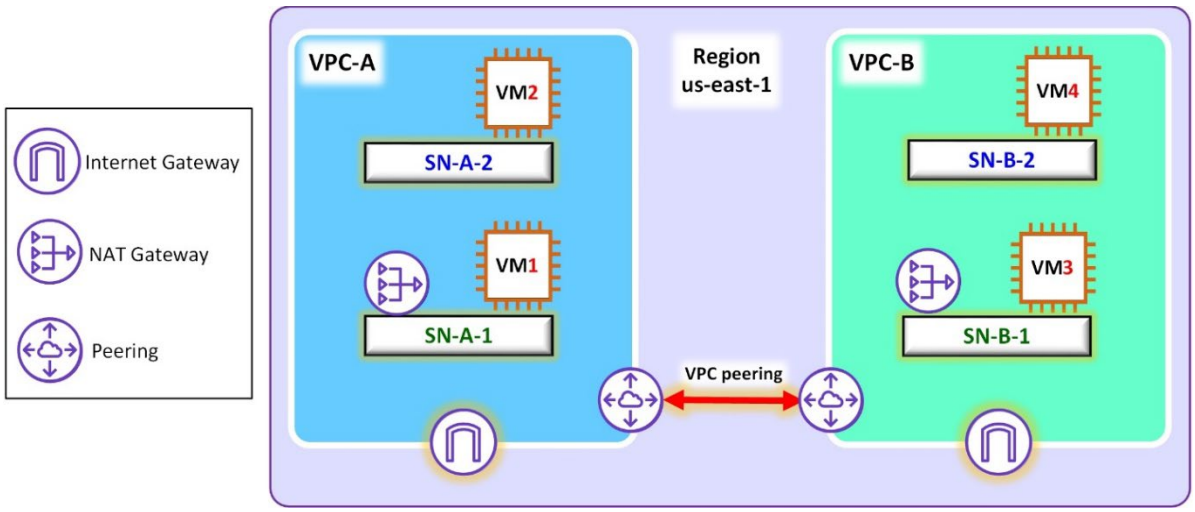
In this section virtual private clouds have been created and configured both using the AWS console and programmatically with python boto3. The main components of the VPC have been configured including the internet gateway, the NAT gateways, and the routing tables,

VPC peering is an AWS software construct that enables the communication across different VPC. Nevertheless, other actions are required to fine tune the connection such as the modification of routing tables and the configuration of security groups. The VPCs might be part of the same organization as this case study have shown or they could be part of different AWS accounts.

Chapter 7 Coursework

Lab deploy two VPCs with peering

The objective of this assignment is to provision a cloud infrastructure of two VPCs. The blue VPC-A is used to stage and develop applications. The green VPC-B is running the production application. The developers continuously improve the application in VPC-A and then the finished product is run in VPC-B.



Requirements:

- Use assigned IPv4 address space 10.#.0.0/16 where # is replaced with a number from 1 to 50.
- Each student **must use** a unique address space.
- Assign one half (½) of your total IPv4 address space to each VPC. Half of the space to VPC-A and the other half to VPC-B.
- Create two consecutive subnets in each VPC.
- Each subnet must have the space for 4,096 addresses including the special cases.
- Subnets SN-A-1 and SN-B-1 are public access subnets.
- Subnets SN-A-2 and SN-B-2 are private subnets.
- The EC2 instances VM1 and VM3 are the administrator's bastions.
- The EC2 instances VM2 and VM4 run web applications.
- VM2's webpage identifies the site with its private IPv4 address. (Hello from $(hostname -f)
- VM4's webpage identifies the site with its private IPv4 address. (Hello from $(hostname -f)

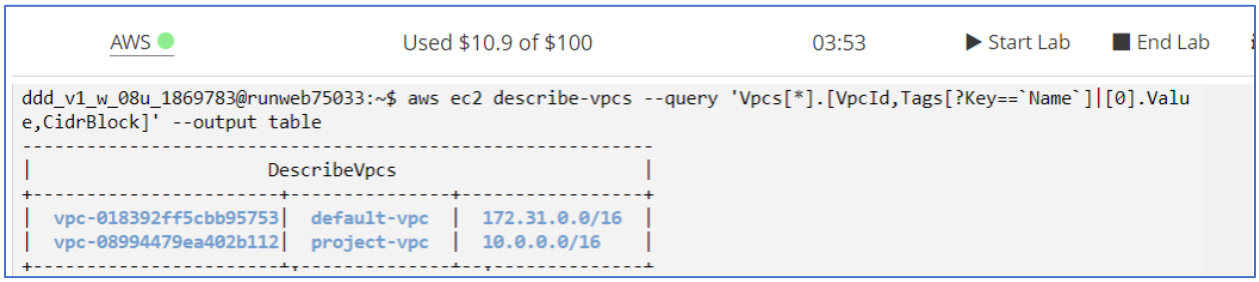
Check list	Item
	Create VPC-A with ½ address space of 10.#.0.0/16 where # is the student number from class list.
	Create VPC-B with the other ½ address space of 10.#.0.0/16 where # is student number.
	Create VPC-A's public subnet SN-A-1
	Create VPC-A's private subnet SN-A-2
	Create VPC-B's public subnet SN-B-1
	Create VPC-B's private subnet SN-B-2
	Create bastion VM1 in SN-A-1
	Create web VM2 in SN-A-2
	Create bastion VM1 in SN-B-1

	Create web VM4 in SN-B-2
	Create VPC peering between VPC-A and VPC-B

- Execute the following AWS CLI command [3] in AWS Academy lab console. Provide the result.

```
aws ec2 describe-vpcs --query
'Vpcs[*].[VpcId,Tags[?Key==`Name`]|[0].Value,CidrBlock]' --output table
```

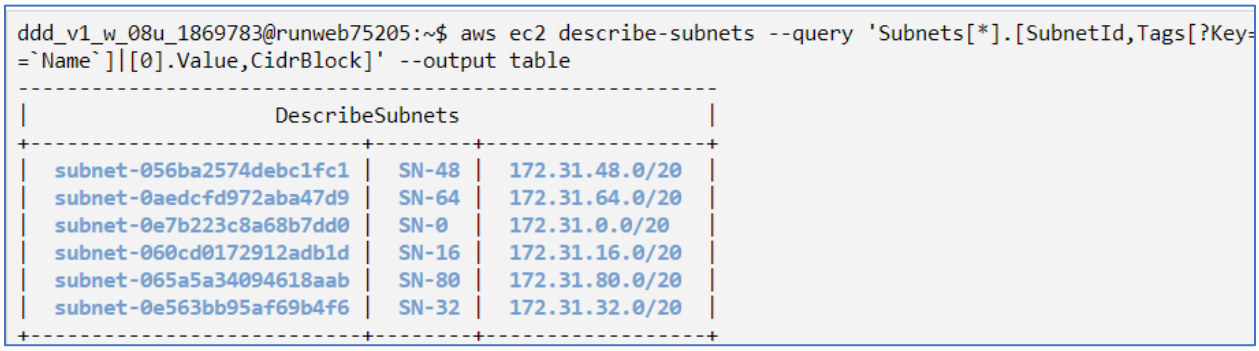
It should output all of your VPCs ids, their names, and the CidrBlocks (see example below).



- Execute the following AWS CLI command [3] in AWS Academy lab console.

```
aws ec2 describe-subnets --query
'Subnets[*].[SubnetId,Tags[?Key==`Name`]|[0].Value,CidrBlock]' --output
table
```

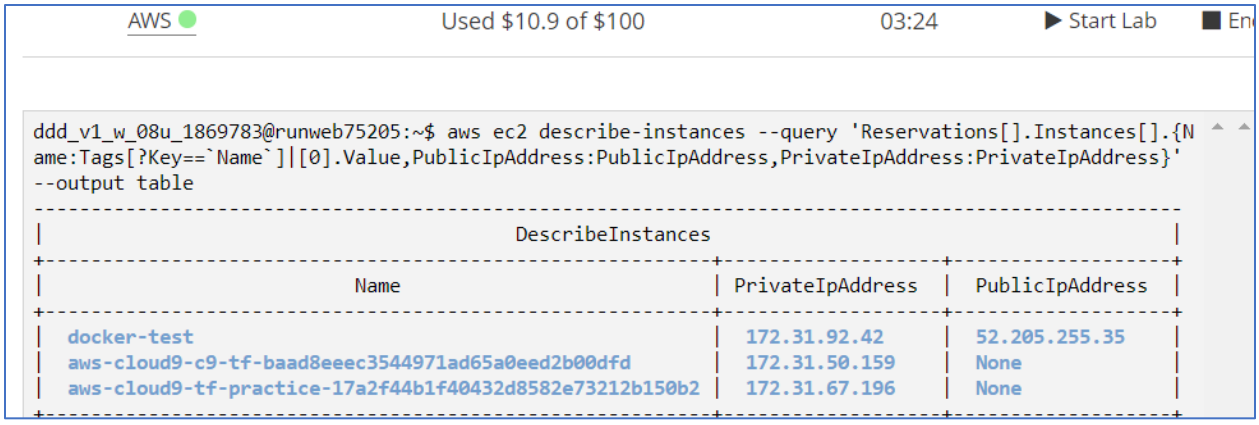
It should output all of your subnets ids, names, and cidr spaces (see example below).



- Execute the following AWS CLI command [3] in AWS Academy lab console.

```
aws ec2 describe-instances --query
'Reservations[].Instances[].{Name:Tags[?Key==`Name`]|[0].Value,PublicIpA
ddress:PublicIpAddress,PrivateIpAddress:PrivateIpAddress}' --output
table
```

It should output all the running EC2 instances with their public and private IP addresses. Example:



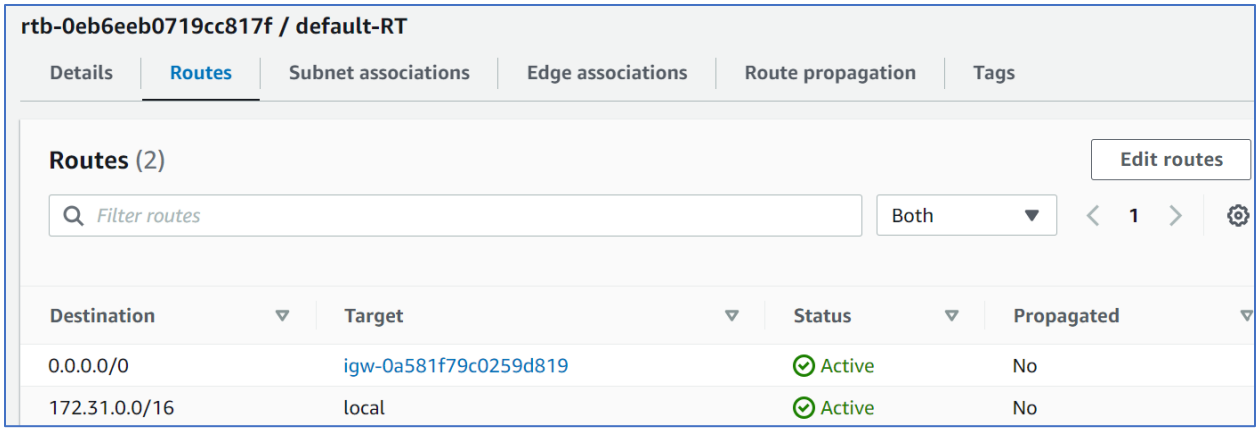
- The following AWS CLI command [3] should describe the peering connection.

```
aws ec2 describe-vpc-peering-connections --query
'VpcPeeringConnections[*].[VpcPeeringConnectionId,RequesterVpcInfo.VpcId
,AccepterVpcInfo.VpcId,Status.Code]' --output table
```

The following AWS CLI command [3] should describe the NAT gateways.

```
aws ec2 describe-nat-gateways --query
'NatGateways[*].[NatGatewayId,SubnetId,VpcId,State]' --output table
```

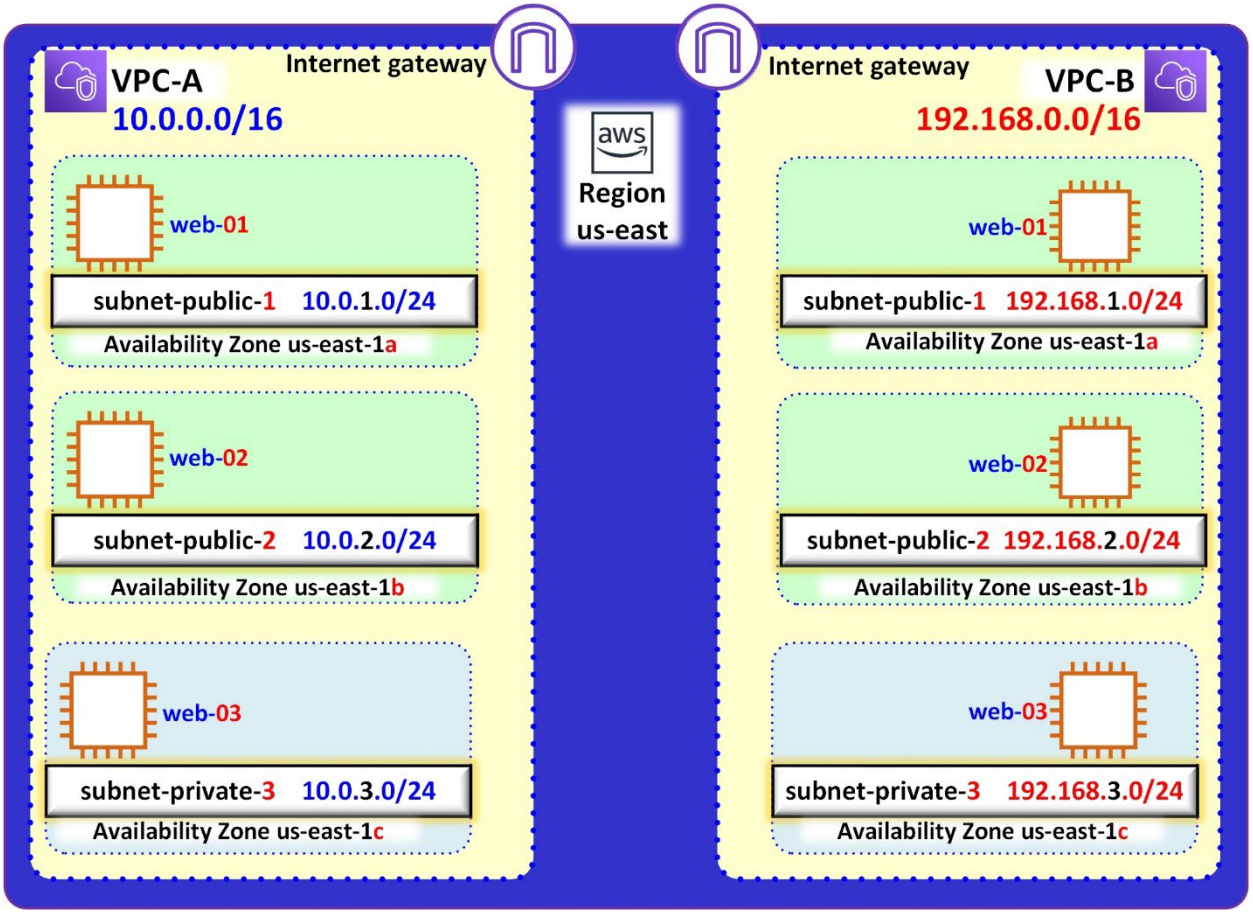
For example, something like this (this is a just an example).



- From VM1 curl to the private IPv4 address of VM2. It should return the homepage of VM2.
- From VM1 curl to the private IPv4 address of VM4. It should return the homepage of VM4.
- From VM3 curl to the private IPv4 address of VM2. It should return the homepage of VM2.
- From VM3 curl to the private IPv4 address of VM4. It should return the homepage of VM4.

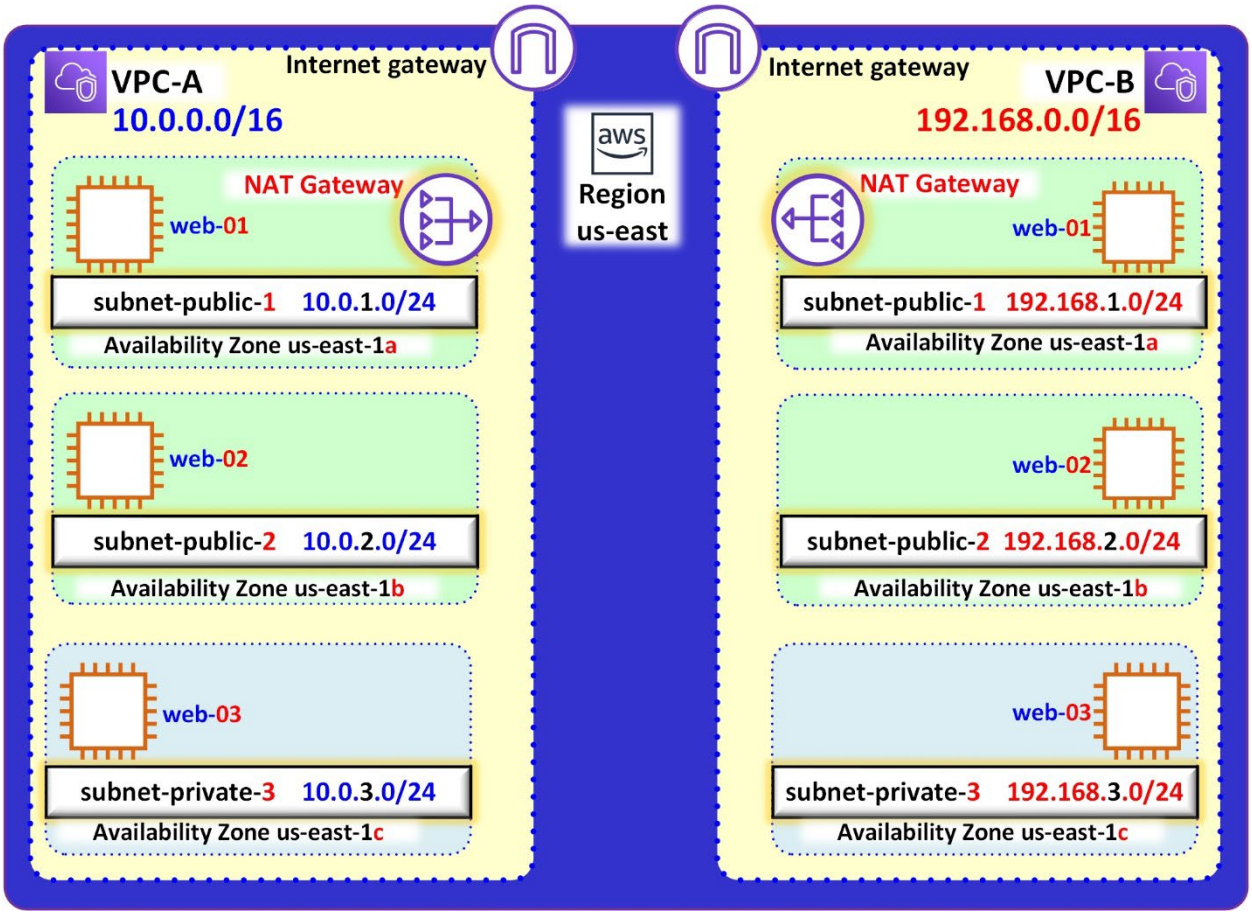
Practice Construction of VPCs

- Build the following topology using either AWS GUI console or Python boto3.
- The web servers web-01 and web-02 should have SSH and HTTP access to the web-03 in their corresponding VPCs.



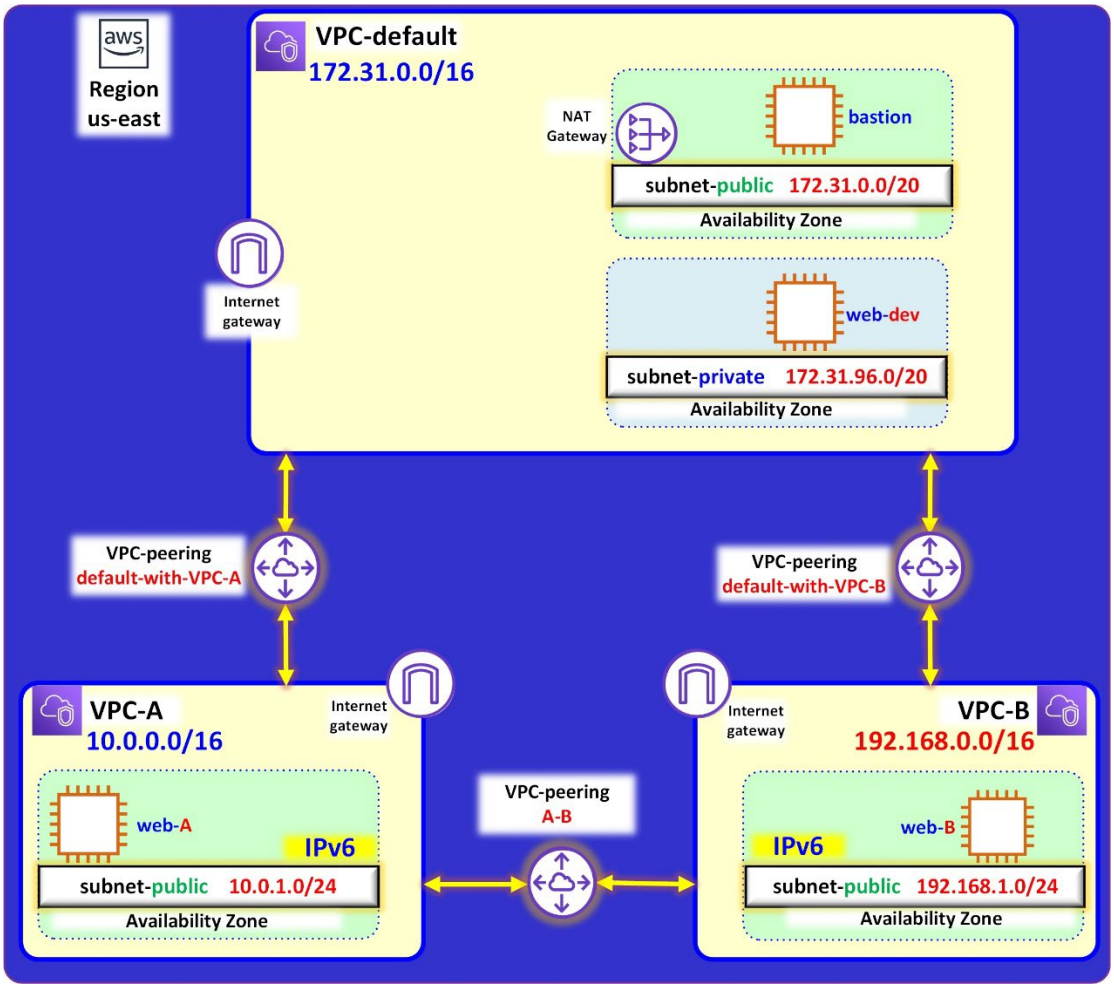
Construction of VPCs with private subnet with NAT gateways

- Build the following topology using either AWS GUI console or Python boto3.
- The web servers web-01 and web-02 should have SSH and HTTP access to the web-03 in their corresponding VPCs.
- The web-03 have access to the Internet via their NAT gateways.
- No one can access the web-03 if the connection is initiated from the Internet.



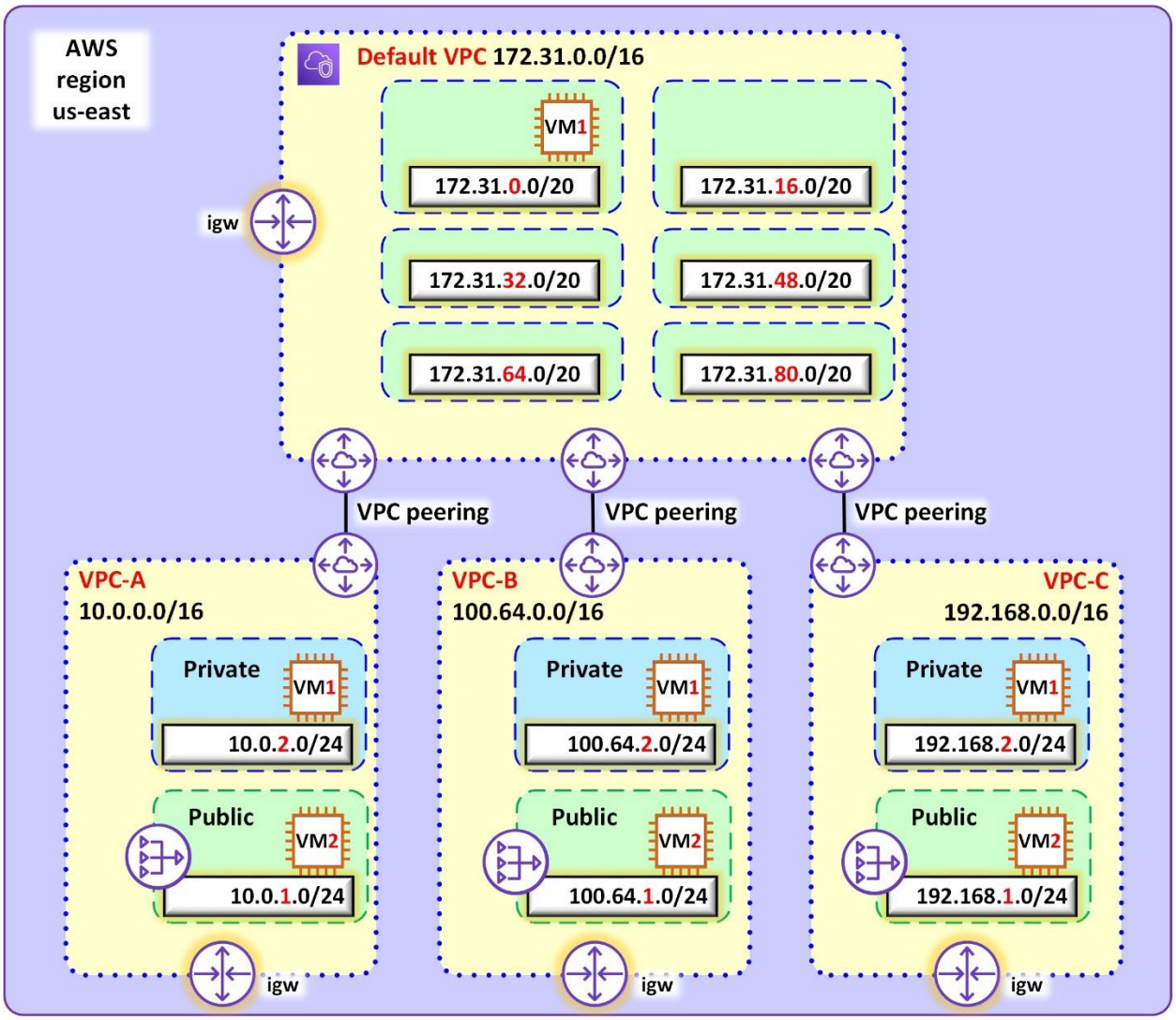
Construction of two VPCs with full mesh VPC peering

- Build the following topology.
- The EC2 instances web-dev, web-A, and web-B are webservers.
- The EC2 web-dev is located in a new subnet which is private.
- The EC2 web-dev has access to the exterior of AWS via a NAT Gateway located in the subnet-public 172.31.0.0/20.
- The user of the account must have SSH access to the EC2 bastion.
- From the bastion, account user must be able to SSH into the web-dev EC2.
- The EC2 instances web-A in VPC-A and web-B in VPC-B must be able to access the web-dev server in the default VPC (via curl to the IPv4 address).
- The public subnets in VPC-A and VPC-B must have IPv6 enabled.
- The EC2 web-A and web-B must be able to reach each other using their IPv6 addresses.
- The EC2 web-A and web-B must be able to reach IPv6 addresses outside AWS (test by pinging 2001:4860:4860::8888 which is an address located in Google).



Construction of three VPCs with VPC Peering to the default VPC

- Build three VPCs are described in the following diagram.
- The default VPC represents a **service provider** VPC.
- The VPC from A to C are **service receivers**.
- The VM1 in the default VPC runs an application (it could be a webserver just to prove the point).
- The VM1s in the private subnets of the VPCs A, B, and C should have access to the VM1-provider.
- The VM2s in the VPCs A, B, and C should have Internet access.
- Additionally, each VM2 should have access to the local VM1.
- The VM2s should not have access to the VM1 located in the service provider default VPC.



Reference

[1] Amazon Virtual Private Cloud. AWS. 2022. [Online]. Available: https://docs.aws.amazon.com/vpc/index.html

[2] VPC peering. AWS. 2022. [Online]. Available: https://docs.aws.amazon.com/vpc/latest/peering/what-is-vpc-peering.html

[3] AWS CLI commands provided by chapGPT. 2023. [Online] Available: https://chat.openai.com/

Chapter 8

Cloud Autoscaling Infrastructure

Description

A robust web application must run on a secure, reliable, highly available infrastructure. To offer the best customer experience, the application code must be robustly written and tested while the underlaying infrastructure must withstand potential failure situations. A cluster of servers running behind a virtual load balancer in the cloud is a solid platform to support an application. The load balancer distributes the workload among identical servers thus providing the customer with a seemliness service quality.

Learning Outcomes

- Configure elastic load balancers to distribute workload traffic.
- Deploy elastic virtual computers in a cloud networking environment.
- Configure webservers on elastic virtual computers.
- Configure target groups of servers.
- Troubleshoot network traffic.
- Building server images.
- Configuring auto scaling groups.

Main concepts

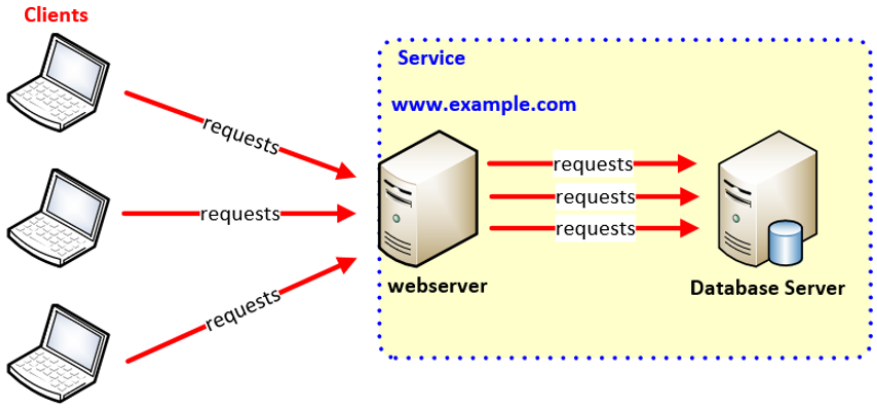
- Fundamentals of HTTP protocol.
- Target groups.
- Load balancing algorithms.
- Elastic Load Balancer.
- Auto scaling groups.
- Server images.

Learning Activities

- Creation of AWS target group of webservers.
- Creation of AWS elastic application load balancer.
- Creation of server image.
- Configuration of auto scaling group.

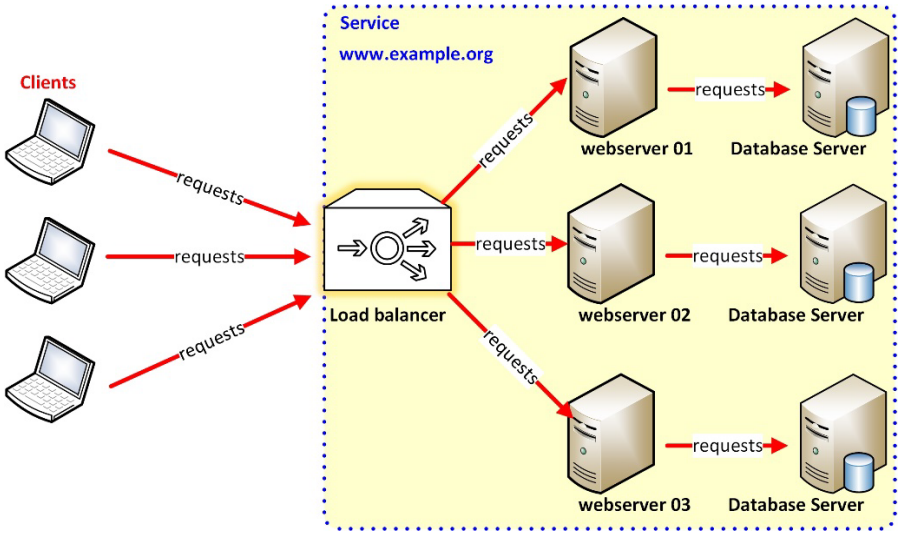
Elastic Load Balancing

Load balancing is a function that **distributes workload** across the backend infrastructure that supports a service. The main objective of load balancing is to maximize the performance of the service. Observe the following diagram to understand the need for load balancing. This is a situation without load balancing. A lone webserver is facing multiple requests from the clients at the same time that it queries the database to retrieve data. In such infrastructure, multiple service requests from clients might overwhelm the only webserver causing longer response time and, in general, a bad customer experience.



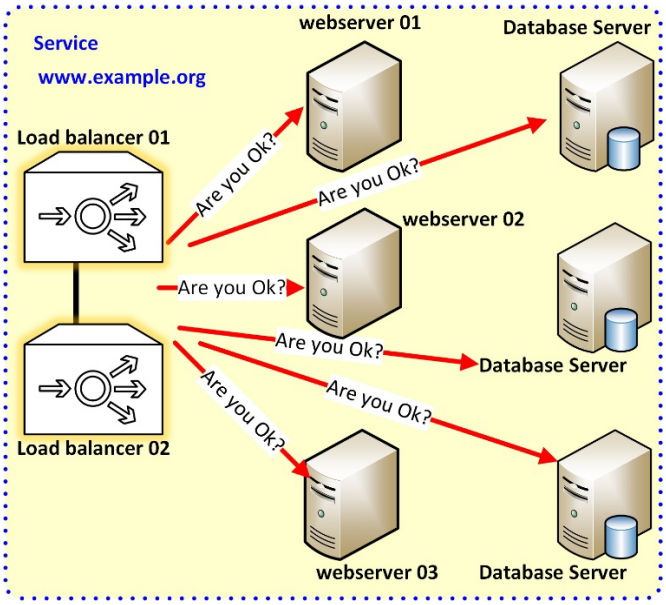
Multiple client requests overwhelm a single server deployment

The logical solution to this problem is to deploy more identical servers to match the demand for the service. The servers are arranged in a **cluster** that is reachable by using an **endpoint**, typically an URL name (for instance www.example.com). Consequently, the details of the infrastructure are opaque from the viewpoint of the clients. The load balancer answers the calls directed to the URL and it distributes the work among the replicated infrastructure. The backend servers supply identical content to the users; thus, they receive consistent content regardless of the server that gets the tasks assigned.



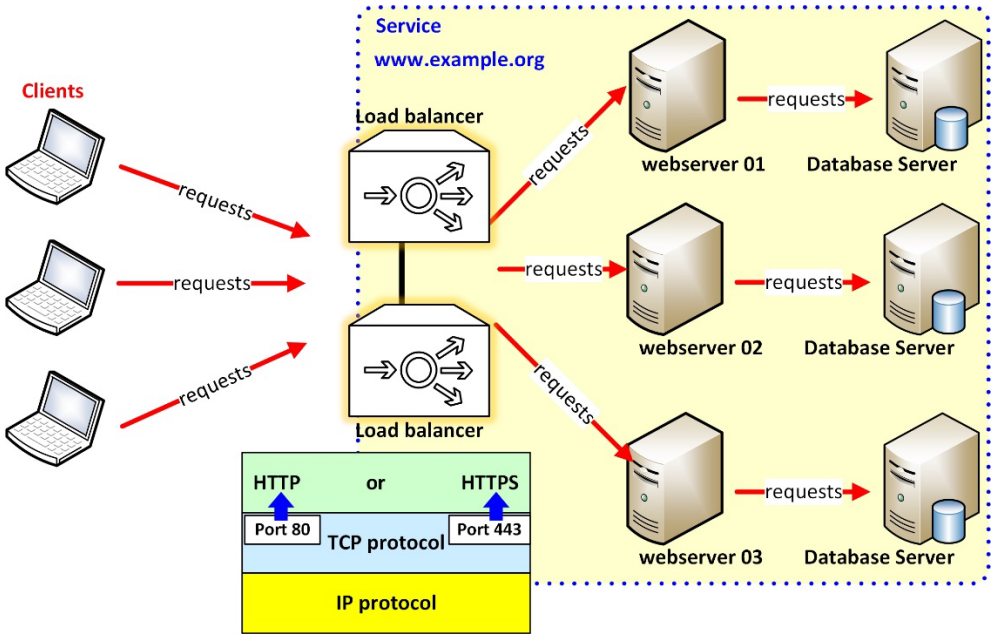
A load balancer distributes the work among identical servers

The load balancer also has an additional function which is to perform **periodic health checks** of the devices and applications that it is serving. For example, if the cluster is a web site, the load balancer sends http queries periodically to the servers to which they must reply back in time. If a server does not respond timely, then the load balancer removes it from the cluster until it starts replying to the queries again.



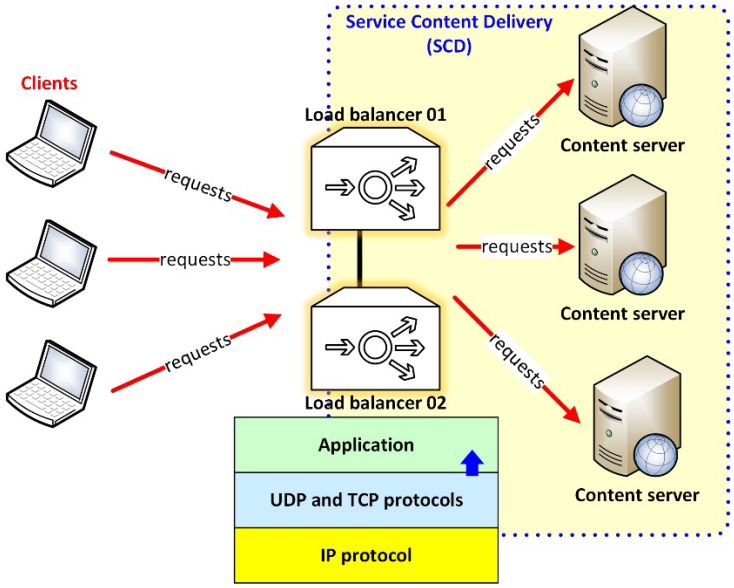
The load balancer also performs periodic health checks to verify if the servers are running properly

Having one physical load balancer represents a single point of failure even though load balancers are designed to handle heavy traffic loads. The solution to this problem is to stack several load balancers to manage the service. In the figure, two load balancers work in tandem to distribute the HTTP/HTTPS traffic load toward a cluster of webservers.



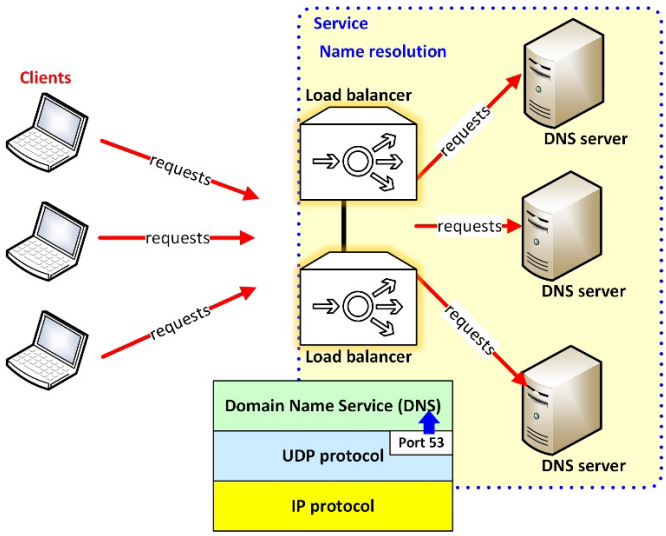
Robust infrastructure with dual load balancers

Load balancers are designed to work at different layers. For example, some load balancers manage the traffic by looking at the networking (IP) layer only. Other load balancers are focused on the transport layer (TCP and UDP) and others are designed to deal with specific applications only. Since load balancers typically handle any protocol encapsulated by either TCP or UDP, it is necessary to understand the protocols mechanisms and encapsulations to set up load balancers properly.



Load Balancers can handle different protocols and ports

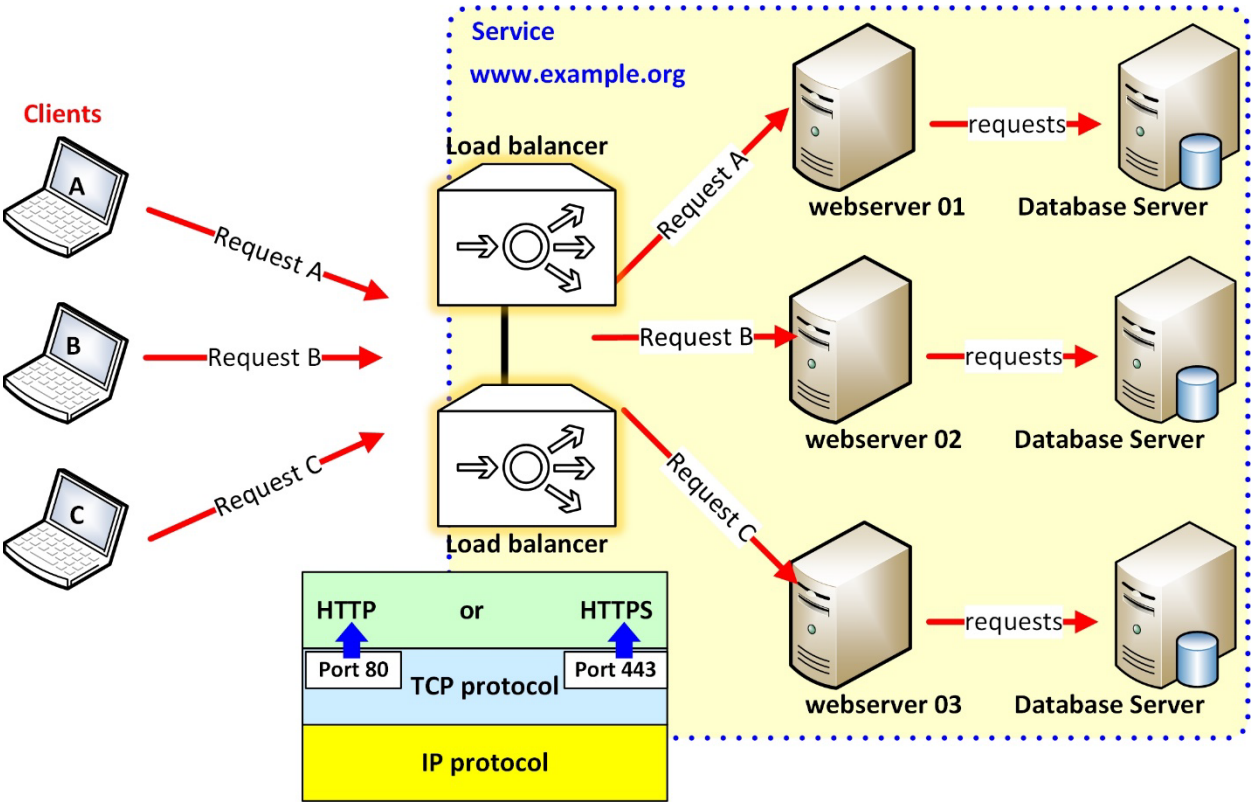
The most common case of load balancing is the handling of web protocols, both secure HTTPS and insecure HTTP. Other common application of load balancers is to act as proxies for domain name resolution requests (DNS) directed at a farm of DNS servers. Also, load balancers are used to handle requests to large database systems. In the following example, the load balancers handle the workload for DNS servers. The devices examine traffic with the destination UDP transport port 53 (DNS). Then, they direct the requests toward the DNS servers. From the viewpoint of the clients, this appears as one front.



Load Balancer handling DNS request

Load Balancing web traffic

- The most common use case for load balancing is the handling of web traffic.
- A Load Balancer is a **proxy** for HTTP.
- Web traffic come in two formats: unsecure HTTP (port 80) or secure HTTP (HTTPS port 443).
- HTTP is un-ciphered. In this case, the load balancer reads several fields in the HTTP header to keep track of the messages being forwarded.
- In the case of HTTPS, the load balancer supplies the required **Security Certificate** to authenticate the service. In such case, the load balancer can also be the ciphering termination point (for HTTPS Transport Layer Security TLS). This action further reduces the work done by the webservers.
- The load balancer must keep track of the TCP session numbers and the source IP address.
- Also, a load balancer can keep track of cookies.



Load Balancer handling web traffic requests

In a typical load balancing deployment, the endpoint URL (DNS name) is tied to a public IP address. The backend servers are located in a private network. Consequently, the load balancer must act as a proxy that makes translations between the public side and the private side of the service.

Furthermore, the load balancer acts as a termination point of a ciphered TLS HTTP session between the external clients and the service. Another proxy session is established between the load balancer to the backend servers. To provide a seamless connection the load balancer must look into the HTTP header.

Specifically, the load balancer manipulates the **Proxy Forwarded** [1] fields in the HTTP header. These fields provide the backends servers with the information about the original request.

The field managed [1] are:

- The **X-Forwarded-For** field in the HTTP header provides the **originating IP address** of a client connecting to a web server through either an HTTP proxy or a load balancer which is the case here. The format of the field is simply: X-Forwarded-For: source IP address.
- The **X-Forwarded-Proto** identifies the **protocol** used by the client. Its format is just: X-Forwarded-Proto: protocol (for example https).
- The **X-Forwarded-Port** contains the **destination port** that the client is targeting.

Handling HTTPS is more complex because the traffic is ciphered. HTTPS involves the use of Security Certificates to prove the authenticity of the web services and the negotiation of a secured TLS (Transport Layer Security) channel between the client and the webserver. There are two ways to manage HTTPS.

- In one format, the HTTPS traffic is passed thru to the backend servers unaltered.
- In another format, the termination of the secure channel is in the load balancer. The traffic between the load balancer and the backend servers is HTTP clear text. In this format, the Load Balancer must present the Security Certificates to the clients and negotiate the secure channels.

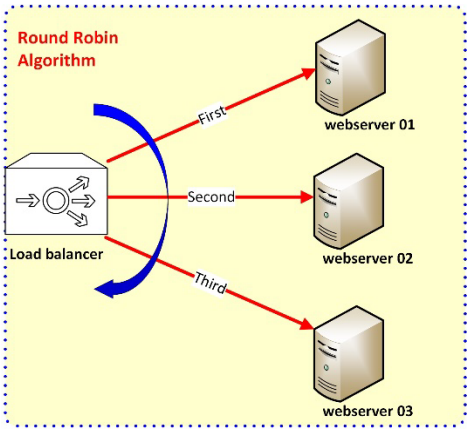
Load Balancing Algorithms

Load balancers use **algorithms** [2] to make their decisions. These are the most common:

- Round Robin
- Least Connections
- Source
- Hash

Round Robin

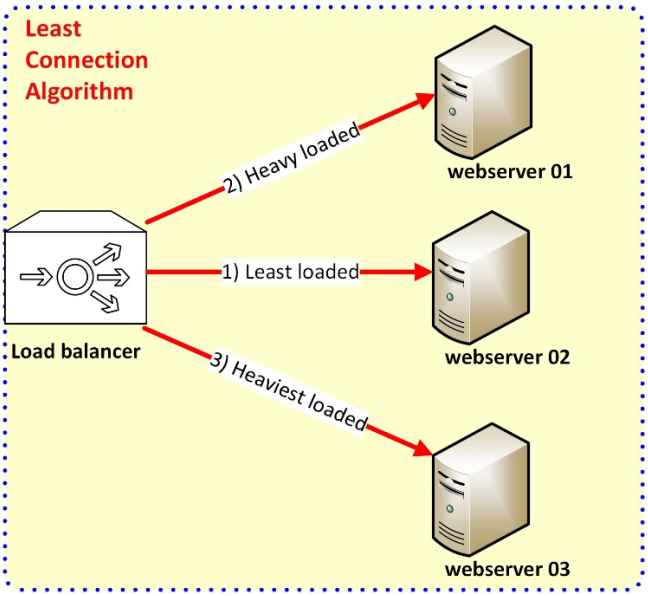
In the round robin load balancing system, the traffic is distributed according to an ordered list of the devices that is declared in the configuration of the load balancer. For example, in the following diagram, the load balancer manages the sessions in a cycled order from the top to the bottom. Round Robin is the simplest to implement algorithm; however, it has a deficiency which is that some servers get heavier loads than others because the time-length and volume of communication sessions are different.



Round Robin Load Balancing Algorithm

Least Connections

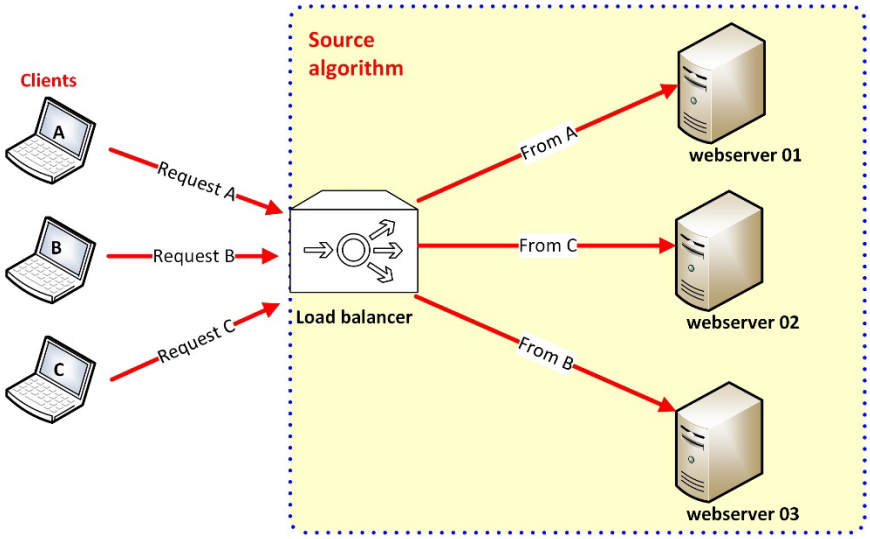
The **least connections algorithm** selects the servers with the least sessions. This algorithm compensates the deficiency of the round robin algorithm by considering the volume and time of traffic that each server must handle.



Least Connections Load Balancing Algorithm

Source

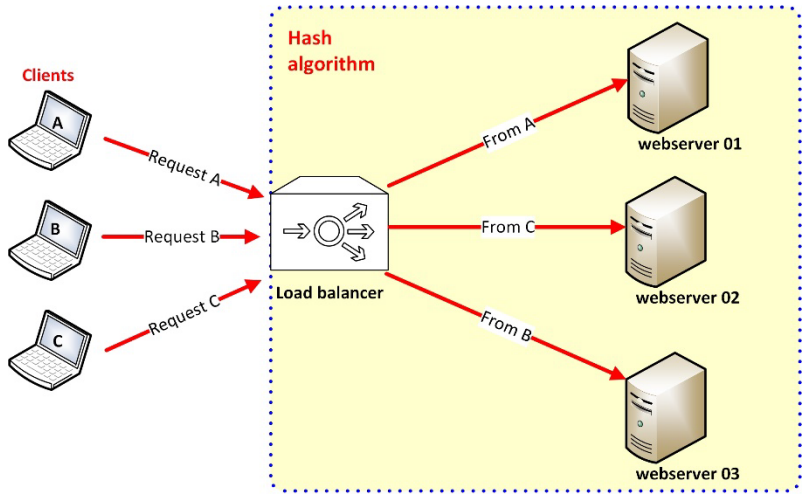
When this algorithm is used, the load balancer keeps track of the source IP address of the client to consistently provide the same server for different sessions. Another way to provision this algorithm treatment is with the use of cookies. These cookies are installed in the client's browser from a previous visit. As long as they are not cleared, the load balancer will look them up to determine the target server.



Source Load Balancing Algorithm

Hash

- In this algorithm, the servers are divided based on the value of an arbitrarily provided hash key. This hash can be formed from text, variables, or a combination of them. This is the only balancing method that requires the user to provide data, which is the key material that would be used for the hash.

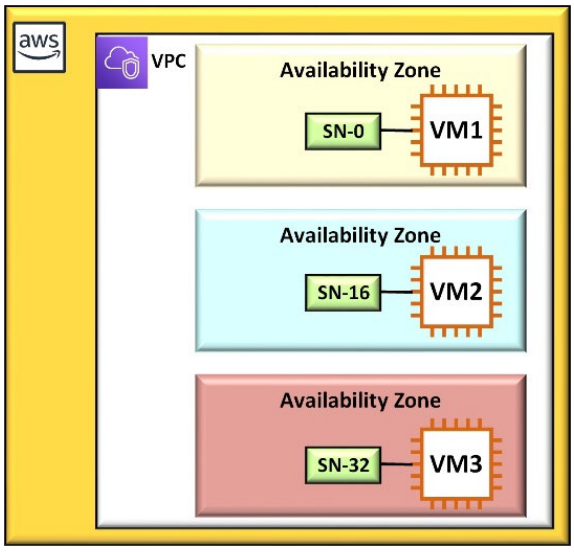


Hash Load Balancing Algorithm

Learning Activity

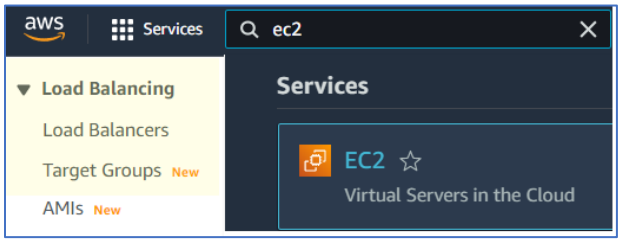
Configuration of an AWS Elastic Load Balancer

In this activity, an AWS elastic load balancer is configured to provide access to a cluster of three webservers connected to different subnets. Having the VM running in different subnets also implies different availability zones and consequently higher reliability.

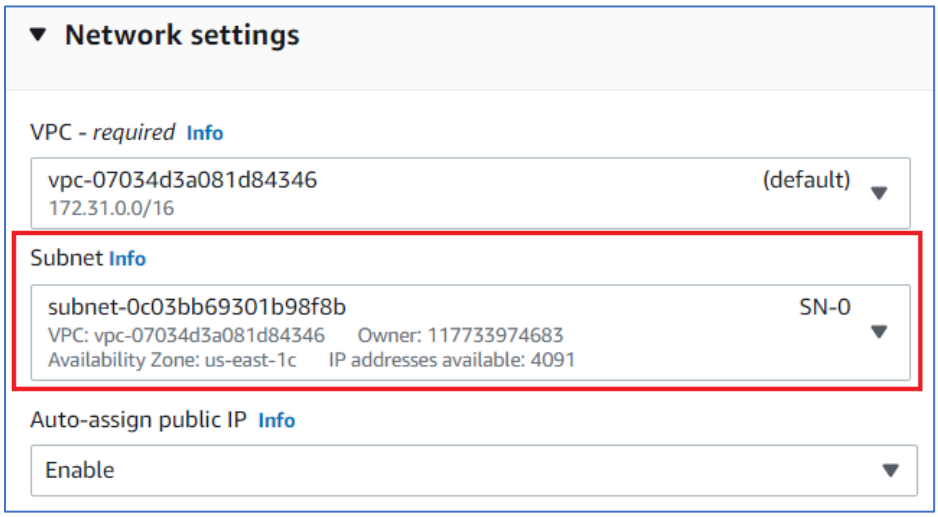


Group of three webservers located in different subnets/availability zones

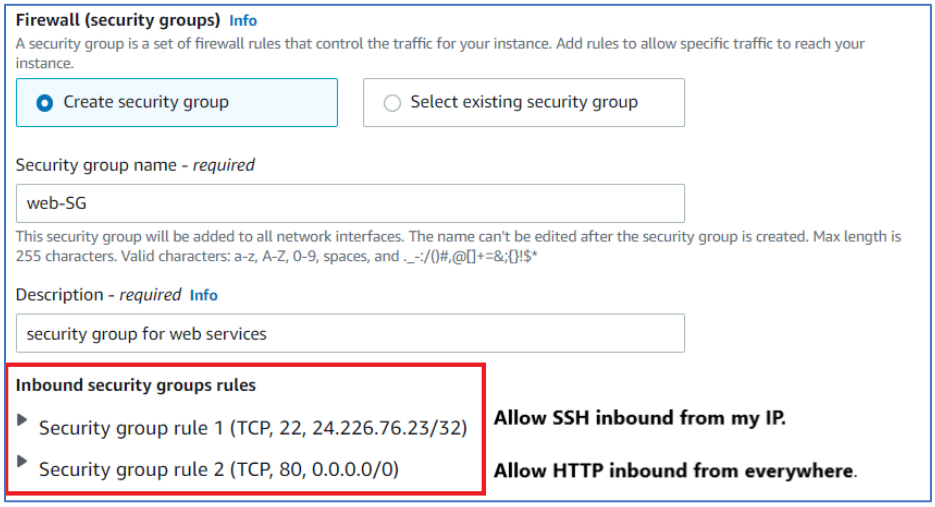
- The first task is to create the three webservers.



- Create an EC2 instance named VM1 and place it in subnet SN-0 (172.31.0.0/20).
- Note: the availability zone or location of the subnets is different for each user account.



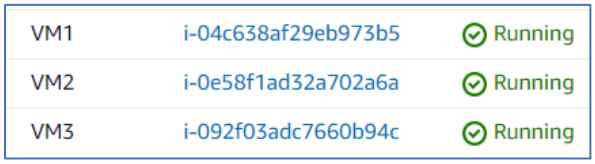
- Create a security group to allow SSH access only from my IP and HTTP access from every address.



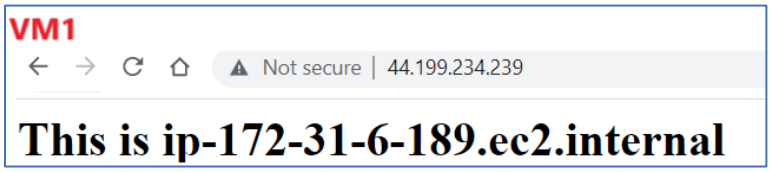
- Bootstrap the VM installation like this:

```
#!/bin/bash
yum install httpd -y
cd /var/www/html
echo "<html><body><h1> This is $(hostname -f) </h1></body></html>" > index.html
systemctl restart httpd
systemctl enable httpd
```

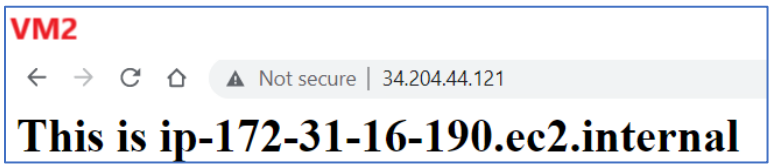
- Repeat the same procedure to create VM2 in subnet SN-16 (172.31.16.0/20) and VM3 in subnet SN-32 (172.31.32.0/20).



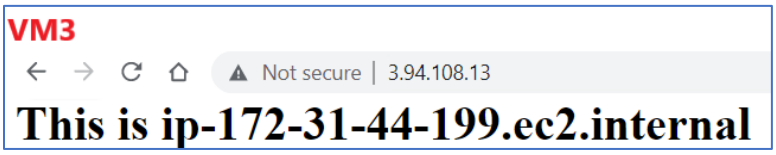
- Test that the VM are functioning as webservers using the individual public IPv4 addresses.



VM1 instance is running in subnet SN-0 (172.31.0.0/20)



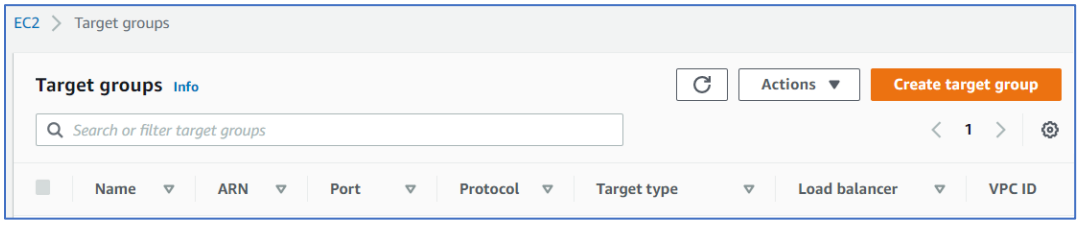
VM2 instance is running in subnet SN-16 (172.31.16.0/20)



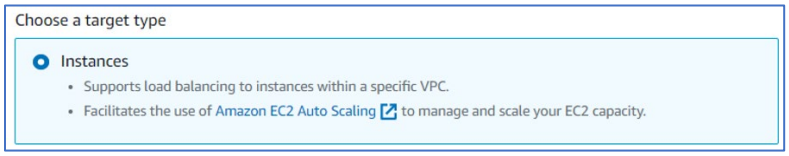
VM3 instance is running in subnet SN-32 (172.31.32.0/20)

Once that the three VMs have been created and tested, they must be placed together into a **target group.** The load balancer interfaces with the target group making easier to administer the service.

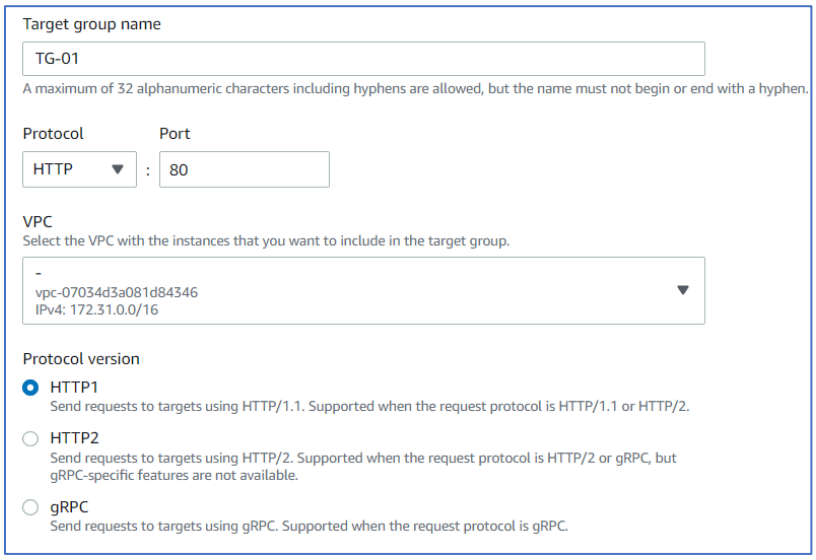
Creation of a Target Group



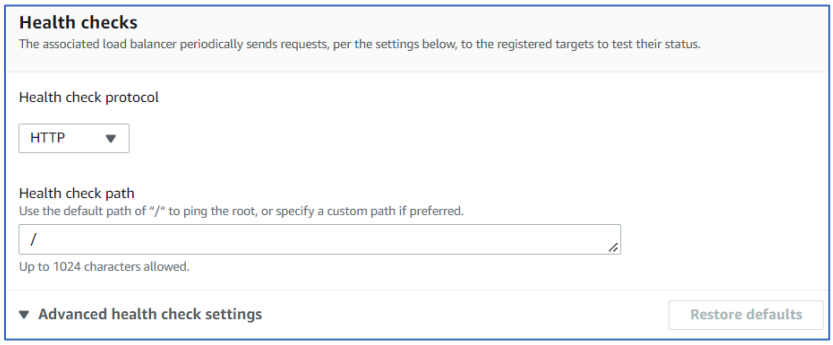
- Target groups can associate different types of resources. In this example, the targets are the three EC2 instances.



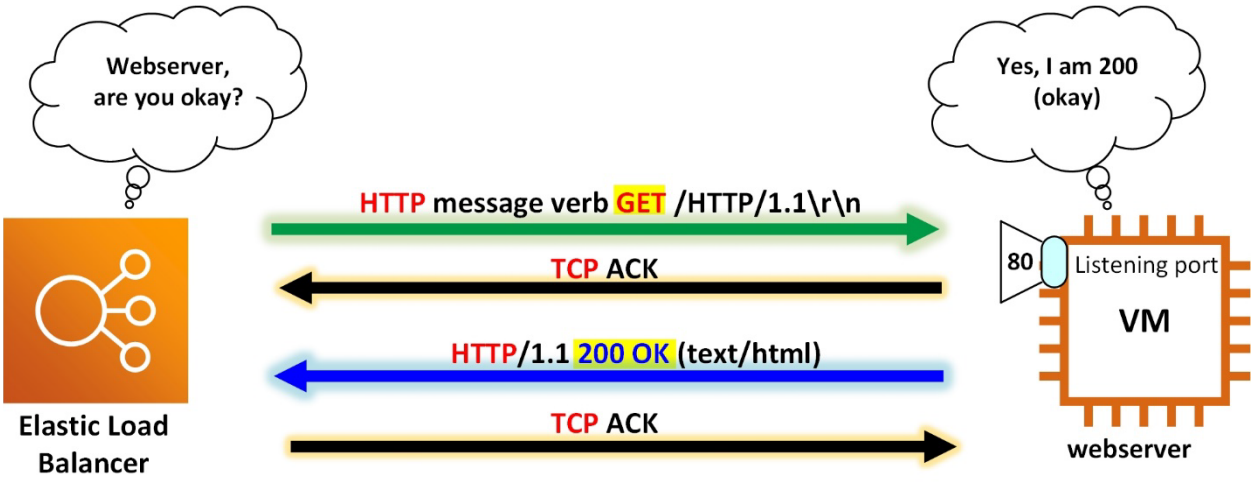
- Configure the protocol HTTP1 with the default port number 80.



- The load balancer will send HTTP requests periodically to each VM in the target group to check the health of the instances.



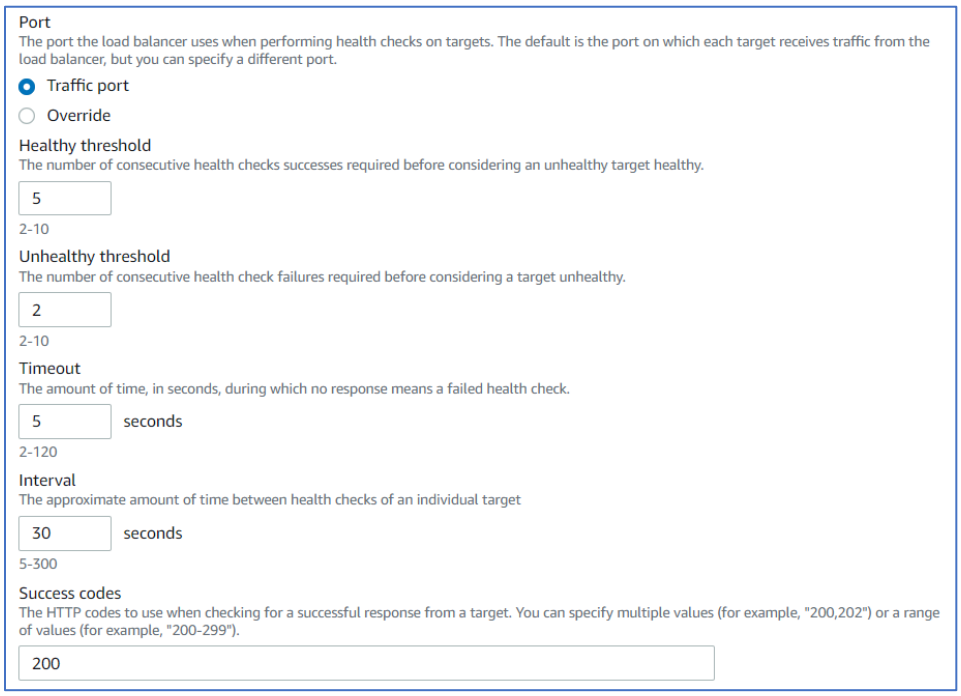
The elastic load balancer sends HTTP GET messages periodically to every webserver in the target group. The servers respond with a text/html message code 200 meaning “I am okay”, literally.



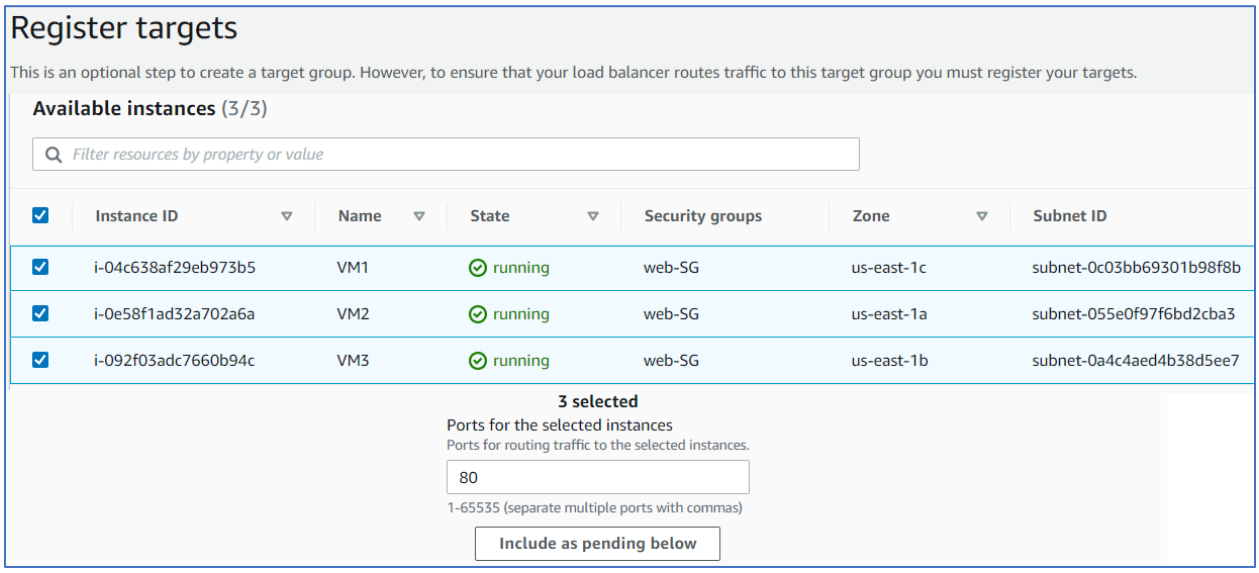
Load Balancer checking the health of a webserver

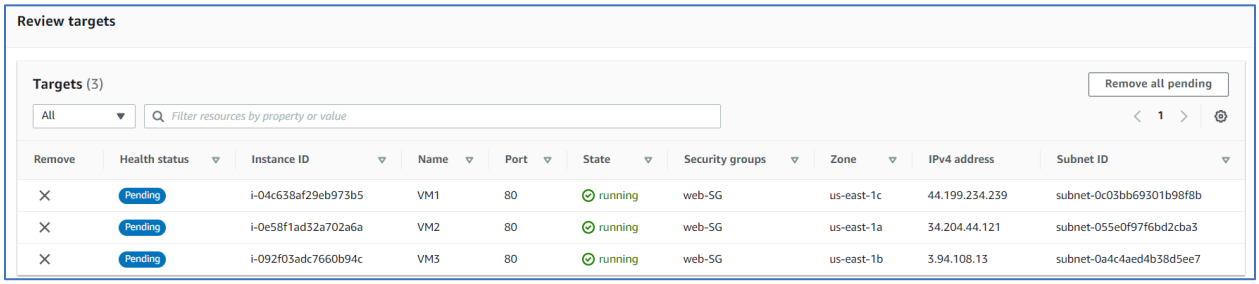
If the webserver does not answer to a number of queries within a **time threshold**, then the elastic load balancer marks the server as unhealthy and it stops sending traffic to it. The default settings indicate that:

- Every 30 seconds a health check is sent by the elastic load balancer.
- Not responding within 5 seconds amounts to a failed health check.
- Missing 2 consecutive health checks makes the target to be considered unhealthy.
- Responding 5 consecutive checks moves a target from unhealthy to healthy status.
- 200 is the okay code in the HTTP protocol.

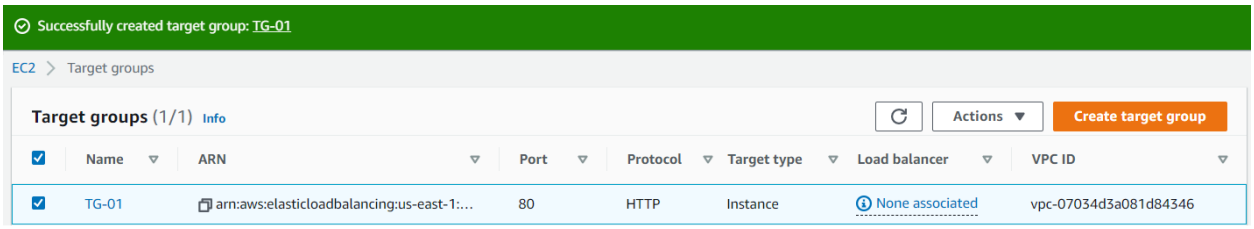


- Proceed to register the targets (the EC2 instances) into the target group but leave them as pending because the elastic load balancer has not been created yet.



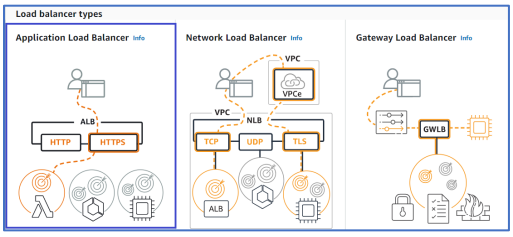


That concludes the creation of the target group. (Notice that there is no load balancer associated yet.)

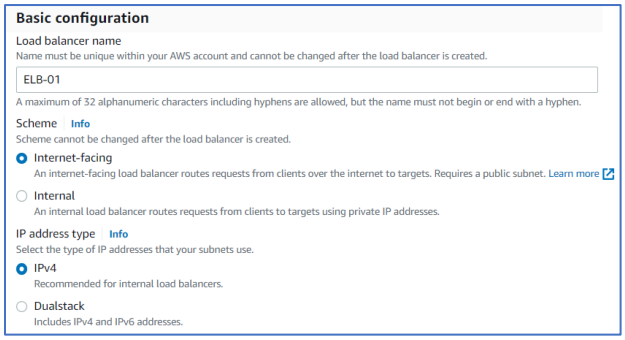


Creation of the elastic load balancer

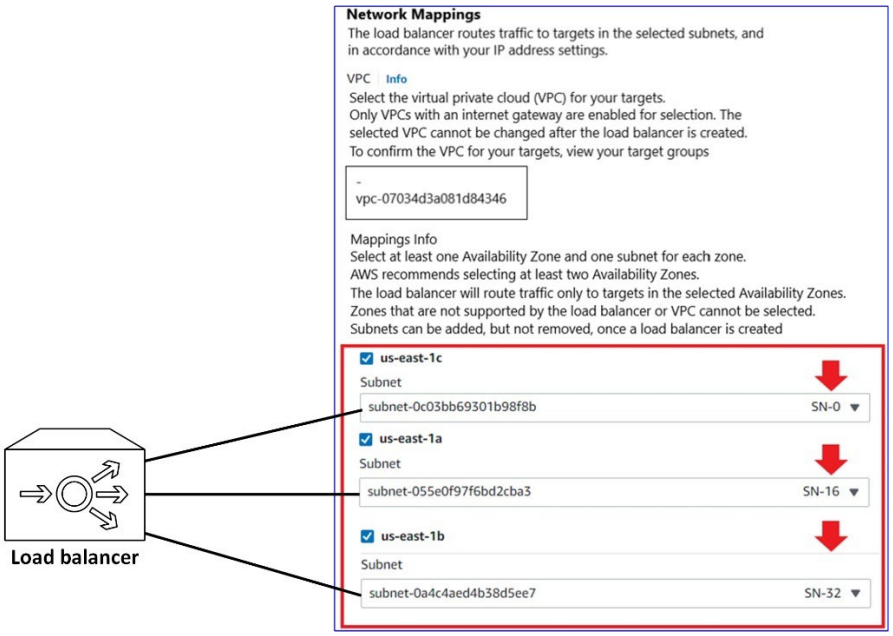
The configuration console offers three types of elastic load balancers. When one of these types is selected, a large number of Application Programmable Interface (API) calls triggers the internal AWS automation system to deploy the virtual resources needed. Among the options, the Application Load Balancer (ALB) is pre-designed to manage the HTTP and HTTPS protocols which is exactly what it is needed to front the webserver cluster.



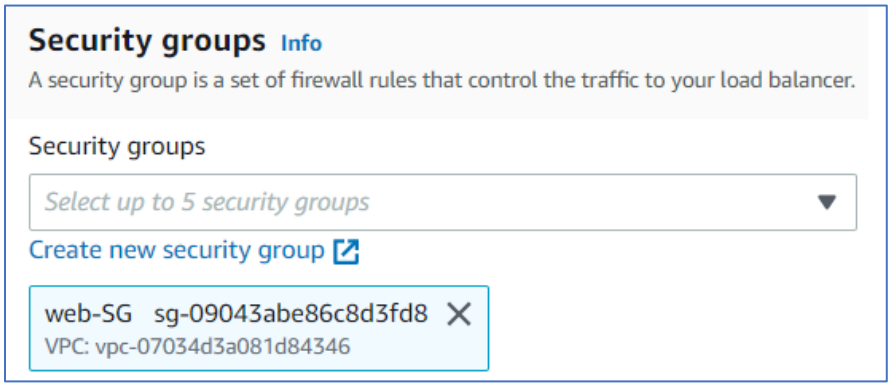
- An ALB can be configured to provide public Internet visibility or only private within the VPC visibility. In the current case, the access required is Internet facing.



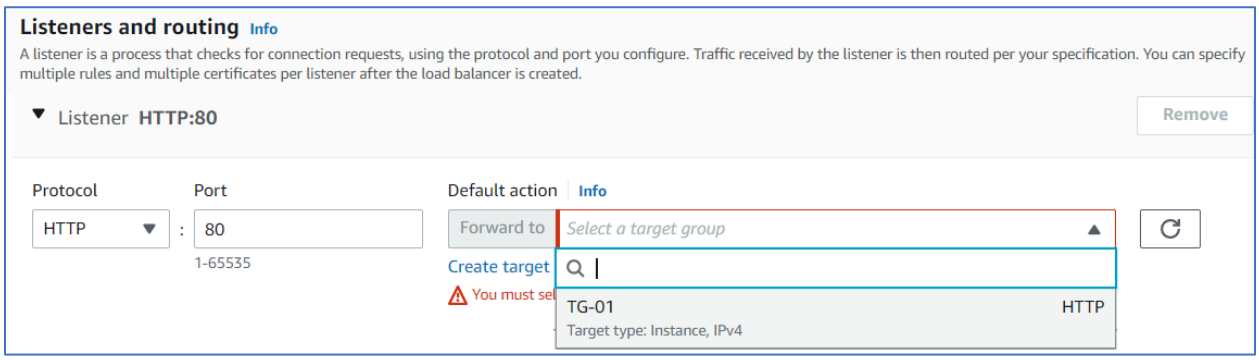
- The ALB must have presence in the same availability zones where the VMs are located.



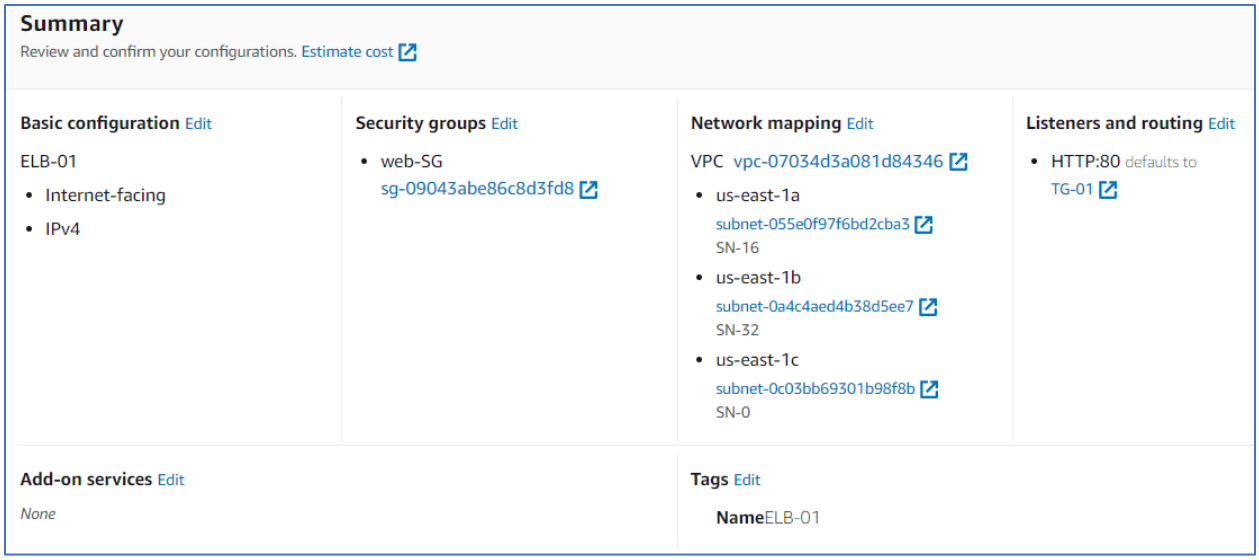
- Choose the same security group used by the webservers.



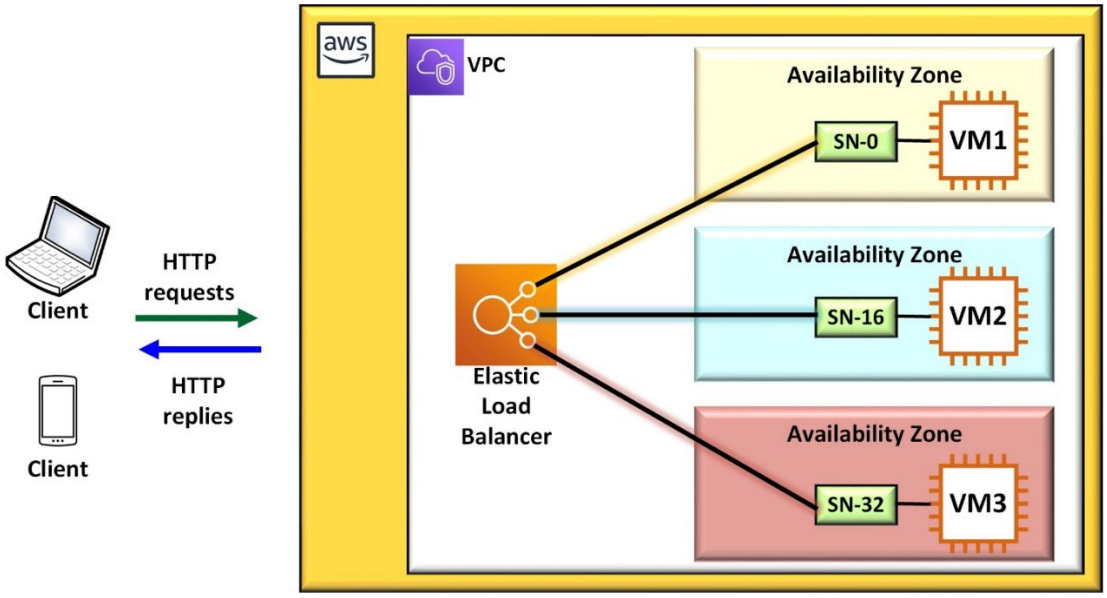
- Select the target group previously configured.



- This is the current configuration of the Application Load Balancer.

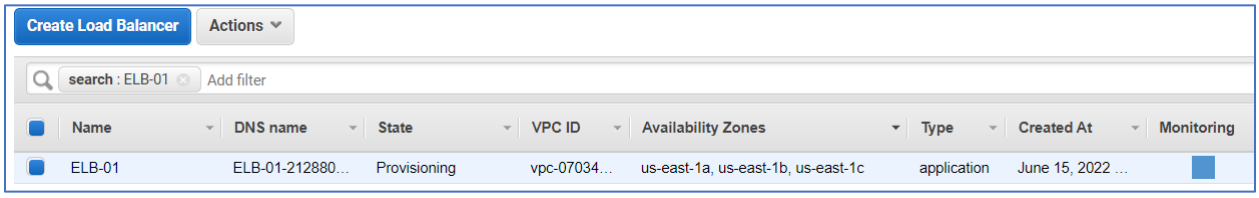


- This is the representation of the final topology.

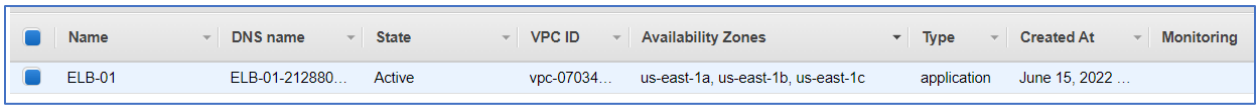


Application Load Balancer serving HTTP traffic to a cluster of three webservers

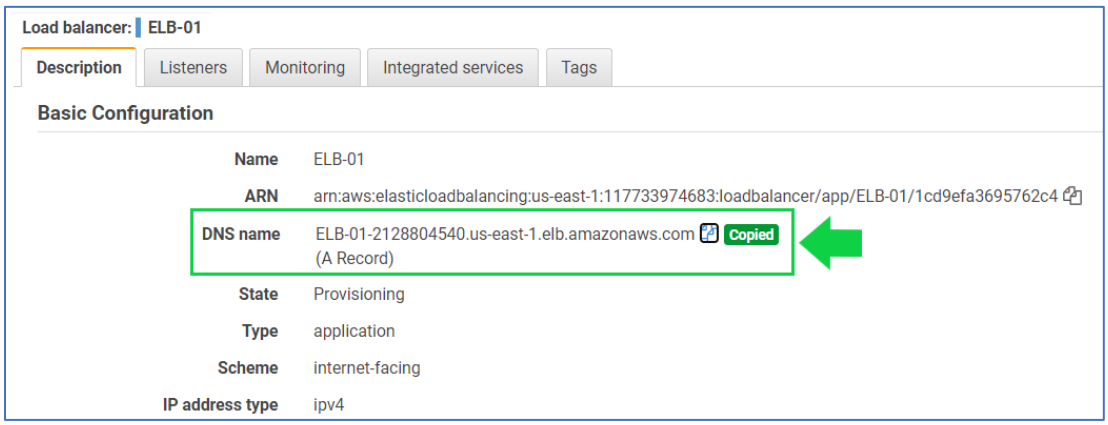
- The load balancer is virtual provisioned by API calls to AWS orchestration systems.



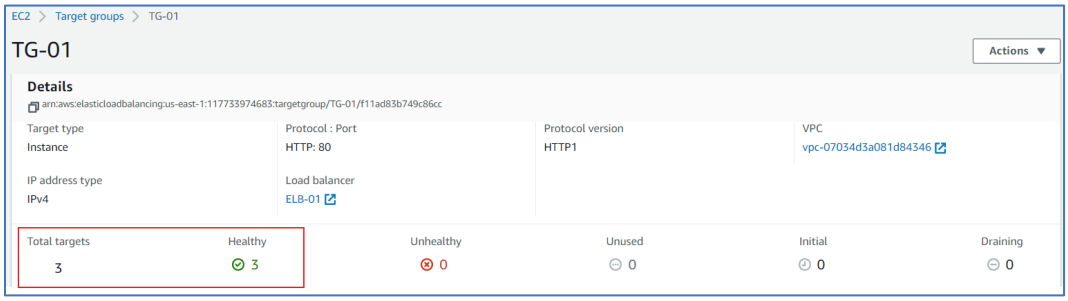
- Until it becomes Active.



The Load Balancer has got a public DNS name **ELB-01-2128804540.us-east-1.elb.amazonaws.com**. This DNS Record is the customer access **service endpoint**.

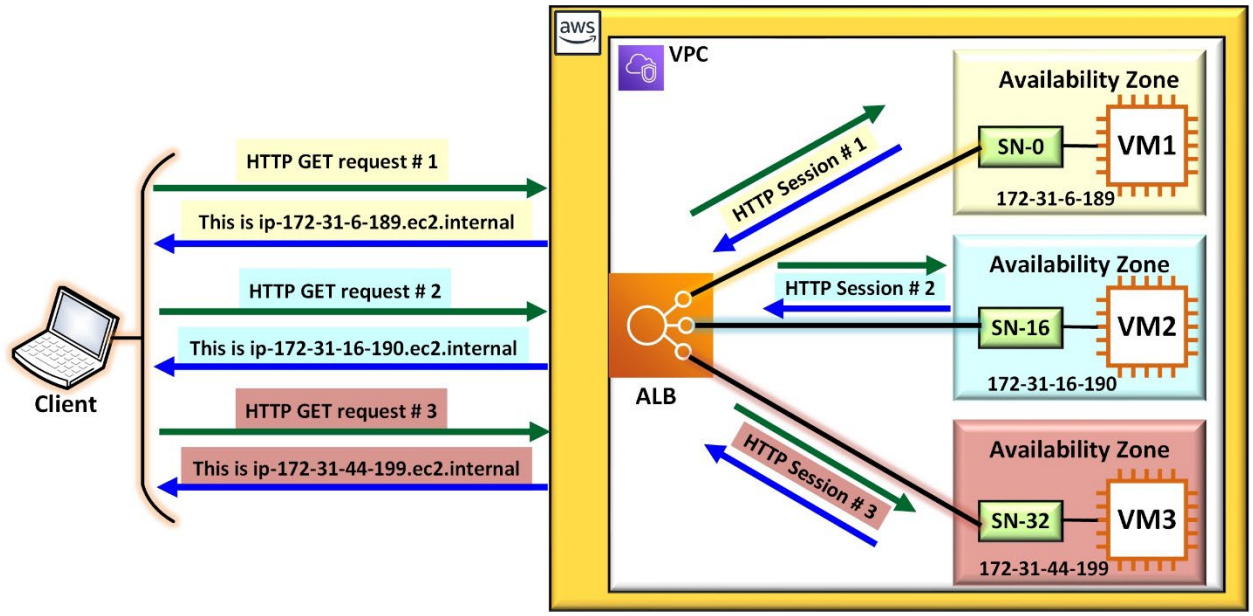


- In the meantime, the target group shows that there are three healthy targets.



- To test, grab the Load Balancer's DNS name and place it in a web browser.
- Use the refresh button to sends new queries.
- The three VMs should be responding at different occasions.

- The three webservers alternatively respond when the load balancer endpoint is used. The load balancer passes the HTTP requests to each server, and then it passes each response back to the customer. In conclusion, this infrastructure is functioning correctly.



The elastic application load balancer in action

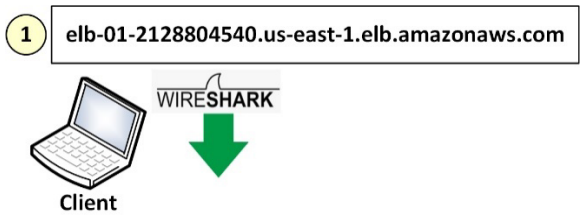
Learning Activity

Further Investigation of the functioning of the Load Balancer

Currently, the load balancer has three interfaces located in three availability zones us-east-1a, us-east-1b, and us-east-1c. That means, three different datacentres sites are involved in this infrastructure. High availability is guaranteed by having location diversity. If one site were to become unavailable, the other sites would still be providing the service.

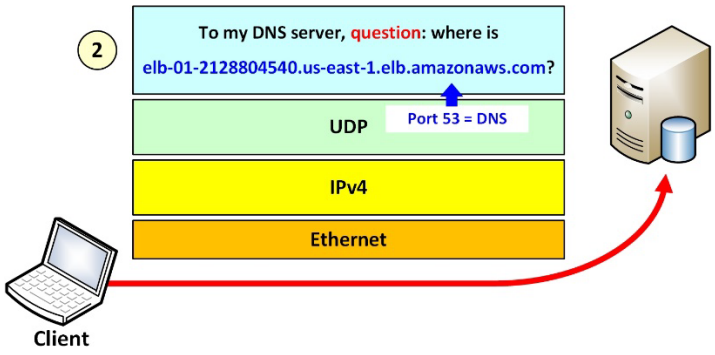
High availability of the elastic load balancer

Let's explore the functioning of the load balancer and at the same time gain some insight about network fundamentals. A protocol sniffer Wireshark was started on the customer home computer to capture the transactions from the very beginning. First, the user types the load balancer DNS name on the browser.



User provides the DNS name to the web browser

However, the PC does not know the IPv4 address of the service. So far, the only information available is the DNS name of the load balancer. Then, the computer proceeds to ask its local DNS server. In this example, the local DNS server belongs to the Internet service provider (Cogeco, Bell, Rogers, etc).

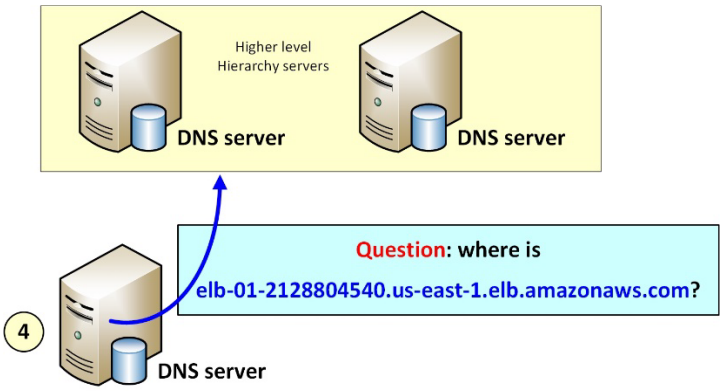


Computer sending a DNS query to a DNS server

The DNS server receives the question and it proceeds to search its name database, but since this is the first time ever that the name is asked, there is no record installed in the cache database.

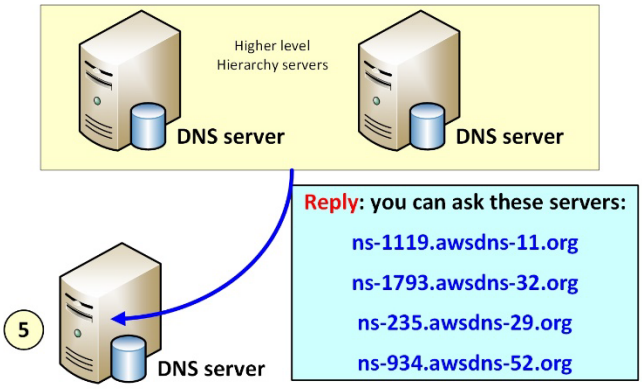
DNS server reads the query and it searches its name database

Since the local DNS server does not have any record of the name, it proceeds to ask other servers. This is a simplified view of the Internet Domain Name System; but fundamentally, if a server does not know a name, it asks other servers higher in a hierarchy.



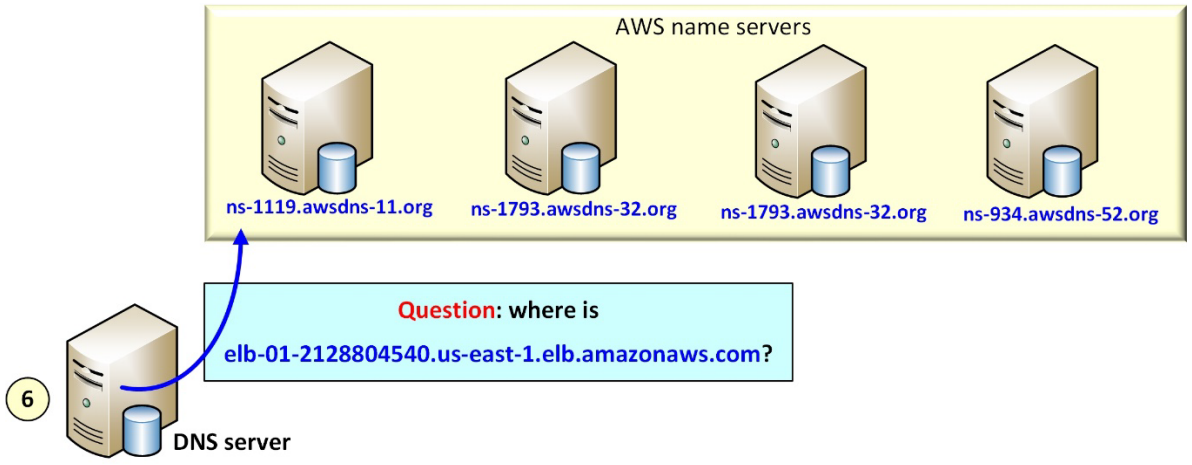
Local DNS server queries other DNS servers for information

A response comes down from a higher-level DNS server, pointing to four different DNS servers that belong to AWS. This information was obtained from the sniffed data, so the server names are real. The prefix **ns** at the beginning of the names indicate that these are Name Servers. Each of these DNS servers contain a database with records about the names that belongs to the AWS domain.



The local server receives the records that point to other Name Servers

Now, the Internet service provider DNS server knows exactly where to find the information, so it sends a query to AWS name servers. In turn, they search their databases looking for an **A record** that matches the name elb-01-2128804540.us-east-1.elb.amazonaws.com. An **Address record** is piece of information that binds the name with an IP address. Since the AWS name servers own the record and the databases, they are called **Authoritative servers**.



The local DNS server asks the Authoritative servers

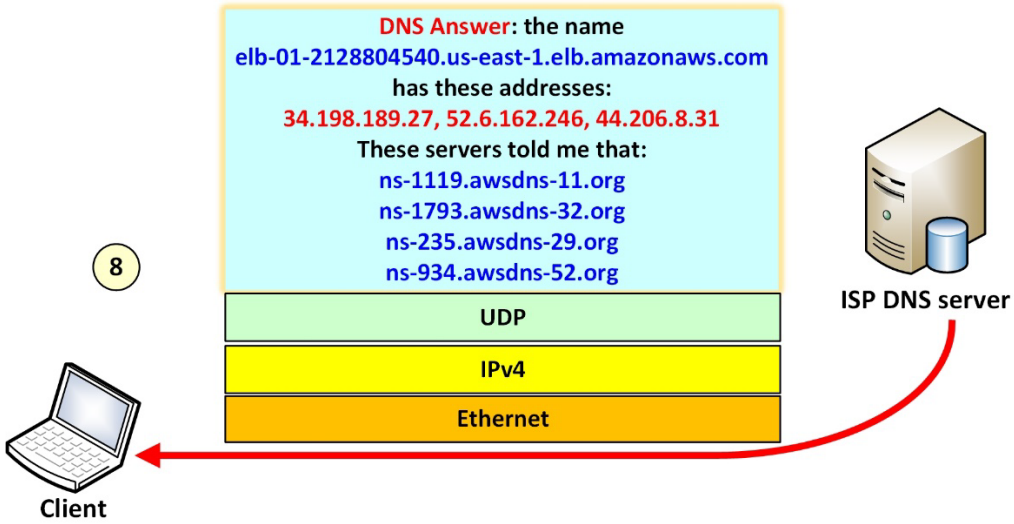
A good question to ask right now is: how did the name elb-01-2128804540.us-east-1.elb.amazonaws.com became an A record in the DNS databases in the first place?

When the customer creates the elastic load balancer, behind the console scene, a large number of API calls trigger orchestration and automation scripts. One of the automated tasks is the assignment of the DNS name in a standard format. The automation system takes the name given by the customer (elb-01 in this example), assigns a unique random number (2128804540) and attach the sub-domain suffix (us-east-1.elb.amazonaws.com). The new name is then automatically registered in the NS server databases.

To continue with the explanation, AWS servers reply to the Internet provider DNS server.

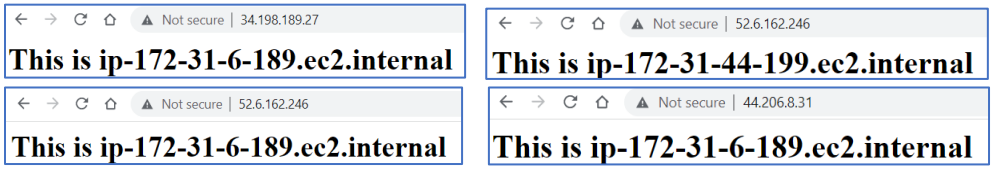
The response comes back from the name servers

The local DNS server stores the records in its name database with the condition that they are temporarily cached. If the information is not requested frequently, then it will be removed out of the database. In a DNS server name database, there are permanent records and temporal records.

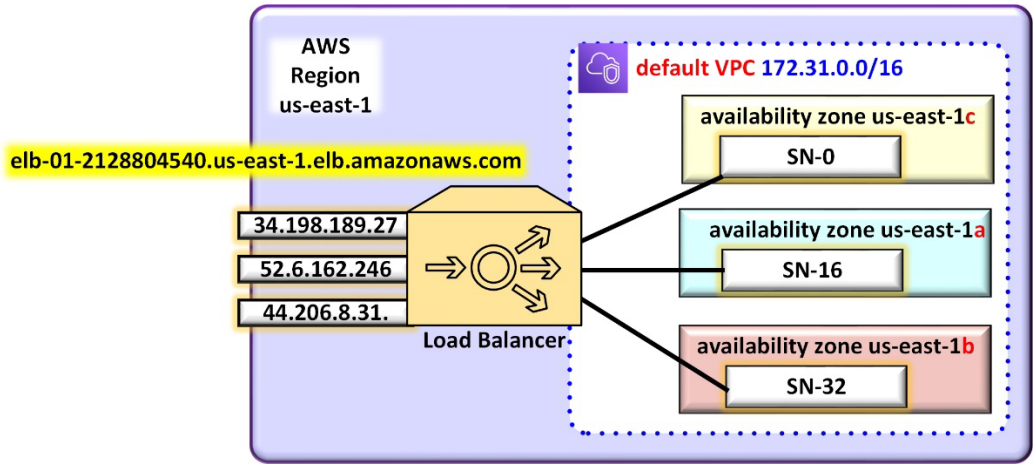


The local DNS server finally answers the question asked by the client

The final response says that the name elb-01-2128804540.us-east-1.elb.amazonaws.com is bound to not one but three IPv4 addresses: 34.198.189.27, 52.6.162.246, 44.206.8.31. In principle, any of these addresses could be used to reach the load balancer. Let's test this concept.

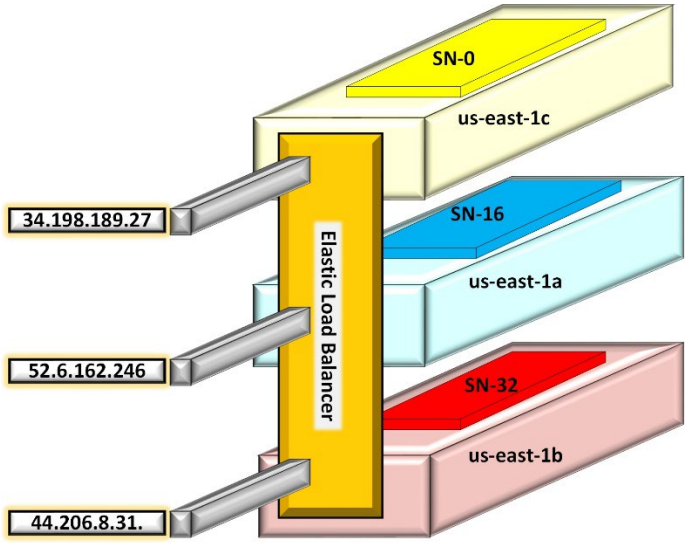


The addresses lead to the webserver clusters. This information suggests the following scenario. The load balancer got three IPv4 addresses from AWS ranges of public addresses.



This load balancer has three public IPv4 addresses

The elastic load balancer is a virtual object instantiated on the datacentres of the three availability zones. Any of the public IPv4 addresses lead to the ELB and consequently to the resources behind it. The following diagram is a free interpretation of the elastic load balancer.



An abstract interpretation of the load balancer infrastructure

Adding a new VM to the current cluster of webservers

A new VM is going to be added to the currently running target group.

- Create a new VM in a different subnet (SN-80 in this example)

Instances (4) Info

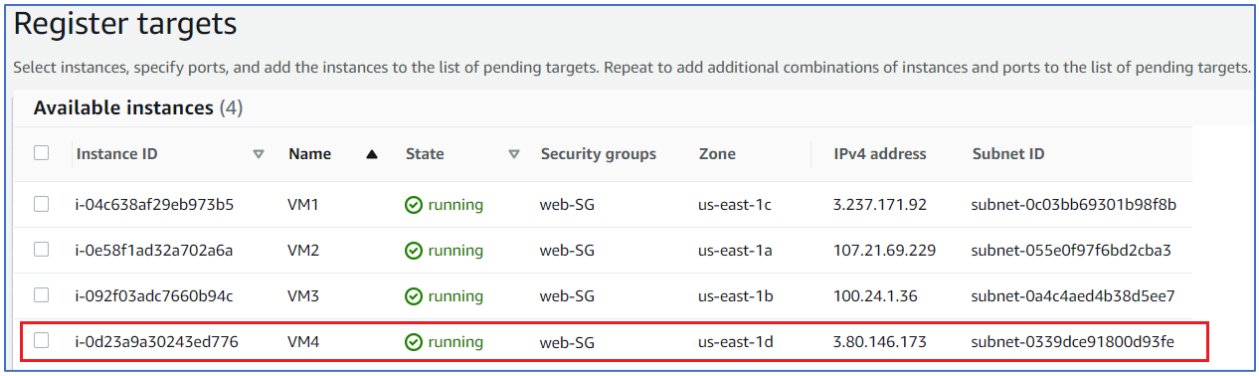
Name	Instance ID	Instance state
VM1	i-04c638af29eb973b5	Running
VM2	i-0e58f1ad32a702a6a	Running
VM3	i-092f03adc7660b94c	Running
VM4	i-0d23a9a30243ed776	Running

- Register the new VM with the Target Group.

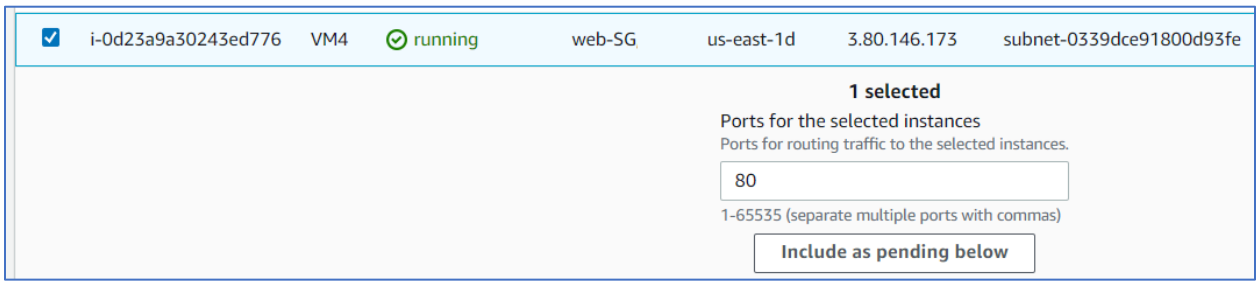
Targets | Monitoring | Health checks | Attributes | Tags

Registered targets (3) Deregister | Register targets

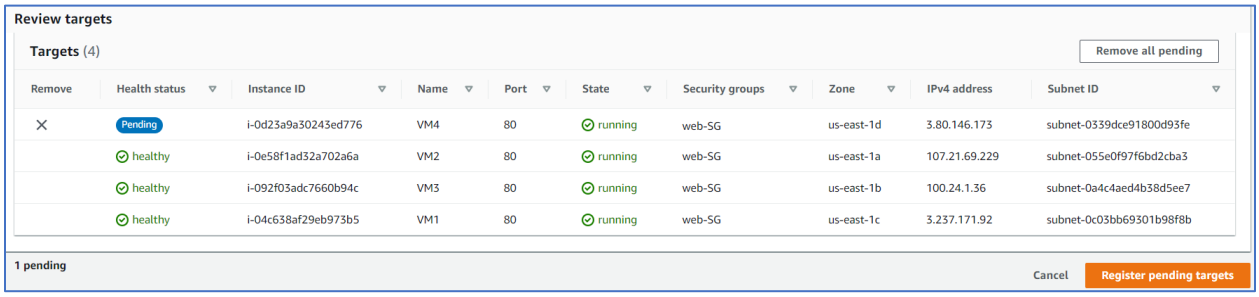
Instance ID	Name	Port	Zone	Health status	Health status details
i-04c638af29eb973b5	VM1	80	us-east-1c	healthy	
i-0e58f1ad32a702a6a	VM2	80	us-east-1a	healthy	
i-092f03adc7660b94c	VM3	80	us-east-1b	healthy	



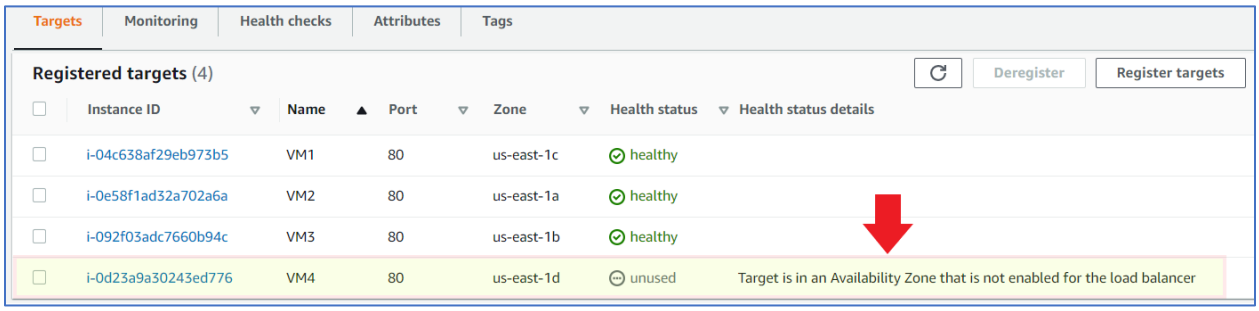
- Include the VM as pending.



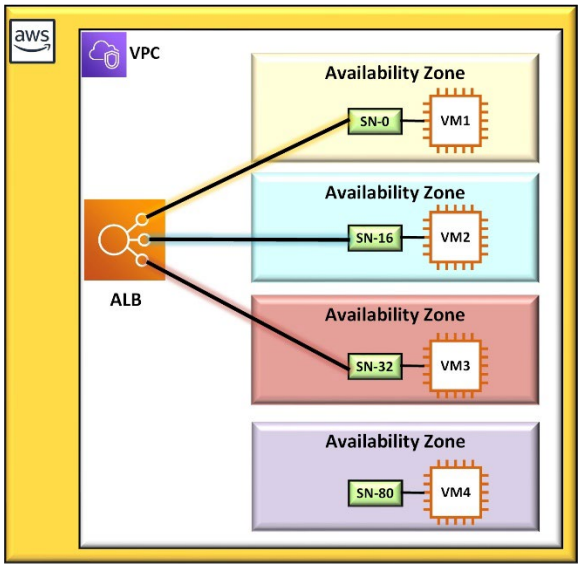
- Verify that it shows in Pending condition under the Health status.
- Try to register the VM with the target group.



However, a warning appears in the target group. It says that the VM is in an Availability Zone where the load balancer has no reach.

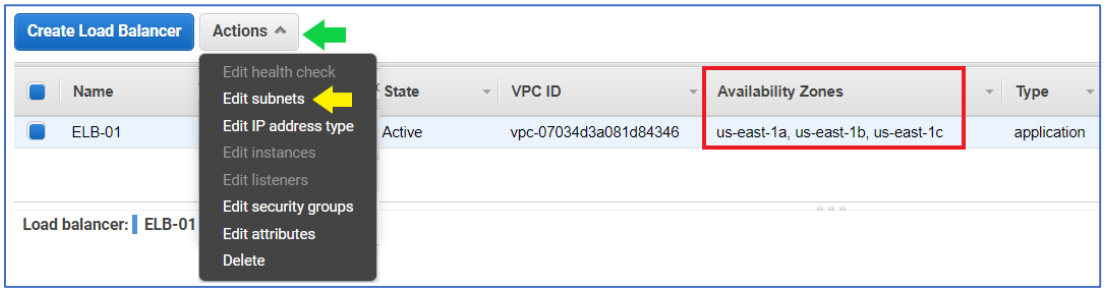


This means that the load balancer does not have a virtual interface in the subnet (SN-80) where the VM is connected. This situation is represented in the following figure.

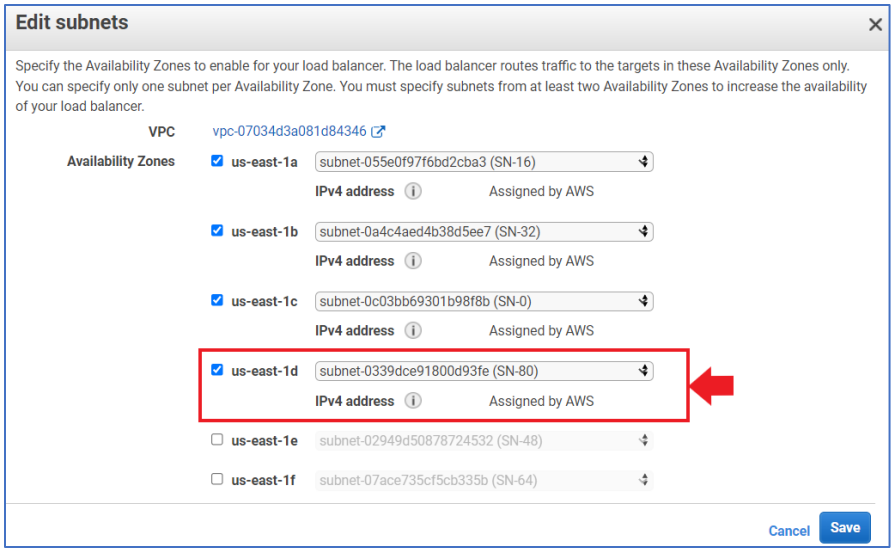


The elastic load balancer does not have an interface to reach the VM

- To fix this problem, edit the load balancer's subnets.



- Add the corresponding subnet.

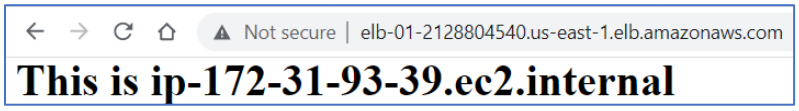


- Verify that the VM is detected by the target group as healthy.

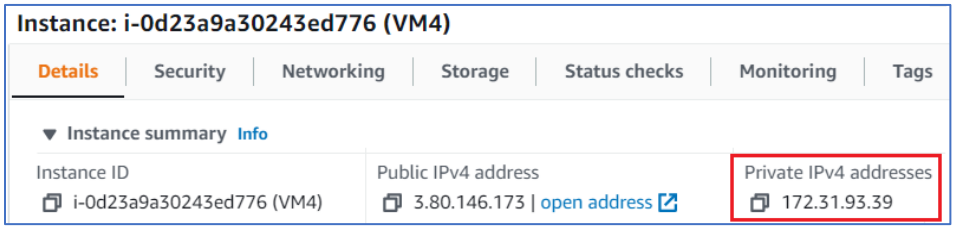
Registered targets (4)

Instance ID	Name	Port	Zone	Health status
i-04c638af29eb973b5	VM1	80	us-east-1c	healthy
i-0e58f1ad32a702a6a	VM2	80	us-east-1a	healthy
i-092f03adc7660b94c	VM3	80	us-east-1b	healthy
i-0d23a9a30243ed776	VM4	80	us-east-1d	healthy

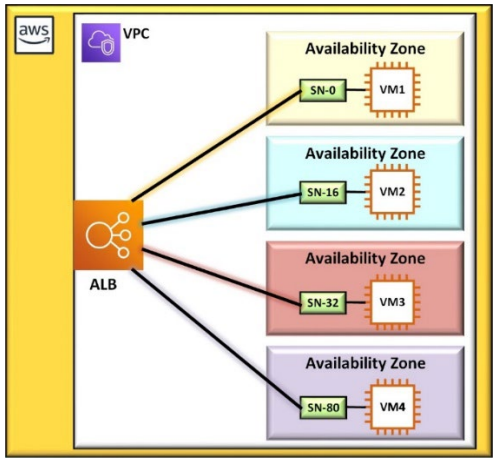
- Verify that the new VM is reachable now via the ELB DNS name.



- Which matches the private IPv4 address of VM4.



- This is the final configuration of the load balancer with its target group.



Elastic load balancer with four EC2 instance webservers

This previous learning activity has investigated the mechanics and principles of the elastic load balancer. Nevertheless, the webservers were not identical thus offering different responses to the clients depending on the server that got the session assigned by the load balancer. Instead, a real webserver application proxied by a load balancer should offer the same experience to the users regardless of the virtual machine that gets the session assigned. This is the topic for the next learning activity.

Learning Activity

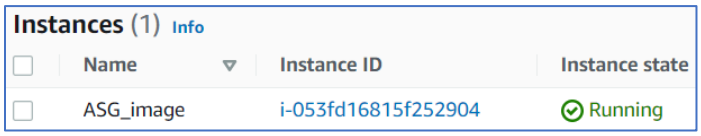
Configuration of an Auto Scaling Group

An **Auto Scaling Group (ASG)** is a set of identical servers that can elastically scale out or in depending on determined conditions. Highly available (HA) applications are based on Auto Scaling Groups that scale out as the requests for service increase and scales in as the requests decrease.

The first step in the configuration of an ASG is to prepare an **image** of an EC2 instance containing the application. This image will be called to dynamically deploy identical copies running in a cluster of EC2 instances. Typically, a load balancer acts as the front proxy to the clients who only interact with the ELB name endpoint. From the view of the clients, it is just one service.

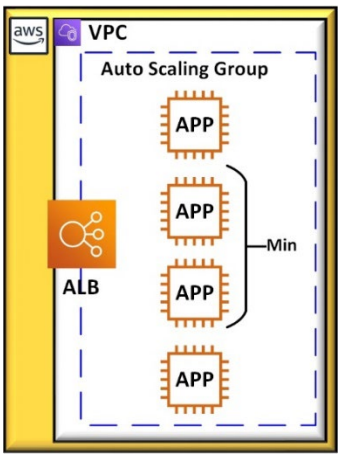
In this demonstration, an EC2 instance hosts a webserver application which is made out of two files; an index.html file and a jpg image. All the servers in the auto scaling group will be identical to this server.

- Create an EC2 instance, AWS AMI, t2.micro, add the key PEM file too.



- Create two files, for example:

```
<html>
    <head>
        <title> TELE20483 </title>
    </head>
    <body>
        <h1> Auto Scaling Group </h1>
        <p>  Cloud Enabled Networks </p>
    </body>
    <img src="AutoscalingGroup.jpg" width=250/>
</html>
```

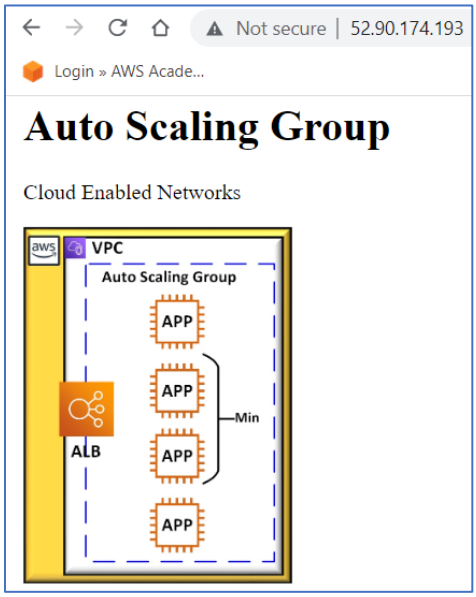


- Upload (scp) the two files to the EC2 instance.

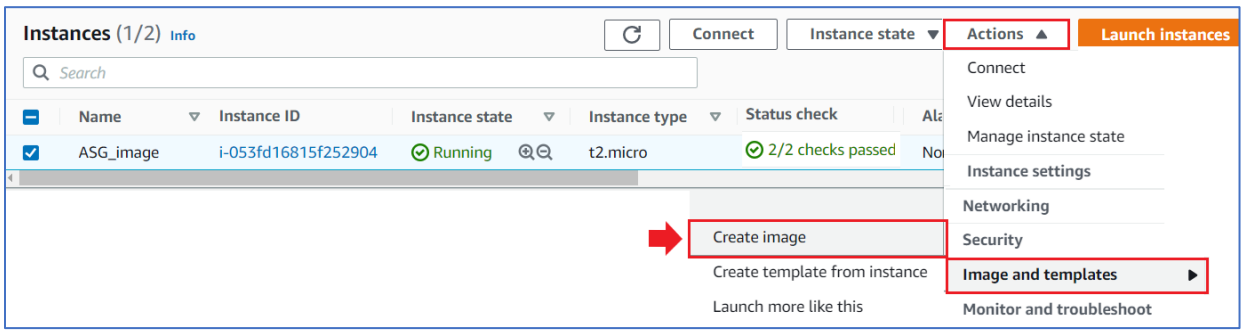
```
scp -i C:\local_files\keyfile.pem
C:\local_files\index.html
C:\local_files\AutoscalingGroup.jpg
ec2-user@public-ip-address:/home/ec2-user
```

```
[ec2-user@ip-172-31-89-25 ~]$ ls
index.html  AutoscalingGroup.jpg
[ec2-user@ip-172-31-89-25 ~]$ sudo mv index.html /var/www/html
[ec2-user@ip-172-31-89-25 ~]$ sudo mv AutoscalingGroup.jpg /var/www/html
[ec2-user@ip-172-31-89-25 ~]$ sudo systemctl restart httpd
[ec2-user@ip-172-31-89-25 ~]$ sudo systemctl enable httpd
```

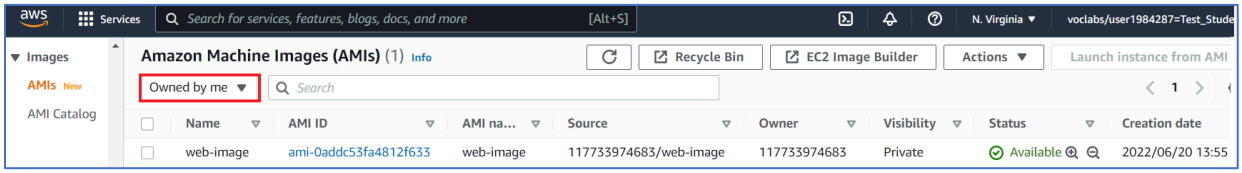
- Test that the webserver is functioning as intended.



- Create an IMAGE out of the EC2 instance.



- The new image will appear under the category “Owned by me”. Give it a name.

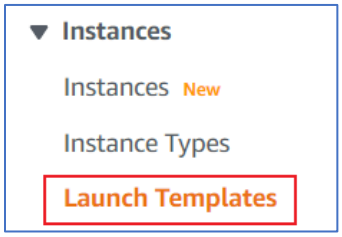


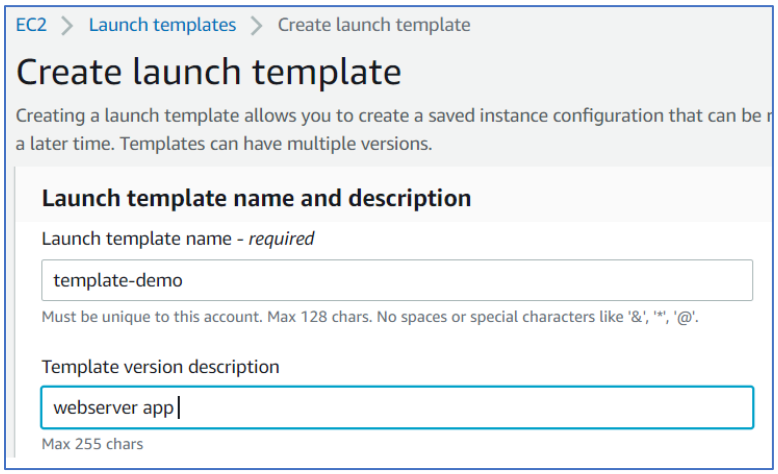
- Any new EC2 instance created after this image, will be identical to the original.
- (The running EC2 instance is no longer needed and it can be terminated now.)

Launch Template

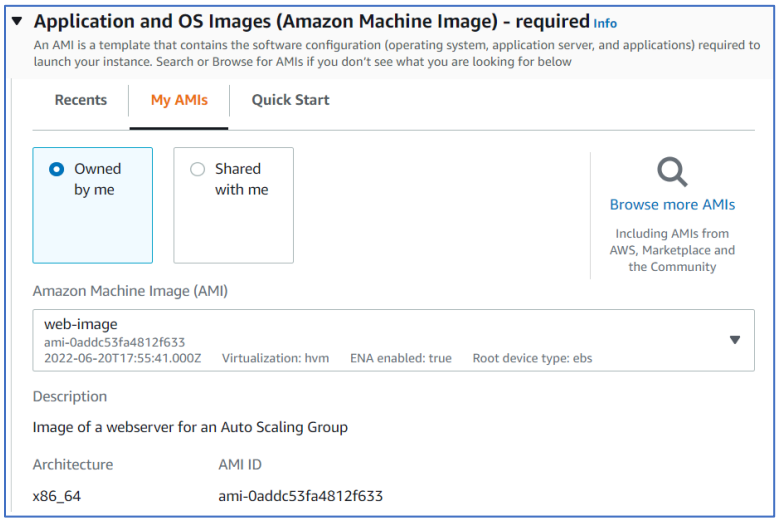
A launch template is a configuration with the specification of the characteristics of the EC2 instances such as the minimum and maximum number of instances, their desired capacity, the configuration of the instance, the networking parameters, and the security group.

- Launch a Template.

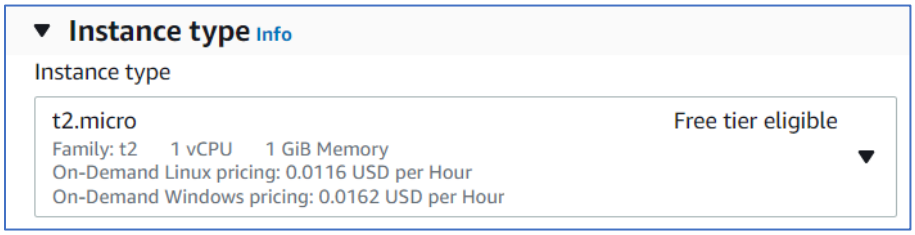




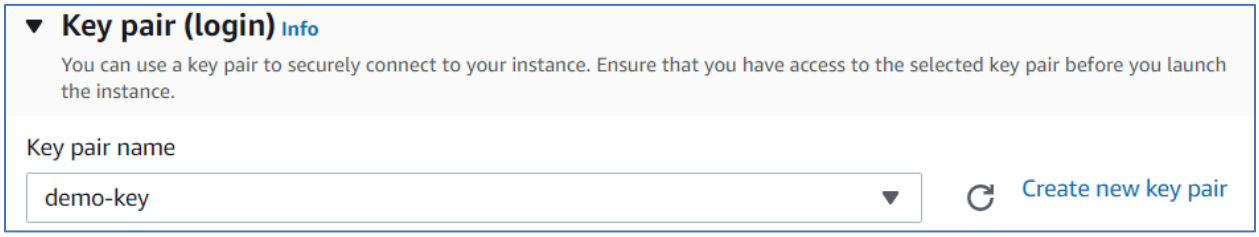
- Choose the saved image under Owned by me.



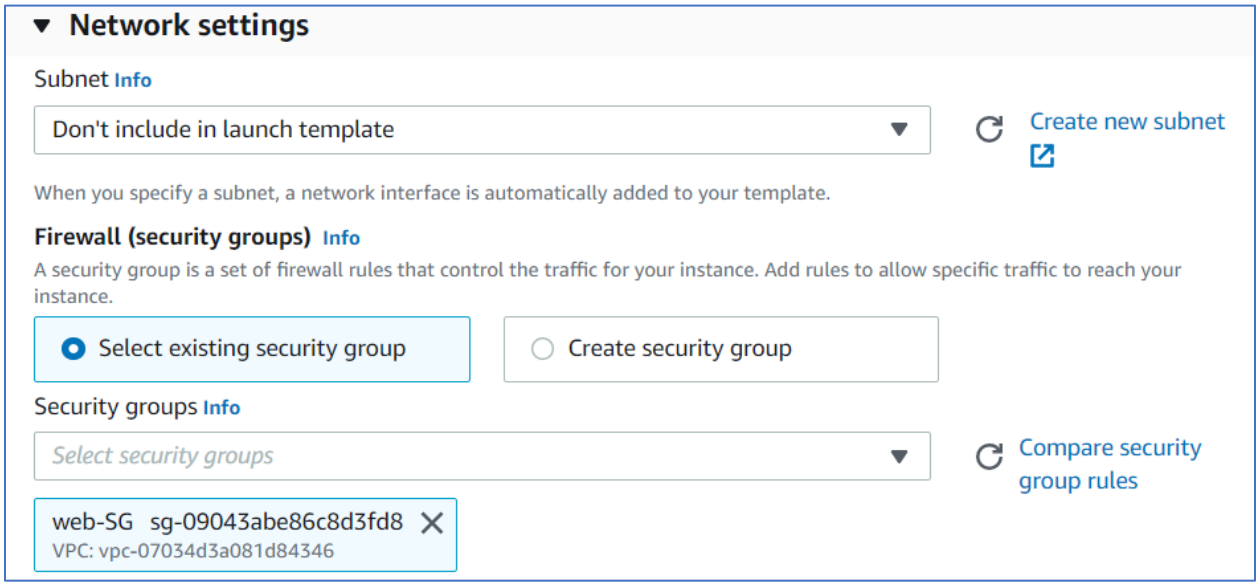
- The virtual chassis can be changed (scaling up or down). In this case, the same t2.micro is chosen.



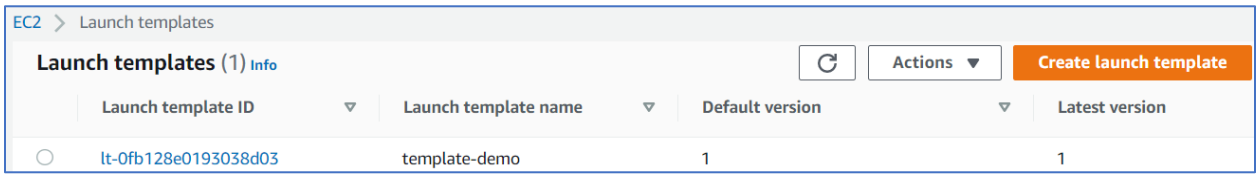
- The same key pair that was used to create the image is required since the corresponding public key is installed in the image.



- The security group allows SSH and HTTP.



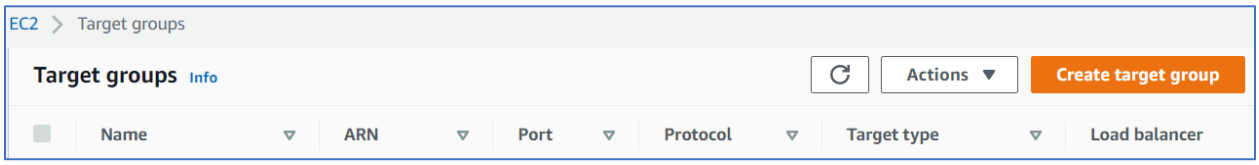
- Create the Launch Template

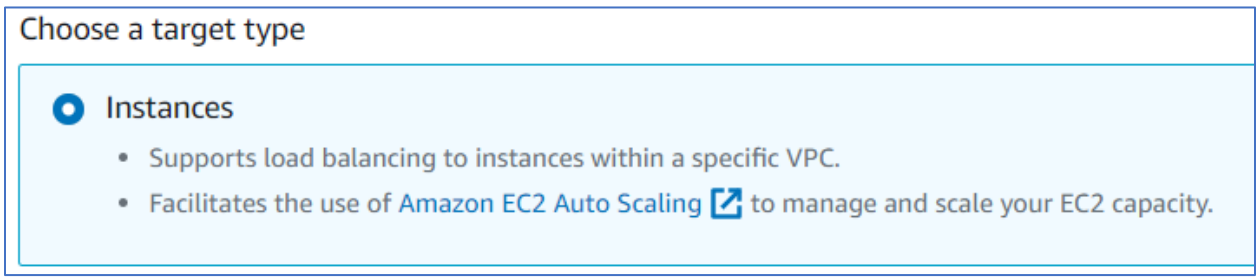


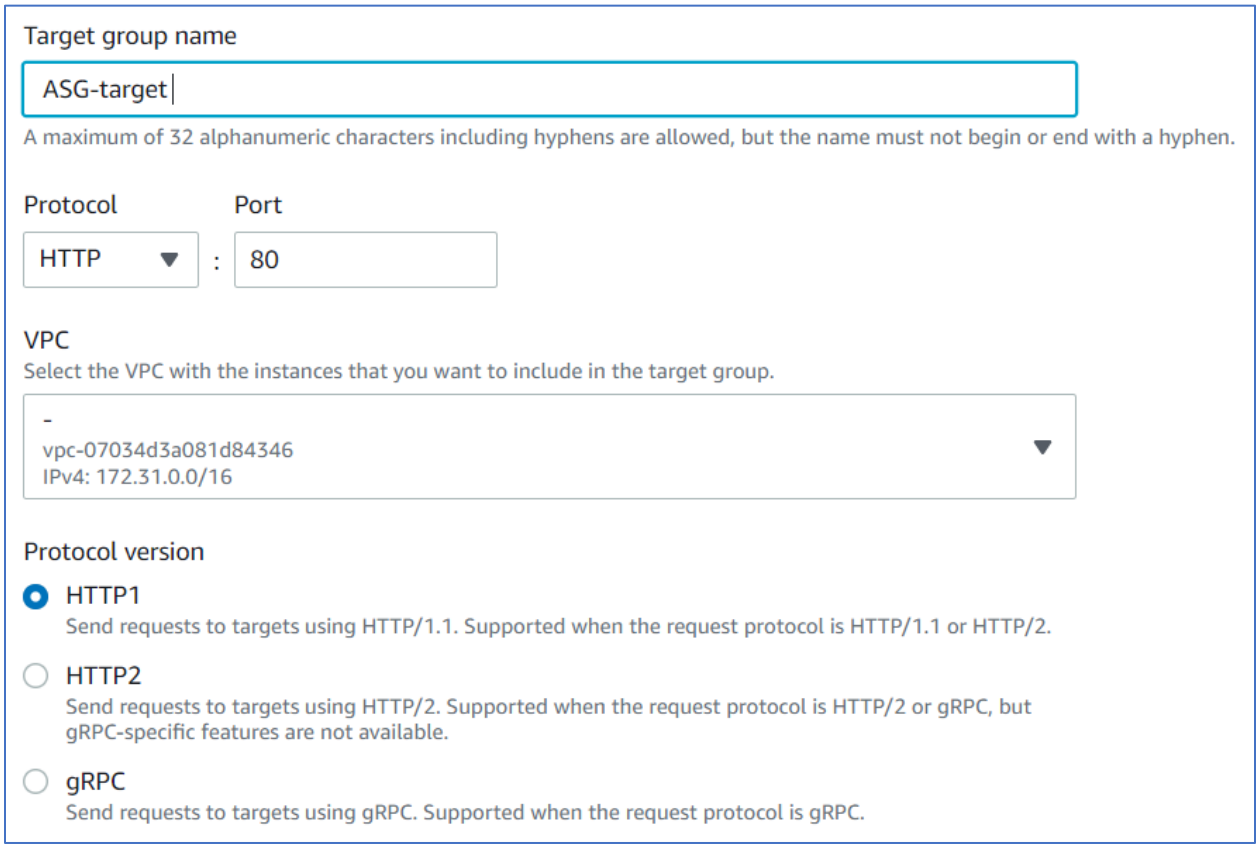
Target Group

The EC2 instance members of the Auto Scaling Group will also be part of a target group for the elastic load balancer to distribute the workload.

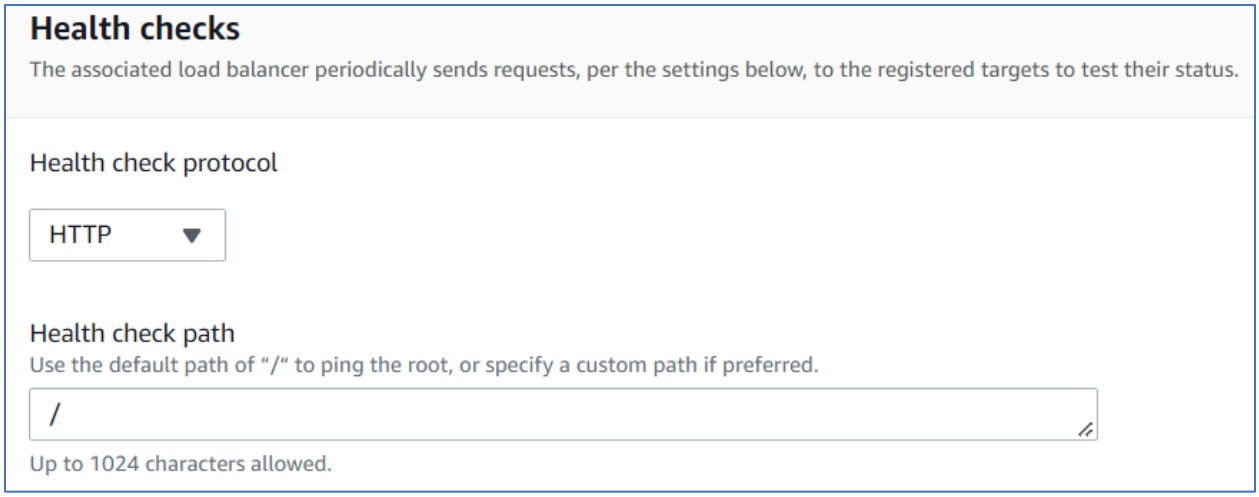
- Create a target group.



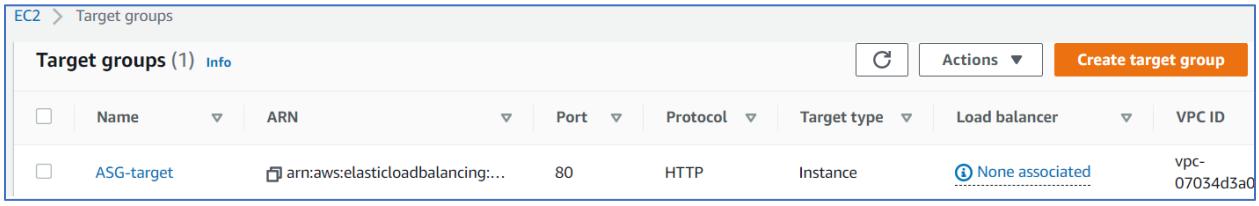




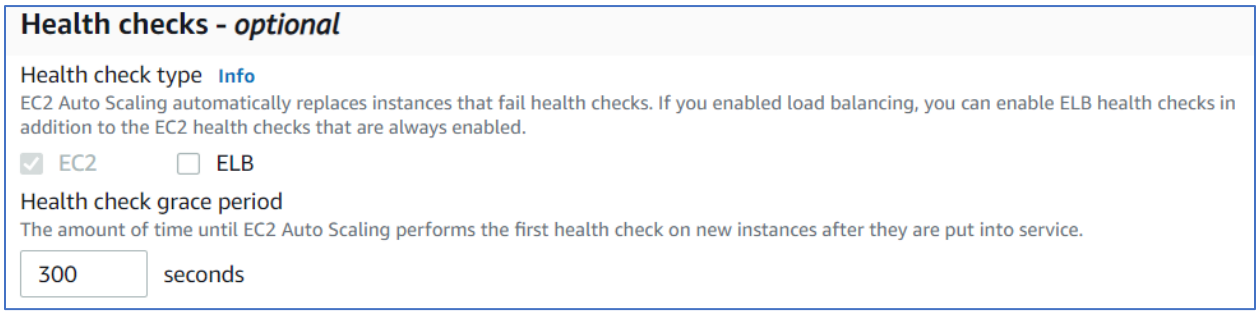
- The health checks will be directed to the TCP port 80 of the webservers root.



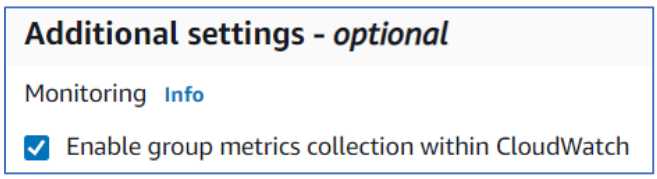
- The target group is created even though there are no EC2 instances added as target pending. This makes sense since the instances of the auto scaling group have not been deployed yet.



- When the EC2 instances fail the health checks, they are automatically replaced.

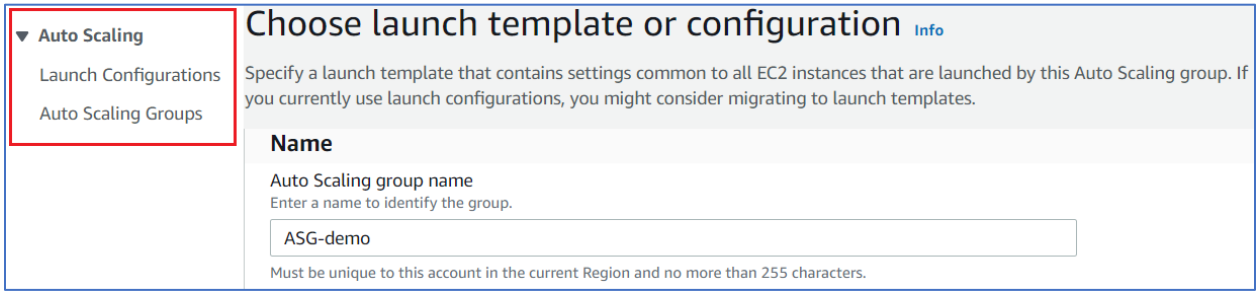


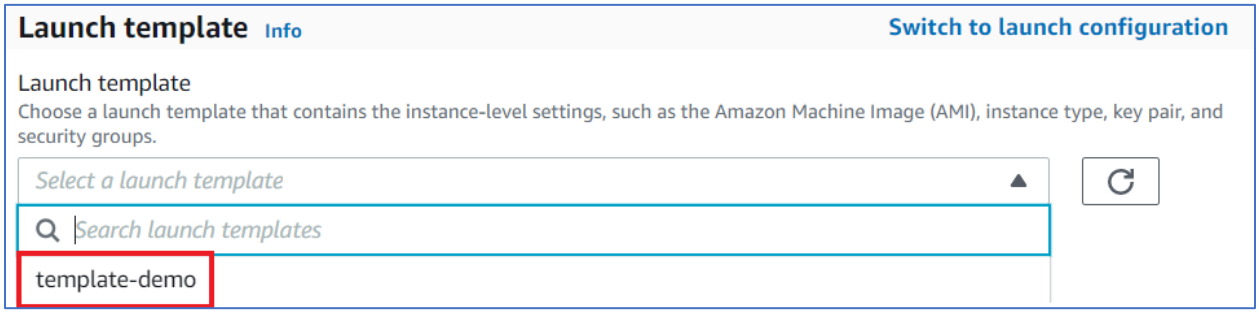
- CloudWatch is a monitoring service that collects metrics about the members of the target group.



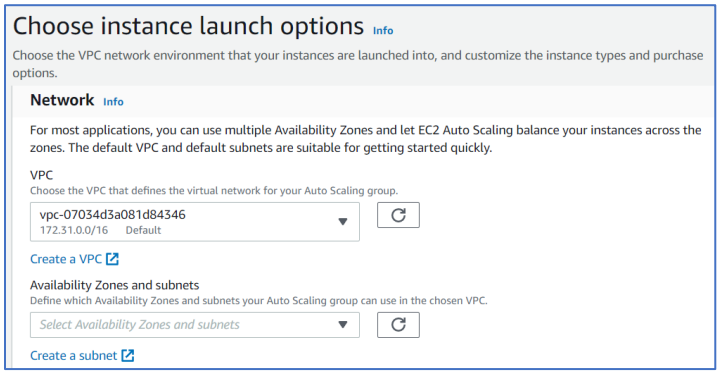
Auto Scaling Group

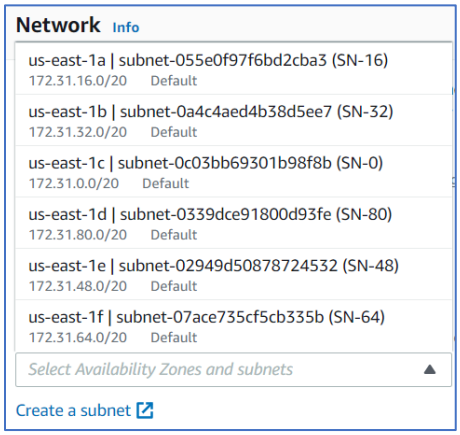
- Choose the Launch Template previously created.

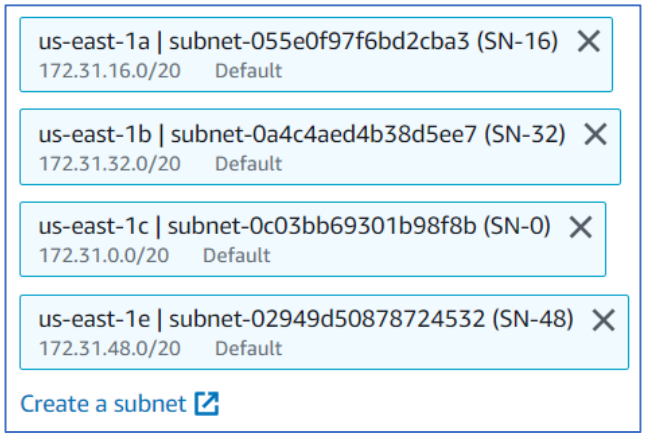




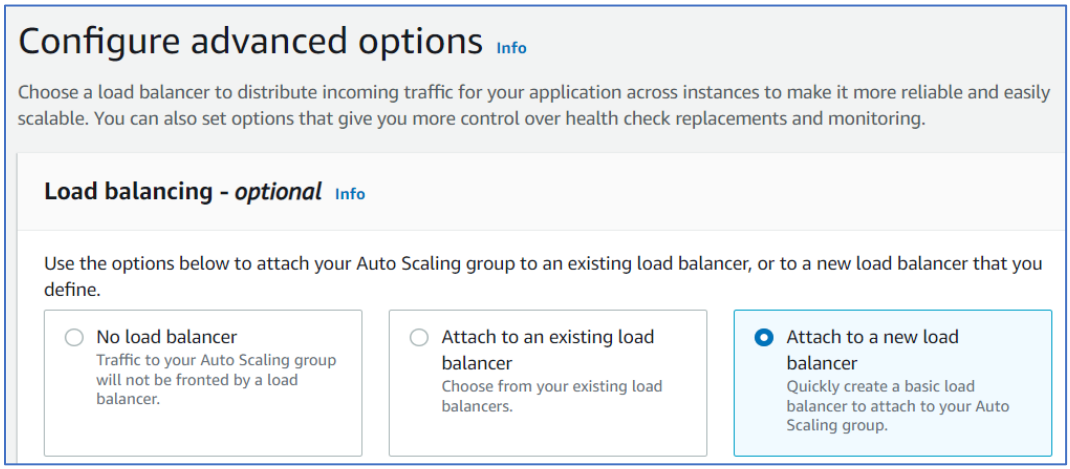
- Choose the availability zones where the EC2 instances will be deployed. In this example, four subnets are chosen.



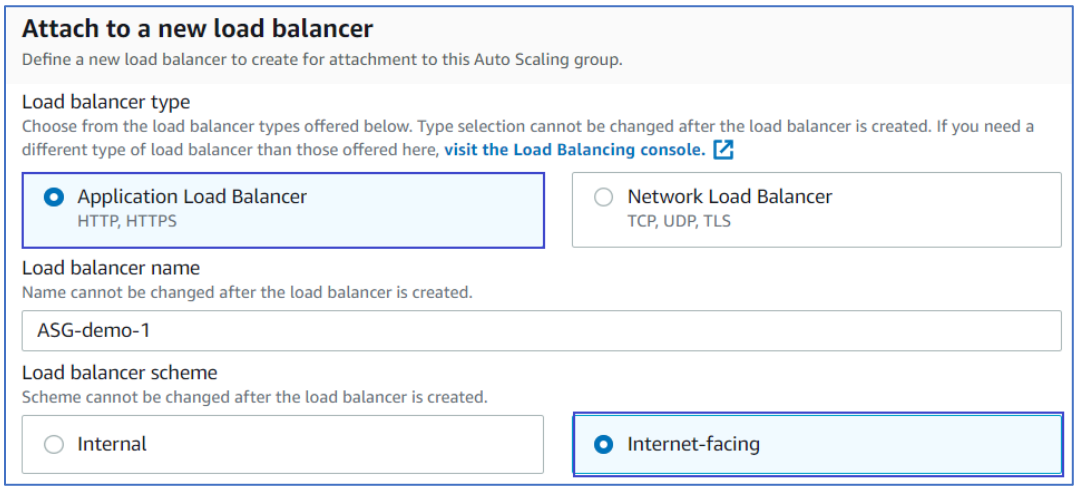




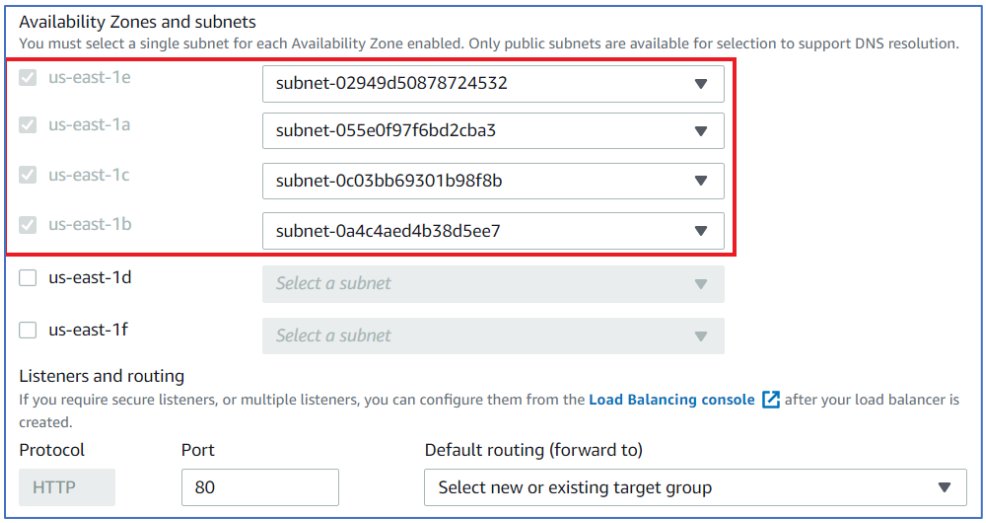
- A new elastic load balancer will front the Auto Scaling Group.



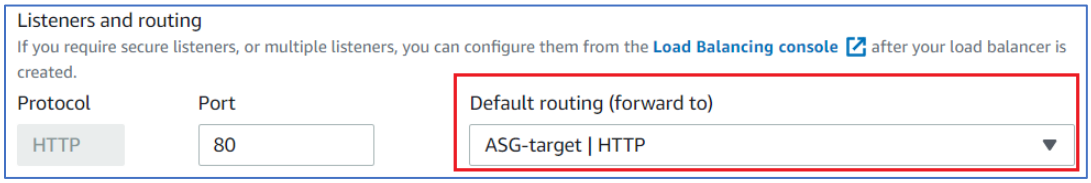
- Application Load Balancer (for web traffic HTTP and HTTPS).



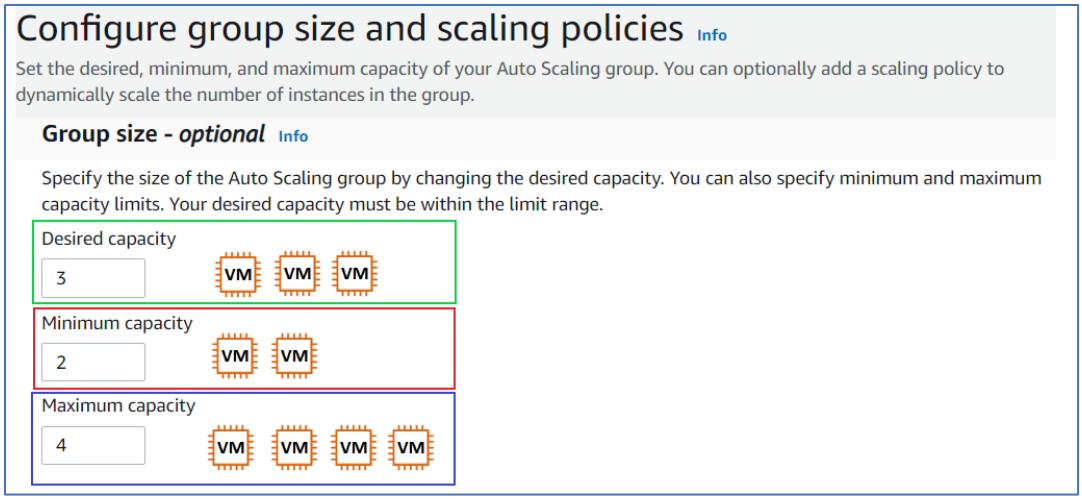
- The load balancer must have interfaces in the same subnets where the EC2 instances reside.



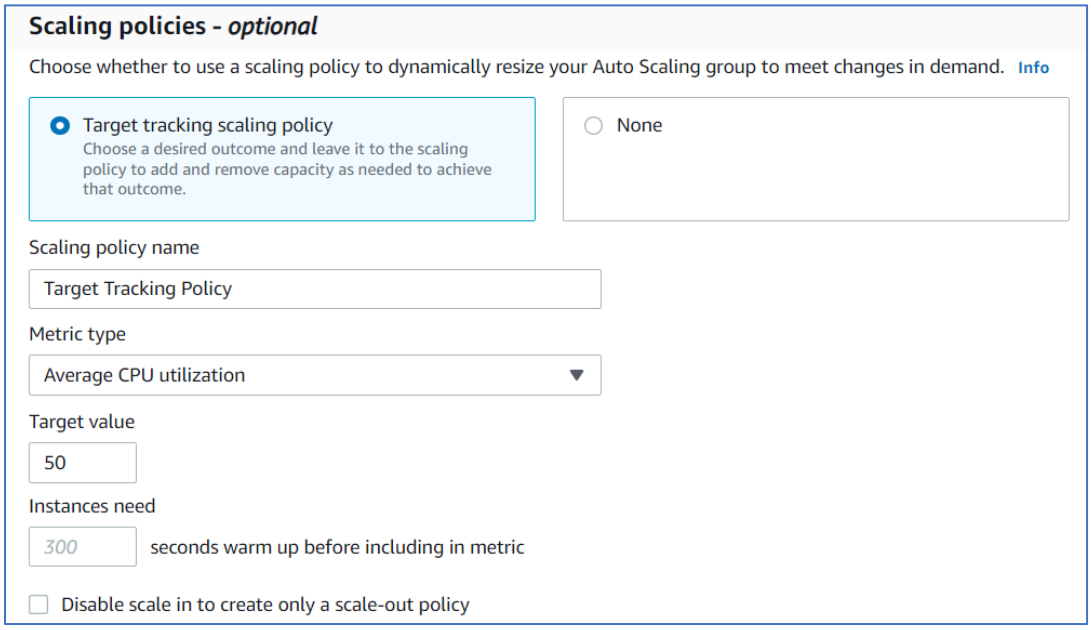
- Select the target group previously created.



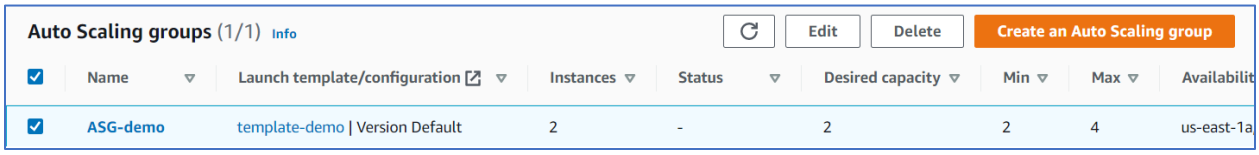
The auto scaling group has three sets: minimum capacity, maximum capacity and desired capacity. In this example, two VMs is the lowest number of EC2 instances that the ASG will be running. The maximum number of EC2 will be trigger by a metric condition.



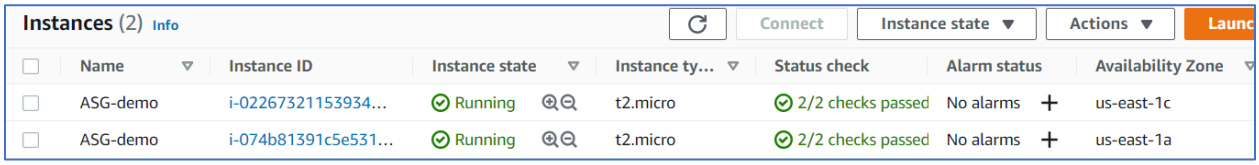
A scaling policy sets the threshold to trigger the creation of a new EC2 within the maximum capacity number. In this example, the metric used is the average CPU usage. If the CPU value exceeds 50%, a new EC2 instance will be added.



- Create the Auto Scaling Group.

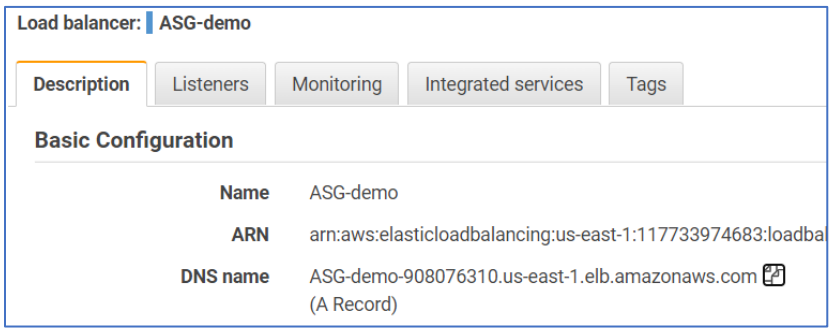


- Verify in the EC2 dashboard if there are new EC2 instances created.

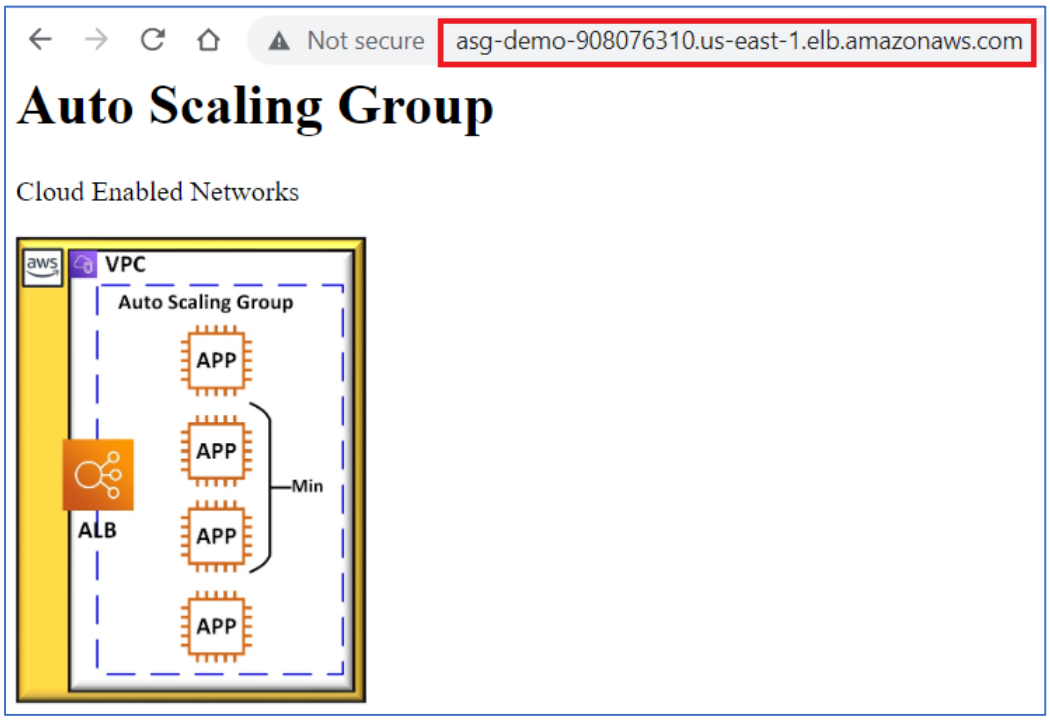


In this case, two new, identical EC2 instances were created automatically.

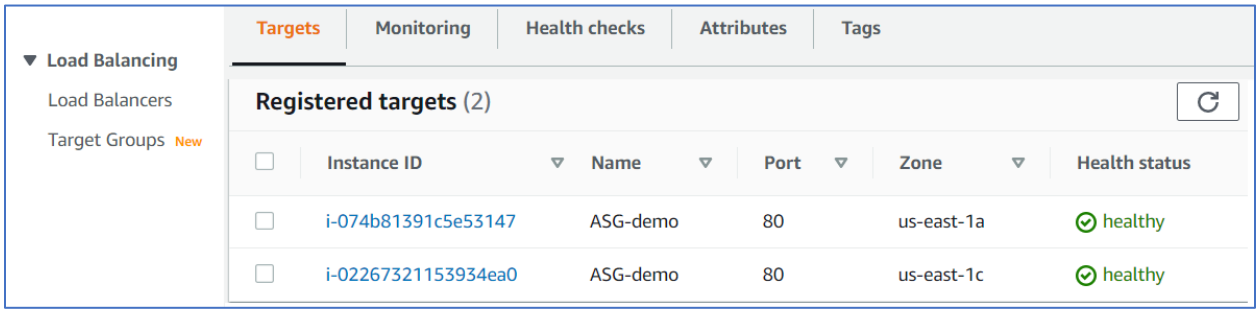
- Grab the DNS name of the Elastic Load Balancer.



- Test the service.



- Notice that the target group got two targets (the EC2 instances) registered automatically.



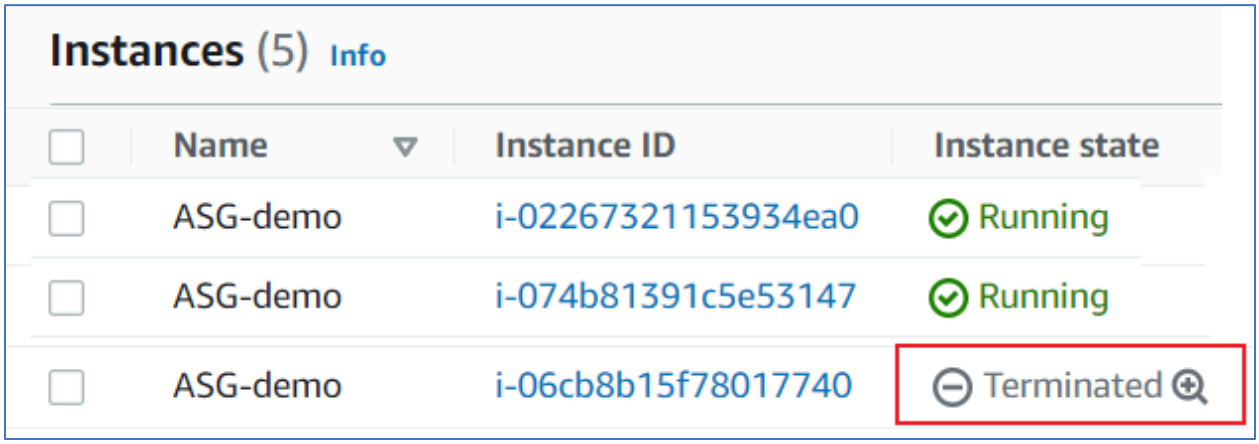
- SSH into one of the EC2.
- Install the following packages.

```
[ec2-user@ip-172-31-4-206 ~]$ sudo amazon-linux-extras install epel -y
[ec2-user@ip-172-31-4-206 ~]$ sudo yum install stress -y
```

- Run a stress test of the CPU.

```
[ec2-user@ip-172-31-4-206 ~]$ sudo stress –c  4
stress: info: [3591] dispatching hogs: 4 cpu, 0 io, 0 vm, 0 hdd
```

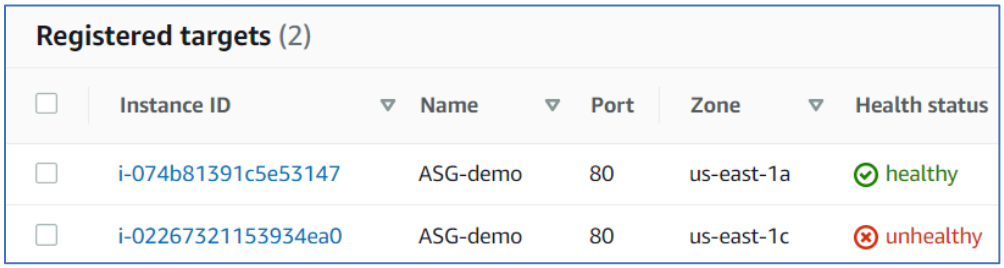
- After a while, the CPU average utilization surpasses 50% which is the threshold set in the scaling policy. The EC2 instance becomes unhealthy and a new one is created to replace it. The older one gets terminated.



- Run another test, disable the HTTP web service.

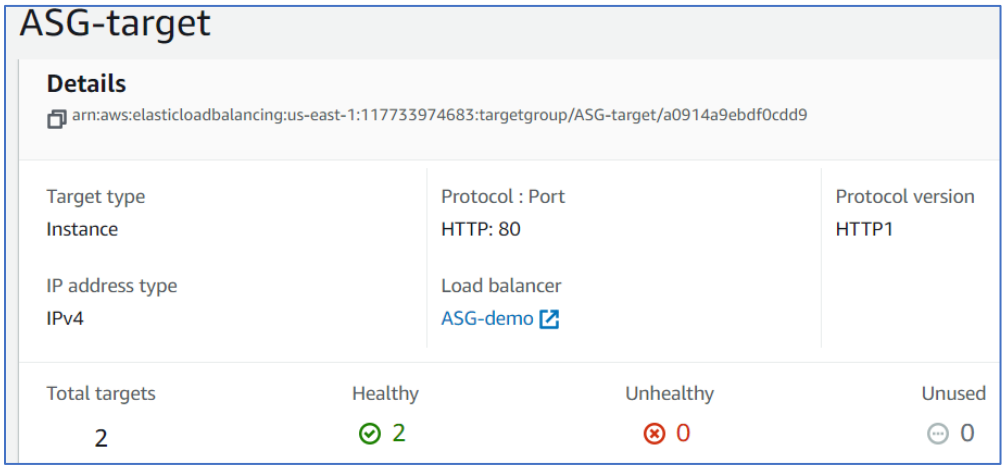
```
[ec2-user@ip-172-31-4-206 ~]$ curl 127.0.0.1
<html>
  <head>
    <title> TELE20483 </title>
  </head>
  <body>
    <h1> Auto Scaling Group</h1>
    <p>  Cloud Enabled Networks </p>
  </body>
  <img src="AutoscalingGroup.jpg" width=200/>
</html>
[ec2-user@ip-172-31-4-206 ~]$ sudo systemctl stop httpd
[ec2-user@ip-172-31-4-206 ~]$ curl 127.0.0.1
curl: (7) Failed to connect to 127.0.0.1 port 80 after 0 ms: Connection refused
```

- Notice that the EC2 becomes unhealthy under the target group panel.

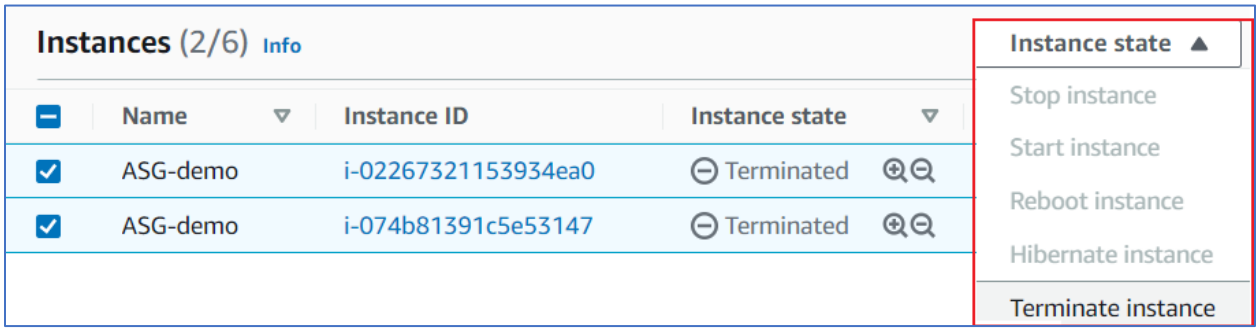


- Restart the service and the EC2 becomes healthy again.

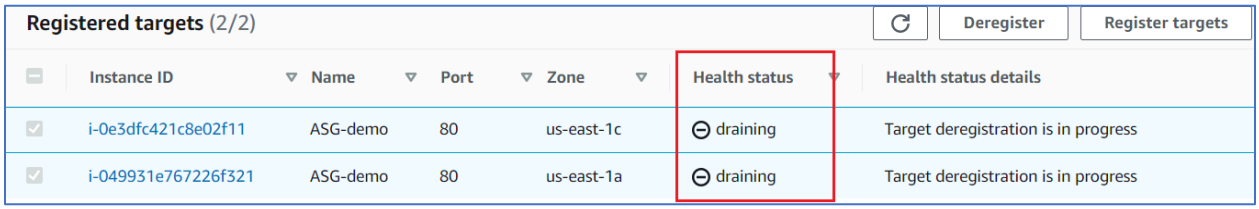
```
[ec2-user@ip-172-31-4-206 ~]$ sudo systemctl restart httpd
```



- Run another test. Terminate both EC2 instances.



The target group shows that the terminated EC2 instances are not eliminated immediately. Instead, the VM are placed in a "draining" state. This is to allow for the traffic of potential sessions to be orderly completed and closed.



The Auto Scaling Group was set to have a minimum number of two EC2 instances. This test is meant to verify that the ASG will try to comply with the minimum requirement which it does since it brings up two new EC2 instances as the next snippet demonstrates.

References

[1] Mozilla Developers. Accessed. June 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

[2] Derek DeJonghe. NGINX Cookbook. 2019. O'Reilly.

Chapter 9

Cloud Elastic Object Storage

Description

The purpose of this section is to provide the student with the fundamentals of object storage in the cloud specifically using the AWS simple storage service S3. Object storage is a valuable service to organizations that need to store non-database data such as pictures, videos, audio clips, static webpages, log information, and in general any kind of data that does not have a hierarchical file system. Object storage is a fundamental part in the deployment of microservices where the intelligence is running in apps while the objects are served from the storage system endpoint.

Learning Outcomes

- Write and run basic scripts to deploy resources and to retrieve resource information from the cloud environment.
- Deploy elastic virtual computers in a cloud networking environment.
- Configure webservers on elastic virtual computers.
- Analyze an object storage system.
- Configure elastic object data storage in the cloud.
- Implement virtual endpoints to access cloud storage and other resources.

Main concepts

- Types of data storage. Concept of object data storage.
- AWS simple storage service S3. The object storage bucket.
- Permission security system.
- Static webserver.
- Cross-reference CORS.

Activities

- Creating an S3 bucket.
- Uploading an object to the S3 bucket.
- Generate a Bucket Policy to allow public access to a bucket.
- Setting a Bucket Policy to selectively allow public access to certain objects.
- Set a Bucket Policy to allow access from an IPv4 range to PNG objects only.
- Configure a Static Website on an S3 bucket.
- Configure two Static Websites with Cross-Origin Resource Sharing (CORS).
- Interacting with S3 buckets using AWS CLI.
- Interacting with the S3 service using AWS Python SDK.

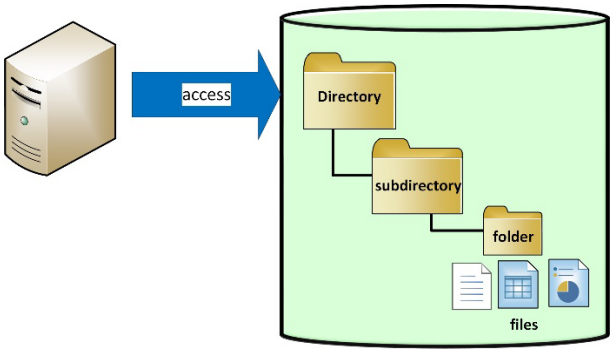
Elastic Storage

Data Storage

Data storage assumes different formats depending on the organization of the data. There are three main types of storage:

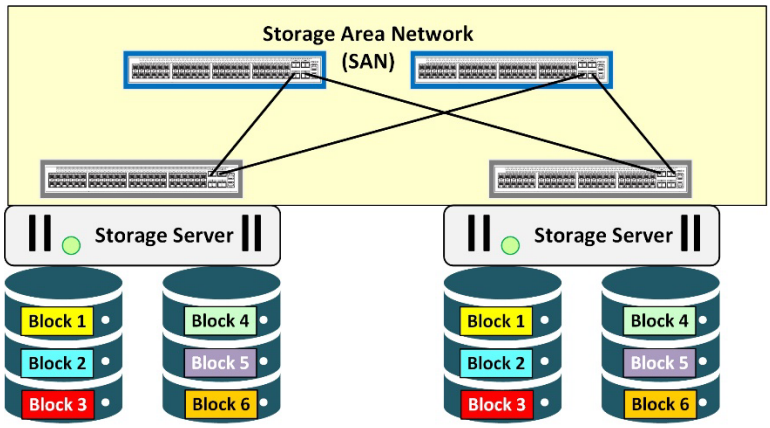
- File Storage
- Block Storage
- Object Storage

File storage is a system that organizes data into folders and typically these folders are within others in a hierarchical structure called a directory hence. The typical hierarchy format of file storage is directory/subdirectory/folder/file. This full path to the file destination is required to search and to reach any data in this storage method. File storage works well for small size scales, but it does not work well as the number of files grows.



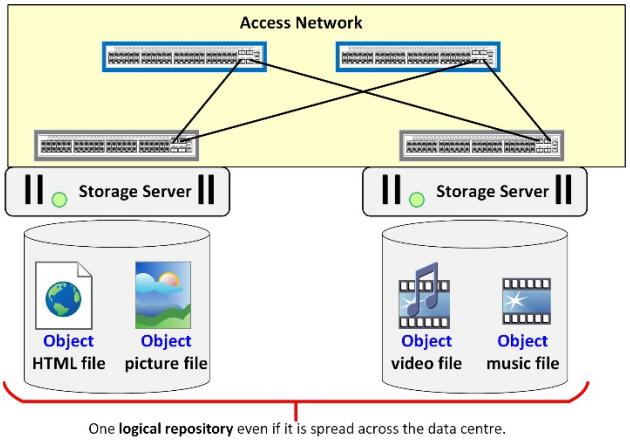
File Storage hierarchy /Directory/subdirectory/folders/files

Block Storage breaks down the data files into equal size blocks which are then saved in an infrastructure of storage servers. A high-speed dedicated Store Area Network (SAN) is used to access the blocks of data. This is possible because every block receives a unique identifier. Additionally, the blocks are replicated across different storage servers thus providing high reliability.



Block Storage

Object storage is the method of interest for this chapter, so it will be described in more detail. Object storage is used to store data items such as video, music, pictures, documents, web data and files in general. These are the types of data that makes up most of the current Internet [1] content. A common property among all these data files is that they do not belong to a hierarchical organization or a filesystem. Instead, an **object** is a distinct unit of data stored in a **logical repository** or storehouse that can be spread out across a network or a datacentre. Each logical repository contains the data object itself, the data (**metadata**) that describes the properties of the object and a **Unique Resource Name** (URN).



Interpretation of an Object Storage system.

Object storage is a versatile service with many uses; usually, but not limited to:

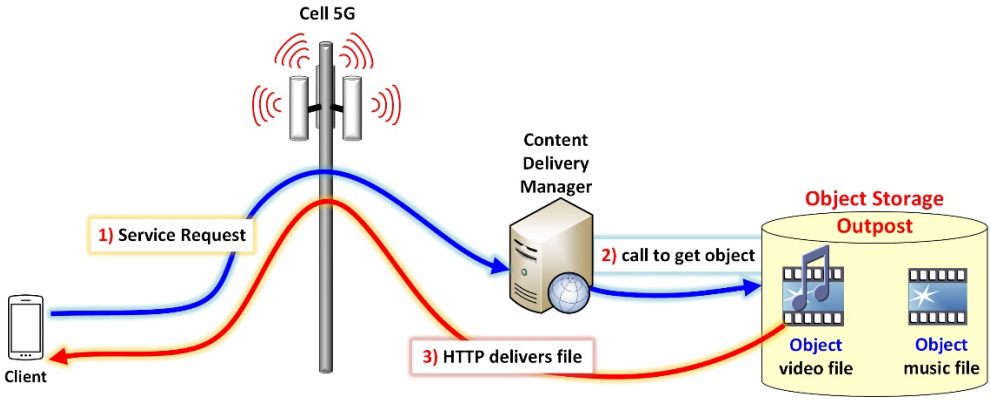
- General storage.
- Big data analytics
- Media storage and delivery
- Backup and archiving
- Cloud-Native applications

General storage places data that does not have an immediate need for organization or it can not be organized in an indexed or hierarchical format. This is due to the very same nature of the data. For example, social media content is of **unstructured** kind. In this case, the content objects can be placed on bulk storage. If needed, the objects are always available by calling their unique resource names.

Organizations that provide a service over the Internet would like to gather intelligence over the content shared by the clients. Again, this content is unstructured and that makes very difficult to search, classify, and analyze with a traditional database engine. **Big data analytics** is a cloud service that analyzes large conglomerates of stored objects to find common factors for instance, the type of content that most users are looking for. This is possible because every stored object have inherent metadata that describes it. Data analytics over object storage provides valuable information for business decisions.

Another use for object storage is **media storage and delivery**. A common way to implement this service is via a Content Delivery Network (CDN). This is a node or endpoint that supplies content, video for example, to users located nearby. The CDN node has access to the object storage service and when a user requests a particular video, the CDN calls the unique name of the object and quickly provides that to the client. This is a desired model for services that require low latency. In the following, oversimplified,

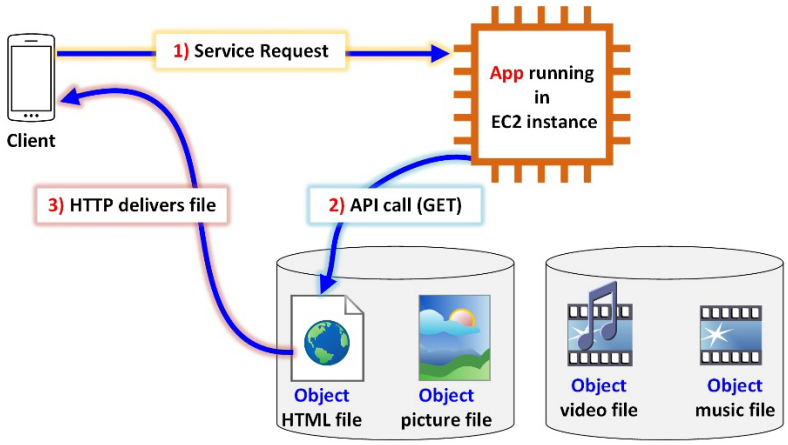
diagram, a cell phone client makes a service request for video-based content. The delivery of the content should be done quickly to provide the user with the best experience. Hence, the content is placed in an object storage outpost close to the location of the user. The Content Delivery Network (CDN) supply the video object to the client.



Object based Content Delivery Network.

The emergence of cloud technologies have caused a shift in the way that applications are designed. The **cloud-native** design paradigm implies that the applications are built using the components and services provided by the cloud. Instead of the traditional monolith, applications are split into functional components. In this sense, object storage is a critical element of this model. To access the storage, cloud-native apps make calls to the unique resource names of the objects or **endpoints**.

For example, a client on a mobile device makes a service call to an application running in a virtual machine (EC2 instance). This triggers the application to make a call to the endpoint using an **Application Programmable Interface (API)**. The API call contains the **RESTful** verb **GET** directed to the object being requested, in this case, a website home page. This HTML file is delivered to the client using the protocol HTTP. Finally, the web browser on the mobile device will interpret the data. In further transactions, the application can make API calls to the other stored objects such as the pictures, video, or music to be delivered to the client device. This example demonstrate a benefit of using object storage. Instead of a monolith, the application and the data storage are separate instances from separate cloud services.



An application integrated with object storage.

Elastic Object Storage

Elastic Object Storage is the cloud service that provisions unlimited, on-demand access to object storage. All cloud providers offer similar object storage services. The following table summarizes some of them:

Cloud Provider	Object Storage Service	Logical repository
AWS	Simple Storage Service S3	Bucket
Azure	Azure Blob	Blob
GCP	Google Storage	Bucket
IBM	Cloud Object Store	Bucket
Digital Ocean	Spaces	Bucket

AWS Simple Storage Service (S3)

The S3 service is organized in a range of **storage classes** based on the factors of cost, frequency of data access, access patterns, performance, data residency, and resiliency. Fundamentally, AWS tries to cover all the use cases with different flavours of the S3 service.

The S3 storage classes [4] are:

S3 standard	For frequently accessed data, low latency, high throughput.
S3 standard Infrequent Access	Less frequency access, but rapid access when required.
S3 one-zone Infrequent Access	For less frequently accessed data. Cheaper than standard infrequent access
Glacier Instant Retrieval	Archive data that needs quick access.
Glacier Flexible Retrieval	(formerly S3 Glacier) Long retrieval time. For rarely accessed long-term data that does not require immediate access.
Glacier Deep Archive	Lowest cost, long time retrieval. for long-term archive and digital preservation with retrieval in hours at the lowest cost storage in the cloud.
Outpost	On premise storage device that interconnects with AWS S3 service. To store S3 data on premises for reasons of data residency requirements that can not be met by an existing AWS Region.

The **S3 standard** storage class is the default setting for S3 storage. It is meant for data that is frequently accessed with required high upload/download throughput (high data bit rate) and low time delay variation (low latency). S3 standard use cases include object storage that is accessed by applications such as microservices and modern apps, content distribution, and static websites. This service is regionally deployed across all the availability zones. For example, in the case of the us-east-1a region, an S3 storage unit will be replicated across all six availability zones. Consequently, the Standard S3 **Service Level Agreement (SLA)** guarantees [4] a high durability (99.999999999% or the 11 9s) and high availability of 99.99%.

Durability is a measure of the likelihood of data loss. The underlay of object storage is actually a form of block storage. When an object is stored in S3, it is broken up into multiple pieces which are stored in different physical servers. These pieces overlap in some parts. If anyone of them get lost, the other pieces can be used to recover it. In short, it is extremely improbable that data can be lost by failure in the cloud.

Availability is the possibility of successfully accessing the data over time. Availability depends on the performance of the data centres that make up an availability zone. It would take a major outage to knock down the access to all the data centres in an availability zone.

The **S3 Standard-Infrequent Access (S3 Standard-IA)** [4] is for data that is accessed less frequently than in the S3 standard service. This storage class offers the same SLA than the S3 Standard, but at a lower storage price per GB precisely because the data is infrequently accessed. When it is, there is a charge per retrieval. S3 Standard-IA use cases that require longer term storage such as backups and log data. There are cases, the user begins the storage in S3 standard and after some time the data is moved to Infrequent Access to reduce the costs.

Amazon S3 One Zone-Infrequent Access (S3 One Zone-IA) is for non-critical data that can live without the high availability provided by the other services since S3 One Zone-IA stores data in a single AWS Availability Zone. Examples of this kind of data are secondary backup copies. The trade-off with the lower availability is a benefit of 20% [4] lower cost that S3-Standard-IA.

For much longer term storage, AWS **Archive** encompasses three classes: S3 Glacier Instant Retrieval, S3 Glacier Flexible, and S3 Glacier Deep Archive. Each of these classes is optimized for combinations of different storage patterns and patterns of access.

S3 Glacier Instant Retrieval storage class is to archive data that do not get access frequently, but when it does, it must be done immediately. The typical use cases are archives of medical records and libraries of media such as birth registries, for instance. it delivers the lowest cost storage with milliseconds retrieval.

S3 Glacier Instant Retrieval can yield savings of 68% on storage costs compared to S3 Standard-Infrequent Access (S3 Standard-IA) storage class. The proviso is that the data should only be accessed as frequently as once every [4] three months. Glacier Instant Retrieval has the same data retrieval time, the same durability (99.999999999%) than S3 Standard. Its availability [4] is a bit lower (99.9%) though.

Amazon S3 Glacier Flexible Retrieval (Formerly S3 Glacier) is 10% cheaper [4] than S3 Instant Retrieval as long as the data is accessed no more than twice per year. Its use case is large amount of archival data that may be accessed over a time ranging from minutes to hours. The key difference of this class that it offers configurable retrieval times depending on the customer needs and the cost.

Amazon S3 Glacier Deep Archive is the lowest cost storage class for long term storage. The main use cases are regulated sectors which are required, either by law or by practice, to keep data for several years such as financial securities, healthcare, law enforcement, civil records and others. The retrieval time is in the order of hours.

S3 on Outposts is a solution to data that is required to be stored on premises either by regulations or by organizational practices. In this case, AWS delivers a physical storage unit to the customer's data centre. The management of the unit is completely done via AWS console and other management tools, so it is completely integrated with AWS services.

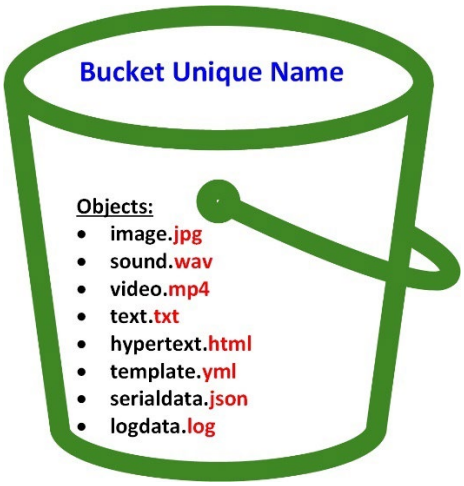
S3 Intelligent-Tiering is not a storage class but a service that enables costs savings by automatically moving data with certain access pattern to the most cost-effective storage class. For example, an object in S3 -Standard that has not been accessed for 30 consecutive days will be moved to the Infrequent Access tier for 40% lower cost. If the same object is not accessed for another 30 days, it will be moved to S3 Glacier Instant Retrieval for a 60% lower cost, all cases compared to S3-Standard. Finally, for objects not accessed for longer than 180 days, S3 Intelligent-Tiering can move them to the Deep Archive Access class to obtain up to 95% in storage cost savings.

The following table copied from the Amazon S3 service console summarizes all the cases. This is the same table that is displayed when an S3 bucket is created.

Storage Class	Designed for	Availability Zones	Minimum storage duration.	Minimum billable object size	Monitoring and auto-tiering fees	Retrieval fees
Standard	Frequently accessed data. More than once a month within millisecond retrieval access.	≥ 3	-	-		-
Intelligent Tiering	Data with changing or unknow access pattern.	≥ 3	-	128 KB	Per-object fees apply for objects >= 128 KB	Per-GB fees apply
Standard-IA	Data infrequently accessed. (Once a month) with millisecond access time.	≥ 3	30 days	128 KB		Per-GB fees apply
One zone IA	Infrequently accessed data stored in one AZ. Millisecond access time	1	30 days	128 KB		Per-GB fees apply
Glacier Instant Retrieval	Long-lived archive data accessed once a quarter with instant retrieval in milliseconds	≥ 3	90 days	-		Per-GB fees apply
Glacier Flexible Retrieval	Long-lived archive data accessed once a year with retrieval of minutes to hour	≥ 3	180 days	-		Per-GB fees apply
Glacier Deep Archive	Long-lived archive data accessed less than once a year with retrieval of hours	≥ 3	-	-		Per-GB fees apply
Reduced Redundancy	Noncritical, frequently accessed data with milliseconds access (not recommended as S3 Standard is more cost effective)	≥ 3		-		Per-GB fees apply

The S3 Bucket

The fundamental unit of storage is the **bucket** which is a virtual case that contains the objects. A bucket must have a unique regional name which is given by the customer at the moment of creation. The S3 service immediately verifies that the name does not exist already in the region. If it is vacant, then it is assigned to the bucket for as long as it exists.



Representation of an S3 bucket with objects stored.

Normally, a customer account can have up to 100 buckets [2]. A bucket can store different data types as shown in the previous illustration. According to AWS [2] the size of a bucket is unlimited and it can hold as many objects as the customer desires.

The Objects

The data objects are stored inside the bucket. Each object can have a maximum size of 5 trillion bytes (5 TB). The objects are uploaded onto the bucket via the HTTPS protocol using either the AWS console or AWS CLI or AWS SDKs (a Python script for example).

The objects can be reached remotely because they have a globally unique Uniform Resource Locator (URL) that includes the name of the bucket, the S3 service name, the Internet domain name of amazonaws.com and the path to the object.

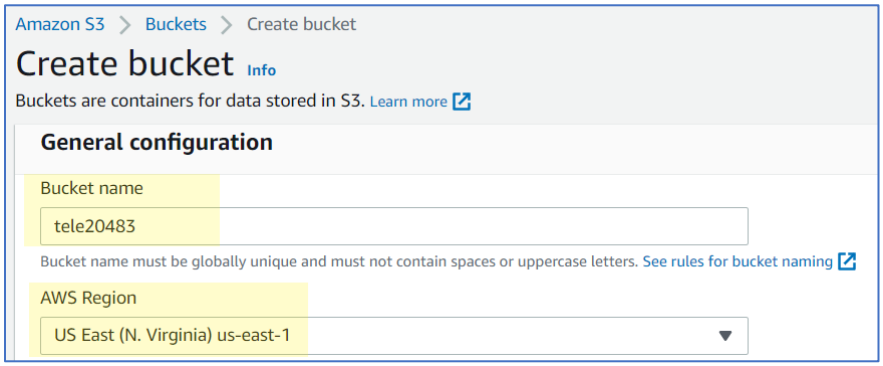
Objects are described in a **representational state (REST)**. The objects have a **key**, a **version ID**, a **value**, **metadata** and **sub-resources**. Because of this structure of the data, any application designed in a RESTful way can interact with the object. When an application makes a call for an object, the service responds with all the relevant information (the state) about the object. This representational data is interpreted by the client application. This is a powerful idea that enables most of the modern applications.

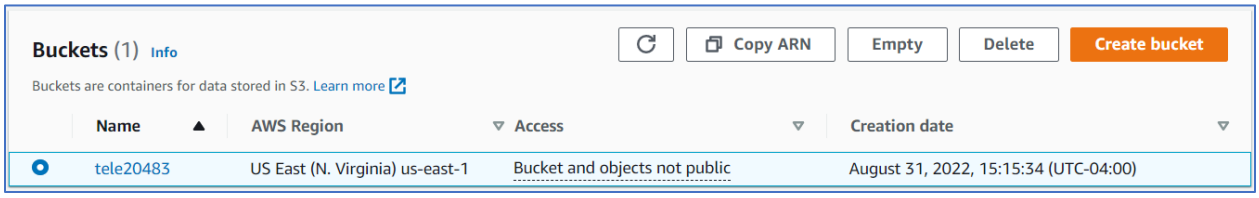
Learning Activity

Creating an S3 bucket

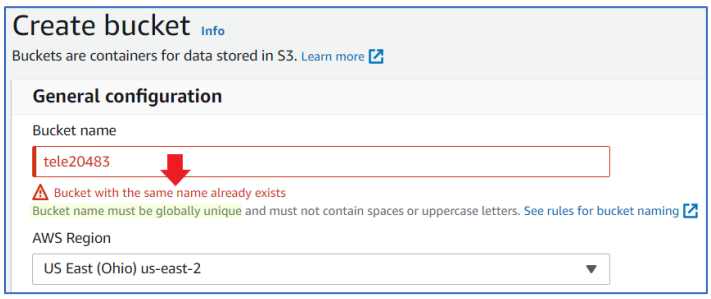
- Create a new bucket in the Amazon S3 service and explore its options and features.

Assign a name that has not been taken yet. If the name is taken, the service will immediately let you know.



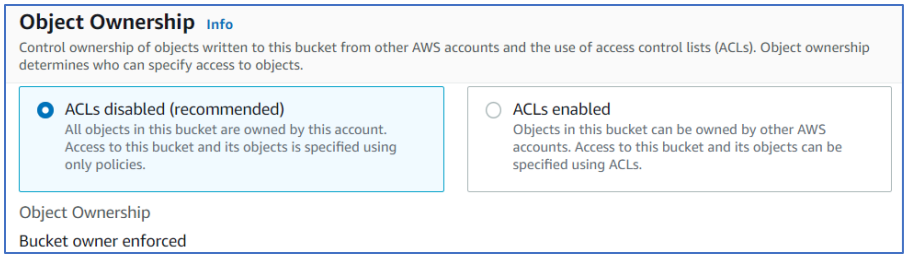


In the example above, the name was available and the bucket was created with all the settings left on their default values in the us-east-1 region. To test the concept, let's attempt to create another bucket with the same name in another region. The S3 service immediately recognizes that another bucket with the same name already exist even if it is in another region. The error message indicates that the name must be globally unique.

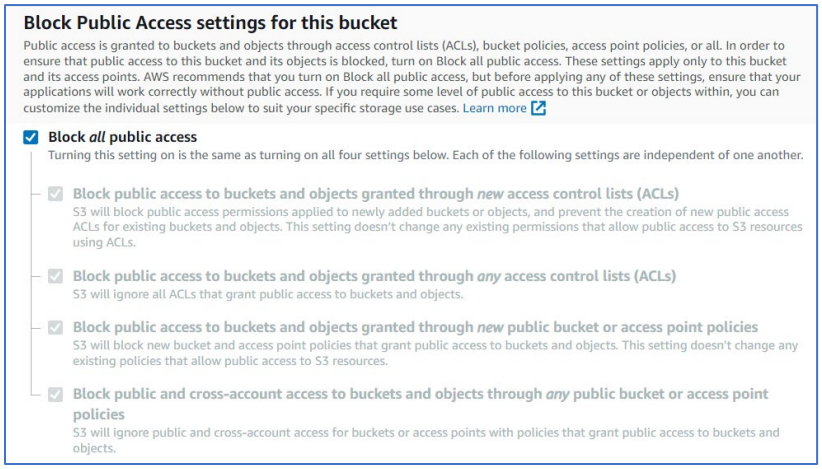


Let's explore now the new **Bucket Default Settings** and at the same time learn some concepts.

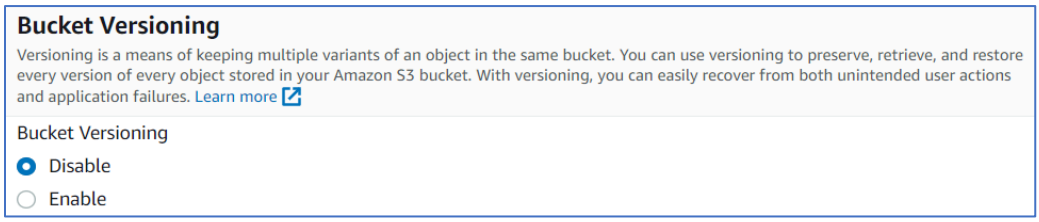
Object Ownership. There are several ways to control the ownership of the objects stored in a bucket. In principle, the account's owner is also the owner of the objects. In such case, the access to the objects is controlled via **bucket policies**. It is possible to allow other accounts to own objects stored in the bucket. In this case, the access to the bucket and its objects is controlled in a granular way with **Access Control Lists (ACL).**



It makes sense that the new bucket will have the public access blocked by default. There are many cases when objects are for internal consumption only, so there is no need to open the access to the Internet in general. This new bucket has a completely closed Access Control List blocking the external access to the objects.

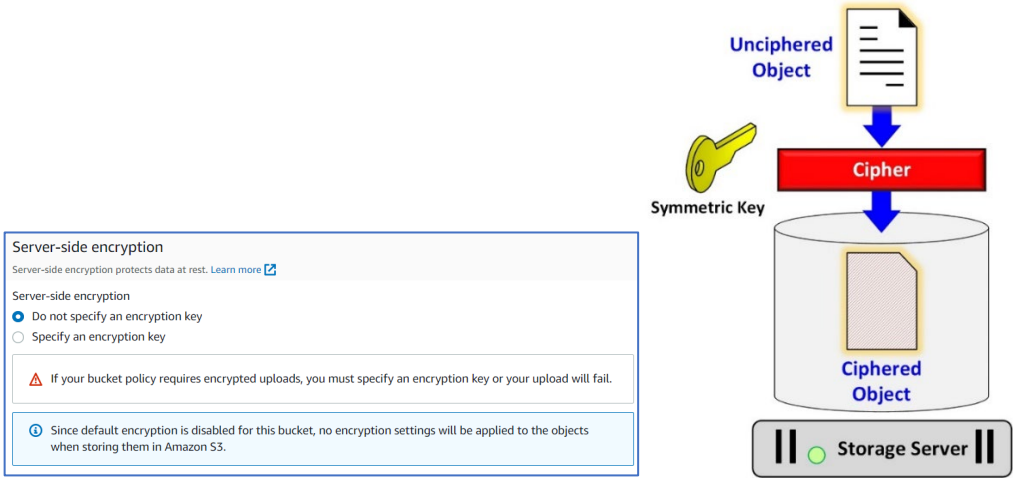


Bucket Versioning is a valuable feature to keep control of the editions of an object. When this is enabled, the S3 service will keep tracking of new versions of the same object by slapping labels on them. Hence, older versions of the object are available in case that they are needed. Bucket versioning is very valuable in infrastructure automation where virtual resources are deployed using configuration scripts. Every time that a resource is deployed or modified, the state of the infrastructure is logged in an S3 bucket as a new version. Therefore, bucket versioning is also helpful to recover from failed upgrades since the previous version is kept in the bucket.



Encryption refers to the protection of the data which can be done in different ways. In the default mode, the objects are ciphered by the Amazon S3 service when they come to rest in the AWS datacentre storage.

AWS manages the ciphering key and the ciphering methods. This is the default method of data protection and it is called **Server-Side Encryption (SSE-S3)**. This is the simplest method to use because everything is managed by AWS. The Amazon S3 service encrypts each stored object with a unique key using the AES-256 [3] encryption algorithm which is a symmetric cypher. The key never leaves AWS and it is protected itself [3] by being encrypted with a master key that is regularly rotated. The default encryption is performed at the bucket level. This implies that all files uploaded onto the bucket will be ciphered with the same encryption algorithm with the same key.



Representation of default Server-Side Encryption.

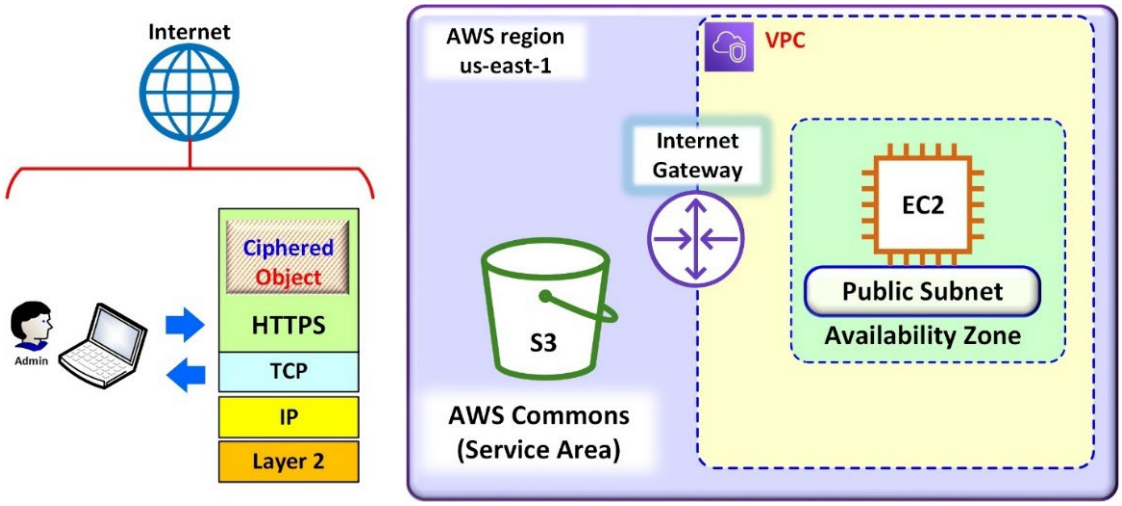
AWS Server-Side Encryption supports two other options beside SSE-S3. They are SSE-KMS and SSE-C.

Server-Side Encryption using AWS **Key Management Service** (**SSE-KMS**). KMS is an AWS service that centralizes the management of the cryptographic keys used to protect the data in the customer account. The customer creates keys using this service and ties policies that control who can use them. KMS also provides an **audit trail** on the usage of the keys. The main advantage of SSE-KMS over SSE-S3 is the additional level of security provided by the **permissions** that control the usage of the KMS keys. Only authorized users and applications can use them.

Server-Side Encryption with **Customer-provided encryption keys (SSE-C)** is when the customer prescinds of the previous options and manages its own encryption key. The ciphering is done at rest in the AWS datacentre (but this is different to the Client-Server Encryption option). In this mode, AWS does not have the keys stored anywhere. Instead, the client has complete control over the creation and use of the key. There is no option in the AWS Console to deal with this SSE-C but rather it is handled with an Encryption Software Development Kit (SDK), AWS CLI or an API. In short, it is part of an application that uses a library to handle the security of the objects.

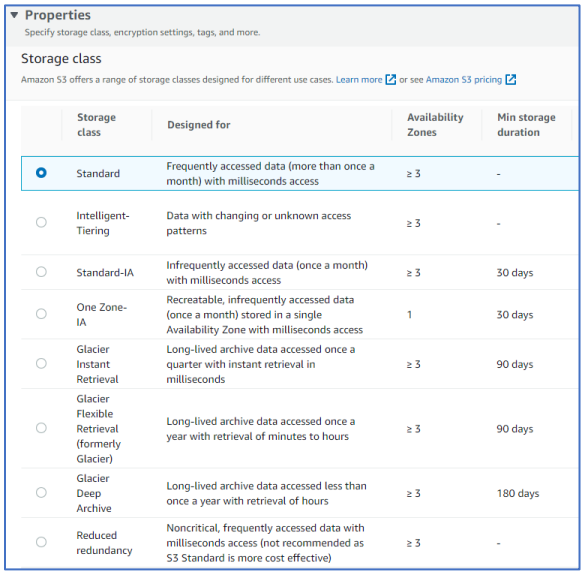
Client-Side Encryption is the only option that instead of doing the ciphering at rest in the AWS datacentres, the customer ciphers the objects locally before uploading them onto the S3 bucket. In this case, the AWS customer manages the security keys and the ciphering methods using a library such as the Amazon S3 Encryption Client. The customer is in full control of the keys and encryption cycle. AWS S3 just stores whatever the client uploads.

There is an additional layer of protection when the objects are in transit over the Internet whether because they are being uploaded or downloaded. This is achieved by the protocol HTTPS which includes Transport Layer Security (TLS) that ciphers the data inside the TCP payload. The same process applies to either type of S3 protection, either server-side or client-side.



Representation of ciphered object data in transit using the secure HTTPS protocol.

The **Storage Class** presents all the available storage tiers (previously explained). The S3 Standard class is the default setting for data frequently accessed.

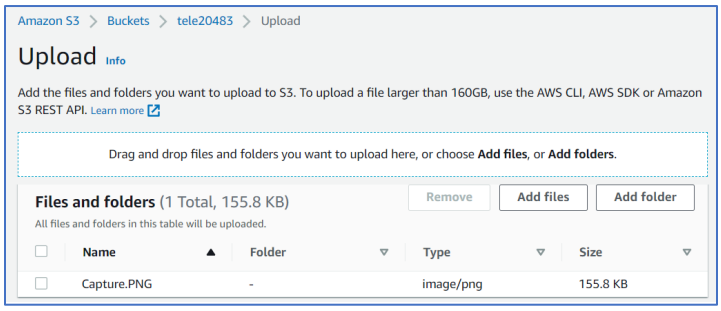


The S3 bucket configuration is complete and it is ready to store objects in the next activities.

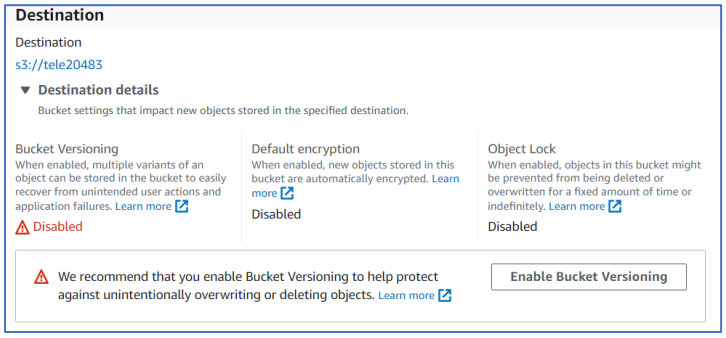
Learning Activity

Uploading an object to the S3 bucket

The bucket created in the previous activity automatically received an AWS Resource Name arn:aws:s3:::tele20483. This is how the Amazon S3 service tracks of the buckets. Let's proceed with the uploading of a picture named Capture.PNG onto the bucket.



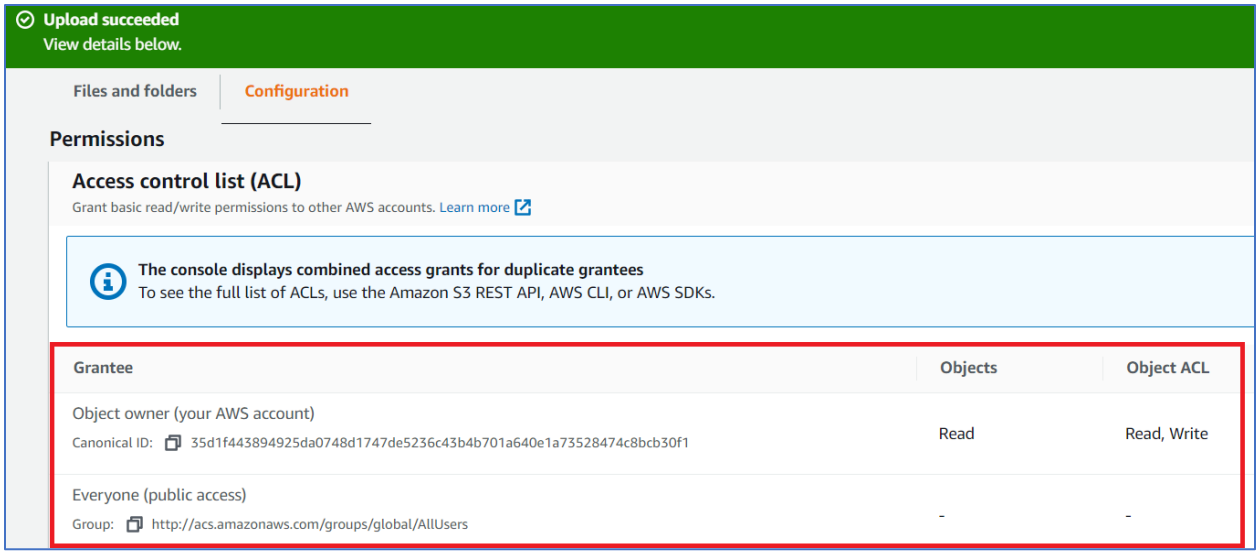
The version and encryption settings are left in default.



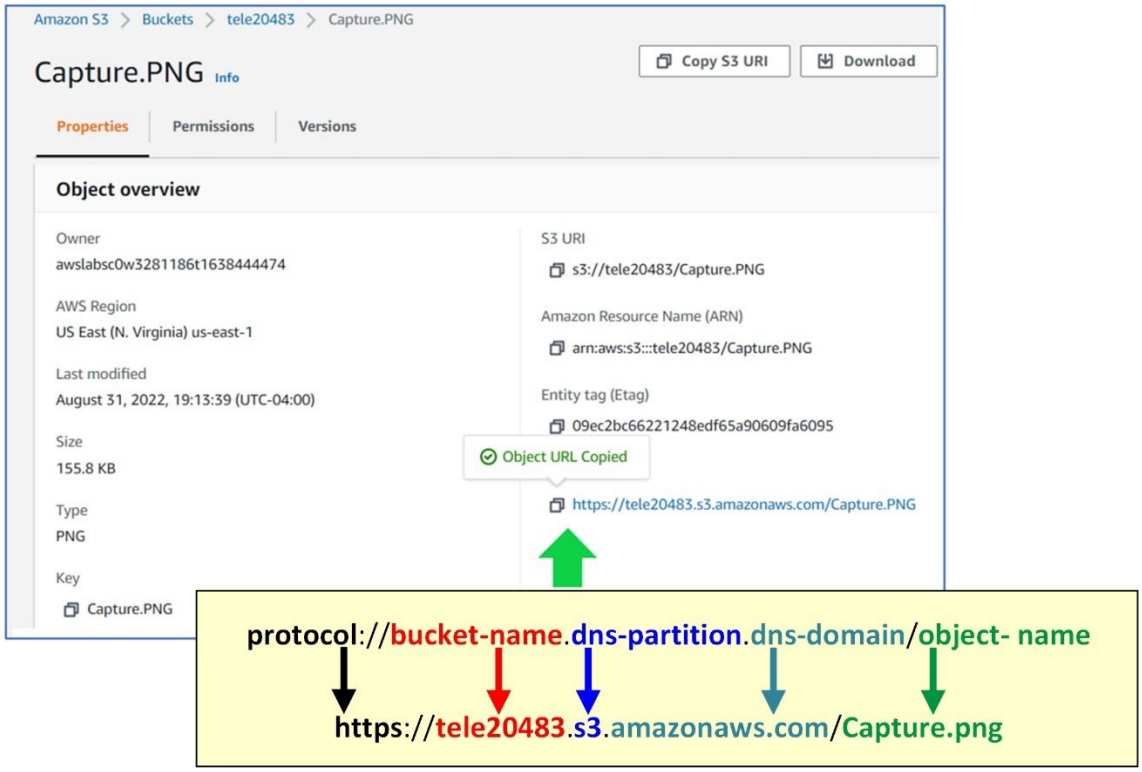
The object is directly available in the AWS console.

This shows the object which is just a picture of an orange plastic bucket.

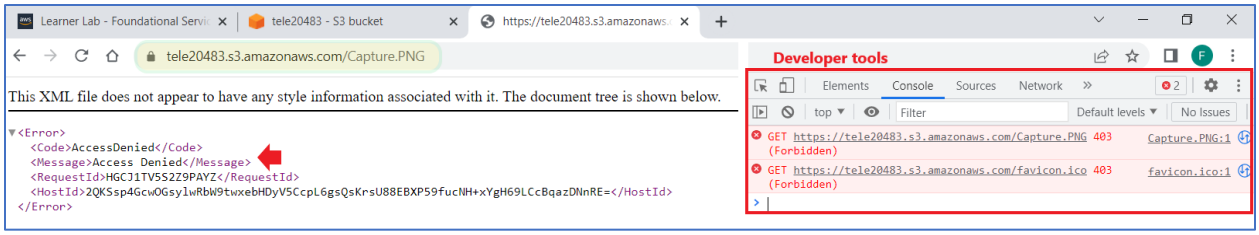
This simple demonstration shows that the object has been uploaded from the customer laptop to the S3 storage service and that the account owner has direct access to the object. Furthermore, the **permissions** reiterate that the owner of the account can read and write but no one else has any rights on the object.



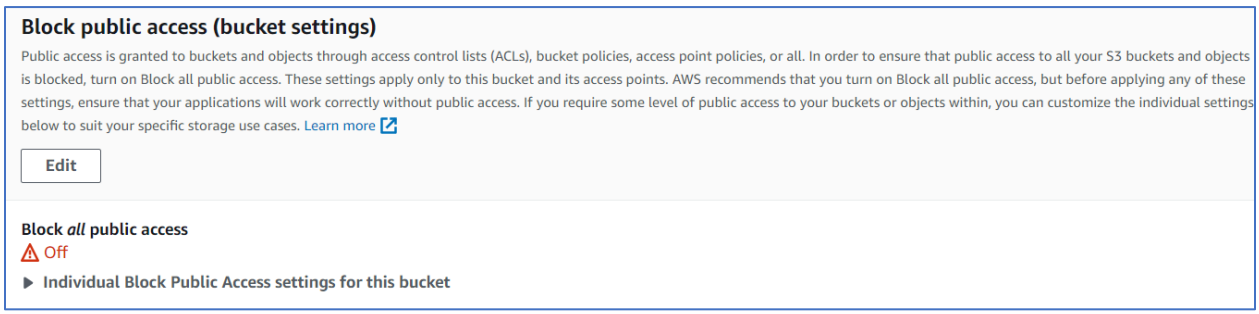
The attributes of the object are listed under the **Properties** tab. The Object URL displays a universally reachable Internet locator. The format of the URL is:



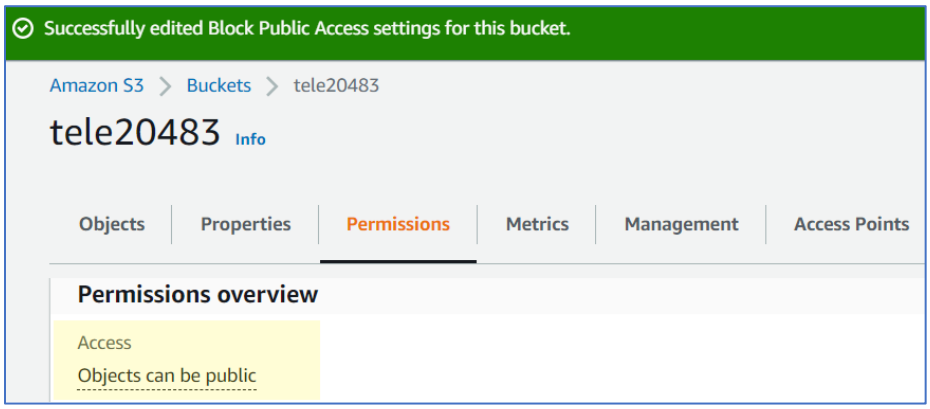
In principle, if the URL is pasted on a web browser, it should return the picture of orange bucket.



But it does not. An error message hints at the reason. It says that the access is denied. The browser's developer tools affirms that the access to the object is forbidden. The bucket's Access Control List (ACL) blocks all public access by default. This is correct, the S3 bucket should not be open to anonymous access unless there is a good reason for that. Nevertheless, let's uncheck that setting to see what happens.



The bucket permission say now that the "Object can be public".



Is the object publicly accessible? No, it is still unreachable.

```
<Error>
  <Code>AccessDenied</Code>
  <Message>Access Denied</Message>
```

This is the moment to analyze this security system. The default ACL is highly restrictive and it blocks all the access to the bucket by default. This is the extreme application of the **principle of least privilege**. When the bucket is created, it should not be open at all. However, to allow access to other entities rather than the account owner, permissions must be set to control such access. Since the ACL has been dropped, there must be another security component to provide granular access to the objects stored in the bucket. That is the role of the **Bucket Policy** which is further explored in the next activities.

Bucket Policy

This is a document written in **Java Script Object Notation (JSON)** that describes the permissions to allow or deny the access to the objects in the bucket. The bucket policy JSON has a structure with the following main keys:

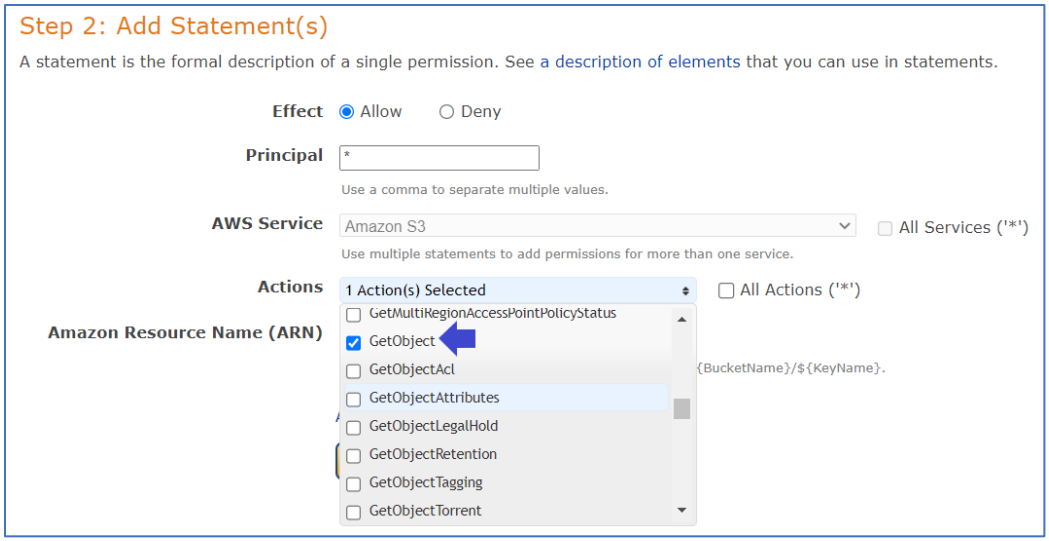
- A **unique identification**
- The **version** in the format year-month-day
- A **statement** that contains an identification and the action to be performed.
- The **effect** which is either allow or deny
- The **Amazon Resource Name (ARN)** on which the action is performed.
- The **principal** element specifies the user, account, service, or other entity that is allowed or denied access to a resource.

Bucket policies might be complex to write, for that reason, AWS provides an example library and even better, a **Policy Generator**. The ARN of the bucket is required to write the policy so it is displayed conveniently.

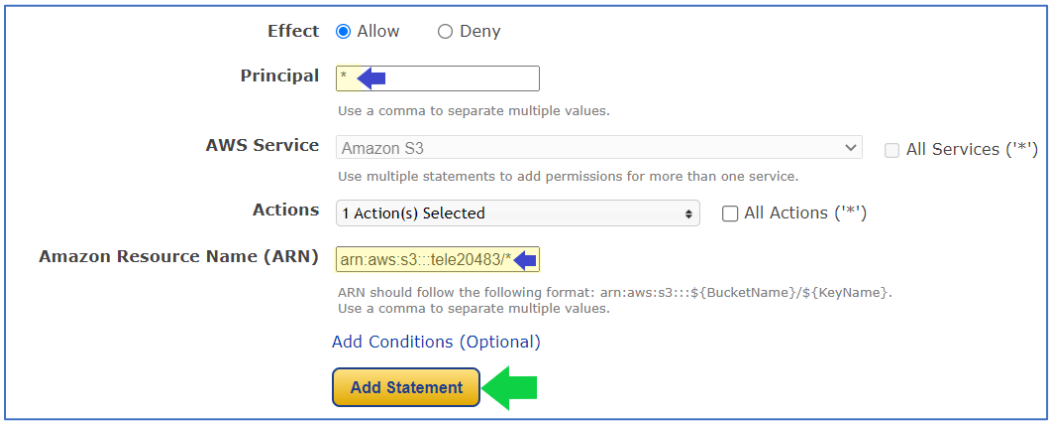
Learning Activity

Generate a Bucket Policy to allow public access to a bucket

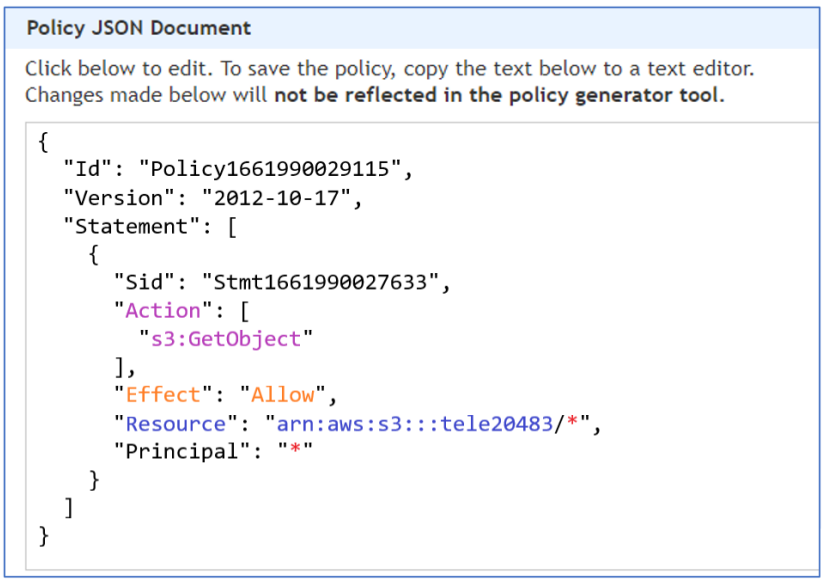
- Edit the **bucket policy**, **Policy Generator**.
- The effect is to allow.
- The principal is *
- The policy generator is used for multiple AWS services, choose Amazon S3.
- In the Actions choose GetObject.
- Paste the Amazon Resource Name (ARN) ending with /* which means everything in the bucket (for the sake of showing a simple example).



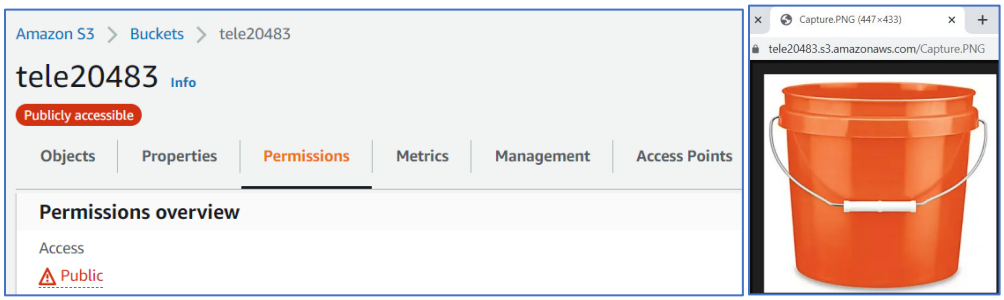
- Proceed to Add Statement.



- Copy the resulting policy and paste it into the bucket policy space. Save it.



The bucket permissions indicate that it is public now. Browsing to the URL proves that the object can be downloaded via HTTPS protocol.

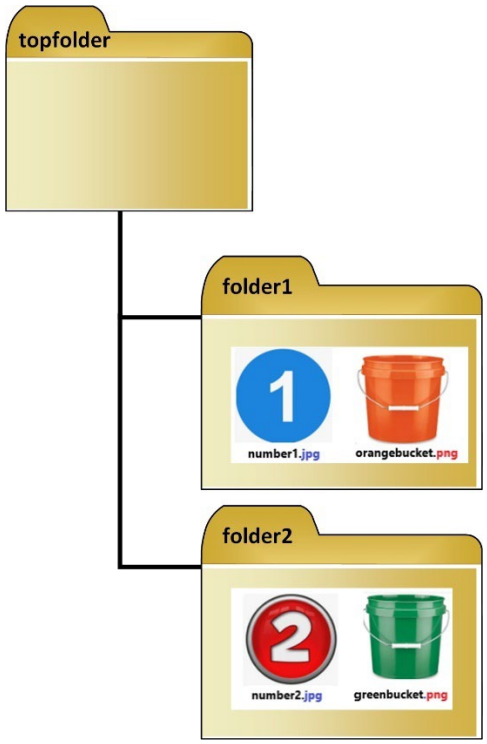


This was done as a demonstration on how to make an object public nevertheless this is an incorrect setting for most case scenarios. The next activity shows how to enable a more selective access to the objects.

Learning Activity

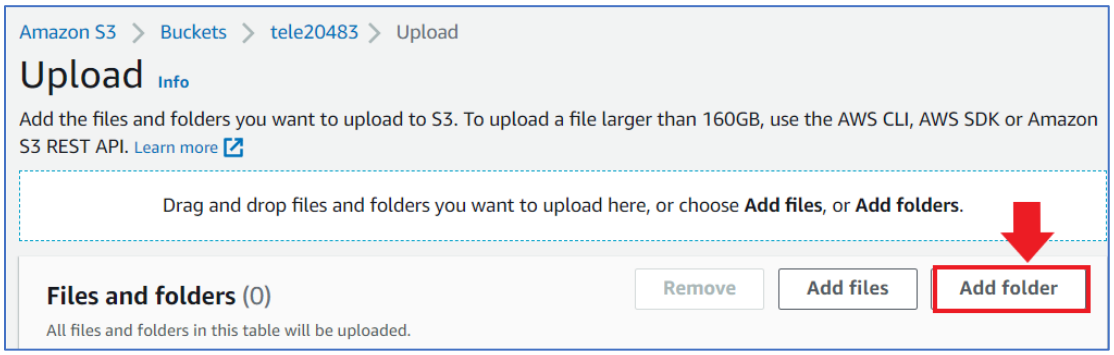
Set a Bucket Policy to selectively allow public access to certain objects

The previous example of a bucket policy opens all the contents of the bucket. In this activity, a folder structure is uploaded to the bucket and then some of the objects are open to the public access. This is the folder structure in the customer's laptop. Each subfolder contains one JPG file and one PNG file.

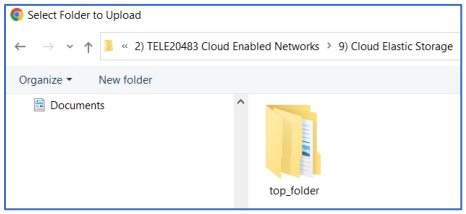


Folder structure to be uploaded to S3 bucket.

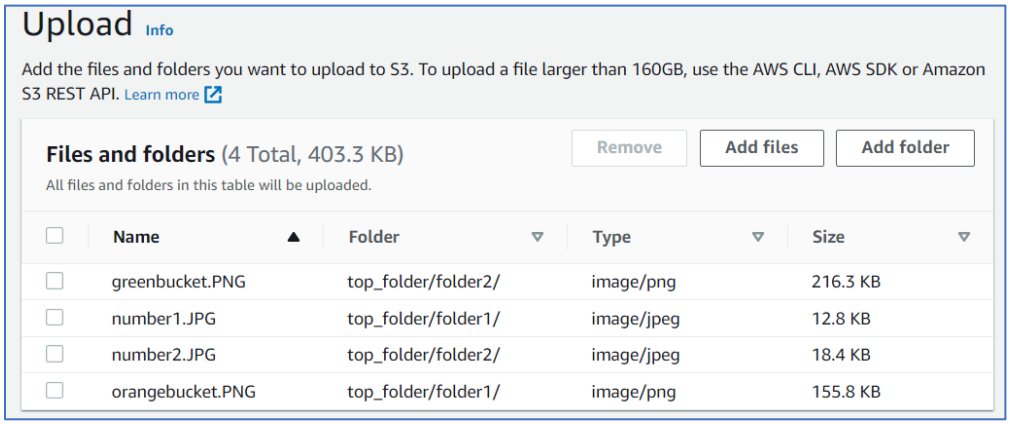
- Let’s proceed to upload the whole structure. Note: S3 does not have a concept of file systems. In this case, it just stores objects inside objects.



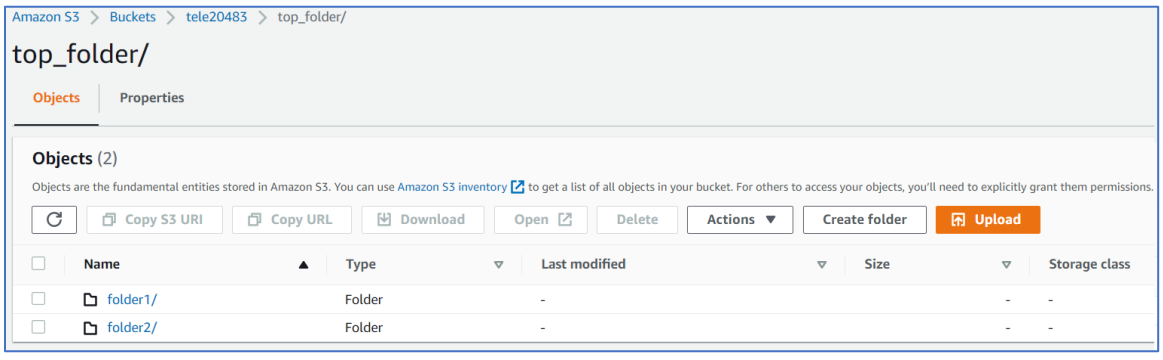
- The folder topfolder is selected in the laptop.



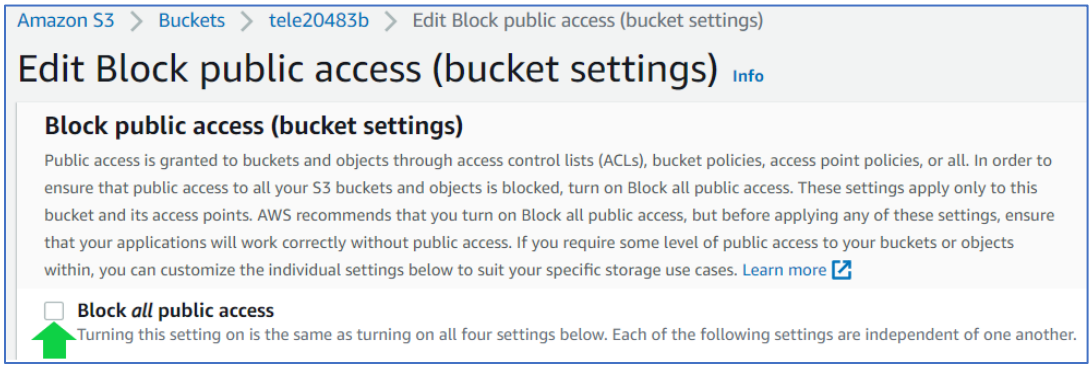
- The files are ready to be uploaded, proceed.



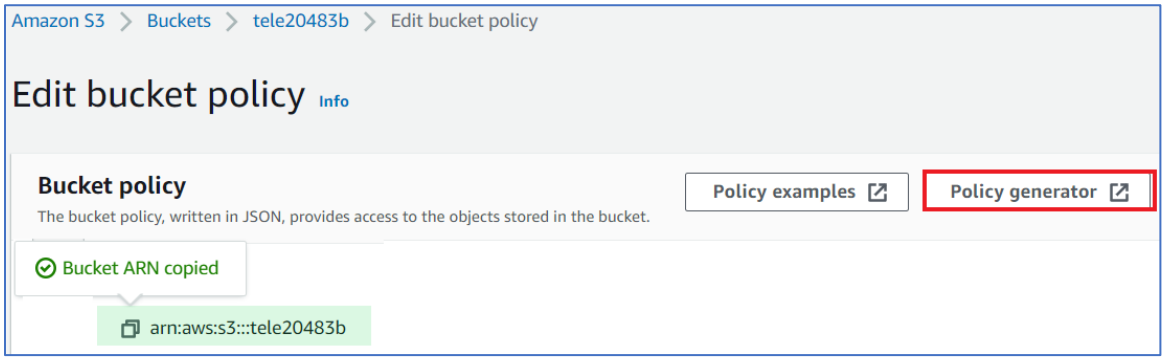
Now the bucket contains an object that contains the folder1 and folder2 object containing the images.



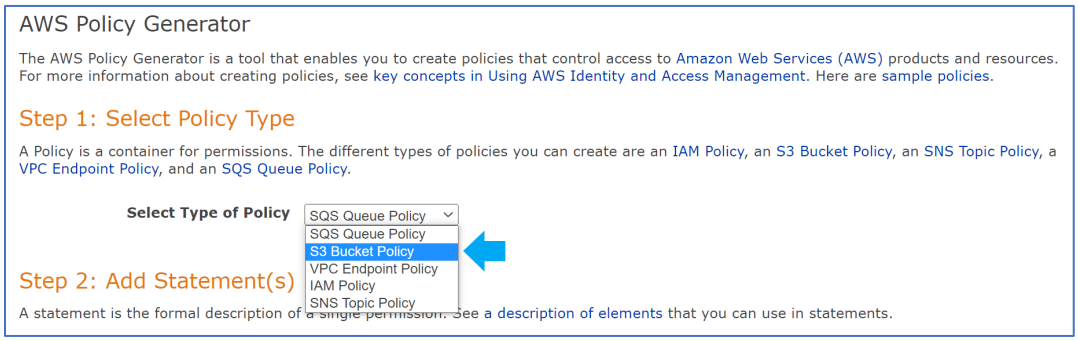
- Under the bucket permissions uncheck the ACL.



- The next step is to configure a bucket policy that will only allow access to the JPEG files.

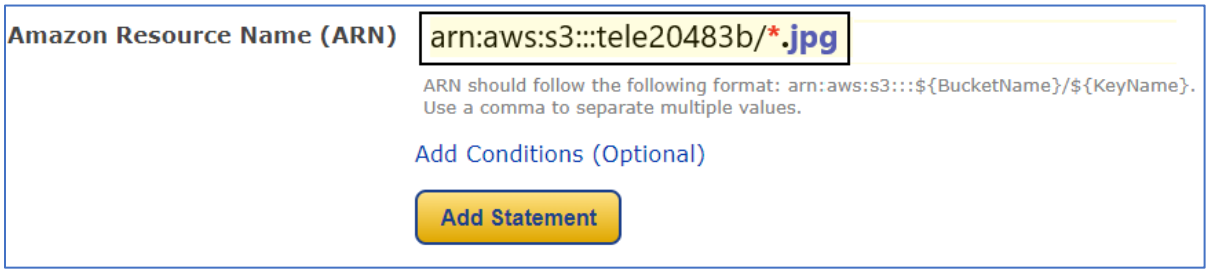


- Choose S3 **Bucket Policy** on the Type of Policy selection.



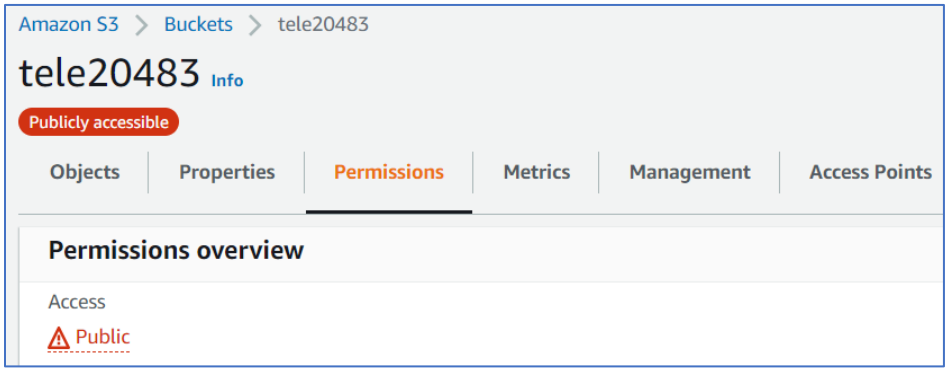
- **Action: GetObject**.

- The wildcard character * means all, so to get all the JPG files, the ARN is ended with /*.jpg



- Result:

- Paste onto the bucket policy and save. The bucket must show that it has public access enabled.



Now comes the testing to verify that the policy is working as intended. Any file of type JPG should be accessible, anything else should not be.

Object	Universal Resource Locator (URL)
number1.JPG	https://tele20483b.s3.amazonaws.com/top_folder/folder1/number1.JPG
orangebucket.PNG	https://tele20483b.s3.amazonaws.com/top_folder/folder1/orangebucket.PNG
number2.JPG	https://tele20483b.s3.amazonaws.com/top_folder/folder2/number2.JPG
greenbucket.PNG	https://tele20483b.s3.amazonaws.com/top_folder/folder2/greenbucket.PNG

- First, the object number1.jpg in folder1:

Surprisingly, it did not work. There must be something mismatching in the name and the policy. In effect, the file type is JGP but the policy has the type as jpg.

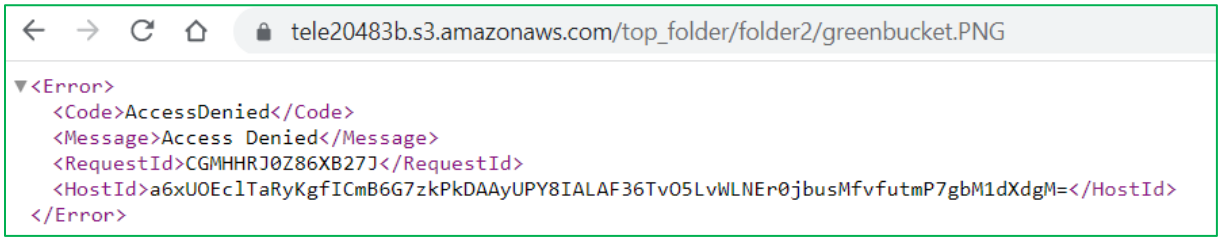
- Let's edit the policy to fix this.

- Try again: https://tele20483b.s3.amazonaws.com/top_folder/folder1/number1.JPG

- https://tele20483b.s3.amazonaws.com/top_folder/folder2/number2.JPG

- https://tele20483b.s3.amazonaws.com/top_folder/folder1/orangebucket.PNG

- https://tele20483b.s3.amazonaws.com/top_folder/folder2/greenbucket.PNG

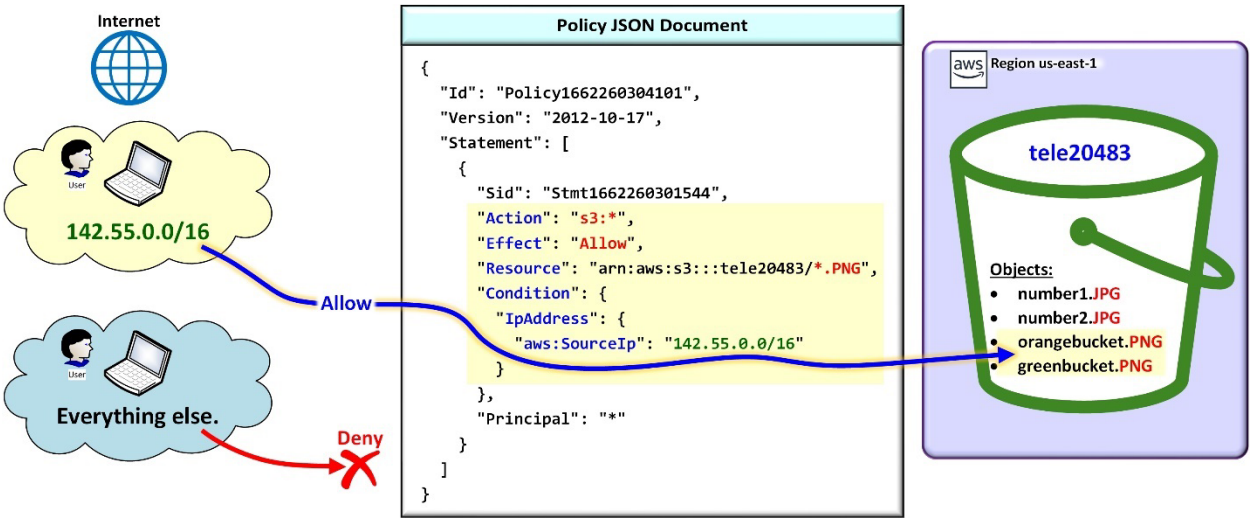


It is working as intended. The JPG files are accessible while anything else is blocked.

Learning Activity

Set a Bucket Policy to allow access from an IPv4 range to PNG objects only

This bucket policy allows selectively access to any PNG type file, but only if the access request comes from an IPv4 address in the network 142.55.0.0/16 (Sheridan College). The resources are reachable by using the combination of the wildcard character and the PNG file type (*.PNG) matches any filename that is of type PNG. The optional **condition** constrains the source IPv4 to the range 142.55.0.0/16. Any other IPv4 range is blocked by default. Bucket policies work by **implicitly denying** everything that is not **explicitly allowed**.



Selective Bucket Policy allows access to PNG files from 142.55.0.0/16.

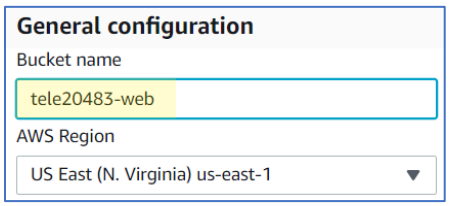
Learning Activity

Configure a Static Website on an S3 bucket

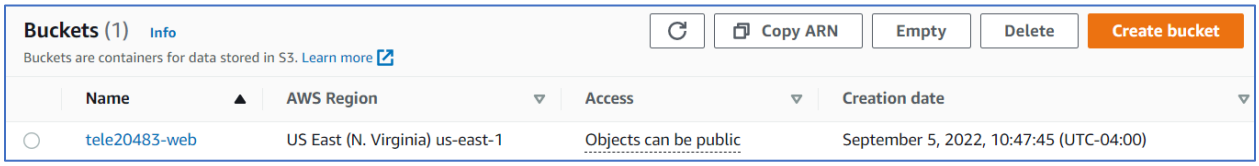
A static website supplies HTML files, images, videos, and other files to clients using the HTTP, HTTPS, and Web Sockets protocols. Static websites do not run server-side scripting or interact with backend databases. All the information needed to render the website is processed by the customer's web browser.

An S3 bucket can supply all these kinds of files and since it uses the web application protocols to deliver the data. Consequently, an S3 bucket can be configured as a static website in this activity. An S3 bucket can not support the dynamic website format. Let's proceed with the creation of a static webserver on S3.

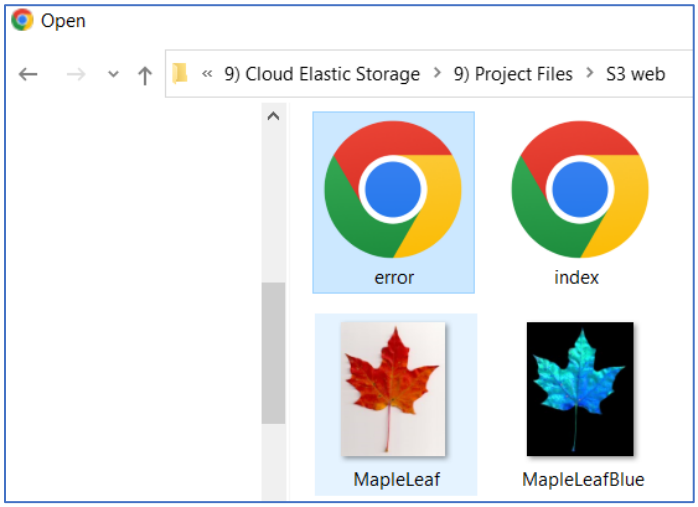
- Create a new S3 bucket.



- By definition, a webserver is open to anonymous access. Hence, unblock all public access.



- Upload an index.html file for the landing home page, an error.html file and some jpeg pictures.



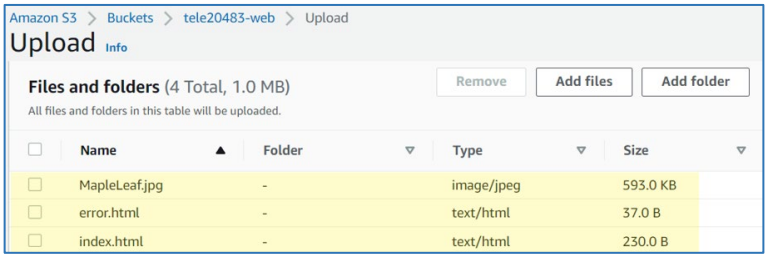
- This is the homepage index.html file. It loads a picture of a maple leaf.

```
<html>
    <head>
        <title> Static Website on S3 Bucket </title>
    </head>
    <body>
        <h1> TELE20483 </h1>
        <p> Cloud Enabled Networks </p>
    </body>
    <img src="MapleLeaf.jpg" width=500/>
</html>
```

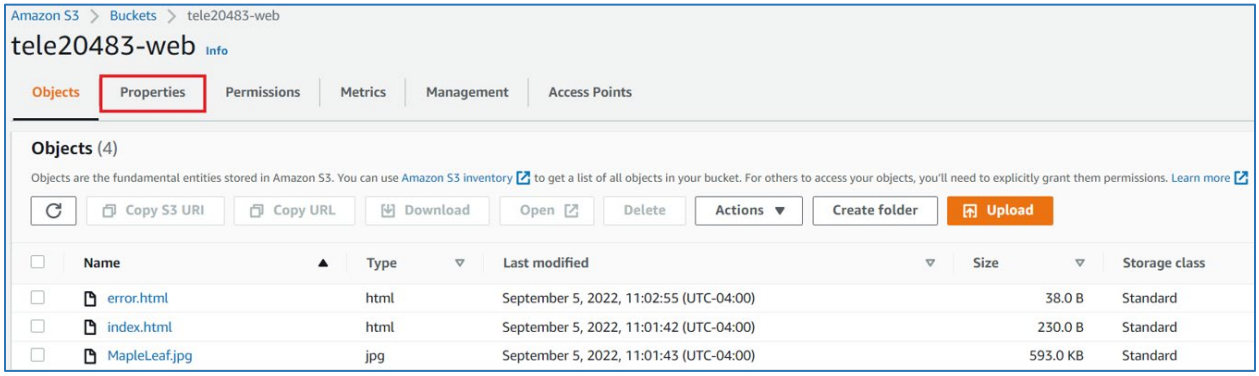
- This is the error message html file.

```
<html> An error has occurred </html>
```

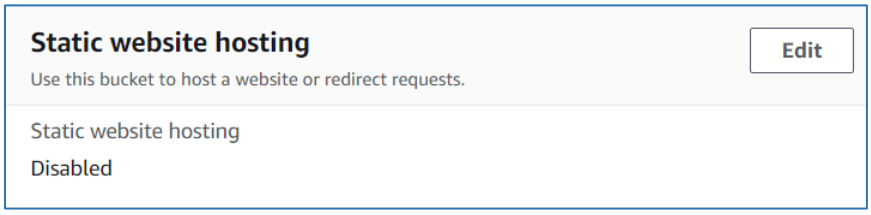
- Upload the three files.



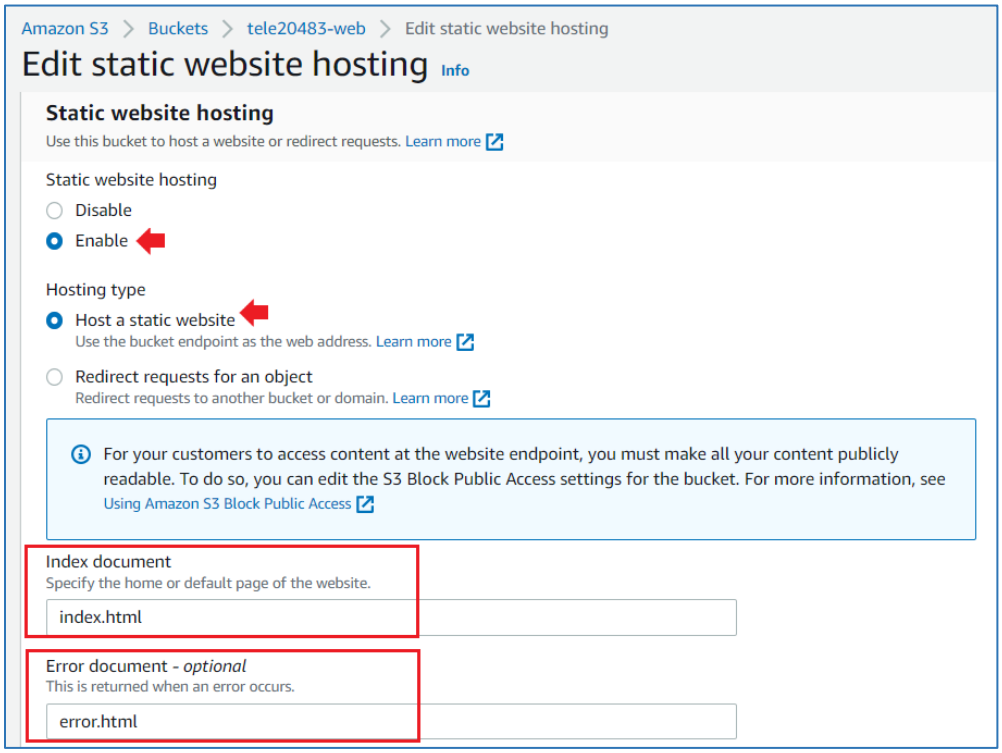
- Modify the **Properties** of the bucket.



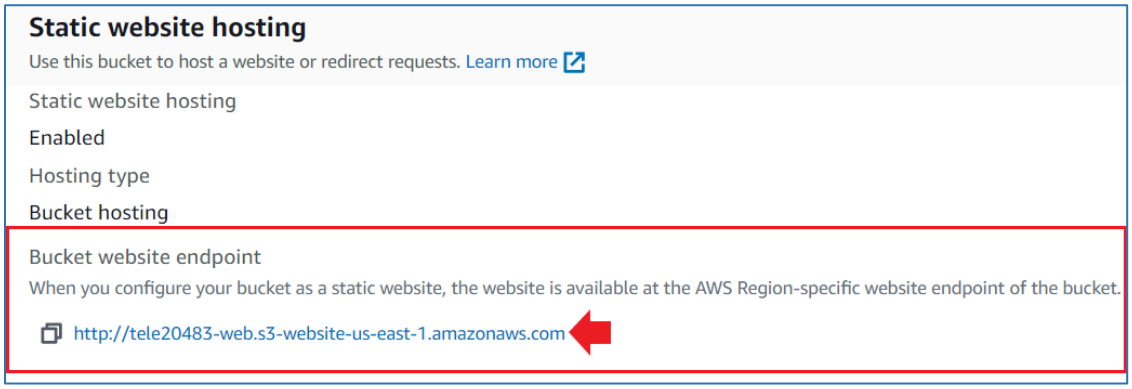
- Enable the website hosting option.



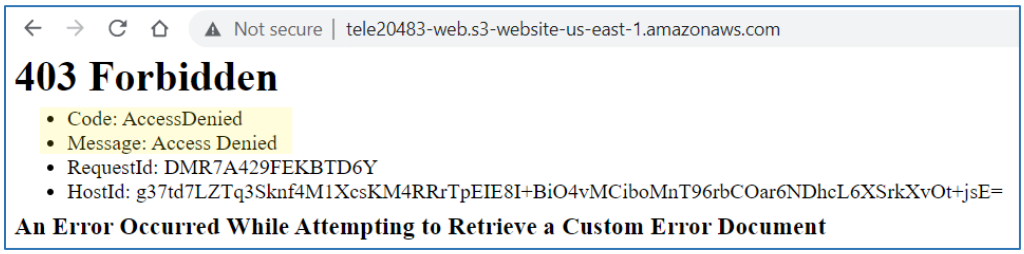
- Specify the home and error HTML files.



- A **Bucket website endpoint** is automatically generated by Amazon S3 and DNS services.

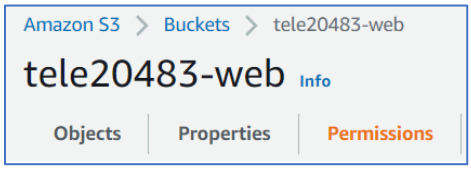


- Copy the endpoint and try to browse to the website.
- An error message “Access Denied” should appear.

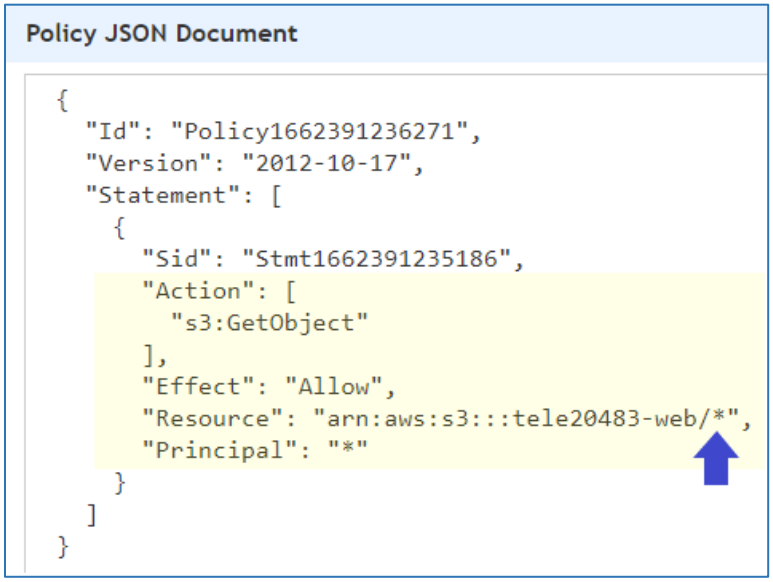


The reason is that the Bucket Policy has not been configured yet.

- Under the bucket's **Permissions**, create a JSON document using the policy generator.



All the objects in the bucket-website should be accessible.



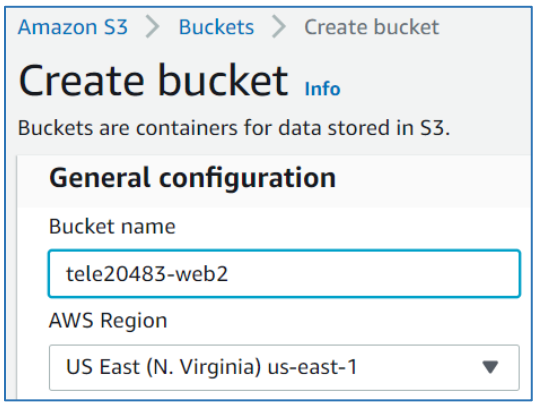
- Test the website again. It should work now.

Learning Activity

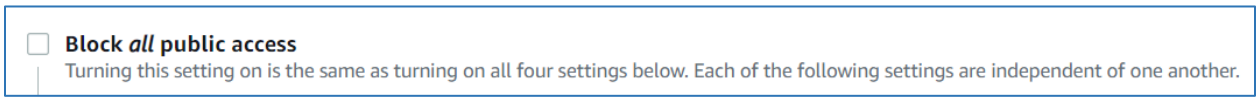
Configure two Static Websites with Cross-Origin Resource Sharing (CORS)

Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources.

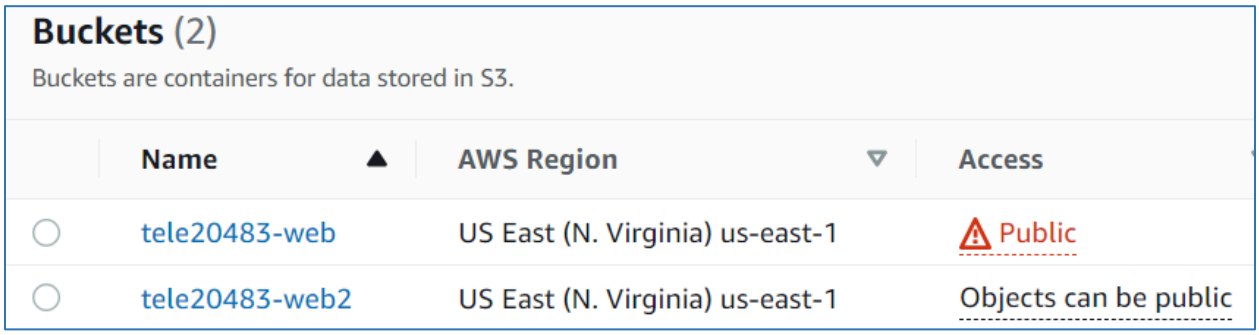
- Keep the website from the previous activity.
- Create another website now.



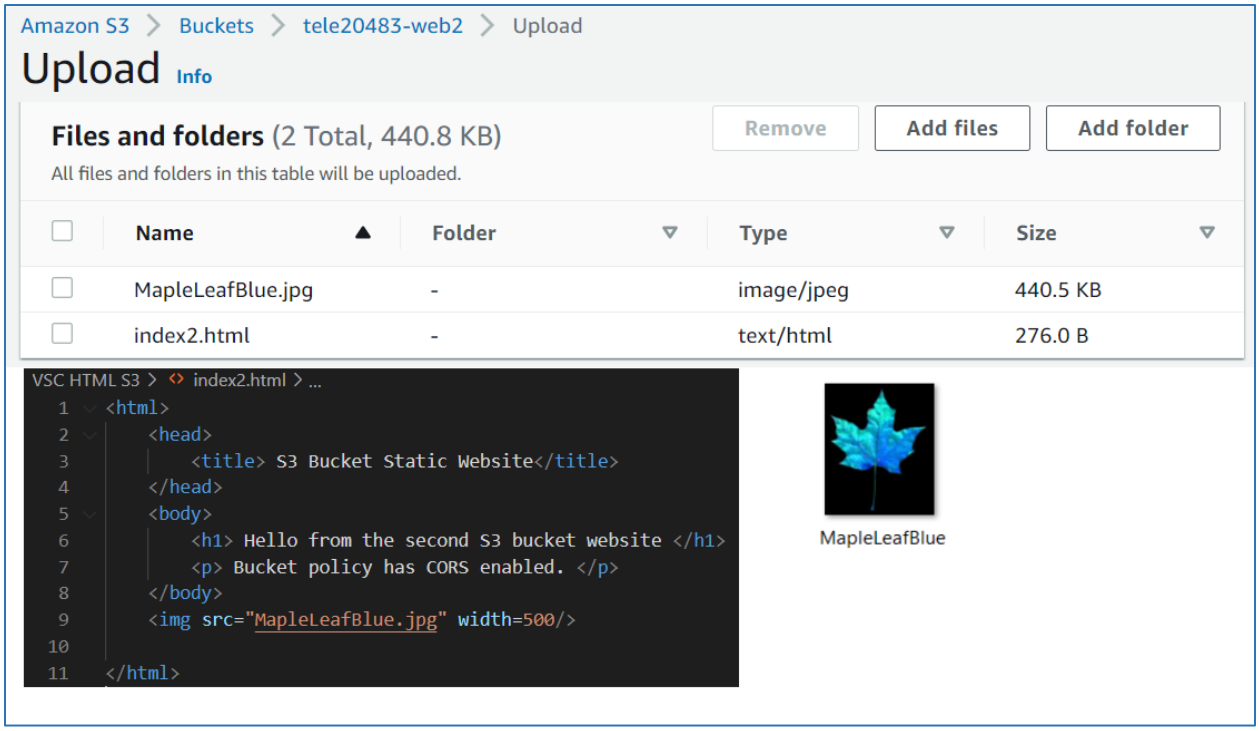
- Uncheck the ACL that blocks the public access.



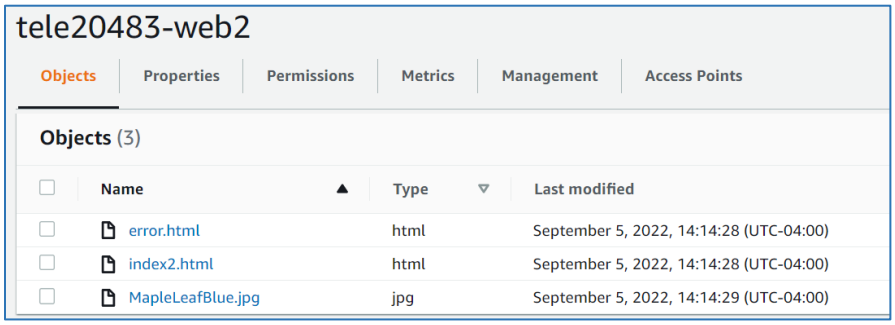
- This is the situation so far. There are two buckets tele20483-web and tele20483-web2.



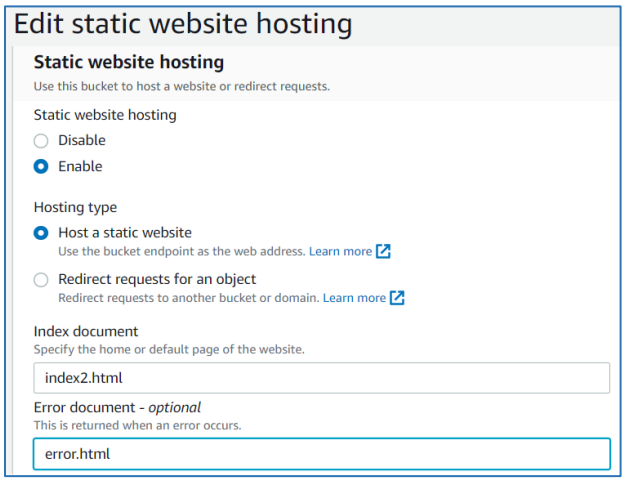
- Proceed to complete the configuration of tele20483-web2.
- Create and upload the index.html, error.html and the image jpeg file as shown below.



- Upload is completed.



- Configure the website.



- Configure the Bucket Policy under the bucket's permissions.

Policy JSON Document

```
{
  "Id": "Policy1662401980020",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1662401979029",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::tele20483-web2/*",
      "Principal": "*"
    }
  ]
}
```

	Name	AWS Region	Access
○	tele20483-web	US East (N. Virginia) us-east-1	Public
◉	tele20483-web2	US East (N. Virginia) us-east-1	Public

- Obtain the Bucket website endpoint and test the access.

Static website hosting

Use this bucket to host a website or redirect requests.

Static website hosting

Enabled

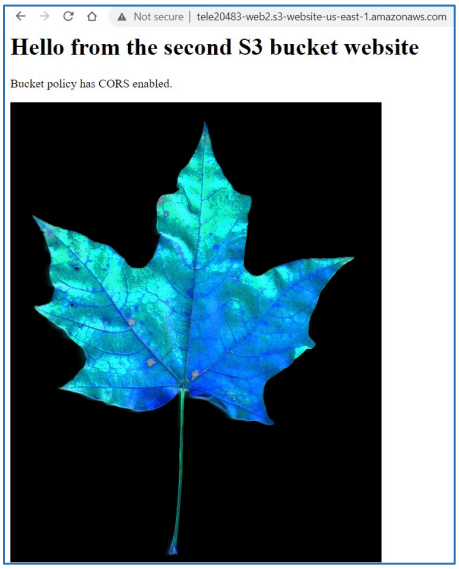
Hosting type

Bucket hosting

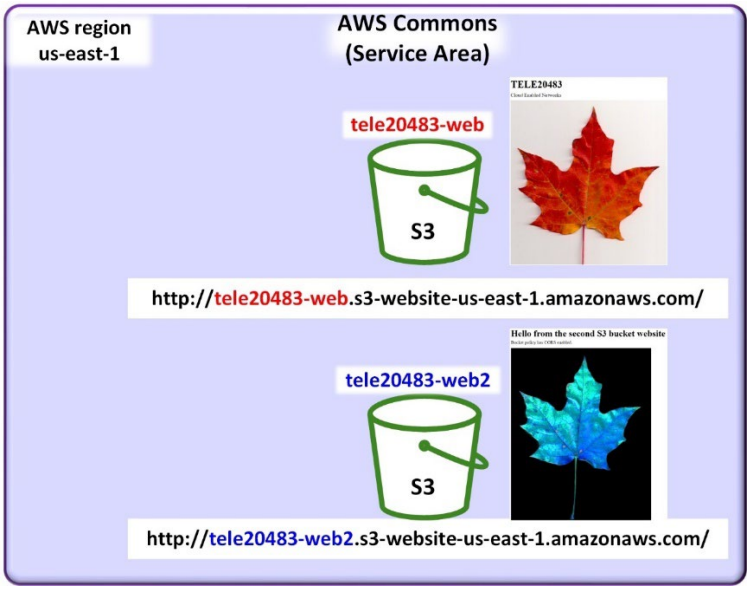
Bucket website endpoint

When you configure your bucket as a static website, the website is available at the AWS Region-specific website endpoint of the bucket.

http://tele20483-web2.s3-website-us-east-1.amazonaws.com



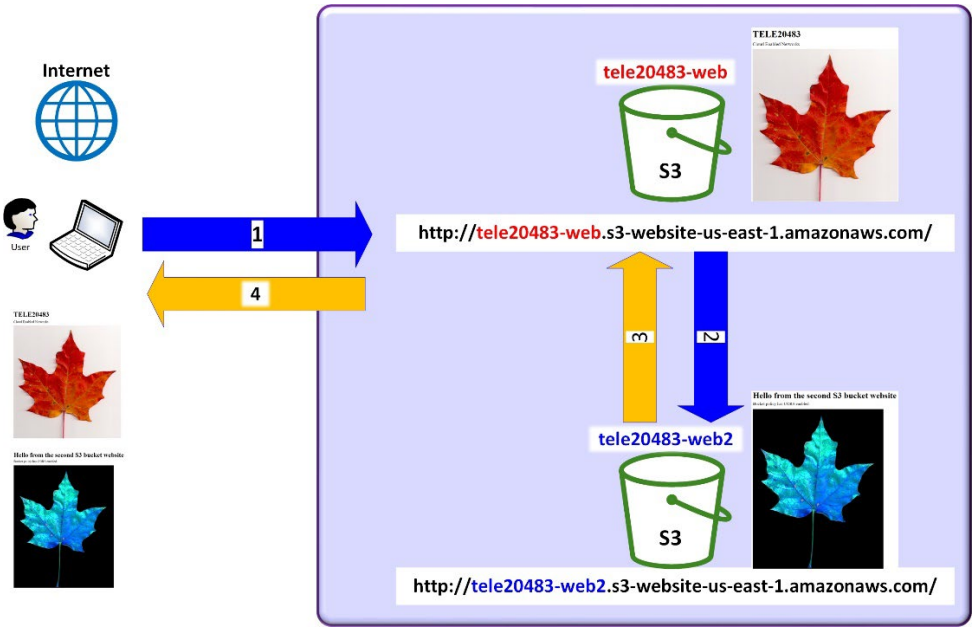
This is the situation right now. There are two functioning websites running on separate S3 buckets.



Representation of two S3 buckets enabled as static websites.

The objective of this demonstration is to build an application that supplies content from another location. Modern websites are built to access content from other domains and trusted third parties. However, this must be enabled carefully because it entices a security risk. This is done with a CORS configuration.

Cross-Origin Resource Sharing (CORS) [6,7] is a mechanism that evaluates the header of the HTTP protocol to allows a server to load resources from another origin. In the current demonstration, CORS will enable a user reaching for the top website (orange maple leaf), to also get the content objects from the second website (blue maple leaf).



The top website application retrieves data objects from the bottom webserver.

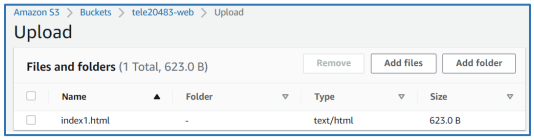
- First, create this new index.html [12] file to replace the current index file of the top bucket.
- Notice the fetch command contains the **bucket website endpoint** of the bottom bucket.

```
<html>
<head>
    <title> Hello from the first S3 bucket website</title>
</head>
<body>
    <h1> TELE20483 </h1>
    <p> Cloud Enabled Networks </p>
</body>
<img src="MapleLeaf.jpeg" width=500 />
<!--Cross Origin Resource Sharing-->
<div id="tofetch" />
<script>
    var tofetch = document.getElementById('tofetch');
    var second = 'http://tele20483-web2.s3-website-us-east-1.amazonaws.com';
    fetch(second + '/index.html')
        .then((response) => {
            return response.text();
        })
        .then((html) => {
            tofetch.innerHTML = html;
            document.getElementById('MapleLeafBlue').src = second + '/MapleLeafBlue.jpg';
        });
</script>
</html>
```

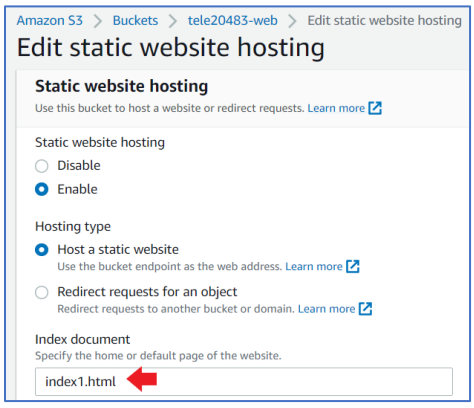
The URL of the second website (the blue maple leaf)

- Upload the new index file.

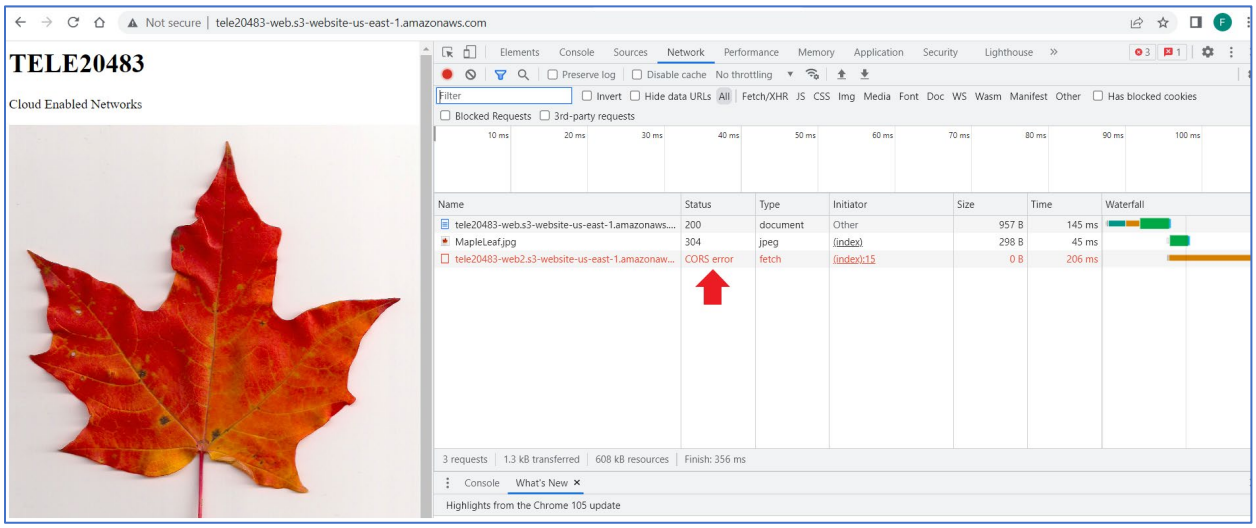
Note: the S3 service has a small latency when a file is replaced by another with the same name. It takes some time for the service to remove the previous file and replace it with the new one. Consequently, wait for a little while before proceeding with the testing.



- Replace the index.html file with the new index1.html.



- Try to access the top website using the bucket website endpoint.
- Open the **Developer Tools, Network** of the **web browser (Chrome** in this example**)** to observe the transactions.



The Developer Tool window of the Chrome browser shows that it downloaded the first website page, but it failed to retrieve the second website page. It shows that a CORS error happens. Basically, the orange maple leaf website tried to access the data, but the blue maple leaf website rejected the request.

The solution to this impasse is to create a **cross-origin resource sharing (CORS) policy** that authorizes the access from the orange maple leaf website to the resources in the blue maple leaf website. The CORS policy is a JSON document that allows the headers with the origin of the accepted website. In this case, the endpoint of tele20483-web.s3-website-us-east-1.amazonaws.com is the allowed origin. The authorized access method is the HTTP verb GET. Once that the CORS policy is enabled, the orange maple leaf website will be able to access the resources in the blue maple leaf website.

- Under the bucket's **Permissions**, create the CORS policy as shown below:

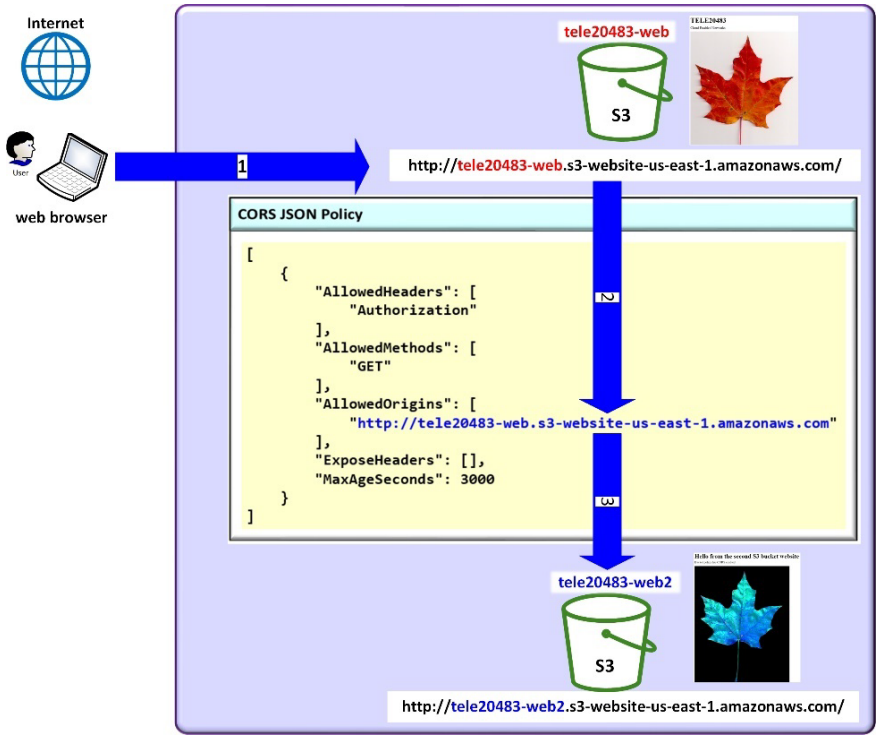
Amazon S3 > Buckets > **tele20483-web2** > Edit cross-origin resource sharing (CORS)

Cross-origin resource sharing (CORS)

The CORS configuration, written in JSON, defines a way for client web applications that are loaded in one domain to interact with resources in a different domain.

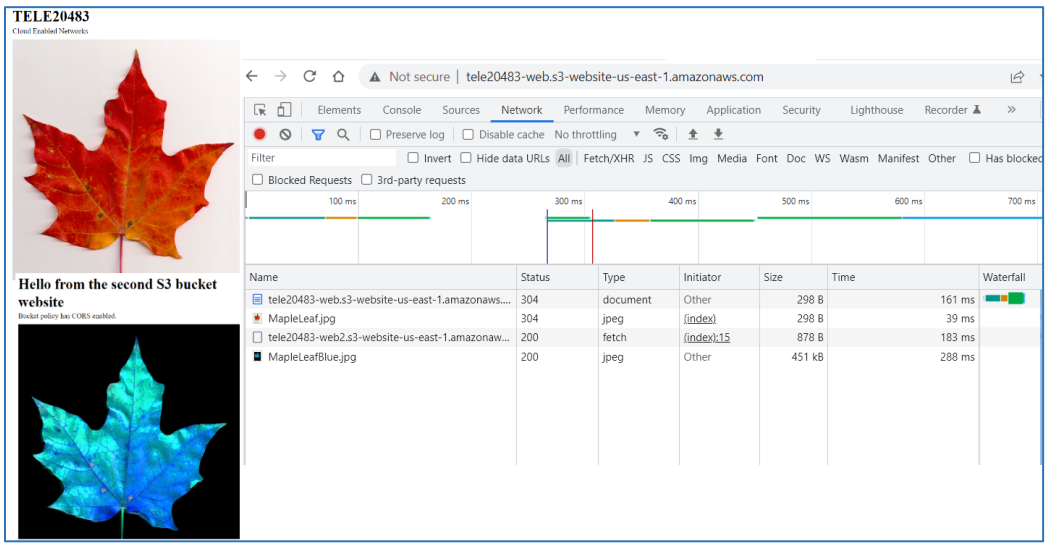
```
[
  {
    "AllowedHeaders": [
      "Authorization"
    ],
    "AllowedMethods": [
      "GET"
    ],
    "AllowedOrigins": [
      "http://tele20483-web.s3-website-us-east-1.amazonaws.com"
    ],
    "ExposeHeaders": [],
    "MaxAgeSeconds": 3000
  }
]
```

A web client attempts to access the tele20483-web site now. The home page HTML file contains a reference to the other website tele20483-web2. The client will be served files located in the second website. The CORS policy on tele20483-web2 allows that action by matching the endpoint of tele20483-web as it is summarized in the graphical representation below.



Logic of a web application accessing another website resources.

A new request shows that it is working now. The developer tool in the client's web browser helps again to explain the transactions. The HTTP code 304 is a **redirection response** telling the client that it is being redirected to another site. The HTTP 200 codes are successful GET actions (CRUD Read verb [7]).



Chrome browser's developer tool showing the HTTP transactions.

Learning Activity

Interacting with S3 buckets using AWS CLI

AWS CLI has two ways to deal with the S3 bucket service, they are the s3api and s3 command sets. The s3api is the direct use of the Application Programmable Interface (API) of the S3 service. The commands are structured around the JSON model of the S3 API. In other words, to issue s3api commands the administrators must be acquainted with the JSON structure of the data being queried and the API transactions. The s3api commands provide a great level of granularity. On the other hand, the AWS CLI S3 commands are part of a higher-level model that abstracts the command line interaction of the user with the S3 service.

- The following **AWS CLI s3api** list all the S3 buckets present in the account with the creation dates.

```
aws-academy-console:~$ aws s3api list-buckets
{    "Buckets": [ { "Name": "tele20483-web","CreationDate": "2022-09-05T14:47:45+00:00"},
                { "Name": "tele20483-web2","CreationDate": "2022-09-05T18:08:09+00:00"}],
    "Owner": {"DisplayName": "awslabsc0w405893068","ID": "1bbc8b33b64b999299d5ed"}
}
```

- The following AWS CLI s3api command shows how to sort throughout the previous JSON data structure using the "Buckets" key and then the key "Name".

```
aws-academy-console:~$ aws s3api list-buckets --query "Buckets[].Name"
[
    "tele20483-web",
    "tele20483-web2"
]
```

- Similarly, the key "CreationDate" is used in the following example.

```
aws-academy-console:~$ aws s3api list-buckets --query "Buckets[].CreationDate"
[
    "2022-09-05T14:47:45+00:00",
    "2022-09-05T18:08:09+00:00"
]
```

- This **AWS CLI s3** command lists all the existing buckets in the account. Notice the simplicity of that the s3 command abstraction provides.

```
aws-academy-console:~$ aws s3 ls
2022-09-05 07:47:45 tele20483-web
2022-09-05 11:08:09 tele20483-web2
```

- This s3 command lists all the files in the target S3 bucket.

```
aws-academy-console:~$ aws s3 ls s3://tele20483-web2
2022-09-05 11:14:29     451079    MapleLeafBlue.jpg
2022-09-05 11:14:28         37    error.html
2022-09-05 11:14:28        276    index2.html
```

- This s3 command creates an S3 bucket in the account's default region.

```
aws-academy-console:~$ aws s3 mb s3://tele20483-demo
make_bucket: tele20483-demo

aws-academy-console:~$ aws s3 ls
2022-09-08 16:00:47 tele20483-demo
2022-09-05 07:47:45 tele20483-web
2022-09-05 11:08:09 tele20483-web2
```

- This s3 command deletes an S3 bucket.

```
aws-academy-console:~$ aws s3 rb s3://tele20483-demo
make_bucket: tele20483-demo

aws-academy-console:~$ aws s3 ls
2022-09-05 07:47:45 tele20483-web
2022-09-05 11:08:09 tele20483-web2
```

- This s3 command deletes a file in the S3 bucket.

```
aws-academy-console:~$ aws s3 ls s3://tele20483-web (before)
2022-09-05 08:35:13     607255 MapleLeaf.jpg
2022-09-05 08:01:43     451079 MapleLeafBlue.jpg
2022-09-05 08:02:55         38 error.html
2022-09-05 08:37:10        234 index.html
2022-09-05 11:43:02        623 index1.html
2022-09-05 08:32:57        230 olderindex.html

aws-academy-console:~$ aws s3 rm s3://tele20483-web/MapleLeafBlue.jpg
delete: s3://tele20483-web/MapleLeafBlue.jpg

aws-academy-console:~$ aws s3 ls s3://tele20483-web (after)
2022-09-05 08:35:13     607255 MapleLeaf.jpg
2022-09-05 08:02:55         38 error.html
2022-09-05 08:37:10        234 index.html
2022-09-05 11:43:02        623 index1.html
2022-09-05 08:32:57        230 olderindex.html
```

Learning Activity

Interacting with the S3 service using AWS Python SDK

Another way to interact with the S3 service is using boto3, the AWS SDK for Python. The following script list all [11] the buckets present for the account's default region.

```
# This python script lists all S3 buckets names.

import boto3
# Create an S3 client.
s3 = boto3.client('s3')

# Make a call to list all S3 buckets.
response = s3.list_buckets()

# Print the names of the S3 buckets in the list
print('These are the S3 buckets:')
for bucket in response['Buckets']:
    print(bucket["Name"])
```

```
aws-academy-console:~$ python list_buckets3.py

These are the S3 buckets:
tele20483-web
tele20483-web2
```

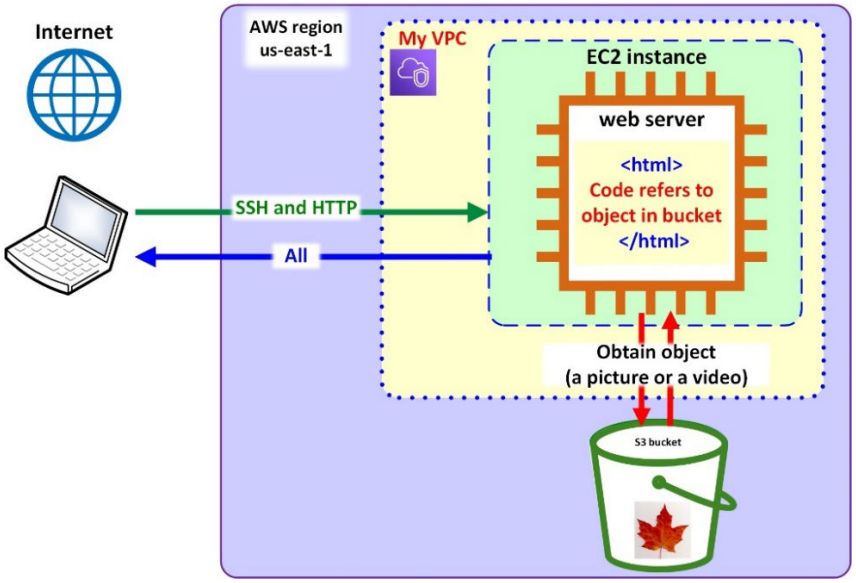
The following Python script creates S3 buckets with the supplied name.

```
# This python script lists all S3 buckets names.
import boto3
name = input('Enter the name of the S3 bucket: ')
s3 = boto3.client('s3')
s3.create_bucket(Bucket=name)
```

```
aws-academy-console:~$ python make_anS3bucket.py

Enter the name of the S3 bucket: tele20483-python
aws-academy-console:~$ python list_buckets3.py
These are the S3 buckets:
tele20483-python
tele20483-web
tele20483-web2
```

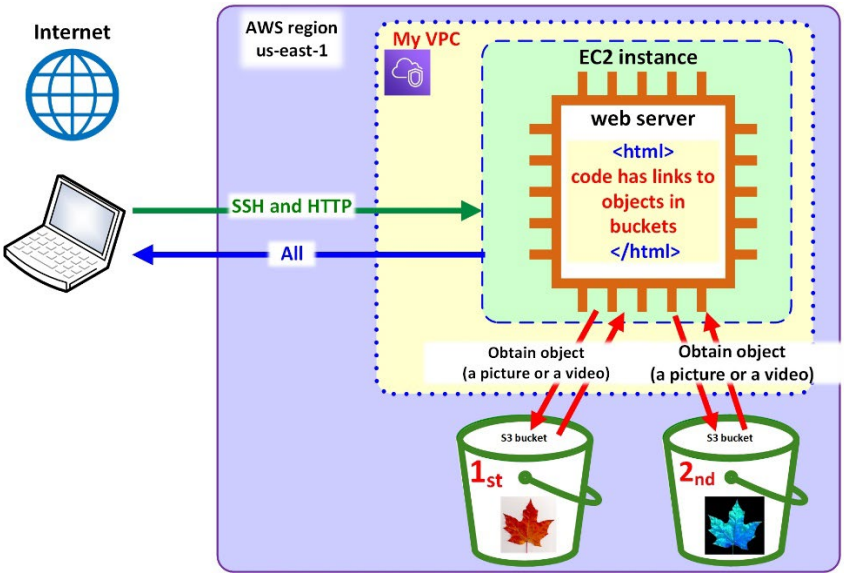
Chapter 9 Coursework

Implement the topology in the following diagram. A webserver running on a virtual machine supplies an object located in an S3 bucket (this S3 bucket is not a static webserver) The object might be a picture, a video, or a song. The HTML code in the home page must have the button that access the object. A client browsing the website has access to the object(s) by clicking HTML button(s).



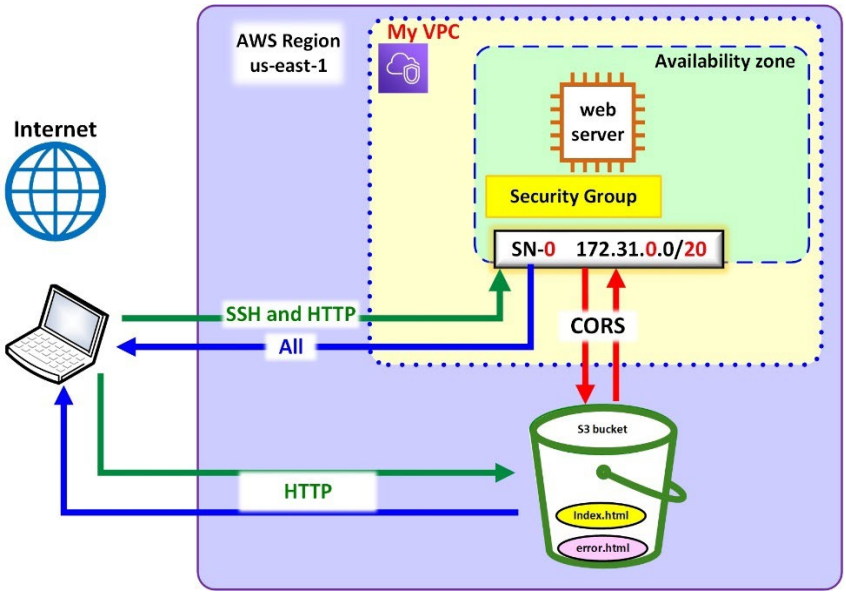
Webserver supplies content from an S3 bucket.

Implement the topology in the following diagram. This is an extension of the previous situation. The web application running in the EC2 webserver has links pointing to objects located in two S3 buckets (these buckets are not static webservers; they are just used for object storage). A client browsing the website has access to the objects by clicking HTML buttons.



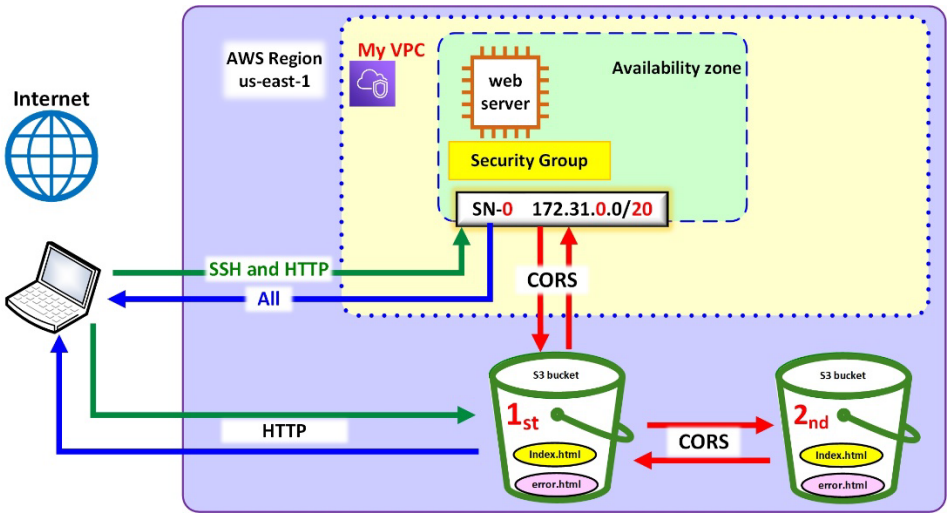
Webserver supplies content from S3 buckets.

Implement the topology in the following diagram. A static website, configured on an S3 bucket, allows cross-origin resource sharing with a webserver installed on an EC2 instance. When the web client accesses the EC2 webserver, it should obtain the EC2's web homepage followed by the S3 bucket's web homepage. Additionally, the S3 bucket website should be accessible directly from the Internet as shown in the diagram.



S3 bucket website allows CORS to the EC2 instance website.

Implement the topology in the following diagram. A webserver on an EC2 instance is publicly accessible from the Internet. Another static website is located on an S3 bucket (labelled 1st in the diagram). This S3 has a CORS policy allowing the access from the EC2's website. Also, another S3 static webserver (labelled 2nd in the diagram) allows CORS access to the 1st bucket website.



S3 bucket website allows CORS to the EC2 instance website.

References

[1] IBM Cloud. What is Object Storage? 2022. [Online]. Available: https://www.ibm.com/cloud/learn/object-storage

[2] AWS. Bucket restrictions and limitations. 2022. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/BucketRestrictions.html

[3] Amazon Simple Storage Service Documentation. 2022. [Online]. Available: https://docs.aws.amazon.com/s3/index.html

[4] Amazon S3 Storage Classes. 2022. [Online]. Available: https://aws.amazon.com/s3/storage-classes

[5] AWS. Grammar of the IAM JSON policy language. 2022. [Online]. Available: https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_grammar.html

[6] Mozilla Developers Documents. Cross-Origin Resource Sharing (CORS). 2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[7] AWS. Using cross-origin resource sharing (CORS). [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/cors.html

[8] REST API Tutorial, Using HTTP Methods for RESTful Services. 2022. [Online]. Available: https://www.restapitutorial.com/lessons/httpmethods.html

[9] AWS Developer Tools Blog. Leveraging the s3 and s3api commands. 2022. [Online]. Available: https://aws.amazon.com/blogs/developer/leveraging-the-s3-and-s3api-commands

[10] Using high-level (s3) commands with the AWS CLI. [Online]. Available: https://docs.aws.amazon.com/cli/latest/userguide/cli-services-s3-commands.html

[11] Creating and Using Amazon S3 Buckets. 2022. [Online]. Available: https://boto3.amazonaws.com/v1/documentation/api/1.9.42/guide/s3-example-creating-buckets.html

[12] Many thanks to Mark Geiman and Irina Geiman for index.html file.

Chapter 10

Cloud Databases

Description

This section introduces the relational database cloud service (RDS). An EC2 instance running Linux Ubuntu is deployed to host a webserver Apache2. This webserver also runs the scripting language PHP to make calls to a relational MySQL database. This conforms a LAMP stack which is the most common platform for web applications. Several underlay concepts are revealed during the configuration of the different components of the stack.

Learning Outcomes

- Deploy an elastic database in the cloud.
- Configure a LAMP stack in the cloud.
- Examine the components of the LAMP stack.
- Configure Linux, Apache2, PHP and MySQL.

Main concepts

- The database.
- The relational database.
- The non relational database.
- The Linux, Apache2, MySQL, PHP, (LAMP) service stack.

Learning Activities

- Deployment of the webserver of the LAMP stack
- Deployment of the MySQL database of the LAMP stack.
- Examining the database parameters
- Adding data to the database
- Configuration of the webserver to talk to the database

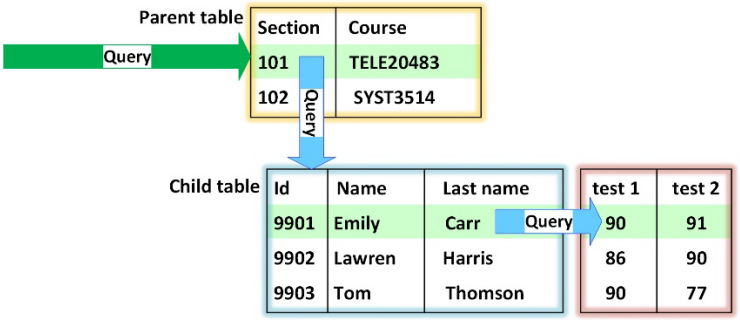
Introduction to database

A database is a collection of information stored in a computer system integrated with a management system. This management tool stores and retrieves the data using a given index or a key. There are two main database types depending on the organizational format of the data: relational and non-relational.

Relational Database

A relational database stores data records in pre-defined columns and rows in a table. A database can be composed of multiple tables which are related in some way to each other, hence the term relational. The structure of the tables and its data is defined by a **schema**. This database schema is described in the language supported by the database management system. All relational databases are ruled by schemas.

Relational database are organized in columns and rows. The **columns** define **attributes**. The **rows** contain the **records of data**. These fields are defined when the tables are created by the administrator. Since the data is arranged in a strict manner, the data can be searched efficiently with the use of keys.



Example of a query on a relational database.

A key is a unique identifier assigned to a row of data in a table. For example, the course section number above is the primary key to initiate a search of a user's data. The management system sends a query with a string that contains the keys to retrieve the data from the different tables. The queries to the databases require to follow rules of syntax and semantics, therefore a defined language is needed. Such language is the **Structured Query Language (SQL)**.

Non-relational databases

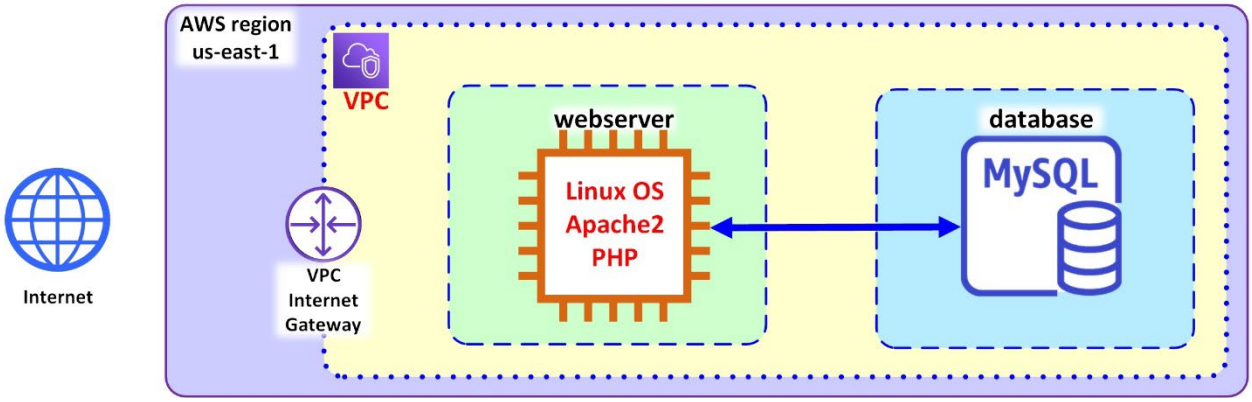
Non-relational databases store data with variable structure. That implies that the data that does not share the same attributes or format. Hence, it is not possible to pre-define a schema with fixed rows and columns as it would be the case with a relational database. Non-relational databases are also called NoSQL databases. Because there are no keys, there is no way to make join queries to different tables.

However, non-relational databases allow the data to have flexible formats. Thus, a non-relational database can store data files, documents, pictures, videos and in general any kind of data format. Consequently, they have ample use in modern applications that store and retrieve content such as Internet social media, data collection and processing applications.

Configuration of a webserver stack in AWS Cloud

The most common deployment of web service applications includes a webserver that faces the clients and a database to store data. The webserver application acts as a proxy between the client and the database retrieving and supplying information.

A web-database stack is typically composed of a server running the operating system Linux, a web server application, a scripting language, and a database in the backend. There are different software combinations to make up this stack. For instance, Linux, Apache2, PHP and MySQL conform a LAMP stack while Linux, NGINX, PHP and MySQL conform a LEMP. There is yet another combination that uses Python Flask instead of PHP for scripting. In any case, the typical topology looks like the following figure.



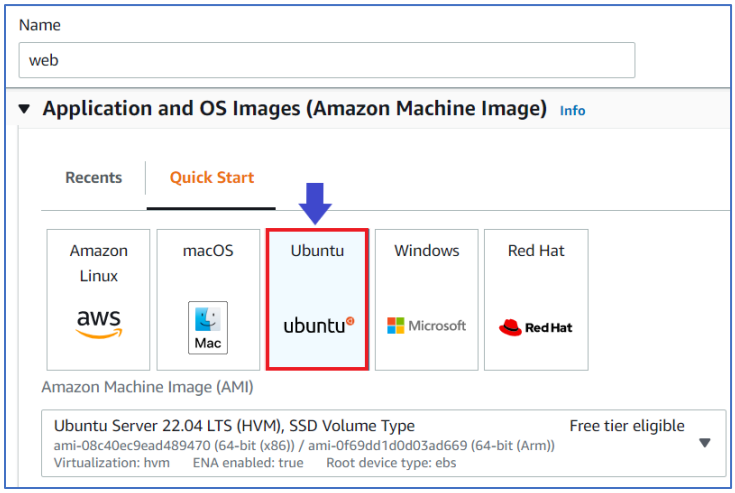
A Linux, Apache2 webserver, PHP, MySQL stack (LAMP)

Learning Activity

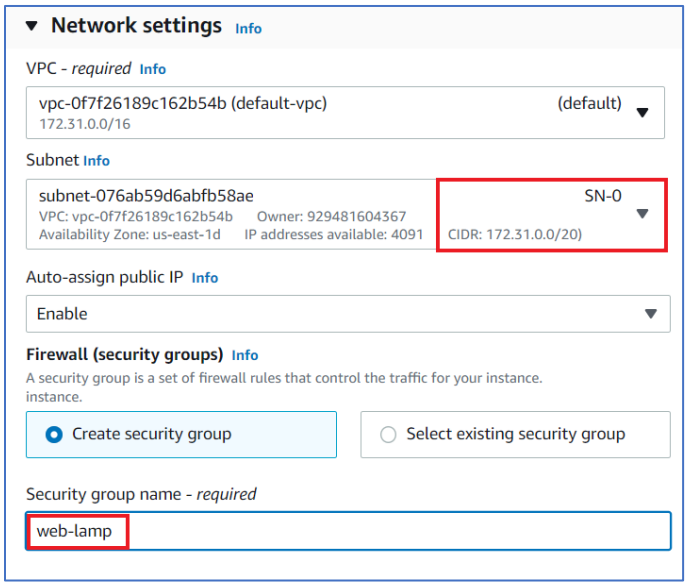
Deployment of the webserver of the LAMP stack

Installation of the webserver.

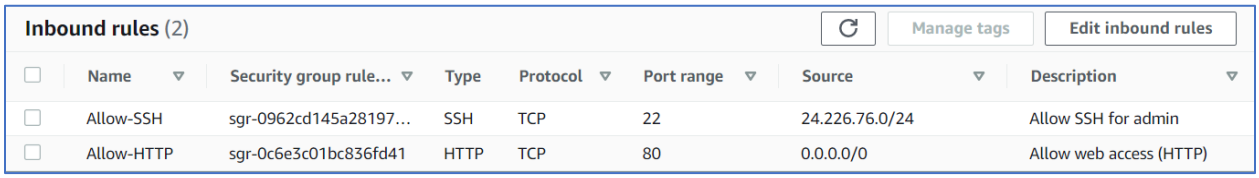
The first step in the LAMP stack configuration is to deploy an EC2 instance on Linux Ubuntu server. Choose the latest ubuntu server image.



This EC2 instance will be deployed in the subnet 172.31.0.0/20 with a security group named web-lamp.



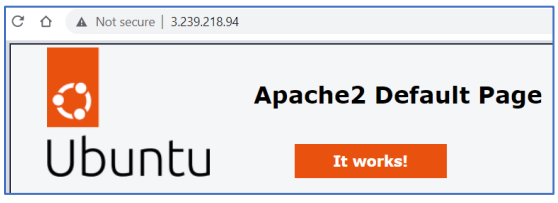
The security group has two rules to allow the administrative access using SSH from a specific subnet (adjust accordingly to location) and a rule to allow inbound web traffic from anywhere on the Internet.



Finally, the user data installs the webserver Apache2. Notice that the installation package manager is **apt-get** since this is a Linux Ubuntu server.

```
#!/bin/bash
apt-get update -y
apt-get install apache2 -y
systemctl restart apache2
systemctl enable apache2
```

The Apache2 webserver test home page shows in the web browser once that the installation is completed.



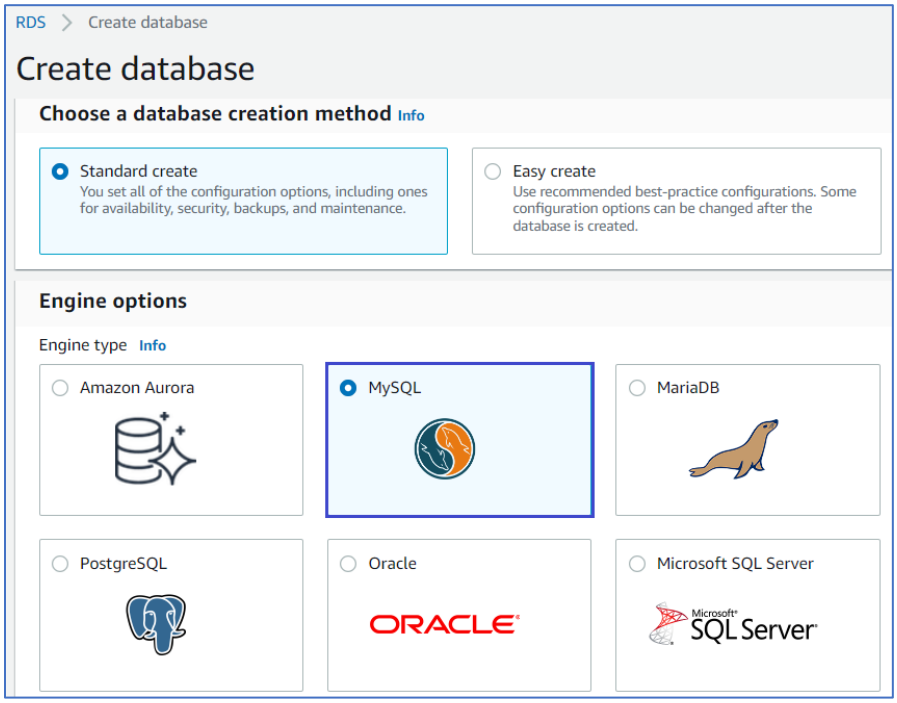
The EC2 instance is ready for now. Additional configuration will be required afterward once that the database is deployed.

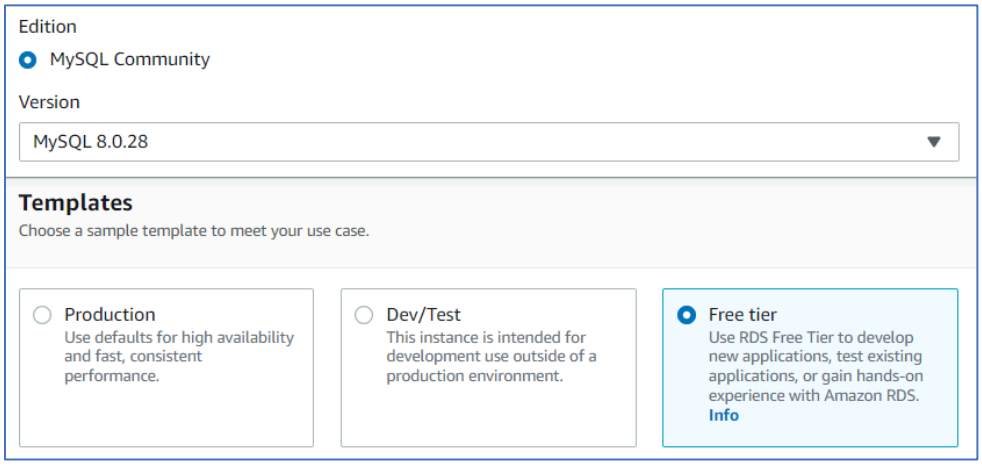
Learning Activity

Deployment of the MySQL database of the LAMP stack

Creation of the database using AWS Relational Database Services (RDS)

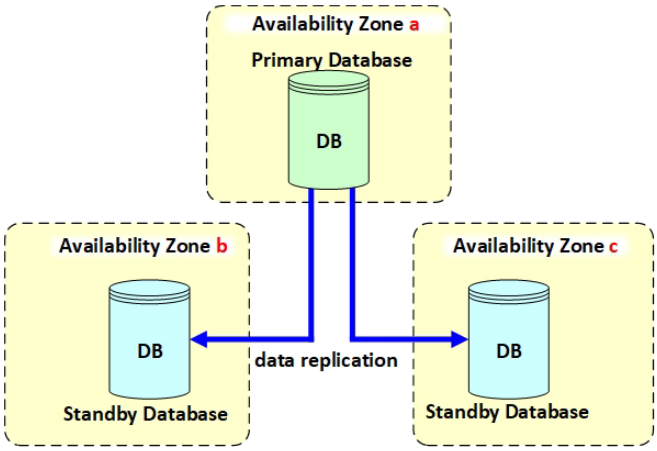
To install the database, Amazon RDS offers six relational database engines: Amazon Aurora, MySQL, MariaDB, PostgreSQL, Oracle, and Microsoft SQL server. In this example, the free tier MySQL community edition is selected. The creation method is standard to observe all the configuration settings.





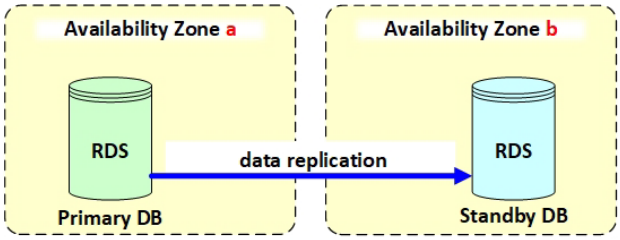
As part of its service commitment, the cloud provider offers levels of guarantee regarding availability and durability. Availability refers to the concept that the data will be available when it is requested. Durability refers to the concept that the data will not be lost or corrupted. AWS offers three deployment options to enable its service level agreements.

The **Multi-Availability Zone database cluster** [1] is a set of three database instances, each deployed in a different availability zone. One of the instances is the primary database while the other two remain in standby mode. A **replication** process maintains the databases data synchronized. The data **writes** can only be performed in the primary database. However, any of the database instances provide **reads** of the data. The Multi-AZ DB cluster offers the highest level of availability and durability but at a higher cost.



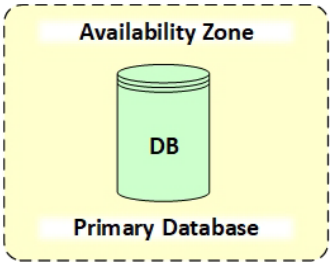
Multi-AZ DB Cluster [1].

The second deployment option is the **Multi-Availability Zone database instance** [1] which consist of one primary database deployed in an availability zone and a standby database replica located in another availability zone. The secondary database is only for backup purposes as only the primary can provide reads and writes of data.

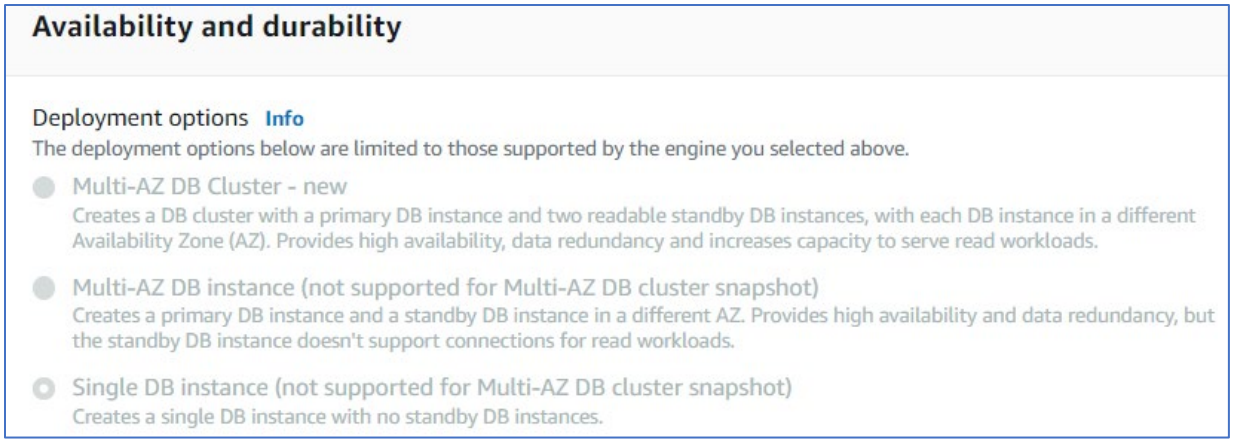


Multi-AZ DB Instance [1].

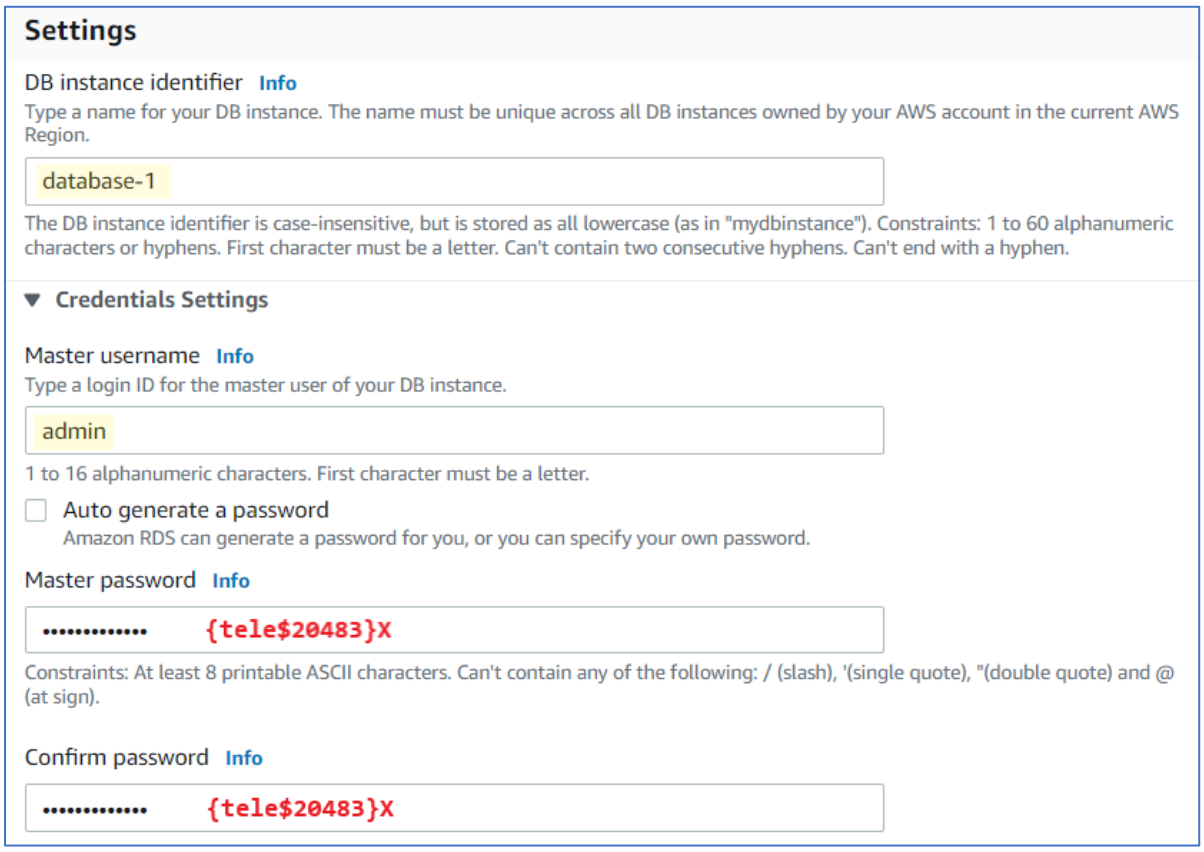
The last option is the **single database instance** [1] which as the name indicates, it is a just one DB instance. This option is automatically selected for the Free Tier case since it is the lowest cost option. This example uses this database instance option.



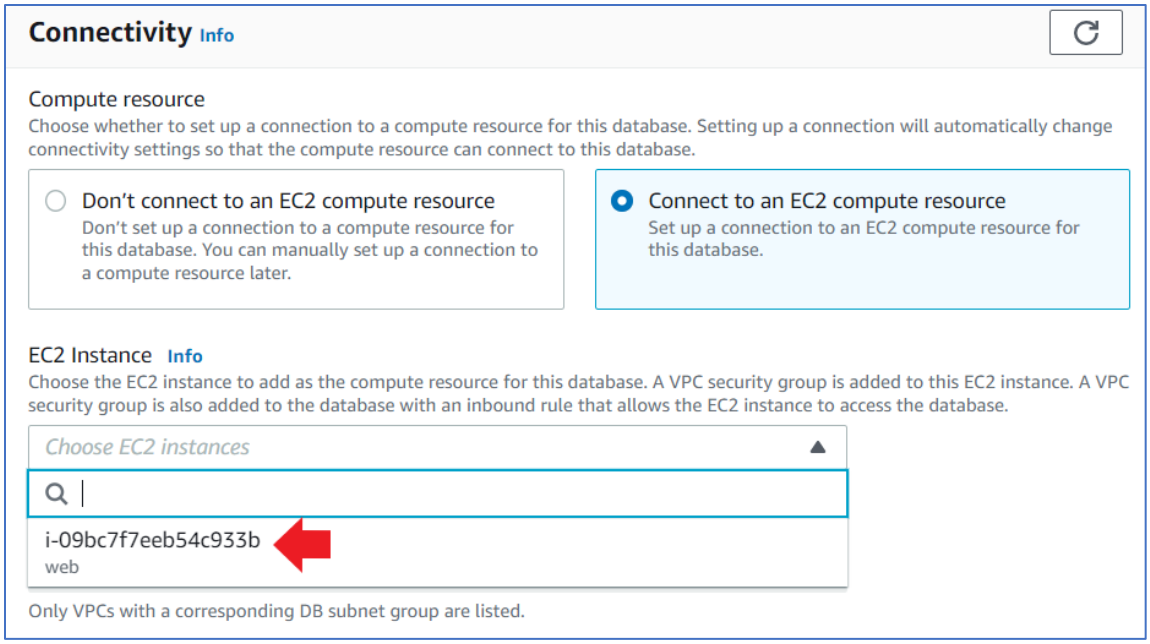
Single DB instance [1].



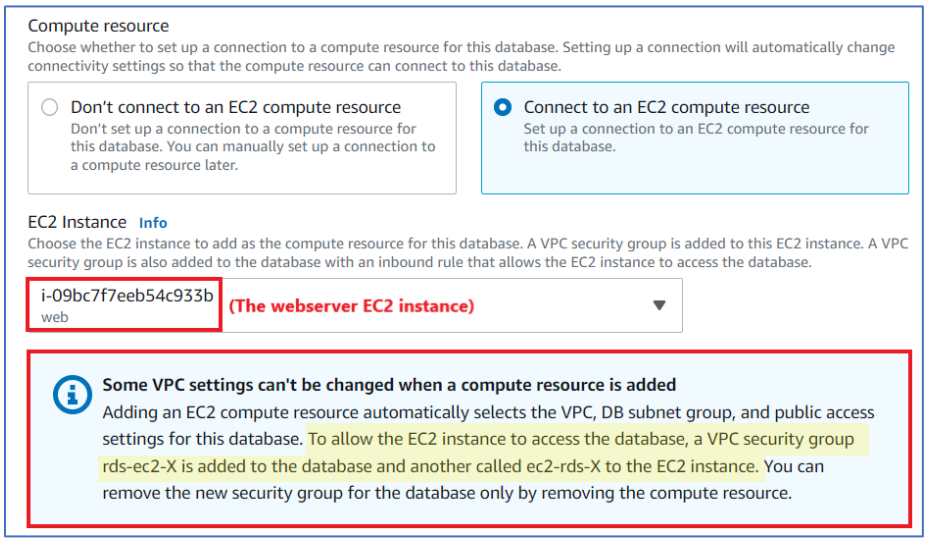
The database instance requires an identifier, a master administrator username, and an administrative access password.



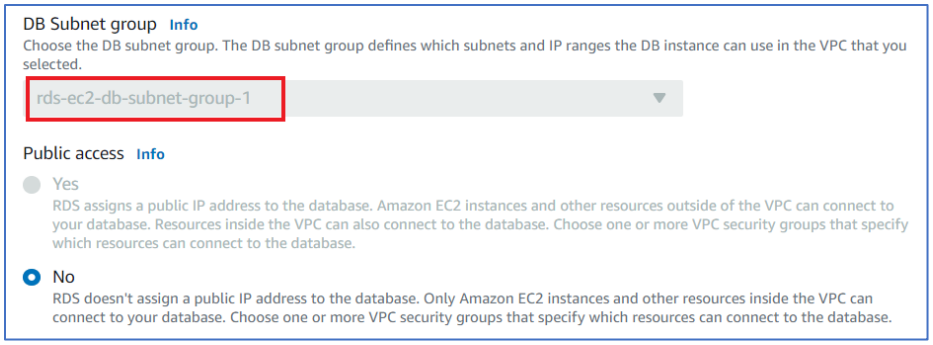
The database can run standalone or it might be part of an application stack which is the current case. Therefore, the security groups must be configured to allow the interconnection of the webserver with the database. This must be done manually if the option “Don’t connect to an EC2 computer resource (yet)” is selected. The other option “Connect to an EC2 computer resource” automatically configures the security groups thus reducing the manual work to set up the connection. Hence, this option is chosen here.



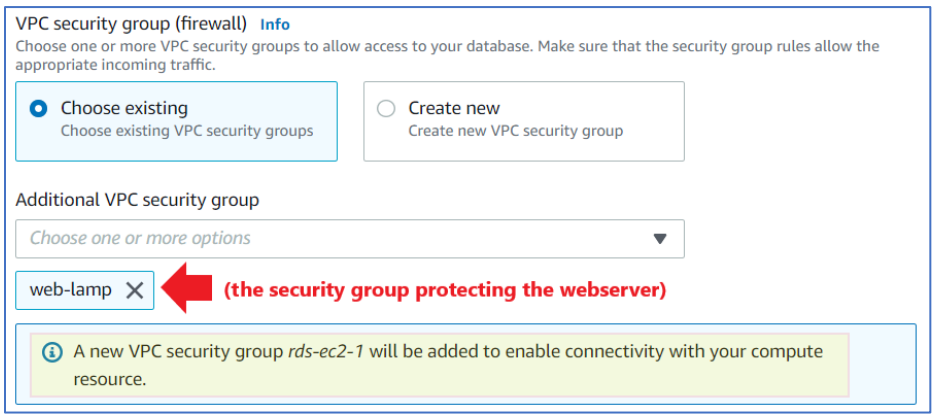
This option automatically configures two security groups. A security group names rds-ec2-1 on the side of the database and another ec2-rds-1 to the EC2 instance. The rules and meaning of these security groups will be analyzed in detail once that the database deployment is complete.



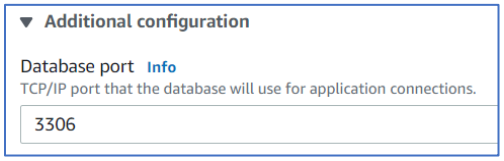
Like any other computer resource, the database must have network access. This can be selected by the administrator by configuring a **DB Subnet Group**. This is a set of existing subnets in the Virtual Private Cloud where the database is connected. In this example, there is no previous DB Subnet Group and instead the configuration of the DB Subnet group is done by the configuration wizard.



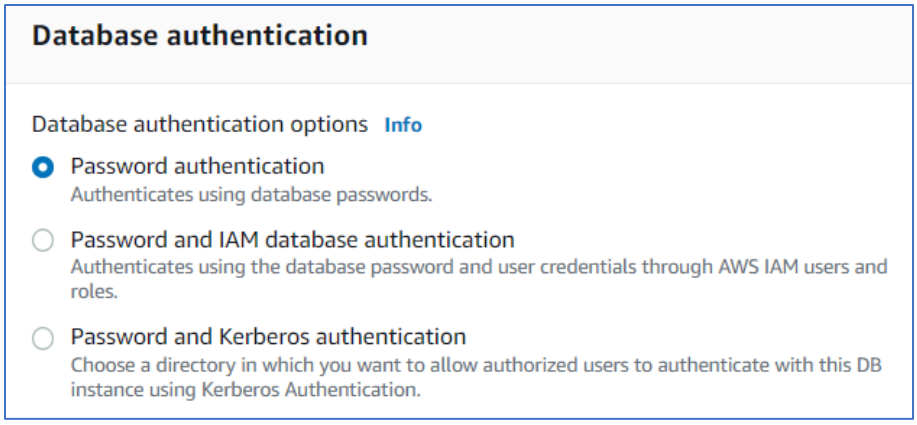
This database will not have public access at all. Consequently, the only way to access it will be by first SSH accessing the EC2, and from there, using MySQL commands to interact with the database. That means that the security group of the EC2 must be instructed to accept inbound MySQL traffic from the database and, conversely, the database security group must accept MySQL traffic coming from the EC2 instance.



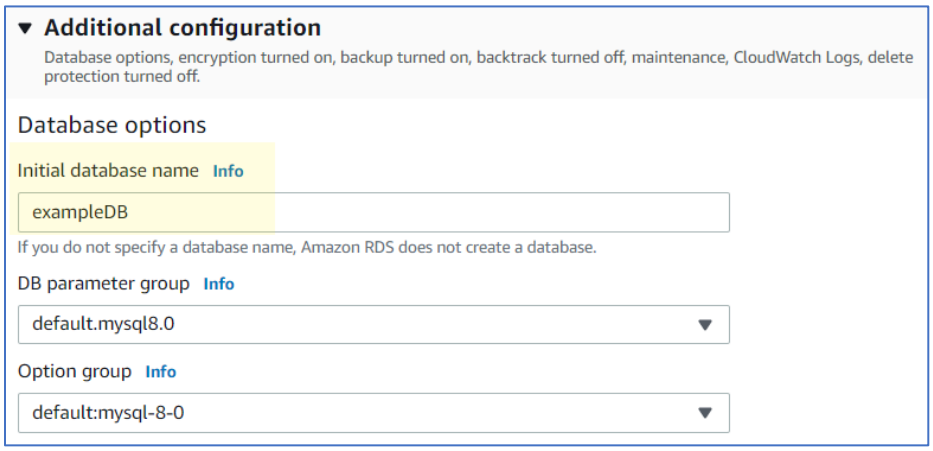
The default TCP port of MySQL is 3306, but the administrator could change that in the database settings. In this case, the default port 3306 is used.



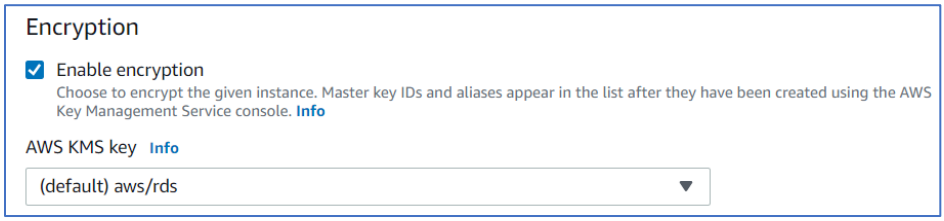
There are three authentication methods. Password authentication is the easiest and the weakest. AWS Identity and Access Management (IAM) is a web service that control the access to AWS resources. IAM controls the permission to access and use resources. The last option consists of a trusted server running the Kerberos protocol to issue authentication and authorization tickets to resources. This example chooses the easiest, and worst, password authentication.



The database instance can run multiple databases with different names. However, an initial database named exampleDB is created just to get started. If an initial database name is not supplied during this step, the database instance will start without a database.



The data exchanged between the webserver and the database will be ciphered by AWS using a KMS (Key Management Service) key under its control.



The configuration parameters are now complete and the database can be created. It takes a while to spin up the database. Once that it is completed, let's examine its details.

Learning Activity

Examining the database parameters

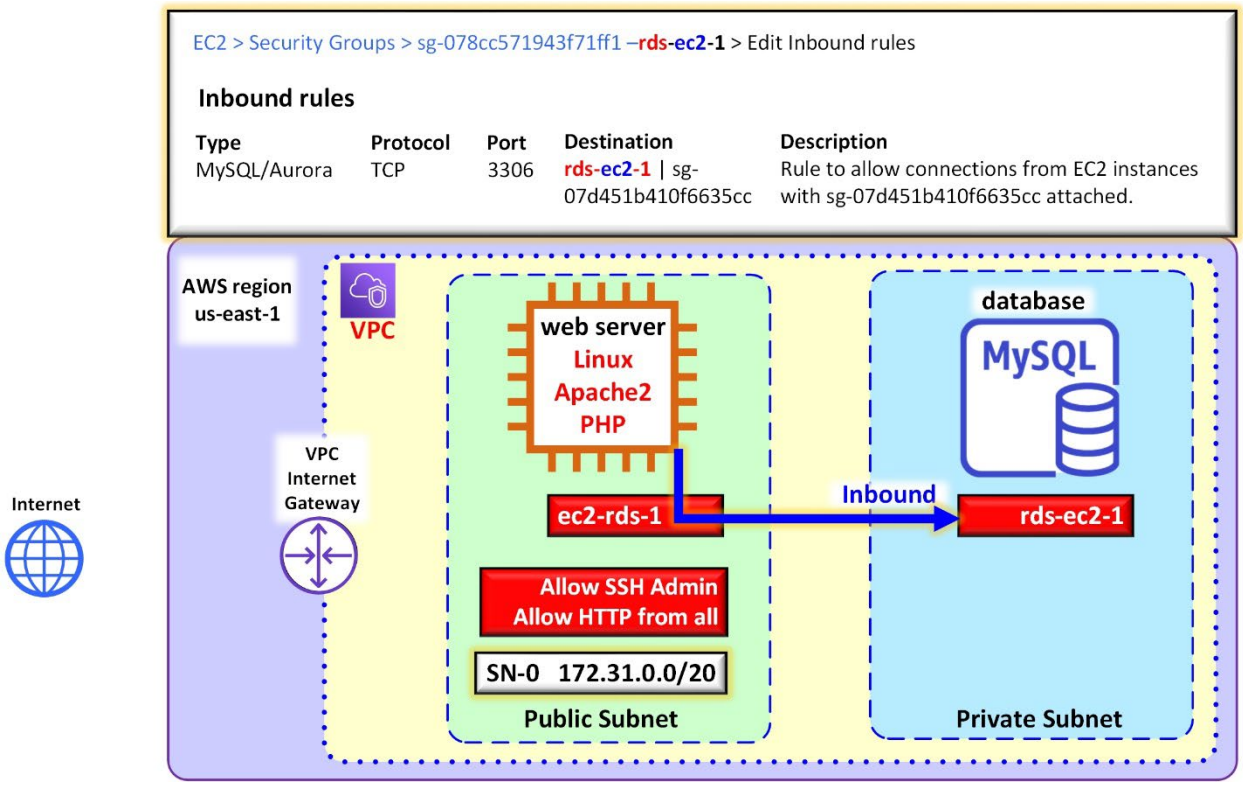
The database endpoint

The RDS service automatically registers a DNS A Record formed with the name of the database-1 together with a random string followed by the region name, the rds service name and the amazon's domain name. This creates a unique name identifier or **endpoint**. This is a critical piece of information because this endpoint is used to reach the database from within the VPC. The calls to the database are directed to that name via the TCP port 3306.

```
Endpoint
database-1.cmgtvogta7kh.us-east-1.rds.amazonaws.com
Port 3306
```

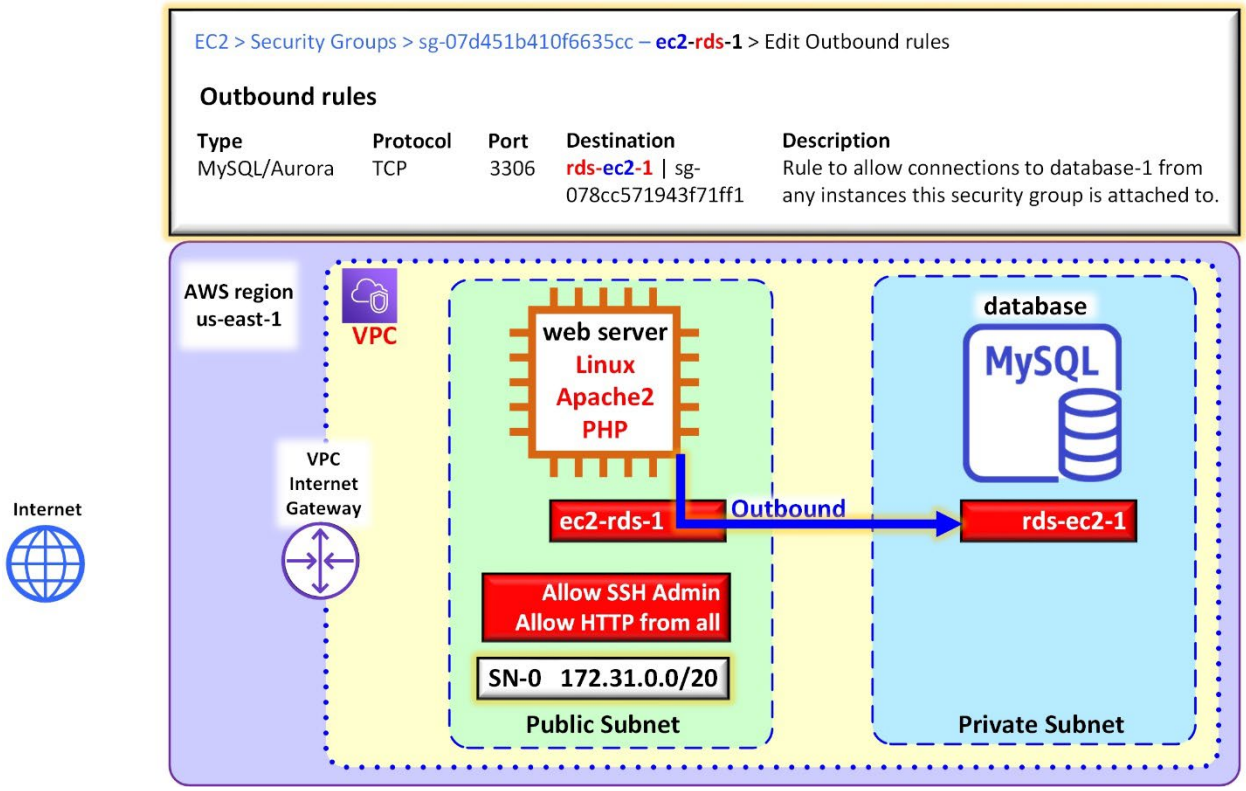
The security groups

The access to the database is controlled by security groups. The firewall rds-ec2-1 controls the **inbound** access from the EC2 instance to the database. Moreover, this security group has only one rule which allows inbound MySQL sourced from any EC2 instance with the security group ec2-rds-1 attached. The security group does not even have an outbound rule because the responses to the requests are automatically granted since security groups are stateful firewalls.



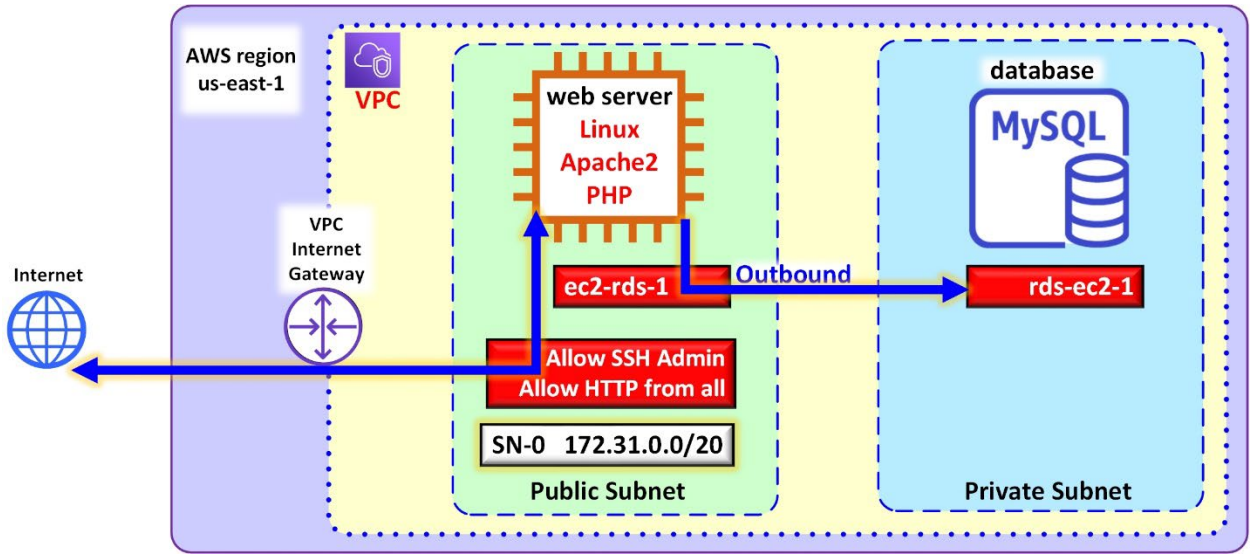
Security Group rds-ec2-1 allows MySQL access with a source in the other security group ec2-rds-1.

Another security group ec2-rds-1 allows the passage of the MySQL traffic leaving to the destination security group rds-ec2-1 that protects the database. The security group has only one rule which is outbound. The response to the traffic is implicitly allowed by the stateful firewall.



Security Group ec2-rds-1 allows the MySQL leaving toward the security group rds-ec2-1.

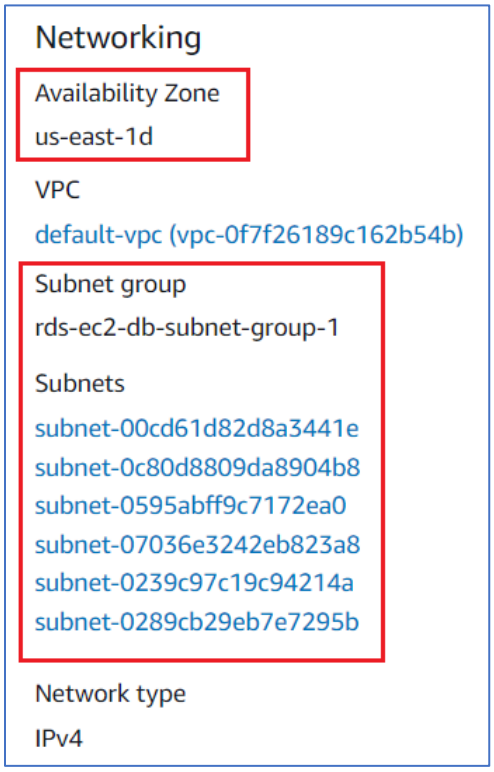
Hence, the EC2 instance is attached to two security groups. Once controls the inbound SSH and HTTP traffic from the Internet and the other controls the access to the MySQL database.

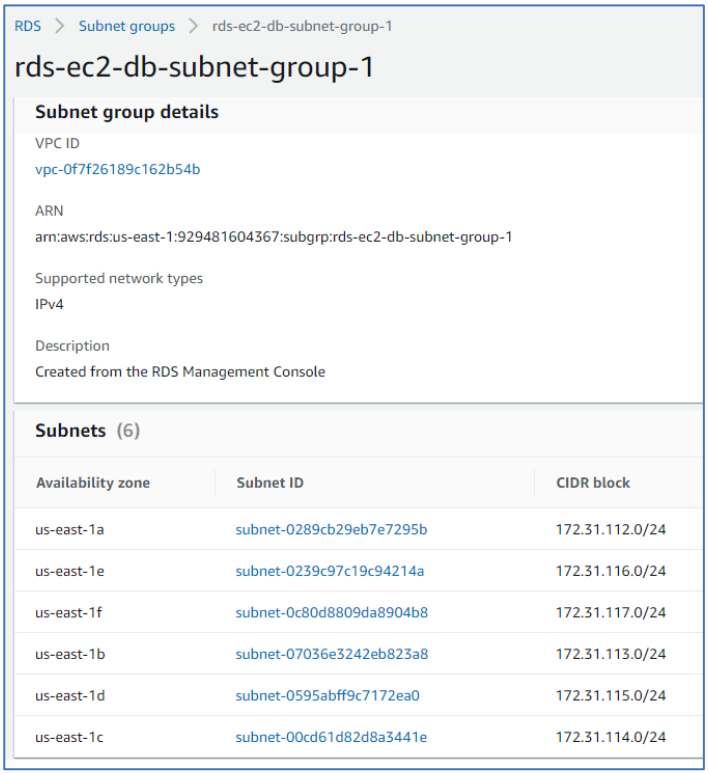


Security groups applied to the EC2 instance.

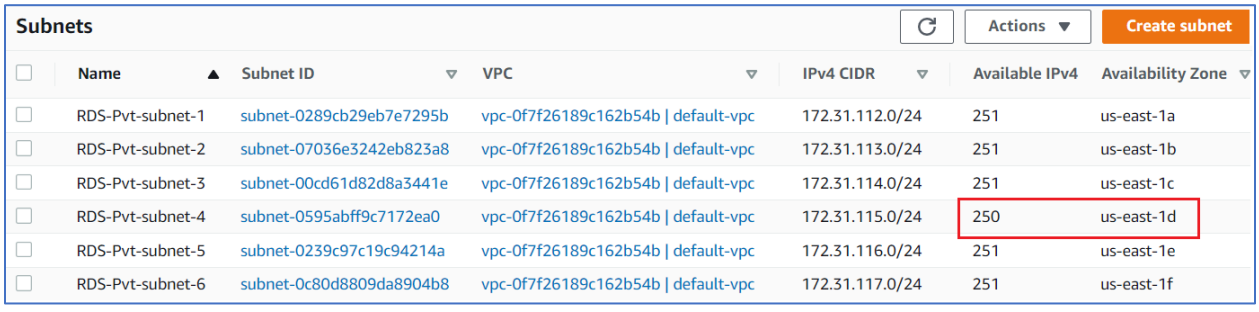
Networking

The RDS service created a subnet group of six new subnets by itself. Even though this is a single DB instance, the service configured the six subnets in case that the database might be migrated later to a more resilient arrangement such as the Multi-AZ DB Cluster.



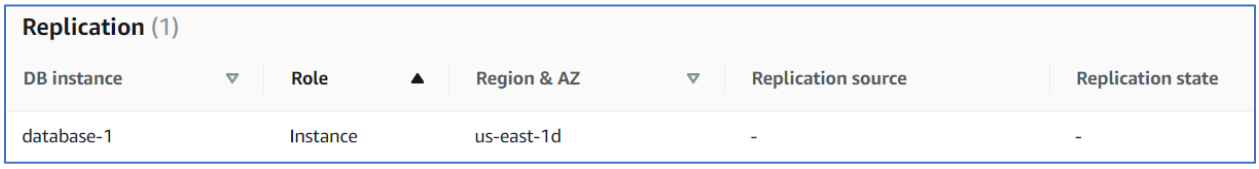


The database instance has been deployed in the subnet 172.31.115.0/24 in the availability zone us-east-1d. Notice that the DB instance has taken on one IPv4 address from the 251 host addresses.



Replication

There is no replication of data since there is just one database instance.



Learning Activity

Adding data to the database

Right now, the webserver is accessed via SSH to install an additional package, mysql-client, just for the purpose of testing MySQL on the database side. (Note: the SSH username is ubuntu, not ec2-user).

```
ubuntu@ip-172-31-4-78:$ sudo apt-get install mysql-client
```

This client is able to format and send the MySQL commands that the database understands. The following command request access for the admin user directed to the database endpoint using the protocol TCP port 3306. If the database responds then it means that the security groups are allowing the MySQL traffic.

```
mysql -u admin -p -h database-1.cmgtvogta7kh.us-east-1.rds.amazonaws.com --port 3306 --protocol=TCP
```

In effect, the database responds with the user prompt. The password is supplied and access to the mysql database is granted.

```
ubuntu@ip-172-31-4-78:~$ mysql -u admin -p -h database-1.cmgtvogta7kh.us-east-1.rds.amazonaws.com --port 3306 --protocol=TCP
Enter password: {tele$20483}X
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 57
Server version: 8.0.28 Source distribution

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

First, let's see what is in the database. There should be the database exampleDB created when the instance was configured. It is there, in fact.

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| exampleDB          |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+

5 rows in set (0.01 sec)
```

Currently, this exampleDB database is empty of any table or any data.

The following MySQL script creates a table named table1 within the exampleDB.

```
mysql> CREATE TABLE exampleDB.table1 (
    -> item_id INT AUTO_INCREMENT,
    -> content VARCHAR(255),
    -> PRIMARY KEY(item_id)
    -> );

Query OK, 0 rows affected (0.03 sec)
```

The following command proves that the table is empty.

```
mysql> SELECT * FROM exampleDB.table1;

Empty set (0.00 sec)
```

The data is added using the INSERT command. Three records are added just for testing purposes.

```
mysql> INSERT INTO exampleDB.table1 (content) VALUES ("This is record 1");
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO exampleDB.table1 (content) VALUES ("This is record 2");
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO exampleDB.table1 (content) VALUES ("This is record 3");
Query OK, 1 row affected (0.00 sec)
```

Finally, the content of table1 in the exampleDB database can be observed with this command.

```
mysql> SELECT * FROM exampleDB.table1;
+---------+------------------+
| item_id | content          |
+---------+------------------+
|       1 | This is record 1 |
|       2 | This is record 2 |
|       3 | This is record 3 |
+---------+------------------+
3 rows in set (0.00 sec)
```

This simple database is ready to be coupled with the webserver to complete the LAMP stack.

Learning Activity

Configuration of the webserver to talk to the database

Back to the EC2 instance, the HTML default configuration will be replaced with another that can the MySQL commands to the database endpoint.

```
ubuntu@ip-172-31-4-78:~$ cd /var/www/html
ubuntu@ip-172-31-4-78:/var/www/html$ ls
index.html
```

The index.html file is removed because a new one named index.php will take its place.

```
ubuntu@ip-172-31-4-78:/var/www/html$ sudo rm index.html
```

A PHP file called header.php is created first with a text editor.

```
ubuntu@ip-172-31-4-78:/var/www/html$ sudo nano header.php
```

Hypertext Preprocessor (PHP) [3] is an open source, scripting language for web development that can be embedded into an HTML file. The code is written, interpreted and run in the server side. The PHP code is enclosed in special tags like <?php and ?> These tags allow the webpage to jump in and out of PHP mode. The end result is HTML code that is delivered to the client via HTTP.

This is the content of the header.php (**note**: code was taken from reference 2). The access parameters are supplied in the PHP script including the database's password (which is a terrible practice, do not do that in real life). The access endpoint name and access parameters are supplied in the script PDO. This is a PHP Data Object that defines the interface to access the database. This is followed by the instruction to query the database to SELECT content showing the data rows. The MySQL reply should carry the three records of the table1.

```
ubuntu@ip-172-31-4-78:/var/www/html$ cat header.php
<?php
$user = "admin";
$password = "{tele$20483}X";
$database = "exampleDB";
$table = "table1";
try {
  $db = new PDO("mysql:host=database-1.cmgtvogta7kh.us-east-1.rds.amazonaws.com;dbname=$database", $user, $password);
  echo "<h2>From the database</h2><ol>";
  foreach($db->query("SELECT content FROM $table") as $row) {
    echo "<li>" . $row['content'] . "</li>";
  }
  echo "</ol>";
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}
```

The website homepage must be configured to include the header.php script. When a client access the webserver via HTTP, the index.php page calls the header.php script to be executed.

Index.php

```
<!DOCTYPE html>
<html>
   <?php
       include('header.php'); ?>
   <body>
        <h1>
            <?=date('h:i:s')?>
        </h1>
   </body>
</html>
```

header.php

```
<?php
$user = "admin";
$password = "{tele$20483}X";
$database = "exampleDB";
$table = "table1";

try {
  $db = new PDO("mysql:host=database-1.cmgtvogta7kh.us-east-1.rds.amazonaws.com;dbname=$database", $user, $password);
  echo "<h2>From the database</h2><ol>";
  foreach($db->query("SELECT content FROM $table") as $row) {
    echo "<li>" . $row['content'] . "</li>";
  }
  echo "</ol>";
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}
```

Once, this is completed, the Apache2 webserver must be restarted.

```
ubuntu@ip-172-31-4-78:/var/www/html$ sudo systemctl restart apache2
```

However, a curl to the local test address (or loopback address) shows that the script did not run. HTML returned the index.php text content, but the header.php file was not executed.

```
ubuntu@ip-172-31-4-78:/var/www/html$ curl 127.0.0.1
<!DOCTYPE html>
<html>
    <?php
        include('header.php'); ?>
    <body>
         <h1>
             <?=date('h:i:s')?>
         </h1>
    </body>
</html>
```

The reason is that the server does not have PHP installed. This is fixed with these commands.

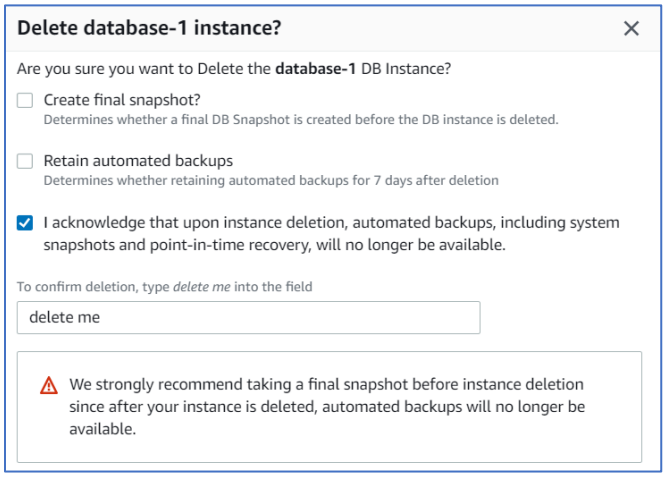
```
ubuntu@ip-172-31-4-78:/var/www/html$ sudo apt install php libapache2-mod-php -y
ubuntu@ip-172-31-4-78:/var/www/html$ sudo apt install php-mysql -y
```

A new test after the installation of PHP proves that the LAMP service is fully running now.

```
ubuntu@ip-172-31-4-78:~$ sudo systemctl restart apache2
ubuntu@ip-172-31-4-78:~$ curl 127.0.0.1
<!DOCTYPE html>
<html>
  <h2>From the database</h2><ol><li>This is record 1</li><li>This is record 2</li><li>This is record 3</li></ol>   <body>
    <h1>
      07:11:17      </h1>
  </body>
</html>
```

The final and conclusive test of the LAMP stack is via the web browser. It shows the three records that were retrieved from the database delivered via HTTP.

Note: once that this activity is completed, proceed to delete all the resources, especially the database. Otherwise, it will consume the credits fast.

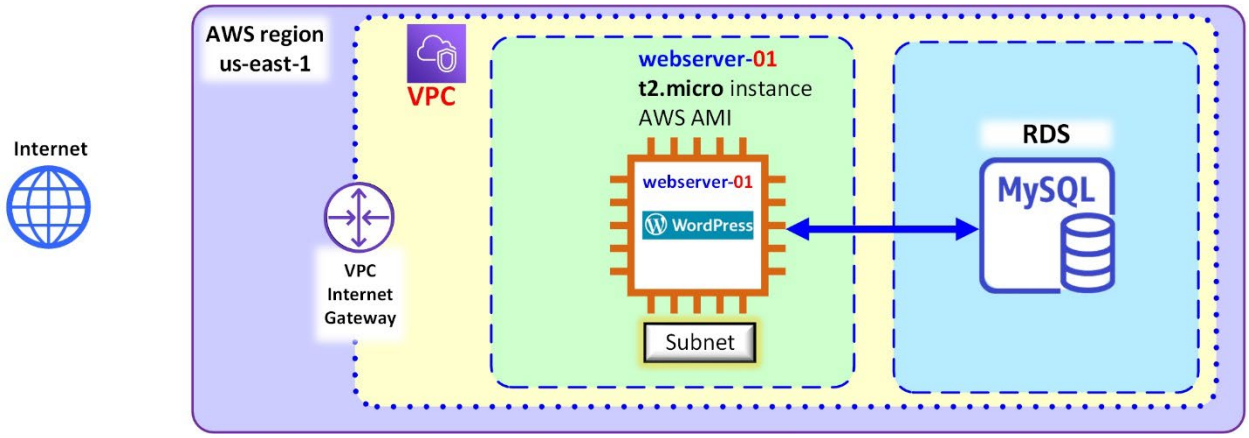


Conclusion

AWS Relational Database Service makes possible to deploy a MySQL database quick and fairly easy. The RDS service provides an endpoint that enables the communication with the webserver fronting an application stack. Being able to deploy a database with this service, instead of installing the database on an EC2 instance, simplifies the deployment of LAMP stacks.

Chapter 10 Coursework

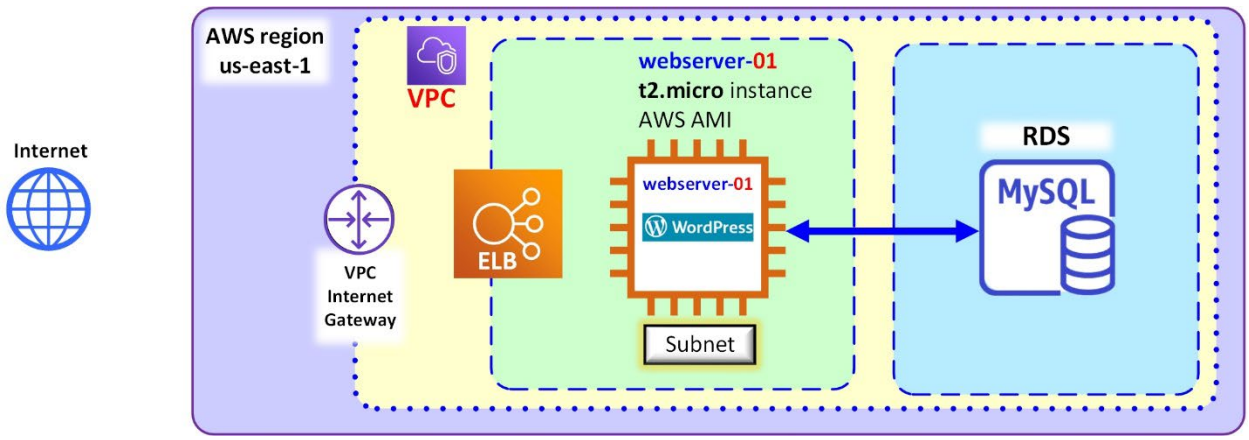
Build the following LAMP topology. The webserver will be running WordPress and the database MySQL. The EC2 instance image is the AWS AMI with HTTPD installed.



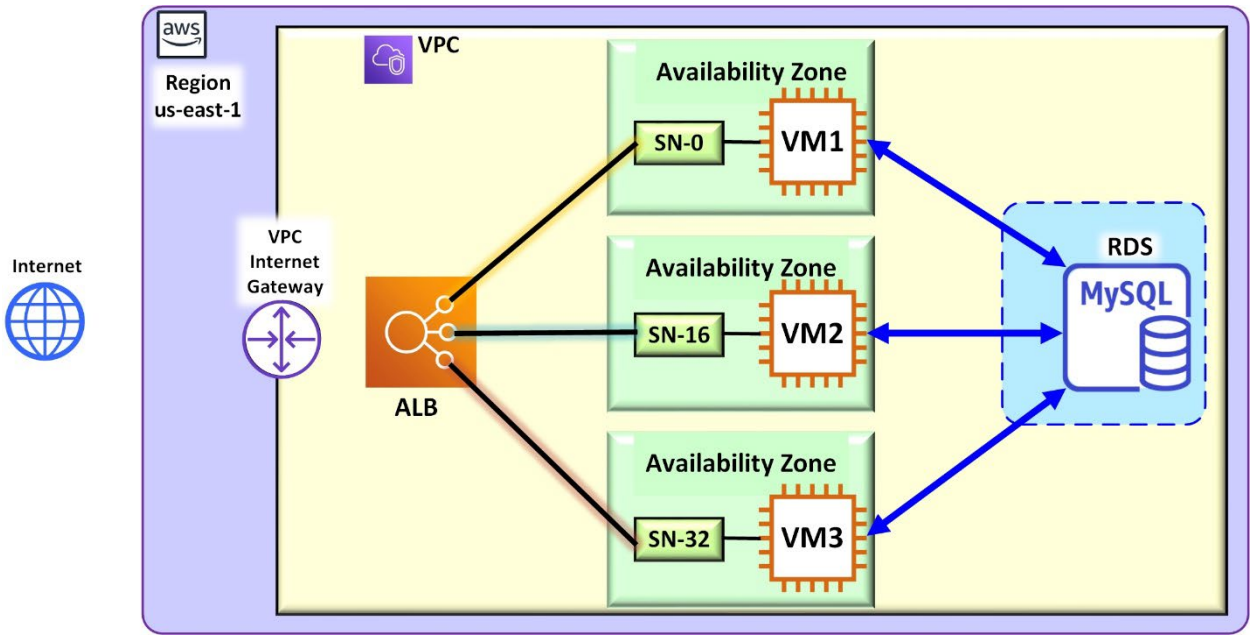
This is the installation script for WordPress, copy that in the User Data at the moment of the EC2 creation.

```
#!/bin/bash
yum install httpd php php-mysql -y
cd /var/www/html
wget https://wordpress.org/wordpress-5.1.1.tar.gz
tar -xzf wordpress-5.1.1.tar.gz
cp -r wordpress/* /var/www/html/
rm -rf wordpress (it is optional to remove the WP package)
rm -rf wordpress-5.1.1.tar.gz (it is optional to remove the tar ball)
chmod -R 755 wp-content
chown -R apache:apache wp-content
systemctl start httpd
systemctl enable httpd
```

Add an elastic load balancer once that the stack is functioning.



Create the following topology. First configure a webserver that can access the database. Then create an image with the functioning webserver. Proceed to create an autoscaling group with the image.



References

[1] Amazon Relational Database Service. User Guide 2022. https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html

[2] Erika Heidi. How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 20.04. 2021. https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-20-04

[3] PHP official manual. 2022. https://www.php.net/manual/en/intro-whatis.php

Chapter 11

Cloud Platform as a Service

Description

This section introduces AWS Elastic Beanstalk which is a Platform as a Service. The customer supplies an application in a compressed bundle to the service interface which takes the necessary actions to deploy the infrastructure required to support the application. After that, the cloud provider automatically maintains the computer resources without the intervention of the customer. Elastic Beanstalk PaaS is intended for the software developer who just want to concentrate in the application coding without being too involved in the infrastructure.

Learning Outcomes

- Observe the main features of PaaS.
- Deploy a single server web project on Elastic Beanstalk.

Main concepts

- The application.
- The environment.
- The platform.
- Security permissions.
- Deployment templates.

Learning Activities

- Deployment of a single webserver application on Elastic Beanstalk.

Introduction to Cloud Platform as a Service (PaaS)

PaaS is the service model where the cloud provider takes care of everything that pertains to the infrastructure that supports an application. The user does not need to be concerned with setting up the computer environment that will supports the application. PaaS offers an interface to the customer to supply the package with the application and the cloud provider instantiates everything that is needed to support it. PaaS is intended for software developers who just want their code running without getting involved with the infrastructure.

AWS Elastic Beanstalk

AWS Elastic Beanstalk is a platform as a service that manages all the resources needed to run **web applications** and also **long running workloads**. Elastic Beanstalk abstracts the infrastructure and the middleware required to run it in an **Environment**. The developer just provides the code to an environment that matches the requirements. Elastic Beanstalk takes it from there and deploys everything needed to support the application. Behind the scene, the service makes API calls to other services, most significantly AWS CloudFormation, to build the environment. A CloudFormation template or **stack** runs and deploys the virtual infrastructure.

Elastic Beanstalk provides several platforms to run web-based applications. The platforms can be programming languages (Go, Java, Python) or application servers (.NET Windows Server, Tomcat). The application developer writes the application in a code editor and then save it in a package or zipped app bundle. Elastic Beanstalk deploys the code to the selected platform version to create an application environment. Elastic Beanstalk takes care of provisioning all the resources needed to run your application. This include virtual machines (EC2 instances), load balancers, firewall, access control, security, etc. The software stack runs on the Amazon EC2 instances. The instances have public IP addresses or a DNS name or both so they can be reached from the Internet.

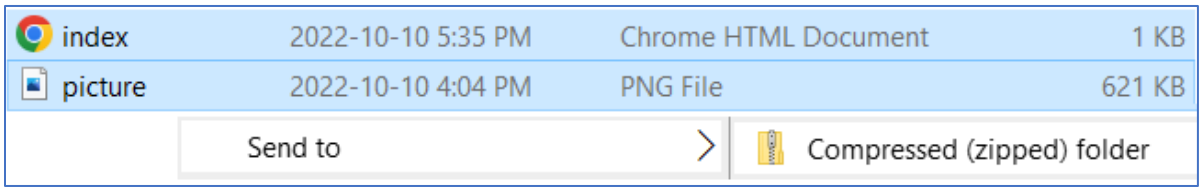
Learning Activity

Deployment of a single webserver application on Elastic Beanstalk

This demonstration will use a simple web project consisting of the HTML index file and a picture acting as the home page. These are the files:

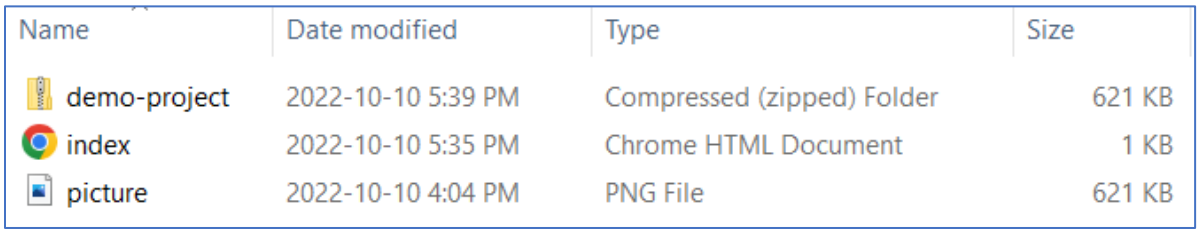
```
<html>
    <head>
        <title> TELE 20483 </title>
    </head>
    <body>
        <h1> PaaS Platform as a Service</h1>
        <p>  AWS beanstalk </p>
    </body>
    <img src="picture.png" width=400/>
</html>
```

The two files are compressed into a zip folder.



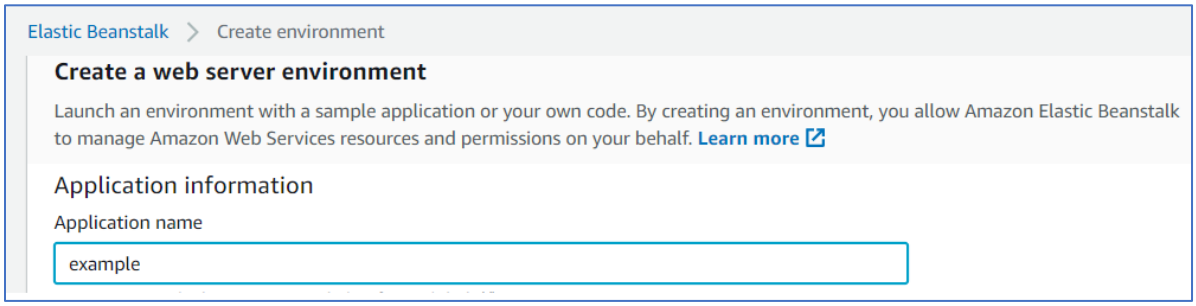
Note: do not make a folder and then compress it. That would not work because when the webserver is created, that folder would be placed inside the directory /var/www/html. The webserver will not be able to find the index file without additional configuration. Just compress the files together in a bundle.

The resulting demo-project zip folder will be uploaded later into Elastic Beanstalk.

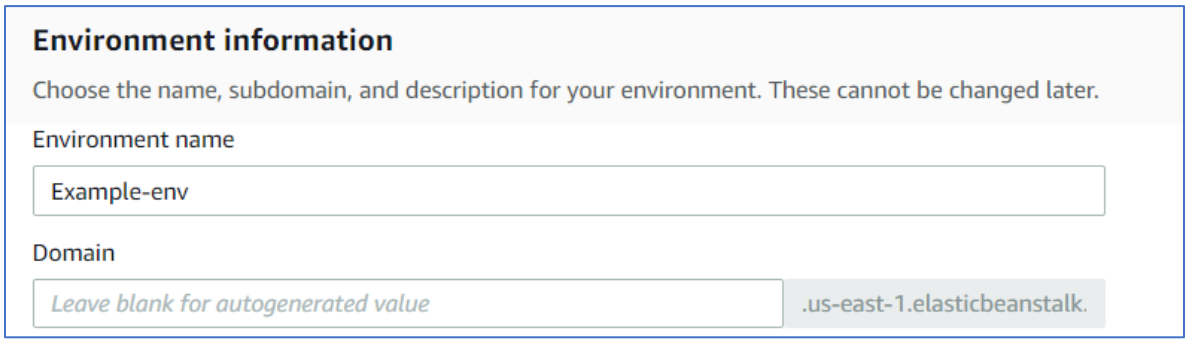


Environment

The **environment** comprehends all the resources needed to run the application. The environment is versioned so it can be updated after it is created.

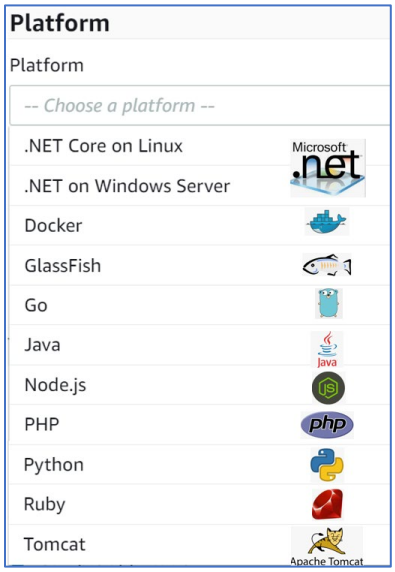


The customer can supply its own domain name in case of having it. Otherwise, AWS supplies a DNS record as part of its us-east-1.elasticbeanstalk naming domain.

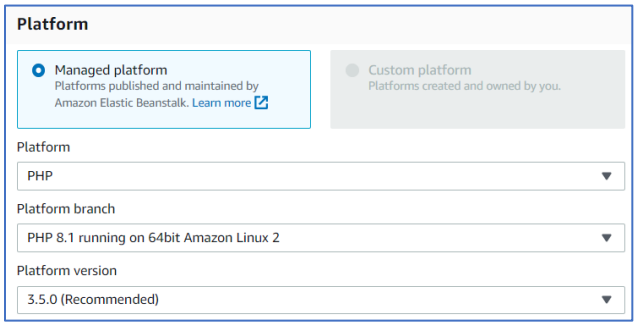


Platform

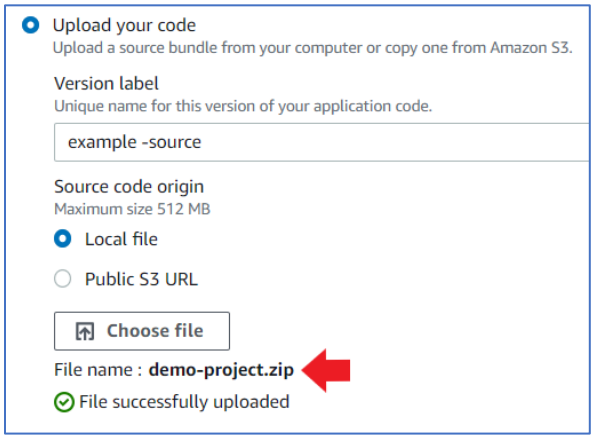
The platform includes the operating system, the runtime program, the server type, and any additional software package required to run the application. There are 11 platforms at the moment of writing this section. Fundamentally, the platform comprehend all the software complement required by the code.



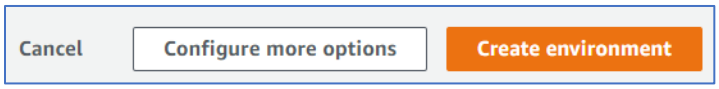
In this example, PHP was selected although, the demo web application is simple and it does not need it.



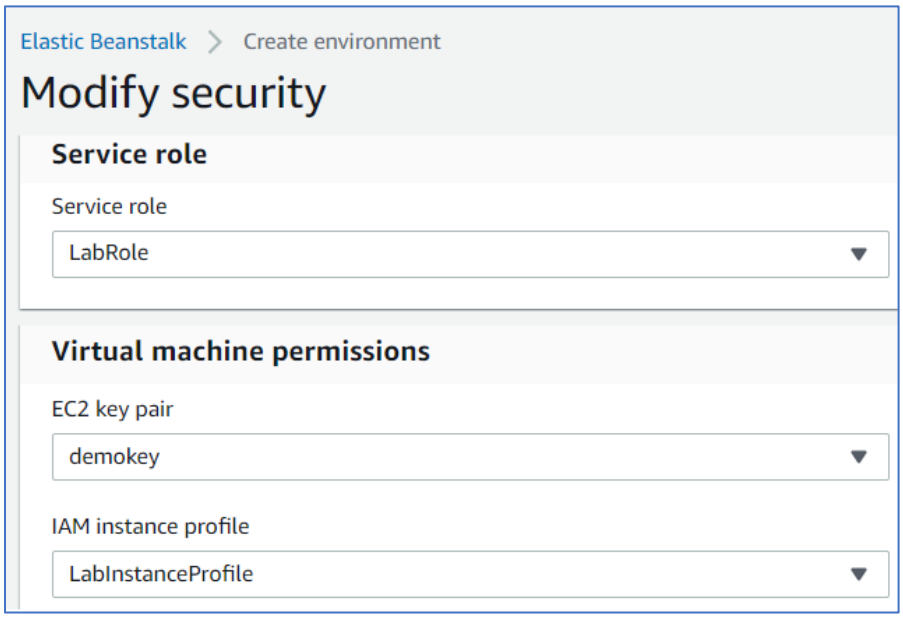
The source bundle with the demo-project.zip is uploaded to the platform. Elastic Beanstalk will install the files onto the /var/www/html of an EC2 instance running NGINX webserver.



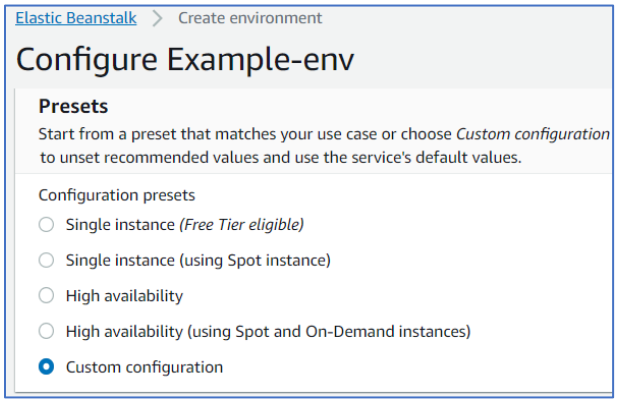
Proceed to **configure more options** before creating the environment.

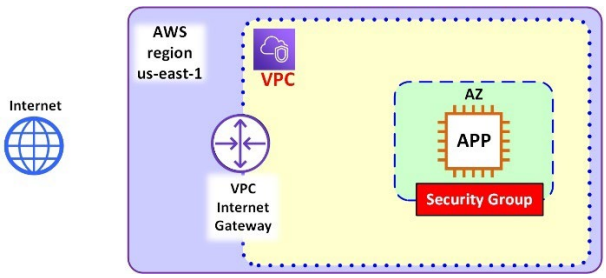


The **security settings** must be modified for the project to run successfully. AWS Academy provides the Service Role LabRole with enough permissions for accessing the resources required for creating the environment. The IAM instance profile LabInstanceProfile grants access to other AWS services.

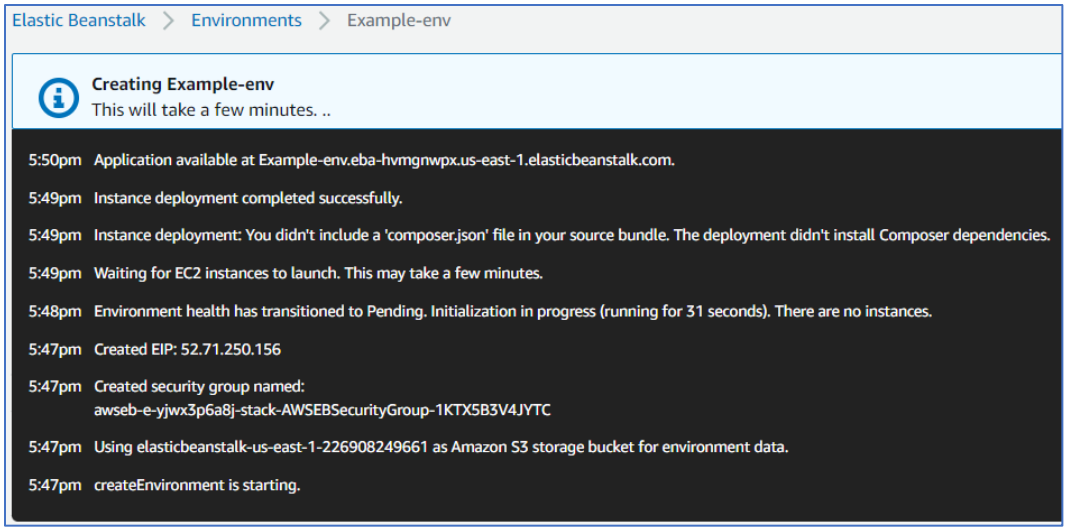


The presets changes automatically to custom configuration when the security values are changed.

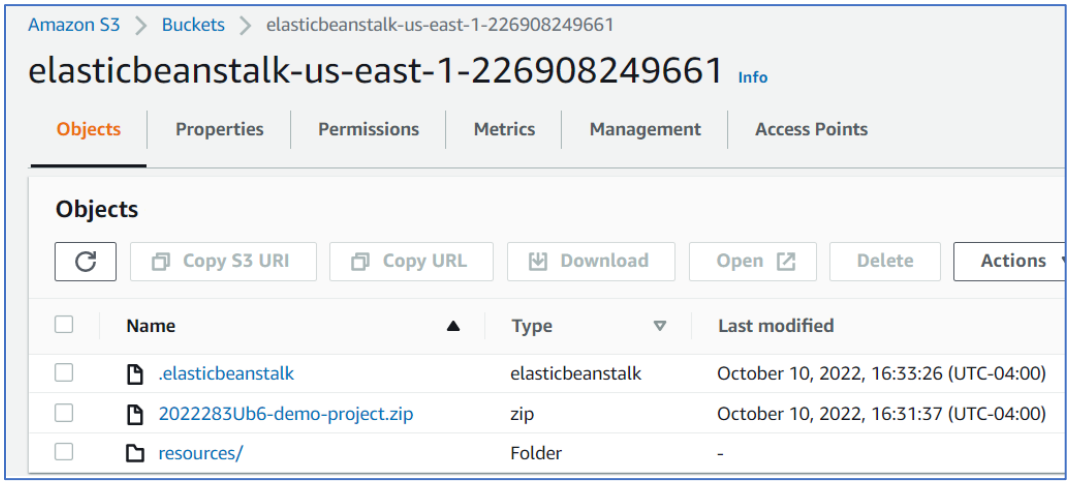




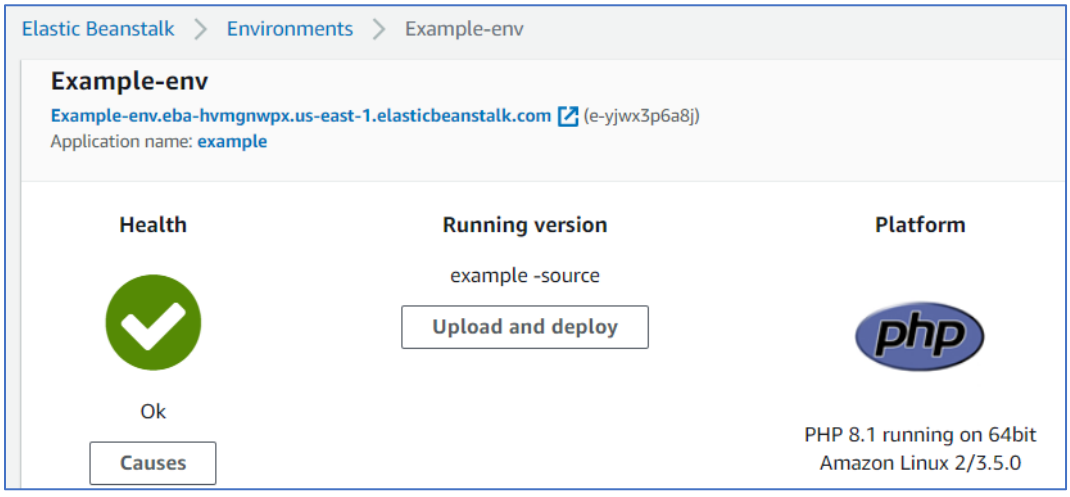
Now, just proceed to create the environment. Behind the scenes, Elastic Beanstalk starts making API calls to other services, most notable the CloudFormation service which uses templates (written in JSON or YAML) to orchestrate the resources required by this environment. In the snippet below, it can be observed that the process begins with a name being assigned to the application. Then the EC2 service is called upon to create an instance. Once that the EC2 instance is ready, an Elastic IP (EIP 52.71.250.156) is assigned to the instance for as long as it exists. A security group is automatically created to allow HTTP and SSH access to the instance. In short, Elastic Beanstalk executes everything that will support the application.



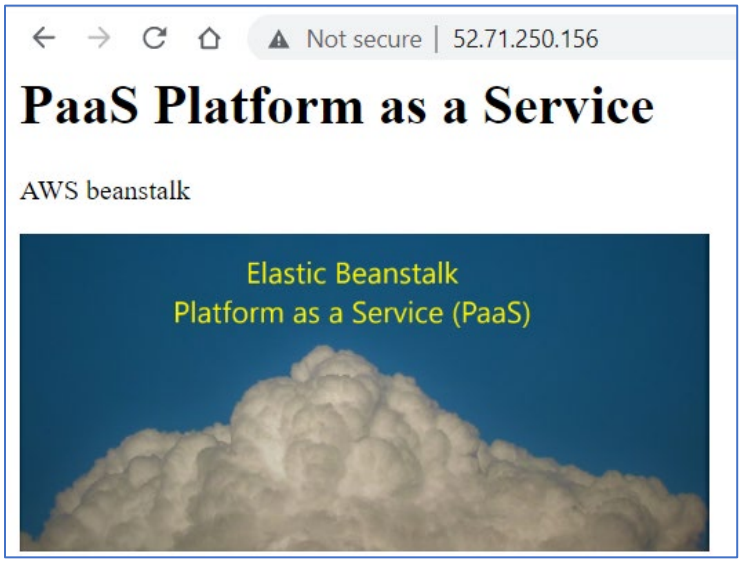
This process also creates an S3 bucket where the files related to the project are stored.



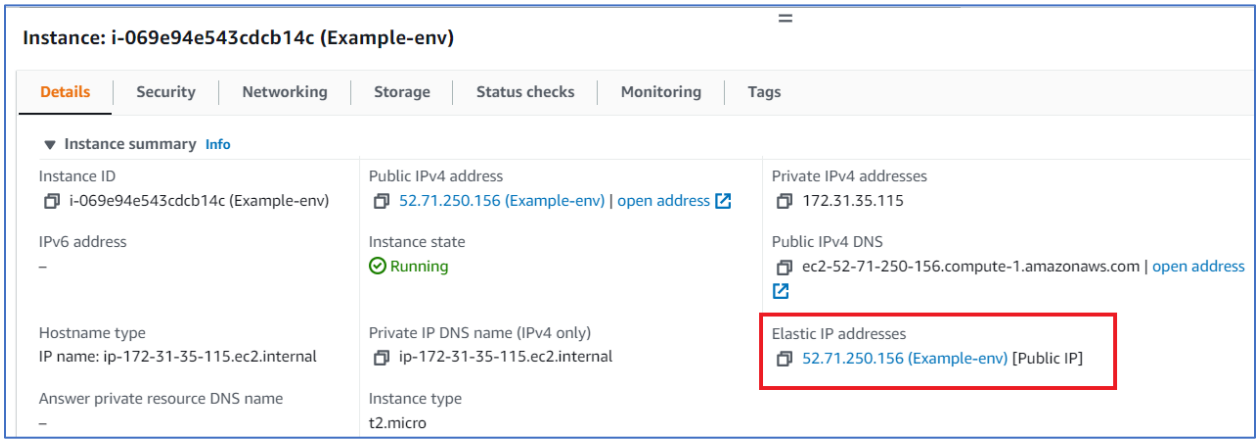
Finally, the setup is ready for testing.



Browsing to either the EIP address 52.71.250.156 or the DNS name assigned Example-env.eba-hvmgnwpx.us-east-1.elasticbeanstalk.com proves that it is working.



Elastic Beanstalk provisioned an EC2 instance named Example-env which can be seen in the EC2 service dashboard. This is the server running the application.

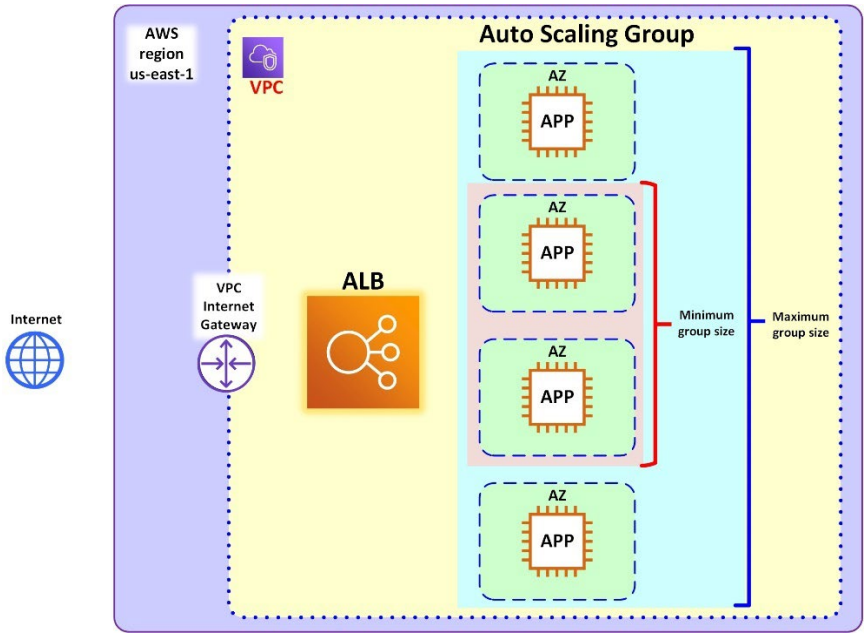


The EC2 instance is manually terminated to test how the management of the environment works. Immediately, a new EC2 instance with the application is spun up automatically. The reason is that Elastic Beanstalk maintains the **desired state** of the application which is to run one EC2 instance all the time. In the same order, if the EC2 instance becomes unresponsive or corrupted, Elastic Beanstalk is notified by the monitoring systems and the EC2 instance would be replaced by a new one. Therefore, the application is maintained automatically.

Learning Activity

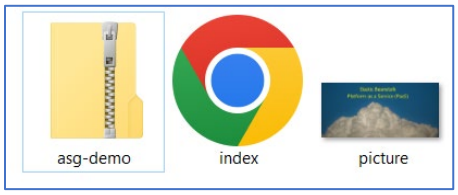
Deployment of a single webserver application on Elastic Beanstalk

In this activity, a highly available Auto Scaling Group of identical servers will be deployed to run behind an Application Elastic Load Balancer. The computer cluster consists of a minimum of two EC2 instance. If the demand for the service increases, a metric threshold will automatically trigger the deployment of more EC2 instances up to a maximum of four EC2 instances.

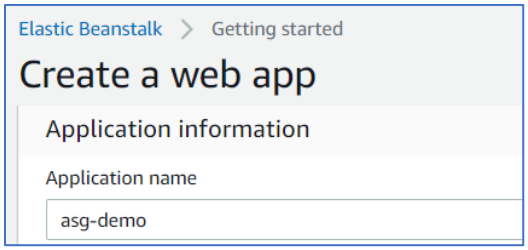


High availability auto scaling group.

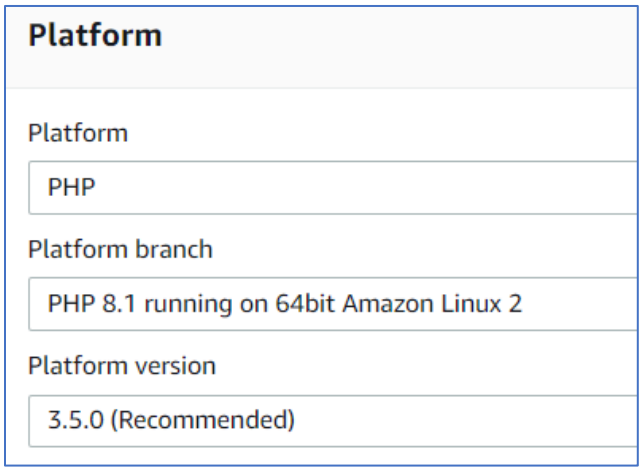
The application bundle is basically the same that was used in the previous learning activity.



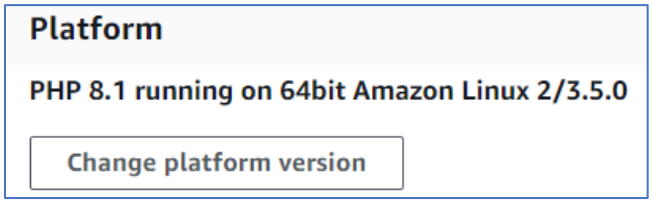
Proceed to create a web application named asg-demo.



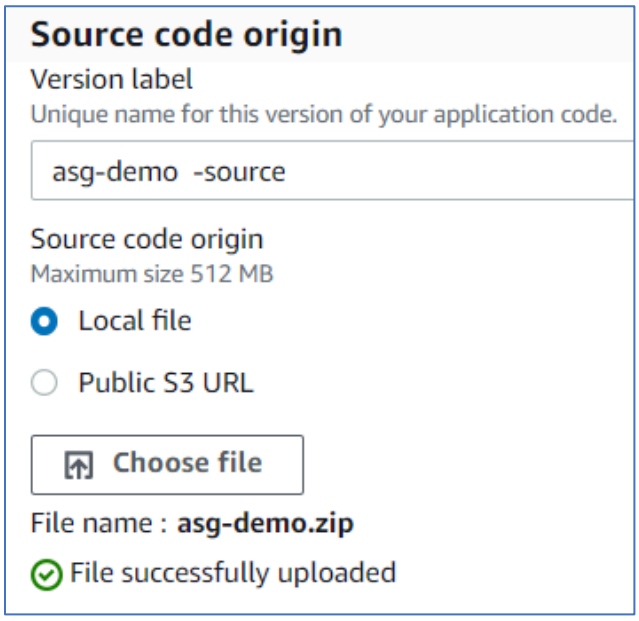
The platform chosen is PHP again.



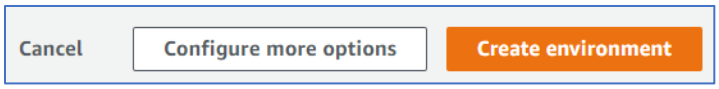
Which runs in Amazon Linux.



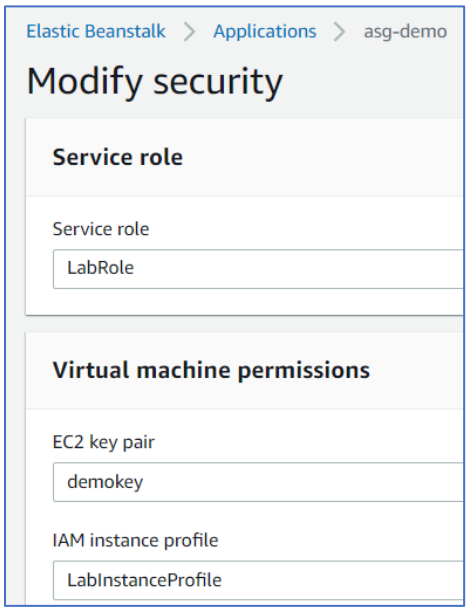
The zipped source code is uploaded next.



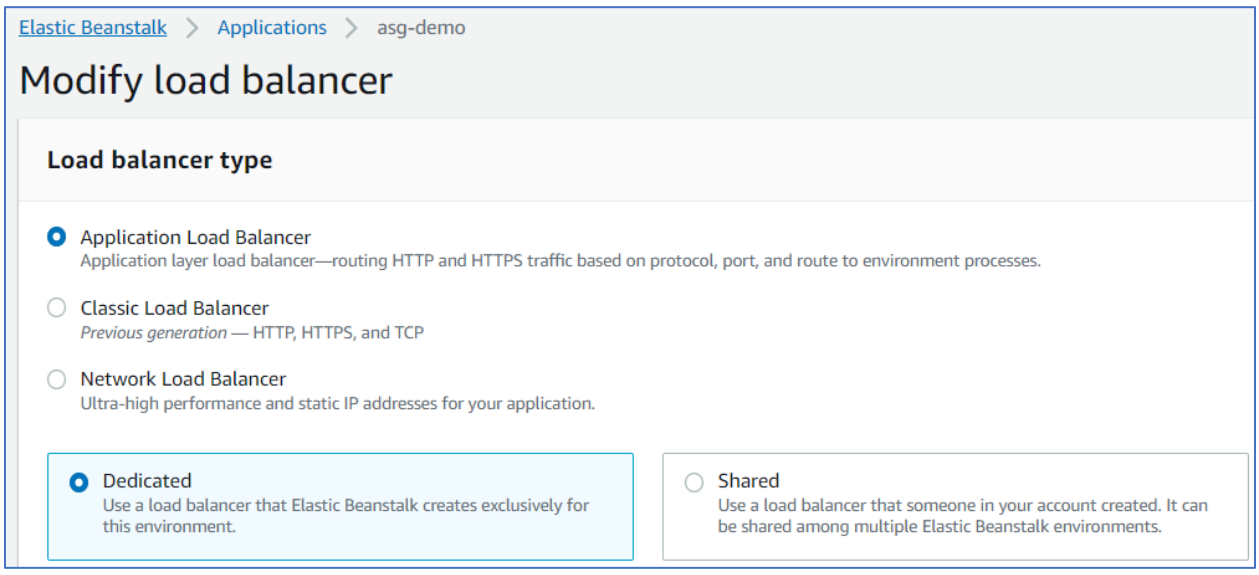
Several options must be configured before proceeding with the launching of the resources.



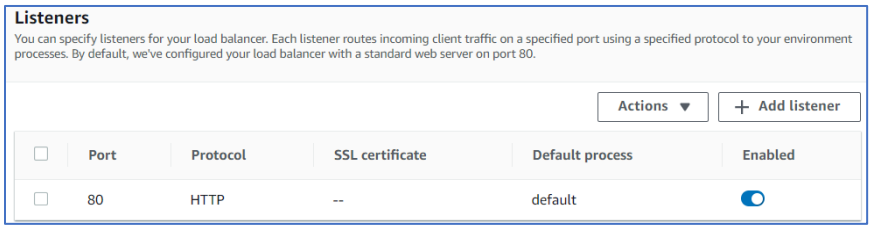
First, the security settings. The service assumes a role with enough rights to perform operations on other services such as calling on CloudFormation to deploy the resources. The instance profile provides an IAM role to the EC2 instance.

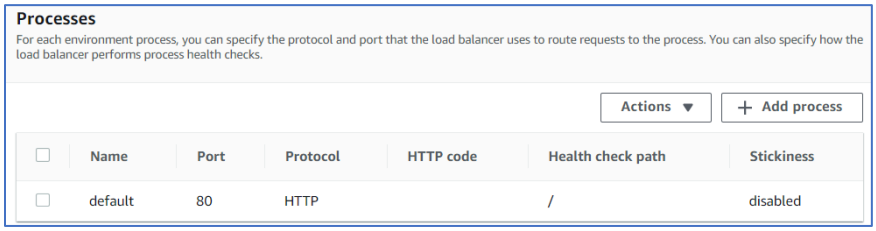


This architecture requires an elastic load balancer for the web application. So, select the application load balancer under modify load balancer.

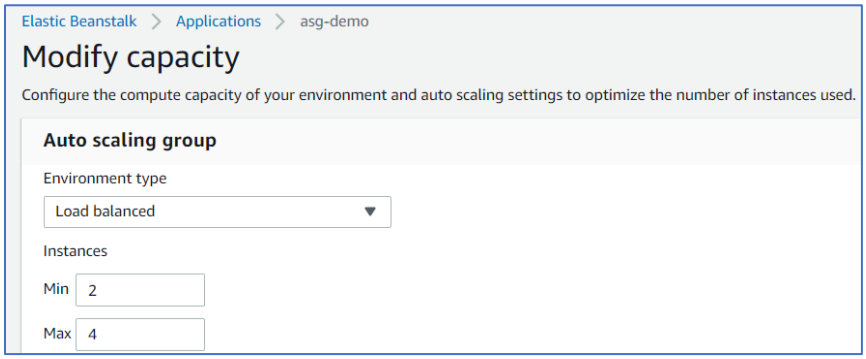


The listeners and processes do not need any modification in this case.

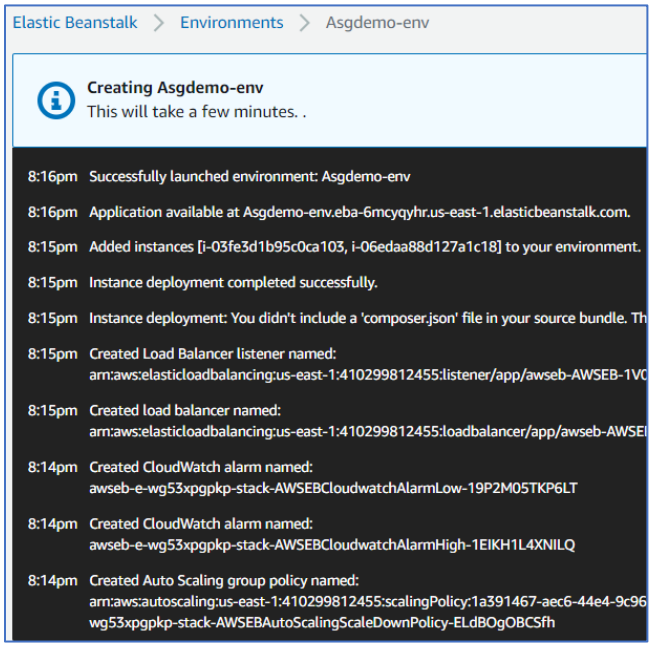




The auto scaling group minimum capacity is selected to be 2 EC2 instances and the maximum 4 instances.



After this point, the environment can be created.



Environment Asgdemo-env launched.

Application DNS name assigned.

Two EC2 instances added to environment.

Instances deployment completed.

Load balancer listener group created.

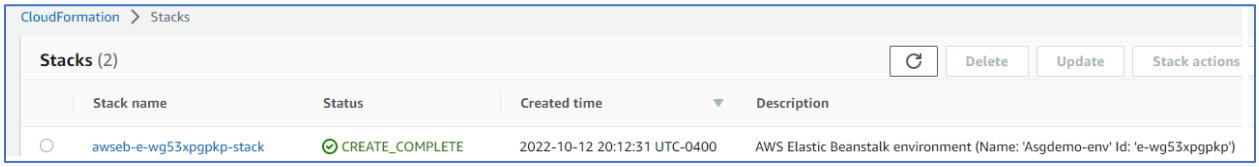
Load balancer created.

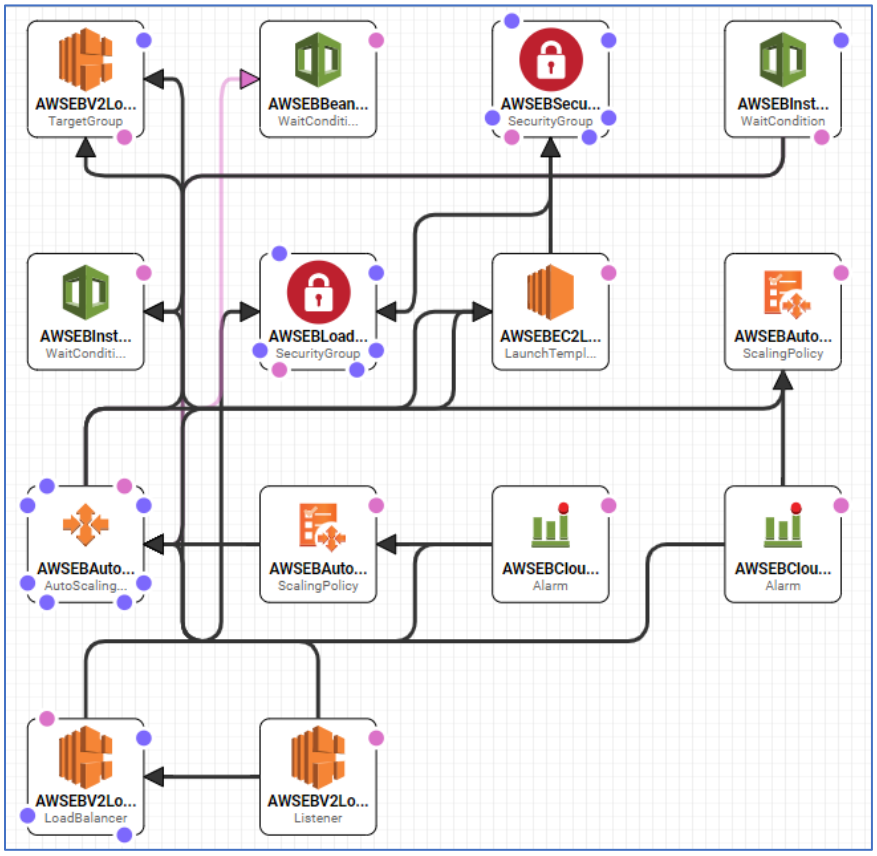
CloudWatch alarms created.

Auto Scaling Group Policy created.

Security group created.

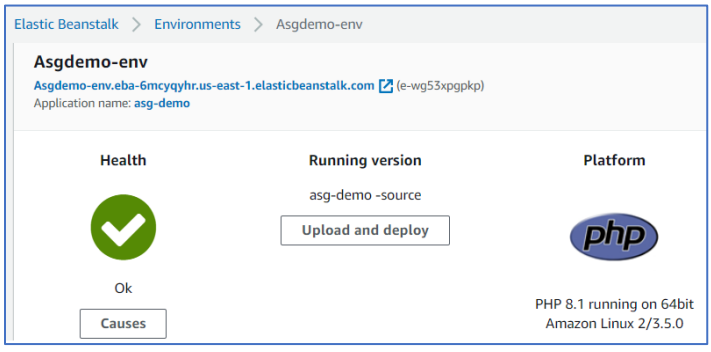
Behind the scenes, Elastic Beanstalk contacted via API calls another AWS service, **CloudFormation**. This service builds AWS cloud architectures using configuration scripts. The following diagram corresponds to the CloudFormation stack showing all the parts for this platform.



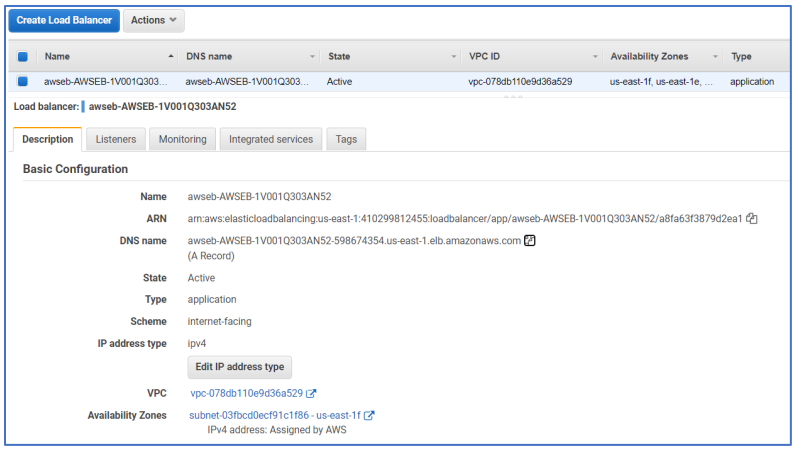


CloudFormation stack that represents all the components of this deployment.

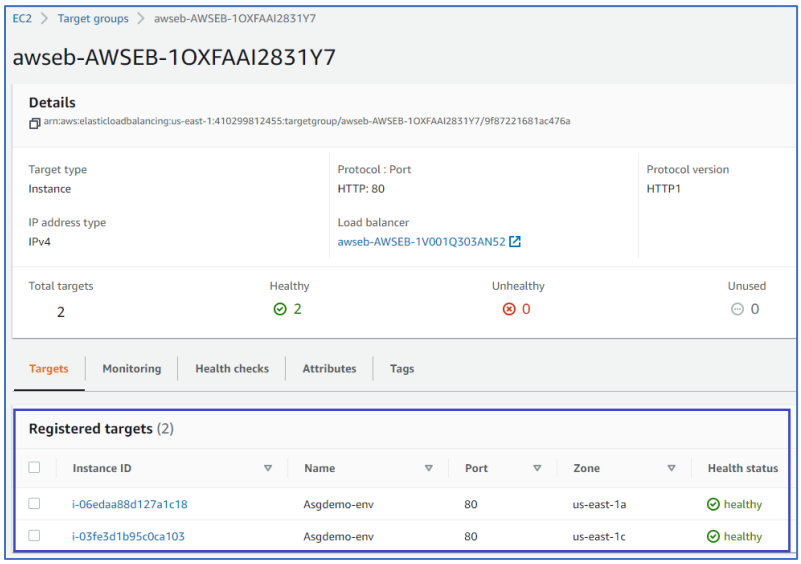
Finally, the environment is ready. The testing proves that it is running fine.



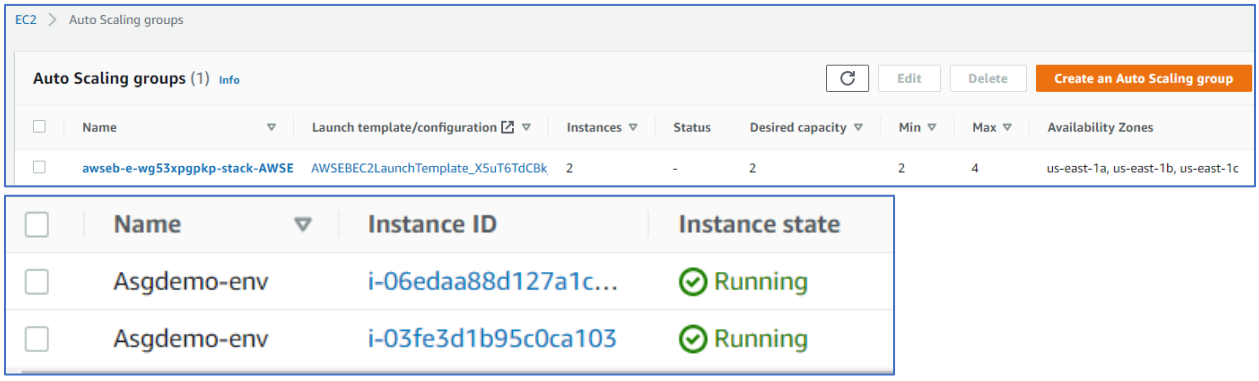
The application load balancer faces the requests for service from the Internet.



The deployment included the Target Group with the minimum required two EC2 instances.



The auto scaling group maintains the desired capacity of a minimum of two EC2 instances and maximum of four instances.

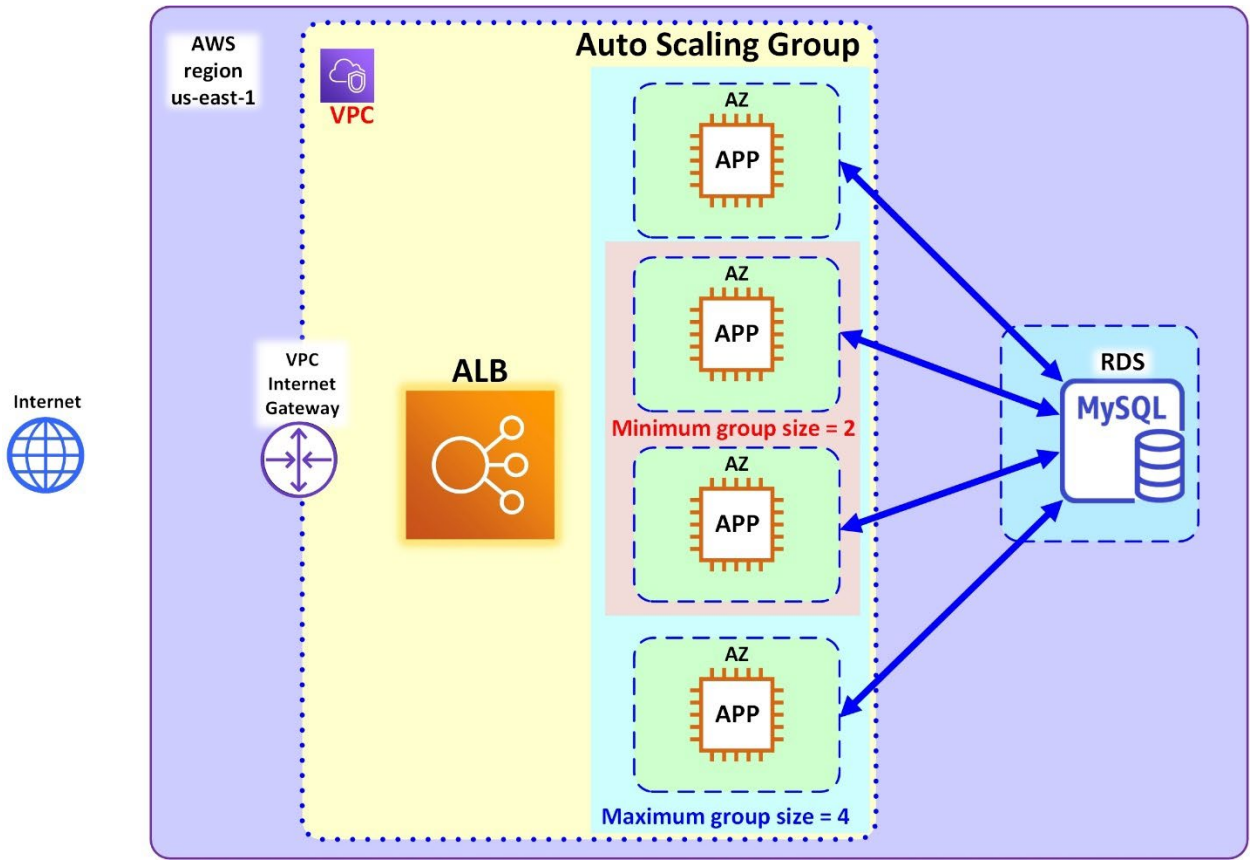


Conclusion

Elastic Beanstalk is one of the PaaS offering from AWS that makes life easier for software developers. By using this service, they can just concentrate on writing their application code and then testing it and deploying it in a quick manner without being concerned much about the underlaying infrastructure. Elastic Beanstalk provides two use case profiles, one for webserver applications and the other for long running workloads. In this section, the web case was implemented using a single web server and then by deploying a high availability auto scaling group fronted by an elastic load balancer.

Chapter 11 Coursework

Deploy the high availability LAMP stack using Elastic Beanstalk as shown in the diagram. The MySQL database must be instantiated using the AWS RDS service before creating the Elastic Beanstalk application. The code in the application must use some scripting, for example PHP, and it must make a call to the database endpoint. That is challenging part of this project. Elastic Beanstalk has a window for setting up variable names for the username, database name, and password to access the database. The PHP code should be dry using the same variables. See reference [2] for guidance.



Deployment of high availability LAMP stack with RDS MySQL database.

Reference

[1] AWS Elastic Beanstalk - Developer Guide. 2022.
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html

[2] Using Elastic Beanstalk with Amazon RDS. 2022.
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/AWSHowTo.RDS.html

Chapter 12

Introduction to Docker Containers

Description

The purpose of this section is to introduce Docker Containers. A container is a unit of code [1] that performs one function running as an isolated virtual computing resource. Modern applications [2] are frequently structured as a choreography of several containers, each performing a particular role. In this section, Docker containers are deployed from pre-existing images with the purpose of showing their functioning. Once the initial understanding is gained, containers images are created for simple projects. Finally, a managed cloud service is used to deploy and support containers in a cloud environment.

Learning Outcomes

- Install Docker on a virtual cloud compute service.
- Deploy several services using existing Docker container images.
- Create containers images from project files.
- Evaluate the composition of a container.
- Deploy containers in a cloud managed service.

Main concepts

- Definition of containers. Main characteristics of containers.
- Obtaining existing images from repositories.
- Running containers from images.
- Elastic Container Registry.
- Elastic Container Service.

Activities

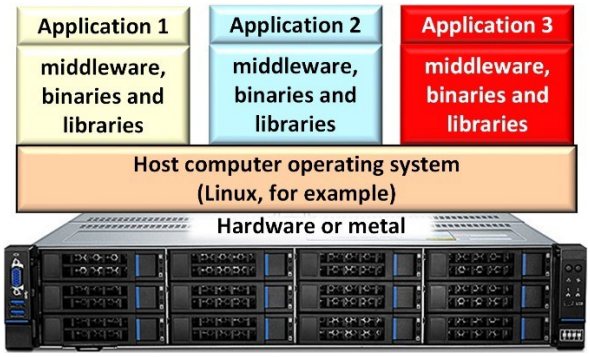
- Configuration of a Docker host on an EC2 instance in AWS.
- Fundamentals commands to manage Docker Containers.
- Exploring the internal infrastructure of containers.
- Introduction to Docker Container Networking.
- Running NGINX webservers on containers.
- Modification of the container.
- Using Volumes to keep persistent data
- Creation of a Docker Image with Dockerfile.
- Creating a basic Docker image for a webserver on Linux Alpine.
- Creating a Docker image for a NGINX webserver on Linux Alpine.
- Creating a Docker container image in AWS Cloud9.
- AWS Elastic Container Registry Service.
- AWS Elastic Container Service.

Introduction

This chapter begins with a comparison among applications deployed on multitask servers, virtual machines and containers to outline the differences and advantages of containers. Afterwards, the container theory is complemented with the deployment of a Docker host to run containers. Images are created once that the most important characteristic of containers are understood. Finally, a cloud platform is used to deploy architecture of either single or fleet of containers.

Running Applications without virtualization

A server running an operating system is multitask by definition. For example, a Linux or a Windows server can run multiple applications at the same time to provide different services. Each of these applications requires the installation of particular software packages such as middleware, binaries and libraries. Since these applications run directly on this platform they become dependent on the operating system.



Applications deployed directly over a host operating system

Running applications directly on the operating system [3] of a dedicated server has been a solution for decades and it is perfectly fine for many cases. However, because this type of deployment is tightly coupled to the server platform, it does not offer flexibility. Frequently, updates and patches to the operating system affect the applications. On the other hand, updates to the applications might cause incompatibilities with the O.S version. If a service needs to be migrated to another server that tends to be a cumbersome process prone to dependency incompatibilities. Furthermore, since the application processes compete for the resources of the server, one of them might starve off the others.

Deploying applications directly on a dedicated server is not the right model when it comes to separation of organizations or tenants. In the case of cloud computing, massive pools of servers are available as per subscription. The elasticity principle states the when the tenants request resources, they will be provided immediately, per subscription. In such scenario, virtualization is the answer to efficiently use the pool of computer power.

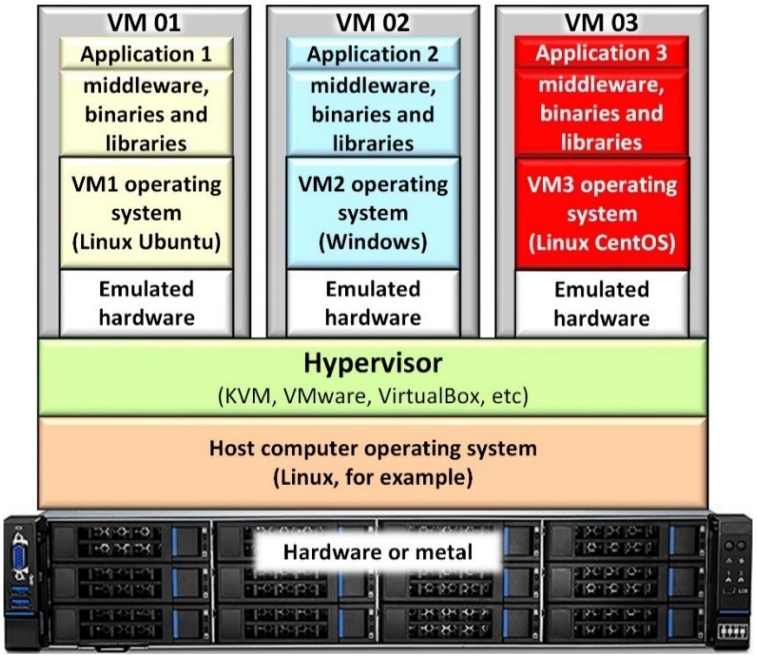
Virtualization

In this model, a **hypervisor** entity enables the creation and management of simulated machines on a physical machine. Each of these **virtual machines (VM)** runs its own operating system and application packages that must be installed and configured on each VM. The configuration of applications on a VM follow the same steps than on a physical server. That means that the operating system and packages must be installed on every virtual machine depending on the particular use case.

A **virtualization hypervisor** maintains the separation among the different VMs. A virtual machine appears as a whole computer including virtualized CPUs and simulated network interfaces. The hypervisor is capable of creating virtual networks either to interconnect the VMs or to provide a bridge to access an external network such as a local area network or even access to the Internet.

A very important property of virtual machines is that the hypervisor supports the packetization of VMs. So, in principle, they can be exported to other physical servers as identical **virtual appliances**. However, there are dependencies that need to be maintained. Virtual machines can also be **cloned** to make identical copies.

The following diagram illustrates the virtual machine concept. A hypervisor sits between the host operating system and the virtual machines. The hypervisor controls the access to the physical resources of the host computer. When a virtual machine makes a call for CPU access, the hypervisor intersects such call, and it talks to the CPU on behalf of the VM. Fundamentally, the hypervisor behaves as a proxy-translator between the VMs and the kernel of the operating system (note: this example shows a type 2 hypervisor).



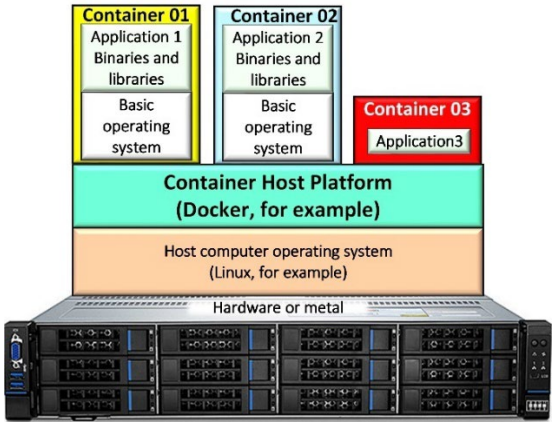
Three applications running on separated virtual machines

From the viewpoint of the VMs, they appear as isolated, real computers. The hypervisor supports multiple VM's operating systems. In the example, one VM runs on Linux Ubuntu, another on Windows Server and the last on Linux CentOS. Let's suppose, as an example, that VM 01 is running a webserver NGINX, the VM 02 is running Windows Server with Active Directory and VM 03 is running a database. Thus, there are three different virtual machines for three different uses running on the same host.

A virtual machine is a solution for efficiently using a physical host. The virtual machines are administered in the similar way that a dedicated server although with a greater deal of flexibility. A very important aspect of virtual machines is that they are isolated units. This is a very important characteristic from the viewpoint of security.

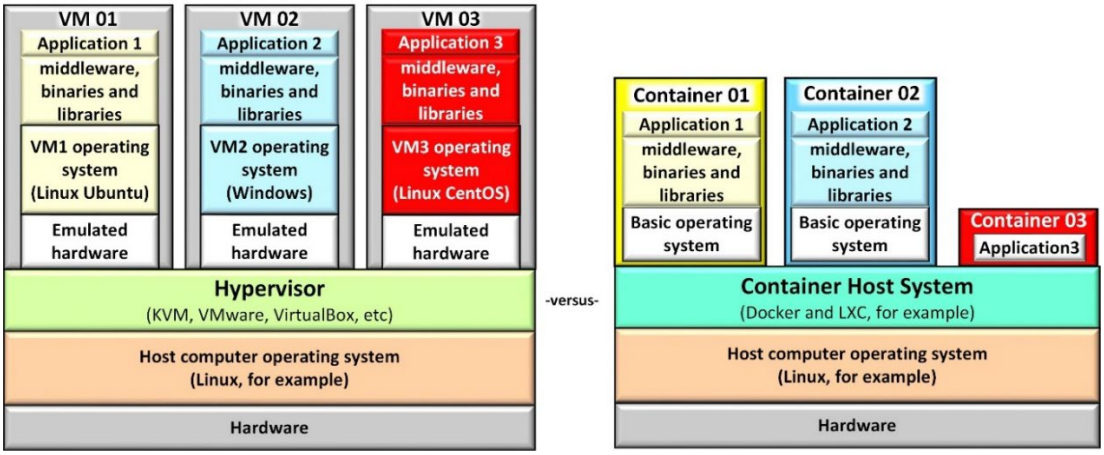
Containers

A container is also virtualization, but it is not a virtual machine. As the name indicate, a container [1] is an **application** with its required **dependencies contained** in a **software package**. A container is designed to host an application based on an existing **image**. For example, a container that runs a web server only does that. The image used to make the container only contains what is needed to run the web server. From this base image, an indefinite number of identical web servers can be made by creating containers.



Three applications running on three containers

Because the container is decoupled from the details of the supporting infrastructure, it can be moved to another host running the same container software. This is the major departure from a virtual machine. A virtual machine is simulated to act exactly like a physical machine. On the other hand, a container is meant to run one application regardless of the underlaying host's operating system. The only requirement is that the host computer must run a container platform [2] such as **Docker** and **Linux Container (LXC)**.



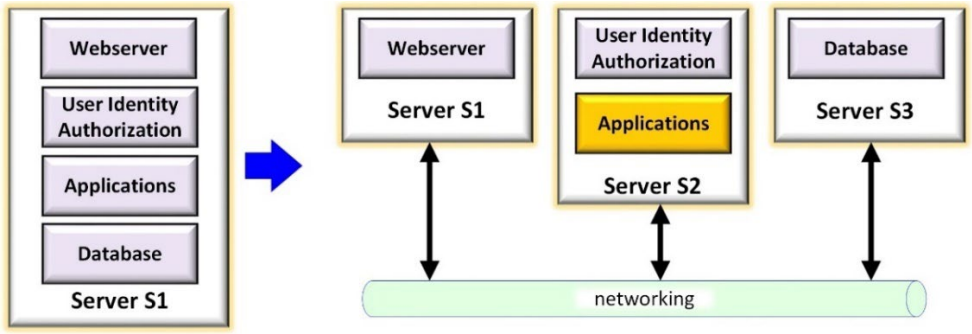
Comparison of Virtual Machines versus Container platforms

In the previous image, each container holds only what it strictly necessary to run one application. For instance, container 01 runs a webservice application. This requires the webserver software and a basic operating system. Container 02 might be running a specific processing job. It only does that and nothing else. Finally, container 03 does not even require an operating system because it just a simple application that runs to display a message. The main point is that containers are built to do one thing well.

Container philosophy

To further explain the container's philosophy, let's observe the design on the left side of the next figure. The service is comprised of four components which perform different functions. However, all these functional parts are integrated in one application running on one server. Having all these components in one **monolith** makes maintenance and development harder. For instance, troubleshooting a problem is very difficult because the application is designed as a block of code. Furthermore, any modification to any part of the service might affect other parts. Improving the service tends to be slow and cumbersome.

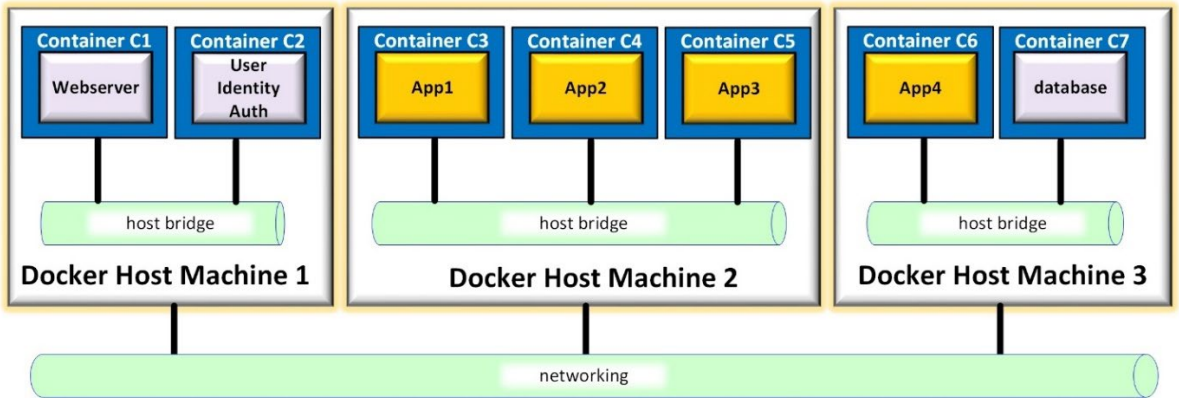
For the reasons above, a monolith design is not desirable. Hence, it is split into different servers, either physical or virtual, in the solution seen on the right side of the diagram.



Monolith design versus functional design

The solution on the right is an improvement over the one on the left. However, it can be even better. The application programs doing different tasks can be separated into different logical units. Since the units of code are separated, software developers can work on improving each unit in a continuous development and deployment **pipeline**. Separating the units also allows different teams to focus on unique tasks.

This is where containers fully shine. Each task unit can be placed in a different container to perform only a specific function. Thus, a monolithic application can be architected into a choreography of containers giving origin to the concept of **microservices**. In its most advanced form, the whole **fleet of containers** can be managed by an orchestrator such as **Kubernetes**, **Docker Swarm** and **Helm**.



A fleet of containers comprehend a service

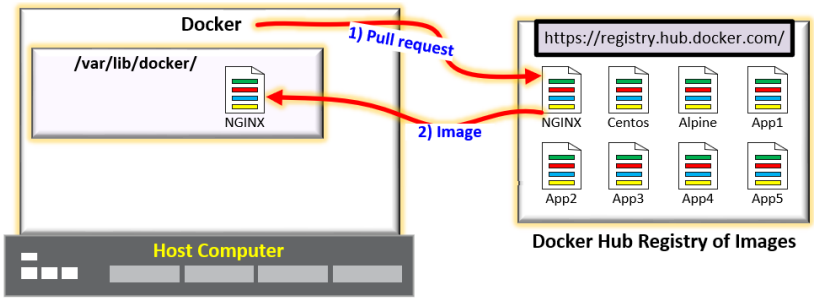
Docker Container Composition

A container is fundamentally a purpose specific service. It is code running to execute one task efficiently. For that reason, it is designed to be a bare bone entity with only the very basic packages needed to provide the service. A **container** is a package of software accompanied by a virtualized environment that allows such software to run. The container has strictly only what it needed to provide a specific function. Therefore, containers run efficiently because they are lean.

A container is not meant to be managed in the same way than a traditional server or a virtual machine VM. Containers are created from images. An **image** is the pre-packaged software that contains everything needed to create specific containers. Images are software defined entities built off a description file called the **Dockerfile**. This dockerfile lists the layering of all the functions and commands needed to build an image. Software developers write their own dockerfiles to **compose** images according to their needs.

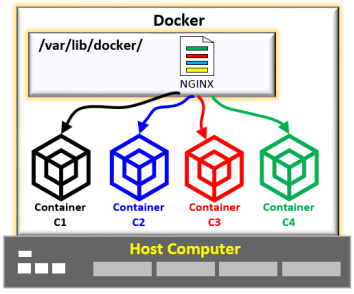
Once that a Dockerfile is written, the image is built following the text description. The resulting image is stored in a **repository** which is a database of images. Organizations typically maintain private repositories for their images. There are also public repositories that contain many general purpose images. These public images are available for downloading. The main public repository is the official docker hub (https://hub.docker.com/).

The following diagram gives an idea of the process of requesting a common image (NGINX webserver) from the docker official registry. An administrator issues a pull request to the repository from a Docker **host** computer. This fetches the image to be downloaded to the directory /var/lib/docker in the host.



Docker image being downloaded from the Docker registry.

The image remains stored in the host location. From such image, an indetermined number of identical containers can be created. In the next diagram, four containers of NGINX webserver are created from the same downloaded image.



Four containers created after an image

Learning Activity

Configuration of a Docker host on an EC2 instance in AWS

In this activity, an EC2 instance will be configured to act as a **Docker host**. Afterward, containers will be deployed on this host virtual machine to explore the fundamentals of containerization. The host instance image chosen is the latest AWS Linux 2 AMI and the type is t2.micro which is good enough for just exploring the basics of Docker containers. (Note: this is just for learning purposes since a t2.micro platform would not be adequate in a production environment).

Create an EC2 instance with the following User_Data to bootstrap the instance at launch.

```
#!/bin/bash
yum update
yum install docker -y
usermod -a -G docker ec2-user
systemctl restart docker
systemctl enable docker
```

The package manager obtains and install the docker application. Once that the installation is complete, the command usermod adds the ec2-user to the docker administrative group. This is needed to avoid typing sudo every time that a docker privileged-level command is issued.

The EC2 instance is accessed via SSH once that it finishes booting up. The following command is used to test if the installation of Docker was successful and also to verify the software version.

```
[ec2-user@ip-172-31-93-252 ~]$ docker --version
Docker version 20.10.17, build 100c701
```

Learning Activity

Fundamentals commands to manage Docker Containers

A newly configured Docker host does not have any image as the following command demonstrates:

```
[ec2-user@ip-172-31-93-252 ~]$ docker image ls
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
```

However, an image is required to create a container. The Docker application always searches first the host computer's local store for the image. If there is no matching image present, then the application makes a call to the Docker public repository located in https://www.docker.com.

Let's begin with the simple testing image Hello World. The command **docker run** *image-name* is used to create a container based on an existing image of hello-world. Since the image does not exist in the local Docker host, it is pulled from the public repository library as shown in the next snippet.

```
[ec2-user@ip-172-31-93-252 ~]$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:aa0cc8055b82dc2509bed2e19b275c8f463506616377219d9642221ab53cf9fe
Status: Downloaded newer image for hello-world:latest
```

The process continues with the running of the container's application. This hello-world image just runs once, it shows the message below and then it shuts down. It does nothing else.

```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash
```

The image "hello-world" is for testing purposes, but it has an educational component too. Its execution explains Docker's general process of creating a container. The messages show that:

- The Docker client called upon the Docker daemon.
- The daemon took control and pulled (downloaded) the **"hello-world" image** from Docker's main repository at https://hub.docker.com/ . (Someone had already built and stored the image there.)
- The daemon created a new container off the base image.
- The daemon started and ran the container.
- The daemon streamed (stdout) the output to the Docker client's screen.
- The daemon stopped the container.

That is all that a "hello-world" container does. Since the image was downloaded, it must be present in the local host store. Let's verify that:

```
[ec2-user@ip-172-31-93-252 ~]$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
Hello-world   latest    bf756fb1ae65   14 months ago   13.3 kB
```

The following command should show all the running docker containers, but nothing is showing:

```
[ec2-user@ip-172-31-93-252 ~]$ docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

Nothing shows because the hello-world container is designed to run once and then to stop. The same command followed by the **flag –a** shows all containers whether they are running or stopped.

```
[ec2-user@ip-172-31-93-252 ~]$ docker container ls -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS   PORTS   NAMES
d4b126852a8b   hello-world   "/hello"   7 minutes ago    Exited   (0)     amazing_borg
```

Since this hello-world container is just for demonstration, it can be removed with this command:

```
docker container rm <the container id number >
docker container rm d4b126852a8b
```

The remove command can only be used after a container has stopped. This is the command to stop a running container:

```
docker container stop <the container id number >
```

This command removes an image that is not required anymore:

```
docker image remove <the image id number >
docker image rm bf756fb1ae65
Untagged: hello-world:latest
Deleted:  sha256:bf756fb1ae65adf866bd8c456593cd24beb6a0a061dedf42b26a993176745f6b
```

The following commands are used for cleaning up unused and unwanted containers and images:

- Remove all stopped containers.

```
docker container prune
WARNING! This will remove all stopped containers.
Are you sure that you want to continue? [y/N] y
Deleted Containers:
a6e11c9a8715b78a12c4b37d2f810992369b31dcd24157e066c06c63a246b0ea95
Total reclaimed space: 1.114kB
```

- Remove dangling docker images.

```
docker image prune

WARNING! This will remove all dangling images.
Are you sure that you want to continue? [y/N] y
Total reclaimed space: 0B
```

Containers are created with the **docker run** command. This is the general format to create a container off an existing image. The docker container form is the older format. Both commands work.

```
docker container run --name container-given-name image-name
```

```
docker run --name container-given-name image-name
```

- For example, to create a webserver NGINX container:

```
docker run --name example01 nginx:latest
```

A docker image for a NGINX webserver is pulled from the Docker repository and a container is created.

```
docker run --name example01 nginx:latest
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
8740c948ffd4: Pull complete
d2c0556a17c5: Pull complete
c8b9881f2c6a: Pull complete
693c3ffa8f43: Pull complete
8316c5e80e6d: Pull complete
b2fe3577faa4: Pull complete
Digest: sha256:b8f2383a95879e1ae064940d9a200f67a6c79e710ed82ac42263397367e7cc4e
Status: Downloaded newer image for nginx:latest
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform
configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of
/etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in
/etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
 Notice that the cursor remains hanging here.
```

In this stage, the container example01 is running, but the cursor control is not returned to the user. That is inconvenient, if control C is issued to regain control of the prompt cursor, this happens:

```
docker container ls

CONTAINER ID   IMAGE   COMMAND   CREATED   STATUS   PORTS   NAMES
```

The container does not show anywhere. Let's try listing all container which are either in running state or stopped by using the –a flag.

```
docker ps -a

CONTAINER ID  IMAGE         COMMAND                  CREATED         STATUS       PORTS     NAMES
36c882ff774a  nginx:latest  "/docker-entrypoint.…"  10 minutes ago  Exited (0)             example01
```

It is there but it is in exited or stopped status. The container can be restarted with the following command.

```
docker start example01

example01
```

So, what happened with container example01? The container was running in the frontend of the host and when the control C command was issued to take back the control of the screen, the container stopped. To avoid this inconvenience the **flag – d** is required to keep the container running in the background.

Let's try creating another container, but this time with the **– d flag**.

```
docker run --name example02 -d nginx:latest

6d6eb2e75bff3fb497578f5f404f8888d7be50bb4cd413d873655f2ae2ff7680
```

This time, the console was not left hanging. The new container is running as shown in the list of containers:

```
docker ps

CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS    PORTS     NAMES
6d6eb2e75bff   nginx:latest   "/docker-entrypoint.…"   a minute ago     Up        80/tcp    example02
36c882ff774a   nginx:latest   "/docker-entrypoint.…"   11 minutes ago   Up        80/tcp    example01
```

- Now, stop the running containers:

```
docker container stop container-name
```

```
docker container stop example01
docker container stop example02
```

- Delete all stopped containers:

```
docker container prune

WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
36c882ff774a17608ad43aaa74d502a4d3153f111f96006616782b59c89b89f8
Total reclaimed space: 2.186kB
```

Thus, at this point the two previous containers example01 and example02 have been deleted.

Learning Activity

Exploring the internal infrastructure of containers

- Create a container using a Centos image using the flags – i, – t, and – d.

The **– i flag** stands for interactive option. It keeps the standard input (STDNIN) device open.

The **– t flag** tells Docker to assign a virtual terminal (tty) session with the container.

```
docker container run -itd --name example centos
```

Because there is a tty session open, it is possible to directly execute **shell** commands on the container. For example, execute the command list (ls):

```
# Either of the commands below works
docker exec -it example ls

bin  etc   lib    lost+found  mnt  proc  run   srv  tmp  var
dev  home  lib64  media       opt  root  sbin  sys  usr
```

The exec command executes the Linux command ls to list the main directories present in this Linux Centos container. Let's run the same action, but this time by dropping the user inside the container. The following command takes the user inside the container's Linux bourne again shell (bash).

```
docker exec -it example /bin/bash

[root@5ca9f0204ee6 /]# (note: this is the prompt inside the container)
[root@5ca9f0204ee6 /]# whoami
root
[root@5ca9f0204ee6 /]# pwd
/
```

Let's try to install a webserver package such as httpd or nginx.

```
[root@5ca9f0204ee6 /]# yum install httpd -y
Failed to set locale, defaulting to C.UTF-8
CentOS Linux 8 - AppStream                                  180  B/s |  38  B     00:00
Error: Failed to download metadata for repo 'appstream': Cannot prepare internal mirrorlist: No URLs
in mirrorlist
[root@5ca9f0204ee6 /]# yum install nginx -y
Failed to set locale, defaulting to C.UTF-8
CentOS Linux 8 - AppStream                                  556  B/s |  38  B     00:00
Error: Failed to download metadata for repo 'appstream': Cannot prepare internal mirrorlist: No URLs
in mirrorlist
[root@5ca9f0204ee6 /]#
```

That did not work. The error message says that there are no URL listed in the mirrorlist of repositories. That means that there is no configuration to find the Linux repositories. Let's try other commands:

```
[root@5ca9f0204ee6 /]# ifconfig
bash: ifconfig: command not found
[root@5ca9f0204ee6 /]# arp -a
bash: arp: command not found
```

This illustrates the philosophy of containers. The previous commands showed that not all tools are available in this centos container. A container is strip-bare to the basics, that is why many utilities are not present. Then the question arises; how can packages be installed on this container if the package manager is not even available? The answer is that this is the wrong way to deal with a container because they are not supposed to be created and then configured. Containers are to be created after an existing image, so the traditional way of adding packages to run on an operating system is not the right practice. The packages should be installed during the creation of the image. This procedure will be explained later when images will be created from scratch.

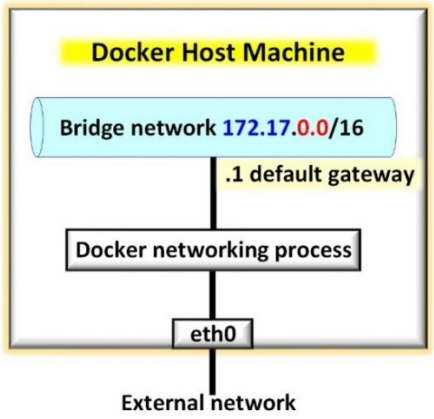
Learning Activity

Introduction to Docker Container Networking

This activity aims to provide the basic understanding of how Docker containers can be accessed across networks. Below is an edited look, for simplicity, of the configuration of the Docker host machine's interfaces. The Linux ifconfig command reveals three interfaces on the EC2 instance: a loopback, eth0, and **docker0**.

```
[ec2-user@ip-172-31-93-252 ~]$ ifconfig
lo:     inet 127.0.0.1  netmask 255.0.0.0, inet6 ::1  prefix length 128
eth0:   inet 172.31.93.252  netmask 255.255.240.0  broadcast 172.31.95.255
        inet6 fe80::10ff:69ff:fef5:5261  prefixlen 64
        Ethernet MAC address 12:ff:69:f5:52:61
docker0: inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        Ethernet MAC address 02:42:94:89:58:c5
```

When docker was installed in the host (the EC2 instance), a **virtual interface** named **docker0** was created. This virtual interface connects to a Docker internal process that handles a **virtual LAN bridge**.



Interpretation of the Docker networking arrangement

Beside the virtual interface for the bridge default gateway, there is another virtual interface for the container named virtual ethernet (vethe) plus a random string 22369c.

```
vethe22369c: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::8441:85ff:fe02:6f37  prefixlen 64
        Ethernet MAC address 86:41:85:02:6f:37
```

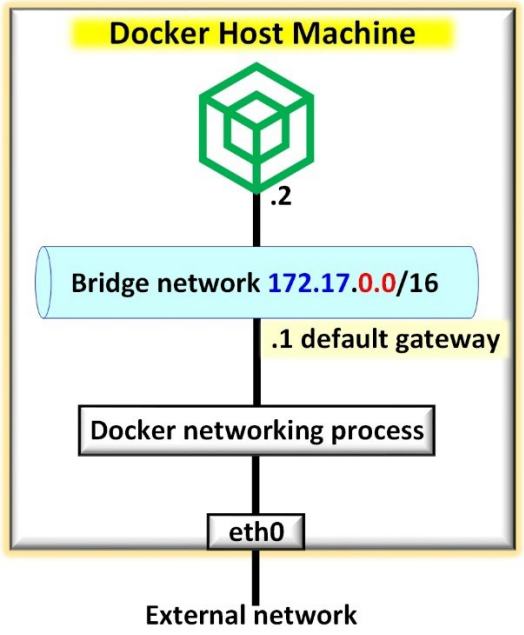
The docker network command lists the networking aspects of Docker:

```
[ec2-user@ip-172-31-93-252 ~]$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
3f4c03e6c6c1   bridge    bridge    local
0521a8e81e5b   host      host      local
9d41c2d9531f   none      null      local
```

When the container was created, it was automatically attached to the default bridge where it received an IPv4 address from an internal DHCP server. This address can be found by issuing the following **docker inspect** command in the container example.

```
[ec2-user@ip-172-31-93-252 ~]$ docker inspect example | grep IPAddress
            "IPAddress": "172.17.0.2",
[ec2-user@ip-172-31-93-252 ~]$ docker inspect example | grep Gateway
            "Gateway": "172.17.0.1",
```

Pictorially interpreted like this:



Docker container networking

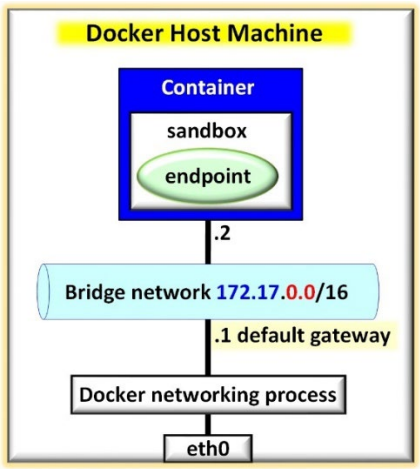
The command inspect bridge offers more details regarding the bridge configuration:

```
[ec2-user@ip-172-31-93-252 ~]$ docker inspect bridge
[
    {
        "Name": "bridge",
        "Id": "3f4c03e6c6c15369645e5aba3460e3910dd3a4e6087ee9b524ac7fcd8d489ce4",
        "Created": "2023-02-02T20:12:08.121068365Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
```

The output of this command also gives the network details of each container attached to the bridge:

```
  "Containers": {
     "61aee97007cf42e2d90c9077766c2f7fe7a74be92664d5c40501eeba044dfb8c": {
        "Name": "example",
        "EndpointID": "af4f83c03d76844e96cefdfe52648a909507a69696ce96f91888b3f4af6c5e1f",
        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16",
        "IPv6Address": ""
      }
  },
```

This information is represented graphically:



Docker container example with host address 172.17.0.2/16 attached to default bridge

Learning Activity

Running NGINX webservers on containers

In this activity, two webserver containers, named C1 and C2, will be deployed based on the latest NGINX image. NGINX is a webserver application. The two containers are created as it follows:

```
docker run -itd --name C1 -p 8081:80 nginx:latest
docker run -itd --name C2 -p 8082:80 nginx:latest
```

The flag –p maps a Docker host external port to the container's internal port. The NGINX webserver image has the port 80 open by default. In consequence, different containers running on the same host must have different exposed ports. This will become more evident in the following analysis.

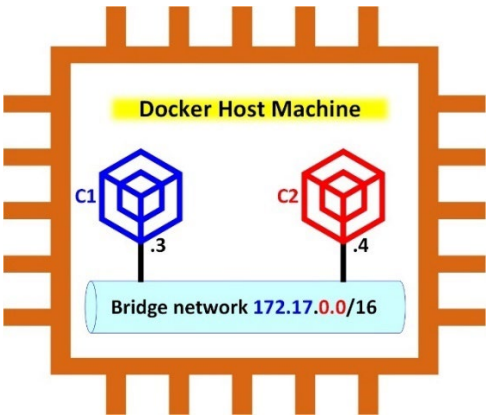
The containers appear like this:

```
[ec2-user@ip-172-31-93-252 ~]$ docker container ls
CONTAINER ID   IMAGE          PORTS                    NAME
3a08e4807674   nginx:latest   0.0.0.0:8081->80/tcp     C1
c8b98e80e6dc   nginx:latest   0.0.0.0:8082->80/tcp     C2
```

Right now, the output of the ifconfig command shows two new virtual interfaces in the docker host computer. Every time that a new container is created, a corresponding virtual interface is created.

```
vethe0b166a1:
        inet6 fe80::e8d4:13ff:fe51:2802  prefixlen 64
        Ethernet MAC address ea:d4:13:51:28:02
Vethe4f84a7a:
        inet6 fe80::8887:f3ff:fe3d:a28d  prefixlen 64
        Ethernet MAC address 8a:87:f3:3d:a2:8d
```

Furthermore, each docker container gets assigned an IPv4 address from the bridge's address space 172.17.0.0/16. The container C1 obtained the address 172.17.0.3/16 while C2 got the address 172.17.0.4/16.

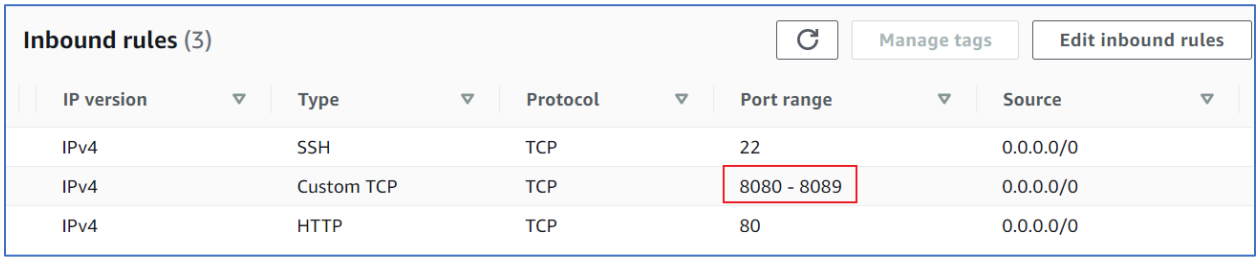


NGINX Containers C1 and C2

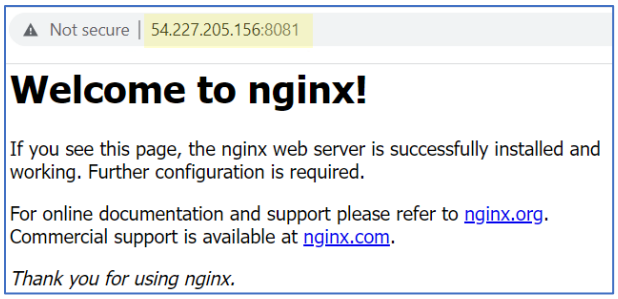
In principle, these addresses should be reachable from the host computer (the EC2 instance).

```
[ec2-user@ip-172-31-93-252 ~]$ ping 172.17.0.3
64 bytes from 172.17.0.3: icmp_seq=1 ttl=255 time=0.033 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=255 time=0.043 ms

[ec2-user@ip-172-31-93-252 ~]$ ping 172.17.0.4
64 bytes from 172.17.0.4: icmp_seq=1 ttl=64 time=0.051 ms
64 bytes from 172.17.0.4: icmp_seq=2 ttl=64 time=0.042 ms
```

Since the ports exposed are 8081 and 8082, the security group requires a modification to provide external access to the webservers. The port range 8080-8089 covers the ports from 8080 to 8089.



The container C1's web application is now accessible using the tuple EC2's public_IPv4_address: port number; specifically, 54.227.205.156:8081.



Testing with the curl command from the Docker host to the container should work too:

```
[ec2-user@ip-172-31-93-252 ~]$ curl 172.17.0.3
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
```

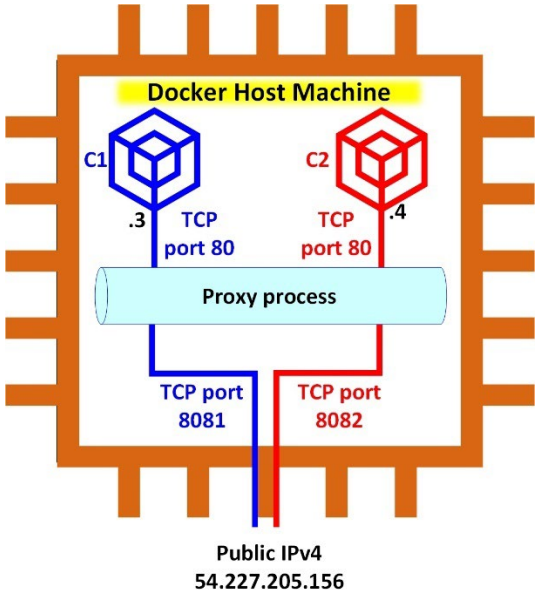
The container answers to calls to port 80 if they are issued in the docker host (EC2 instance).

```
[ec2-user@ip-172-31-93-252 ~]$ curl 172.17.0.3:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
```

However, the container does not listen on port 8081.

```
[ec2-user@ip-172-31-93-252 ~]$ curl 172.17.0.3:8081
curl: (7) Failed to connect to 172.17.0.3 port 8081 after 0 ms:
Connection refused
```

This deserves an explanation. The host computer (the EC2 instance) allows external HTTP communication via the ports 8081 and 8082, but both containers listen on 80. Therefore, the sessions directed to port 8081 go to container C1 while sessions directed to 8082 go to container C2. This is possible thanks to an internal process running in the Docker host which acts as a proxy matching the TCP sessions directed to each exposed port to the corresponding container.



The containers port mapping process

Learning Activity

Modification of the container

The NGINX container are still showing their testing HTML homepage. Let's try to modify such behaviour to better understand the nature of containers. The command exec followed with the bash interpreter is used to drop into the container. NGINX keeps the default index.html file in the /usr/share/nginx/html directory.

```
[ec2-user@ip-172-31-93-252 ~]$ docker exec -it C1 /bin/bash
root@3a08e4807674:/# cd /usr/share/nginx/html/
root@3a08e4807674:/usr/share/nginx/html# ls
50x.html  index.html
```

Since the file exists, it could be possible to edit it with a text editor.

```
root@3a08e4807674:/usr/share/nginx/html# vi index.html
bash: vi: command not found
root@3a08e4807674:/usr/share/nginx/html# nano index.html
bash: nano: command not found
```

However, the NGINX image does not have any text editor installed. This is consistent with the container concept of having only the packages and tools strictly necessary to run the application. Since there is no text editor, this trick might work to get a simple text home page.

```
echo "<html><body><h1> Hello from C1 </h1></body></html>" > index.html
```

```
root@3a08e4807674:/usr/share/nginx/html# cat index.html
<h1> Hello from C1 </h1>
```

The echo command is available so the index.html file has been overwritten. Let's test the result.

```
root@3a08e4807674:# curl 127.0.0.1
<html><body><h1> Hello from C1 </h1></body></html>
```

Now, the container has been effectively modified. Let's make an image out of the running container. First, the container id is listed.

```
docker container ls
d3c79da8c281 (the container id)
```

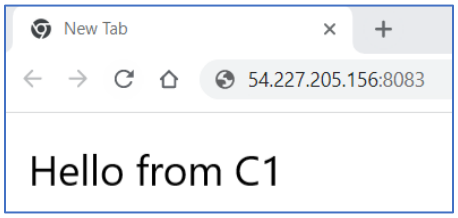
The following command makes an image with a given name using the container id. (In this example, it was called repository/new_nginx).

```
docker commit -m "new nginx" -a "user" d3c79da8c281 repository/new_nginx
```

Now, create a new container with the modified image.

```
docker run -itd --name myweb -p 8083:80 repository/new_nginx
```

Test the new container.



It worked, but this is not the way to create new containers. The correct way is to write an application project, create a file describing how to conform an image, then create the image to make containers.

Learning Activity

Using Volumes to keep persistent data

Containers have an ephemeral nature. As such, they are created, used and then they get replaced by improved versions of the application. The life of containers is a pipeline of continuous integration and continuous development. Nevertheless the containers transient nature, the data that they use is not always ephemeral. Therefore, a storage for persistent data must be available.

Volumes are permanent stores of data located in the Docker host server in a way that can be accessed by the containers. This activity demonstrate the creation and use of a volume to load the index.html file that NGINX containers will use. First, let's create a volume in the Docker host machine:

```
[ec2-user@ip-172-31-93-252 ~]$ docker volume create VOL-EXAMPLE
VOL-EXAMPLE
```

The volume VOL-EXAMPLE exists now in the host's **/var/lib/docker/volumes/** directory.

```
[ec2-user@ip-172-31-93-252 ~]$ sudo su
[ec2-user@ip-172-31-93-252 ~]$ cd /var/lib/docker/volumes/VOL-EXAMPLE
```

This newly created volume automatically contains a folder **_data** which is empty.

```
/var/lib/docker/volumes/VOL-EXAMPLE$ ls
_data
/var/lib/docker/volumes/VOL-EXAMPLE$ cd _data
/var/lib/docker/volumes/VOL-EXAMPLE/_data$ ls
Nothing here, it is empty.
```

The next step is to create a new container based on the NGINX image. Webserver NGINX containers load the webserver files from **/usr/share/nginx/html**. Thus, the creation command includes the – v flag to map the volume to that file path.

```
[ec2-user@ip-172-31-93-252 ~]$ docker run -itd --name C1-VOL -p 8084:80 \
> -v VOL-EXAMPLE:/usr/share/nginx/html nginx:latest
```

A new look into the VOL-EXAMPLE/_data folder shows that two files has been added by the creation of the NGINX container.

```
/var/lib/docker/volumes/VOL-EXAMPLE/_data$ sudo ls
50x.html  index.html
```

When the container spun up, it looked into the volume and it found that it was empty. So, it copied the contents of its default **/usr/share/nginx/html** directory (the index.html and 50x.html files) to the volume in **/var/lib/docker/volumes/** in the docker host computer. Notice that the directory path where the container has its NGINX files is different than the directory path in the docker host's volume.

To demonstrate the functioning of the external volume, the index.html file is modified next with the nano editor in the Docker host machine:

```
 GNU nano 2.9.8            index.html
<html><body>
<h1>Demonstrating the VOLUME concept!</h1>
</body></html>
```

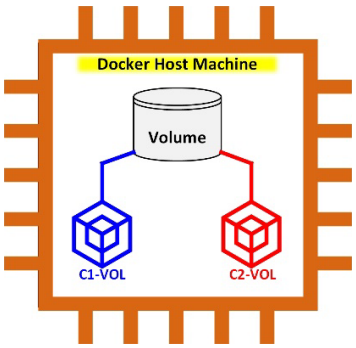
The web browser offers the conclusive demonstration of the container's welcome web page.

Until this point, there is one container running using the data stored in the volume VOL-EXAMPLE. Let's create a second container to use the same volume.

```
[ec2-user@ip-172-31-93-252]$ docker run -itd --name C2-VOL -p 8085:80 \
> -v VOL-EXAMPLE:/usr/share/nginx/html nginx:latest
```

Again, the web browser offers the proof that the data is coming from the volume VOL-EXAMPLE:

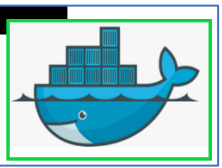
To summarize, a volume was created in the Docker host machine. Initially, this volume contained no files. A NGINX container was created with the instruction that the webserver html files should be located in the volume by mapping it to the path /usr/share/nginx/html. Since, the initial container did not find any files there, it created the default test homepage files there. The index.html file was edited from the Docker host machine. Similarly, a second container was created but this time, it encountered the modified index.html in the volume thus loading that file.



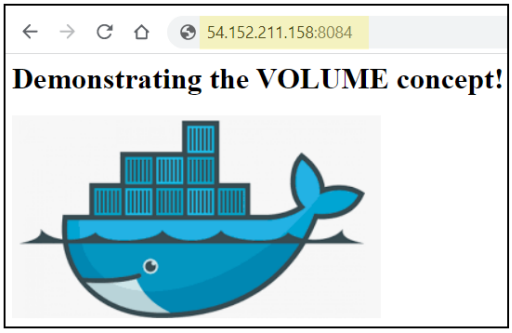
EC2 instance hosting two containers accessing the same volume

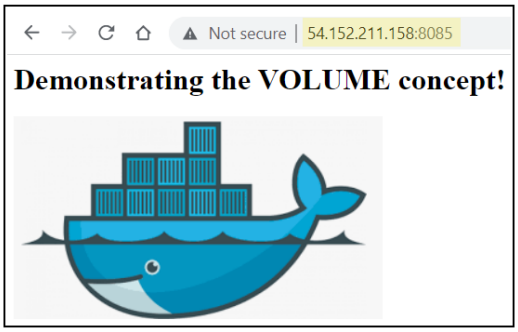
Finally, to further demonstrate the functioning of the Docker volume, an image is uploaded to the host machine, then moved into the volume _data. The index.html file is again edited to load the image.

```
 GNU nano 2.9.8          index.html
<html><body>
<h1>Demonstrating the VOLUME concept!</h1>
<img src="docker-icon.png" 250 />
</body></html>
```



Both containers show the common data from the same index.html file proving that they are accessing the same data from the volume VOL-EXAMPLE.



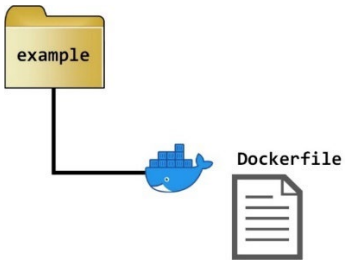


Learning Activity

Creation of a Docker Image with Dockerfile

A Dockerfile is a list of commands and actions that yields a Docker image when executed on a Docker host. In this section, basic Docker images are created from Dockerfile definitions. Let's being with a super basic image. First, create an example directory to hold this little Docker project.

```
[ec2-user@ip-172-31-93-252 ~]$ pwd
/home/ec2-user
[ec2-user@ip-172-31-93-252 ~]$ mkdir example
[ec2-user@ip-172-31-93-252 ~]$ cd example
[ec2-user@ip-172-31-93-252 ~]$ nano Dockerfile
```



Project file structure

The Dockerfile only has two lines. The FROM instruction pulls a base image of the operating system Alpine which is a very basic, lean Linux distribution. The CMD instruction just echoes a text message on screen.

```
OpenSSH SSH client
 GNU nano 2.9.3                    Dockerfile

FROM alpine
CMD ["echo", "this image only shows this message, it does nothing else"]
```

Save file. CRTL-X, yes.

Build the Docker container image with the command **docker build .**

```
[ec2-user@ip-172-31-93-252 ~]$ docker build .
```

The **dot** at the end means that "the Dockerfile is located in this directory". If the file were in another directory, then the path to the file would have to be indicated.

```
docker build .
Sending build context to Docker daemon 2.048kB
Step 1 out of 2 : FROM alpine
latest: pulling from library/alpine
59bf1c3509f3: Pull complete
Digest: sha256: 50d858e0985ecc7f60418aaf0cc5ab587f42c2570a884095a9e8ccacd0f6545c
Status: downloaded newer image for alpine:latest
---> c059bfaa849c
Step 2 out of 2 : CMD ["echo", "this image only shows this message, it does nothing else"]
---> Running in 96e158fe8668
Removing intermediate container 96e158fe8668
---> 8f827eee9036
Successfully built 8f827eee9036
```

- When the command was executed, Docker first found an instruction (FROM) to get Linux Alpine. Since there was no such local image, it pulled the image from the public Docker repository.
- That formed the substrate or first layer of the Docker image.
- In a second step, the CMD provides a message that will show when the container runs (similar to the Hello World Docker container).
- The intermediate container that was created is replaced and that renders the final image.
- Images are built in layers executed one after the other.
- The resulting image 8f827eee9036 is shown here.
- Additionally, the alpine image used in the construction of the image is also here.

```
[ec2-user@ip-172-31-93-252 ~]$ docker image ls
REPOSITORY      TAG          IMAGE ID        CREATED          SIZE
<none>          none         8f827eee9036    9 minutes ago    5.59 MB
alpine          latest       c059bfaa849c    44 hours ago     5.59 MB
```

Let's test the image by creating a container named DEMO-01.

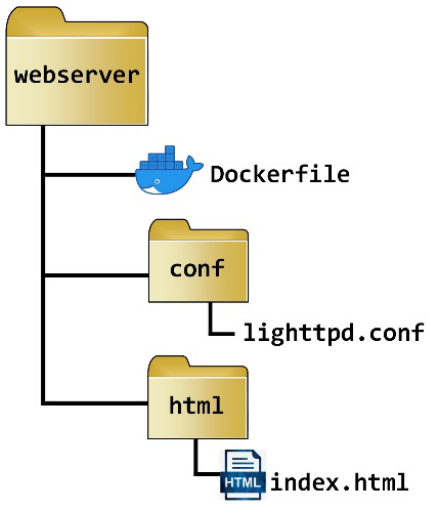
```
docker run --name DEMO-01 8f827eee9036
this image only shows this message, it does nothing else
```

That's it, the container was created off the image. It did not do much, but it made the point about how images are created. The next example is a bit more elaborated.

Learning Activity

Creating a basic Docker image for a webserver on Linux Alpine

This docker project will be running the webserver software lighttpd on Linux Alpine. A main folder named webserver contains all the files required to create the image. A folder conf contains the basic configuration of the software lighttpd. Another folder named html contains an index.html file for the web home page. Finally, the Dockerfile puts everything together to create a Docker image. From there, Docker containers can be created. This is the directory filesystem.




```
[ec2-user@ip-172-31-93-252 ~]$ ls
Dockerfile    conf    html
```

Docker project filesystem

Write a basic index.html file under the folder html.

```
[ec2-user@ip-172-31-93-252 ~]$ cd html
[ec2-user@ip-172-31-93-252 ~]$ nano index.html
[ec2-user@ip-172-31-93-252 ~]$ cat index.html
<html><body><h1> Basic Docker webserver on Alpine </h1></body></html>
```

The package **lighttpd** is Linux Alpine webserver software, but it is so basic that it needs some basic configuration to be able to process web requests. That is done in the next snippet.

```
[ec2-user@ip-172-31-93-252 ~]$ cat lighttpd.conf
server.document-root = "/var/www/html"
server.port = 80
server.indexfiles = ("index.html")
mimetype.assign = (
  ".html" => "text/html",
  ".txt" => "text/plain",
  ".jpg" => "image/jpeg",
  ".png" => "image/png"
)
```

The Dockerfile ties all together.

```
[ec2-user@ip-172-31-93-252 ~]$ cat Dockerfile
FROM alpine
RUN apk update && apk upgrade
RUN apk add --update lighttpd
COPY ./conf/* /etc/lighttpd/
COPY ./html/* /var/www/html/
EXPOSE 80
CMD ["lighttpd","-D","-f","/etc/lighttpd/lighttpd.conf"]
```

This is the explanation of the commands on this Dockerfile:

- FROM alpine gets the base operating system. Alpine is a lightweight Linux operating system. Since its size is very small, the resulting image and containers are very lean too.
- RUN updates and upgrades obtain and install respectively the latest packages.
- The command apk is specific to Linux Alpine. This is Alpine's package manager that handles the distribution and software for this operating system. In this case, the packages of lighttpd web server are downloaded and installed by the apk.
- The COPY instruction takes the configuration file to the proper directory in /etc/lighttpd.
- Another COPY instruction sends the index.html file from the local /html folder to the /var/www/html directory in the image.
- The port 80 is exposed. Hence all containers derived from this image will be internally listening on port 80 as the server port.
- Finally, the CMD instruction defines the default behavior of the container created after this image.
- Specifically, the "-D" flag sets the containers to start lighttpd in daemon mode. That means, that the containers will run without taking control of the host's console.
- The "-f" flag tells lighttpd to use the provided configuration in place of the default config file.

Save the files and create the image with the command:

```
[ec2-user@ip-172-31-93-252 ~]$ docker build –t  myweb:1.0 .
```

- The flag –t (tag) enables the assignment of a name to the image.
- The **:1.0** is used to indicate a version number.
- (Notice that there is a **dot** at the end of the command)
- This is the image building process:

```
[ec2-user@ip-172-31-93-252 ~]$ docker build -t myweb:1.0 .
Sending build context to Docker daemon  5.12kB
Step 1/7 : FROM alpine
 ---> c059bfaa849c
Step 2/7 : RUN apk update && apk upgrade
 ---> Running in 1d23d364e73c
fetch https://dl-cdn.alpinelinux.org/alpine/v3.15/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.15/community/x86_64/APKINDEX.tar.gz
v3.15.0-21-g41a764a23f [https://dl-cdn.alpinelinux.org/alpine/v3.15/main]
v3.15.0-21-g41a764a23f [https://dl-cdn.alpinelinux.org/alpine/v3.15/community]
OK: 15829 distinct packages available
OK: 6 MiB in 14 packages
Removing intermediate container 1d23d364e73c
 ---> d7207fc64bad
Step 3/7 : RUN apk add --update lighttpd
 ---> Running in 089d02763791
(1/9) Installing brotli-libs (1.0.9-r5)
(2/9) Installing libdbi (0.9.0-r0)
(3/9) Installing gdbm (1.22-r0)
(4/9) Installing libsasl (2.1.27-r14)
(5/9) Installing libldap (2.6.0-r0)
(6/9) Installing lua5.4-libs (5.4.3-r0)
(7/9) Installing pcre (8.45-r1)
(8/9) Installing zstd-libs (1.5.0-r0)
(9/9) Installing lighttpd (1.4.61-r1)
Executing lighttpd-1.4.61-r1.pre-install
Executing busybox-1.34.1-r3.trigger
OK: 10 MiB in 23 packages
Removing intermediate container 089d02763791
 ---> ddbe7e9914be
Step 4/7 : COPY ./conf/* /etc/lighttpd/
 ---> e8a931065af9
Step 5/7 : COPY ./html/* /var/www/html/
 ---> 1cf06ab50893
Step 6/7 : EXPOSE 80
 ---> Running in aa2f9b77fa4f
Removing intermediate container aa2f9b77fa4f
 ---> ba15b669a9ed
Step 7/7 : CMD ["lighttpd","-D","-f","/etc/lighttpd/lighttpd.conf"]
 ---> Running in ee102d73e9ca
Removing intermediate container ee102d73e9ca
 ---> 9f1b0a443c4f
Successfully built 9f1b0a443c4f
Successfully tagged myweb:1.0
```

The image exists now:

```
[ec2-user@ip-172-31-93-252 ~]$ docker image ls
REPOSITORY   TAG   IMAGE ID       CREATED          SIZE
myweb        1.0   9f1b0a443c4f   53 seconds ago   12.5 MB
```

Make a container with the image:

```
[ec2-user@ip-172-31-93-252 ~]$ docker run -d  --name web1 -p 8080:80  9f1b0a443c4f
0d895d4b47efba666380f1e60b494512ce8ffcb2ac93807bdade0efd8c809995
```

The docker inspect command shows the container's path file:

```
[ec2-user@ip-172-31-93-252 ~]$ docker inspect web1
[
        {
                "Id"0d895d4b47efba666380f1e60b494512ce8ffcb2ac93807bdade0efd8c809995,
                "Created": 2023-03-02 T22:21:53.097464696Z",
                "Path": "lighttpd",
                "Args": [
                        "-D",
                        "-f",
                        "/etc/lighttpd/lighttpd.conf"
                ],
                "State": { "Status": "running",
```

Which is exactly what it was defined in the Dockerfile. The command also shows that the external port is 8080 and that the internal port is 80.

```
"Ports": {
        "80/tcp": [
                {
                        "HostIp": "0.0.0.0/0",
                        "HostPort": "8080/tcp"
                }
```

The container is tested with a curl command:

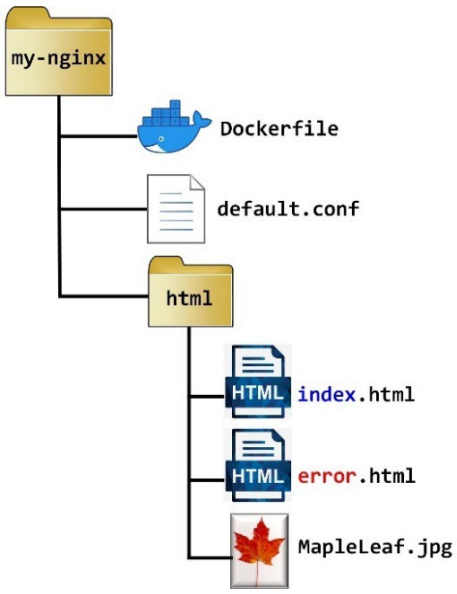
```
[ec2-user@ip-172-31-93-252 ~]$ curl 172.17.0.2
<html><body><h1> Basic Docker webserver on Alpine </h1></body></html>
```

The container should respond from a customer browser (44.193.28.135:8080):

Learning Activity

Creating a Docker image for a NGINX webserver on Linux Alpine

Container images are based on projects. For example, a web application with all its accompanying files must be incorporated into an image and from that image, containers of the application are made. The following diagram presents this basic idea. A main folder contains the Dockerfile, a NGINX webserver default configuration file, and another folder. This child folder contains a simple web app composed by an index html file, an error message html file, and a picture of a maple leaf for the web site homepage.



Docker project my-nginx directory

The following Dockerfile will be used to create a Docker image for a basic NGINX webserver container. It begins by obtaining an initial base image of NGINX build on Linux Alpine. The second step moves all the project files from the folder named HTML to the NGINX [4, 5] directory location /usr/share/nginx/html/. This includes the index.html file, the error html file and any accompanying files such as the homepage image. The third step copies the webserver's configuration file to the location where it should live in /etc/nginx/conf.d/. The fourth step enables the port 80 to listen for the TCP sessions that deliver the HTTP application. The final instruction contains the flag –g which tells NGINX to set the global configuration options for NGINX and the daemon off to run the container for as long as NGINX is running.

```
# 1) Use a pre-built NGINX on Alpine image.
FROM nginx:alpine
# 2) Copy all files from project folder HTML to where NGINX expects to find them.
COPY ./HTML/* /usr/share/nginx/html/
# 3) Copy the webserver configuration file to the location that NGINX uses.
COPY ./default.conf /etc/nginx/conf.d/
# 4) Listen on port 80 for incoming traffic.
EXPOSE 80
# 5) Start NGINX when the container launches. Keep NGINX in the foreground.
CMD ["nginx", "-g", "daemon off;"]
```

This is **default.conf** file that configures the webserver. It indicates the TCP port being used and the location of the serving HTML files. It also instructs the server on how to provide the error message when client requests can not be answered.

```
server {
    listen       80;
    server_name  localhost;

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }
    # redirect server error pages to static page /50x.html
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
}
```

Finally, this is the index.html file.

```
<!DOCTYPE html>
<html><body>
    <h1> Docker container running NGINX on Alpine </h1>
    <img src="MapleLeaf.jpg" width=200/>
</body></html>
```

Let's proceed to create the image with the build command:

```
[ec2-user@ip-172-31-93-252 my-nginx]$ docker build -t web-nginx .
```

The execution of the command pulls a base nginx on alpine image from Docker repository and then superimpose the layers that yield the final image tagged as web-nginx.

```
[ec2-user@ip-172-31-93-252 my-nginx]$ docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
web-nginx    latest    0677d8af16d8   2 minutes ago    41.3MB
nginx        alpine    2bc7edbc3cf2   3 weeks ago      40.7MB
```

Since the image is available, a container is created to test its functionality.

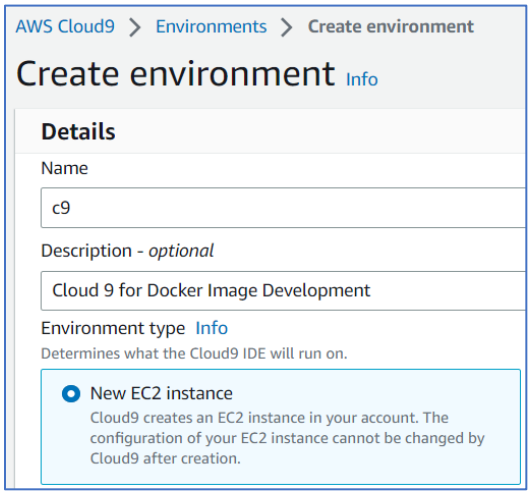
```
[ec2-user@ip-172-31-93-252 ~]$ docker run --name WEB -d -p 5000:80 web-nginx
35d0f5c7c213ddece4d52c69acc650bfd61f8c7da5594b430a6beb09eb0ddeeb
```

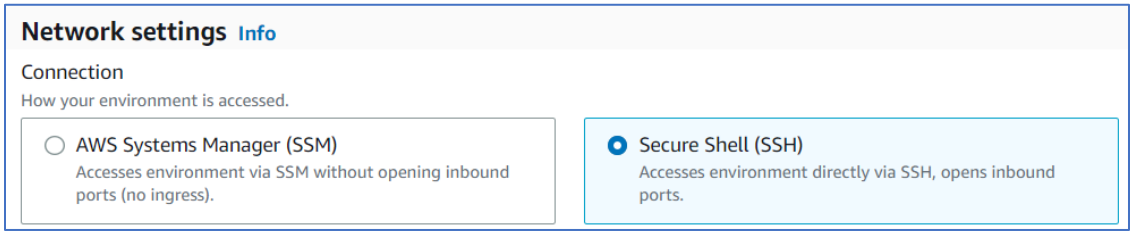
This the final proof of the functioning container (modify the security group to allow port 5000).

Learning Activity

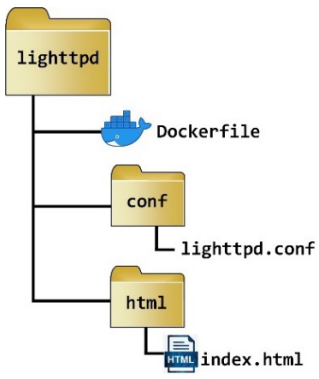
Creating a Docker container image in AWS Cloud9

In this activity, an image will be created to be used to create a container later on. The image must be created in a computer that has Docker installed. A convenient way to create the image is to use an AWS Cloud9 environment. Cloud9 is an AWS service that creates a EC2 instance specially equipped with SDKs for Python, JavaScript, Go and others including Docker. This special instance provides an **Integrated Development Environment (IDE)** for writing the files that compose an application project. Hence, an environment is created in the AWS Cloud9 service:





This triggers the creation of the special EC2 instance that provides the development environment. Once that the instance is ready, the following directory is created. The Dockerfile lives at the root level. A conf folder contains the configuration file of a simple lighttpd webserver. The HTML folder contains the index.html file for the webserver homepage.




```
voclabs:~/environment $ pwd
/home/ec2-user/environment
voclabs:~/environment $ clear
voclabs:~/environment $ mkdir lighttpd
voclabs:~/environment $ cd lighttpd/
voclabs:~/environment/lighttpd $ touch Dockerfile
voclabs:~/environment/lighttpd $ mkdir conf
voclabs:~/environment/lighttpd $ mkdir html
voclabs:~/environment/lighttpd $ ls
conf  Dockerfile  html
```

The file lighttpd [7, 10] contains the configuration file of a light http server on Linux Alpine. It specifies that the server must look into the /var/www/html folder to find the index.html and that the server will listen on TCP port 80. The configuration file also specifies the media types (Multipurpose Internet Mail Extensions) that the server can handle.

```
voclabs:~/environment/lighttpd $ cd conf
voclabs:~/environment/lighttpd/conf $ touch lighttpd.conf
```

lighttpd.conf

```
server.document-root = "/var/www/html"
server.port = 80
server.indexfiles = ("index.html")
mimetype.assign = (
    ".html" => "text/html",
    ".txt" => "text/plain",
    ".jpg" => "image/jpeg",
    ".png" => "image/png"
)
```

The next file is a basic HTML index file

```
voclabs:~/environment/lighttpd $ cd html/
voclabs:~/environment/lighttpd/html $ touch index.html
```

index.html

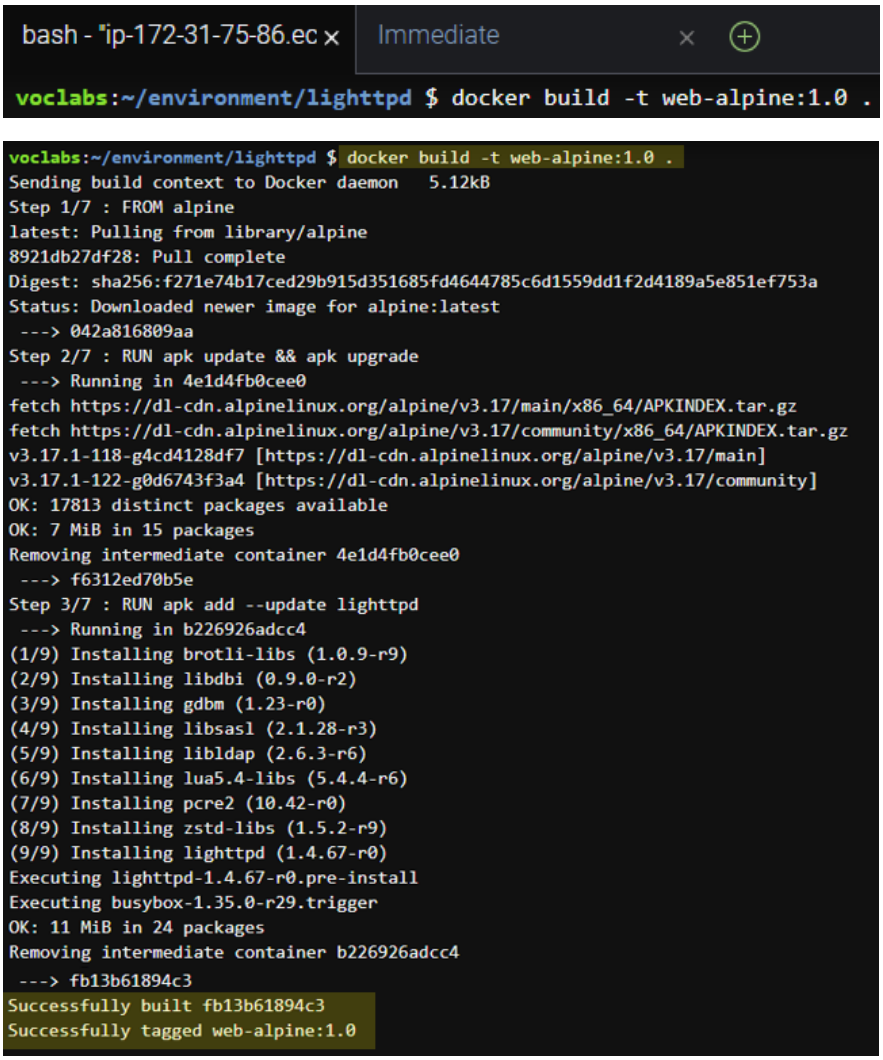
```
<html>
    <body>
        <h1>
            Simple webserver lighttpd on Linux Alpine.
        </h1>
    </body>
</html>
```

The Dockerfile [7, 10] gets everything together. The image will start with a basic operating system layer provided by Linux Alpine. This initial layer is actually a Docker container image of Alpine that will be pulled from the public repository. This image will be complemented with the updates, upgrades and the installation of the webserver application lighttpd. The files from the folders conf and html are copied onto the proper directories on the webserver image.

Dockerfile

```
FROM alpine
RUN apk update && apk upgrade
RUN apk add --update lighttpd
COPY ./conf/* /etc/lighttpd/
COPY ./html/* /var/www/html/
EXPOSE 80
CMD ["lighttpd","-D","-f","/etc/lighttpd/lighttpd.conf"]
```

Finally, the Docker image is built with an assigned name web-alpine. The number 1.0 is an arbitrarily assigned version number.



The image web-alpine tagged version 1.0 has been created as the next command proves. Notice that the starting point image alpine is there too.

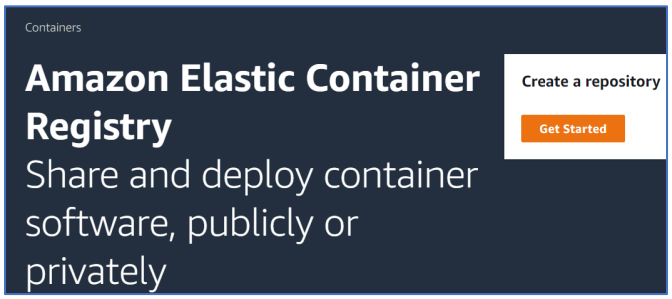
```
voclabs:~/environment/lighttpd $ docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
web-alpine   1.0       fb13b61894c3   3 minutes ago    13.7MB
alpine       latest    042a816809aa   2 weeks ago      7.05MB
```

The image is ready in the EC2 instance that hosts the IDE environment. The image will be pulled into a repository in AWS Elastic Container Registry Service in the next activity.

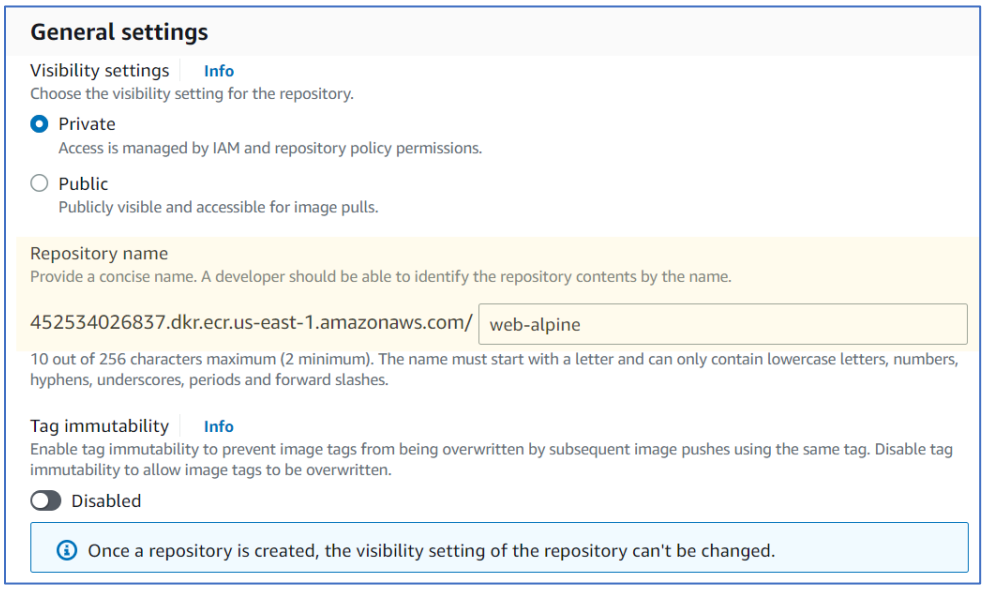
Learning Activity

AWS Elastic Container Registry Service

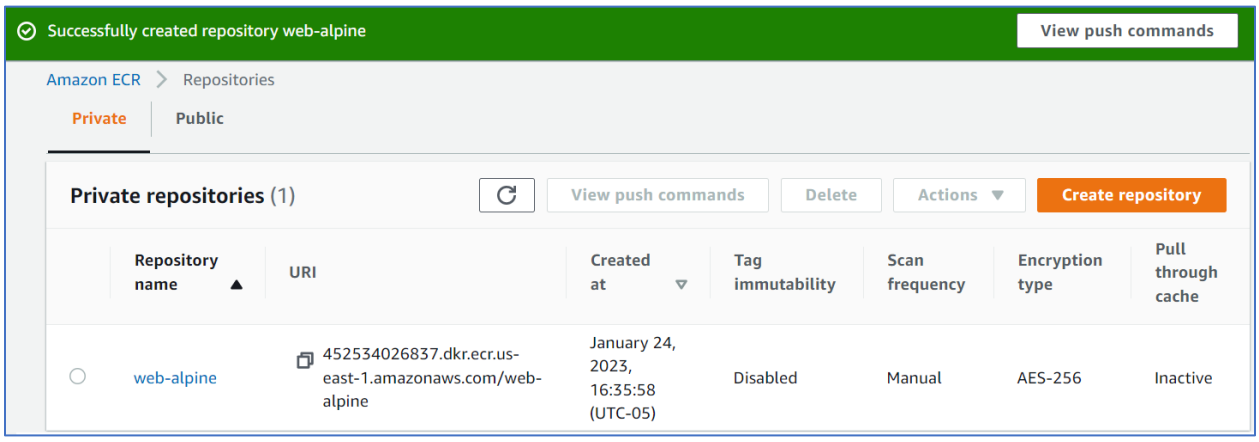
The first step is to create a **repository** to store the container image(s). This is done in the Amazon Elastic Container Registry (ECR). This is a service [8] that enables customers to store the container images in AWS.



This registry is set for private use only which would be the use case of a company.

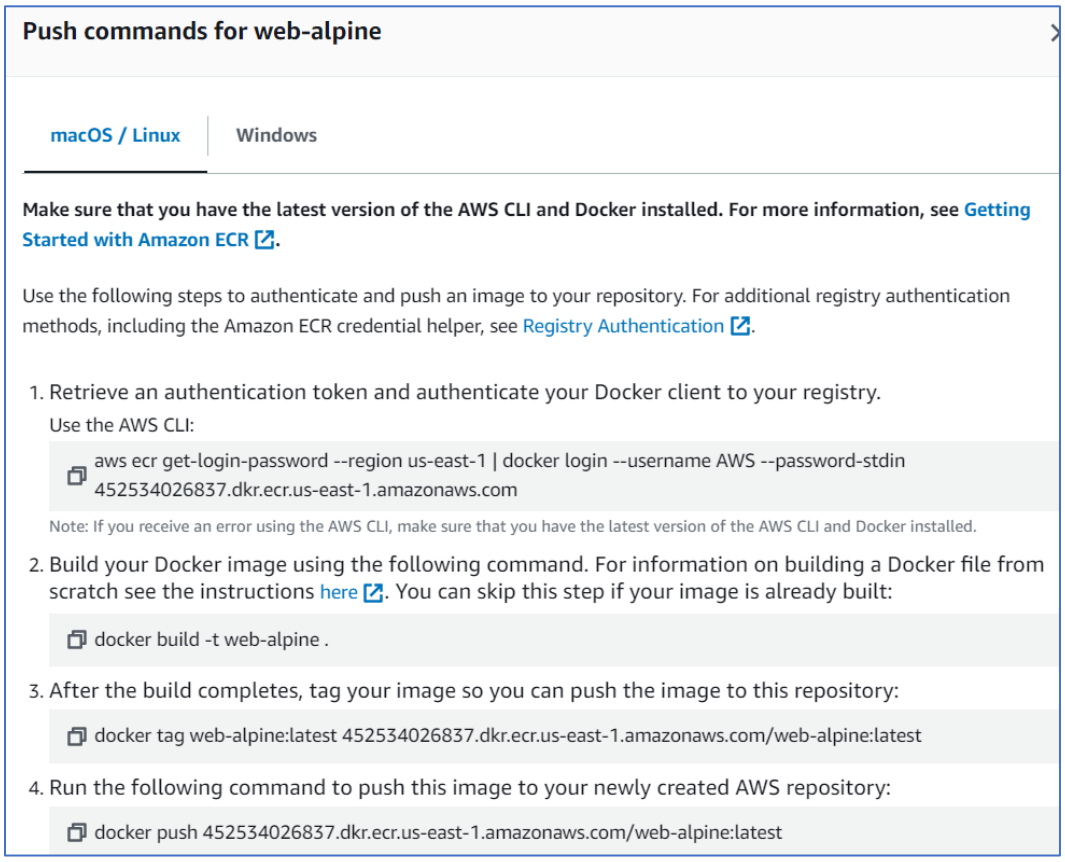


The repository is ready to incorporate the recently created image web-alpine from the previous activity.

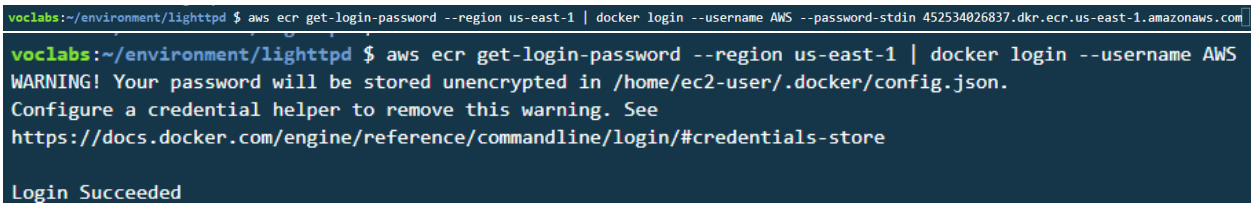


The repository service conveniently suggests the commands to push a copy of the image from the current location in the EC2 Cloud9 instance to the repository.

Four commands are listed. Command number one creates a security token to authenticate the access from the EC2 instance to the Elastic Container Registry. Command number two was already issued in Cloud9 when the image was created in the previous activity. However, for the sake of following the procedure, it will be reissued. Command three tags the image to be pushed into ECR. Command four copies the image from the Cloud9 location to the ECR repository.

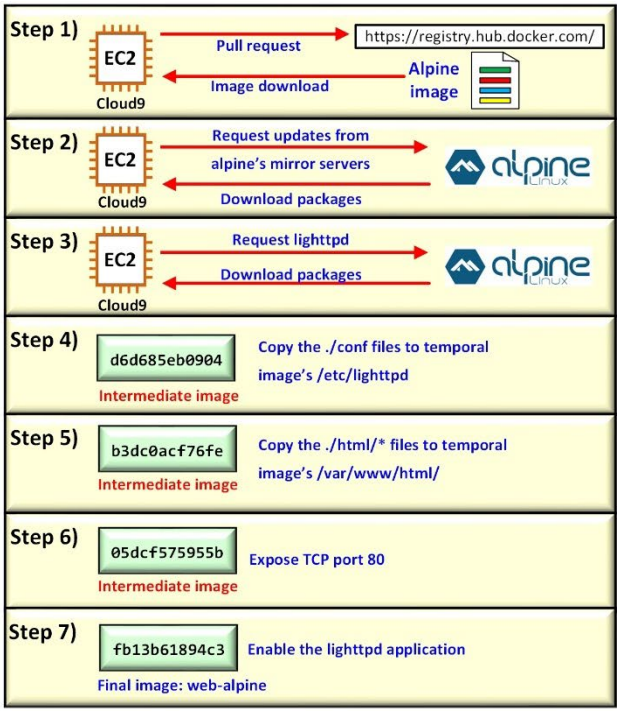


The first command is issued on the Cloud9 EC2 instance. The login is successful.



The second command recreates the image based on the existing project files. This diagram shows the seven transactions that occur based on the steps defined in the Dockerfile to create the image.

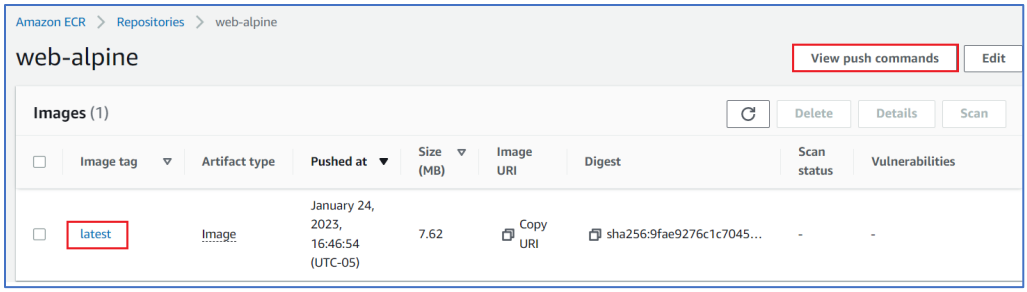
```
voclabs:~/environment/lighttpd $ docker build -t web-alpine .
Sending build context to Docker daemon  5.12kB
Step 1/7 : FROM alpine
 ---> 042a816809aa
Step 2/7 : RUN apk update && apk upgrade
 ---> Using cache
 ---> f6312ed70b5e
Step 3/7 : RUN apk add --update lighttpd
 ---> Using cache
 ---> 3e0f2fe1b340
Step 4/7 : COPY ./conf/* /etc/lighttpd/
 ---> Using cache
 ---> d6d685eb0904
Step 5/7 : COPY ./html/* /var/www/html/
 ---> Using cache
 ---> b3dc0acf76fe
Step 6/7 : EXPOSE 80
 ---> Using cache
 ---> 05dcf575955b
Step 7/7 : CMD ["lighttpd","-D","-f","/etc/lighttpd/lighttpd.conf"]
 ---> Using cache
 ---> fb13b61894c3
Successfully built fb13b61894c3
Successfully tagged web-alpine:latest
```



- The command docker tag sets a tag web-alpine:latest to the image.
- The command docker push brings the image over the Elastic Container Registry service.
- The name assigned is 452534026837.dkr.ecr.us-east-1.amazonaws.com/web-alpine:latest.

```
voclabs:~/environment/lighttpd $ docker tag web-alpine:latest 452534026837.dkr.ecr.us-east-1.amazonaws.com/web-alpine:latest
voclabs:~/environment/lighttpd $ docker push 452534026837.dkr.ecr.us-east-1.amazonaws.com/web-alpine:latest
The push refers to repository [452534026837.dkr.ecr.us-east-1.amazonaws.com/web-alpine]
e2976eab1462: Pushed
5ebd9c1b47db: Pushed
85b1347dfed5: Pushed
204e6156b45f: Pushed
8e012198eea1: Pushed
latest: digest: sha256:9fae9276c1c7045df5ee5fb9600a76456cefc75922fbccb70fe15aa7faf02a7f size: 1364
```

- The result can be observed in the repository web-alpine.



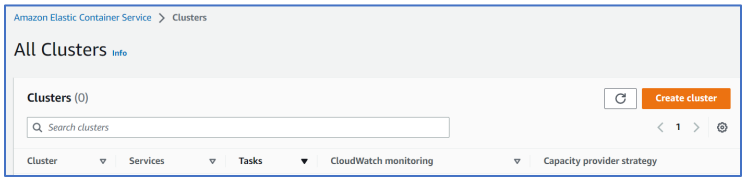
Now, the image is available for use in the next learning activity.

Learning Activity

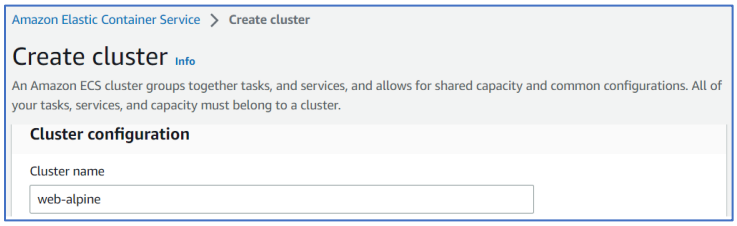
AWS Elastic Container Service

AWS Elastic Container Service (ECS) [8] is a container management service used to deploy and maintain from one container up to a fleet of containers. The applications are containerized in images, then they are stored in repositories in the Elastic Container Registry (ECR). From there, the images are used to create container-based services.

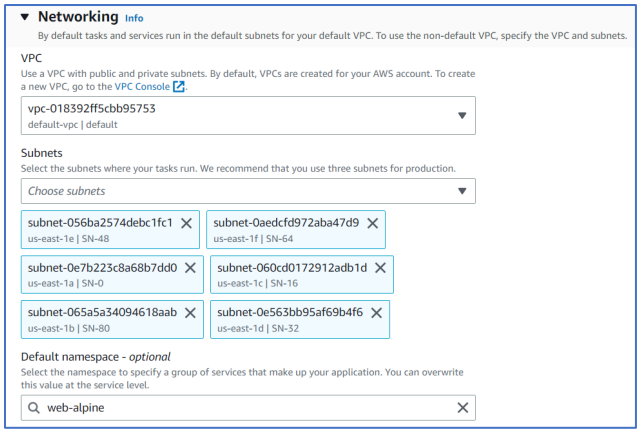
The first action is to create a **cluster**.



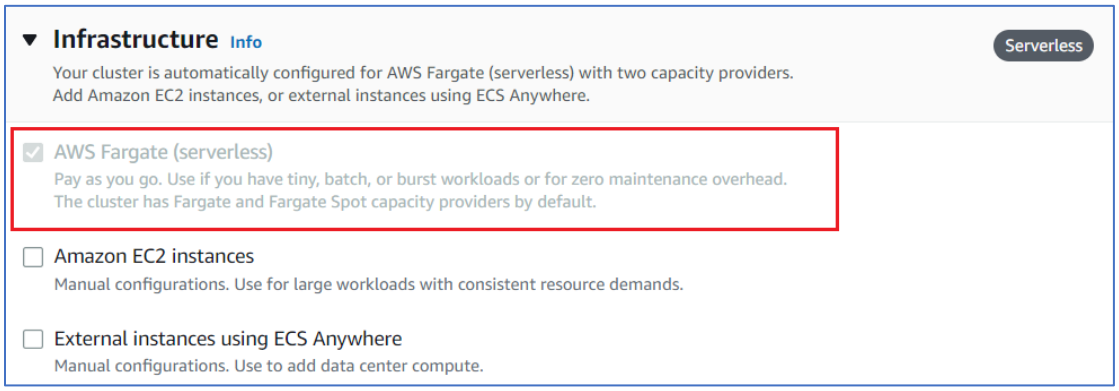
ECS requires the creation of a cluster whether it is to run just one container or a set of tasks and services.



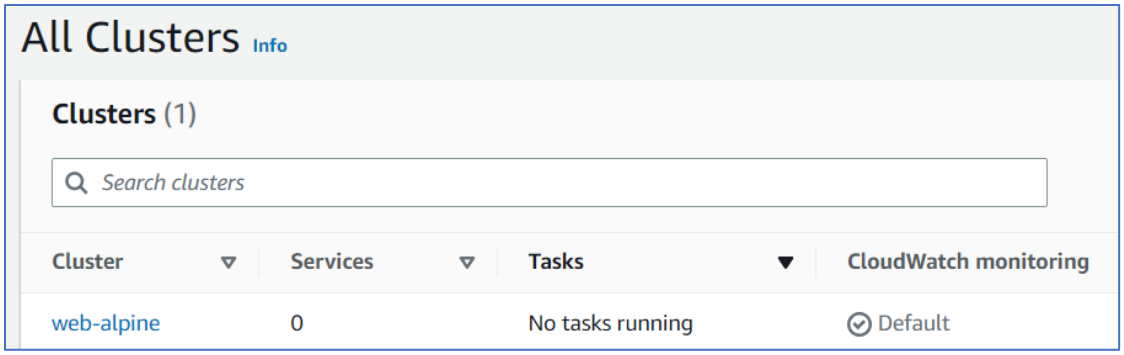
The containers are like any other resource that requires network access. In this case, the six default subnets of region us-east-1 have been selected. That means that the cluster will have EC2 instances running on every availability zone of the region.



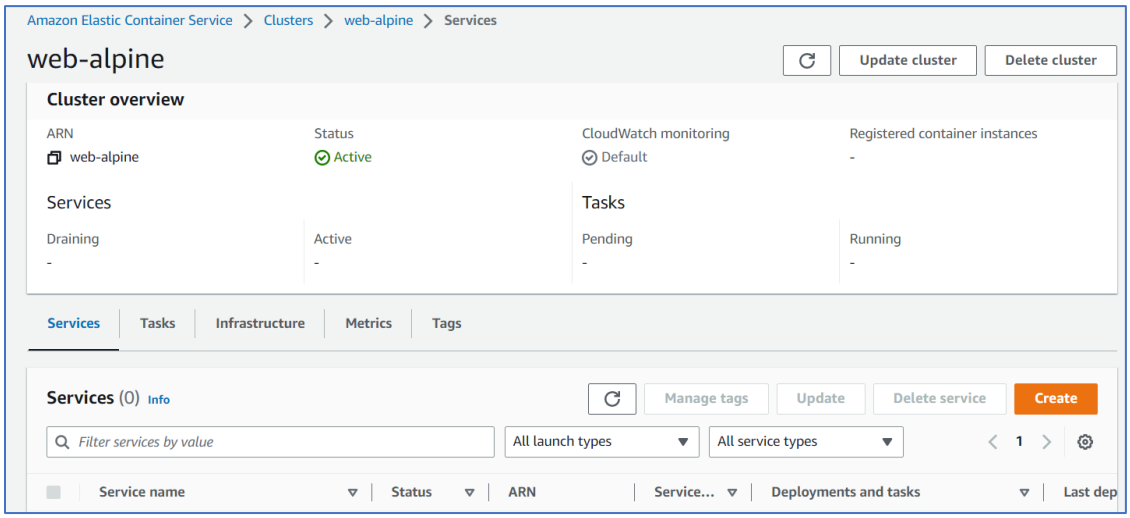
There are three ways to configure the infrastructure where the containers will run. The customer might choose to take care of the provisioning and management of it either by using AWS EC2 instances or external on-prem servers. A third attractive option is Amazon **Fargate** [9] which is a service which takes care of the infrastructure so the customer only needs to take care of the applications. This kind of service receives the generic name of "**serverless**" since the customer does not handles the infrastructure. This is a misleading name because in reality; there are servers someplace supporting the infrastructure. It is just that the customer does not deal with it.



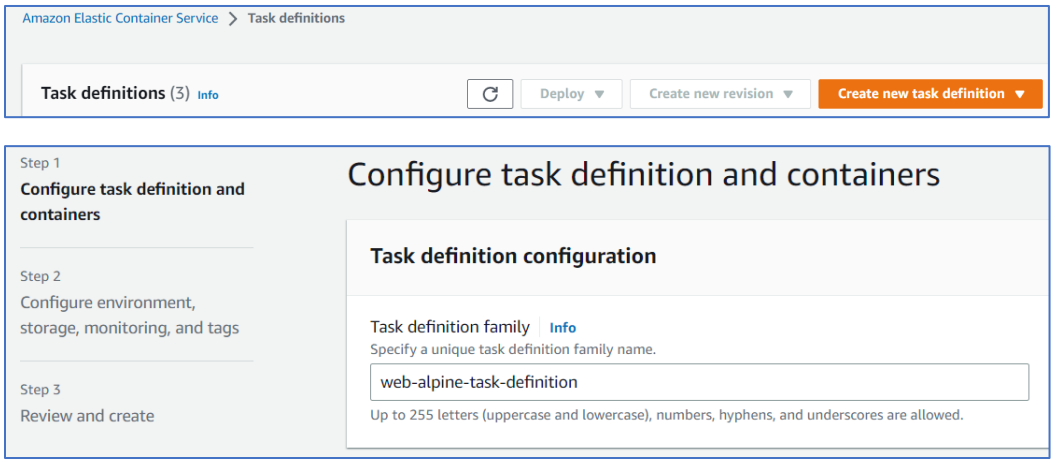
Finally, the cluster is ready.



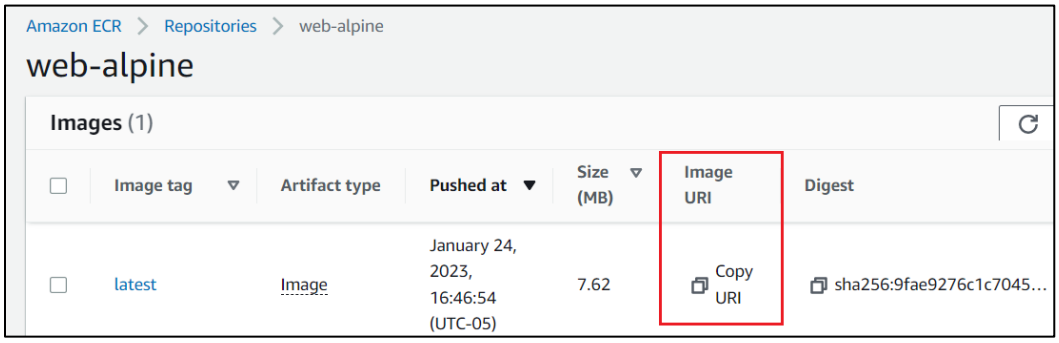
The cluster is fundamentally a group of cloud resources that support the application service.



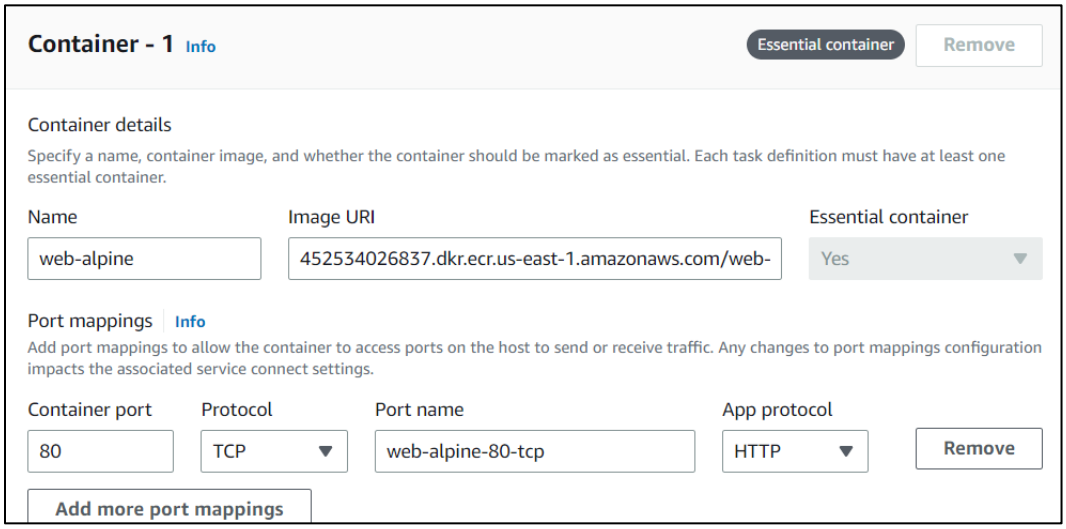
The **task definition** is the configuration of all the properties and settings of the task.



The name of the Docker image stored in the Elastic Container Registry will be required in the next step. The URI 452534026837.dkr.ecr.us-east-1.amazonaws.com/web-alpine:latest can be obtained from the ECR repository.

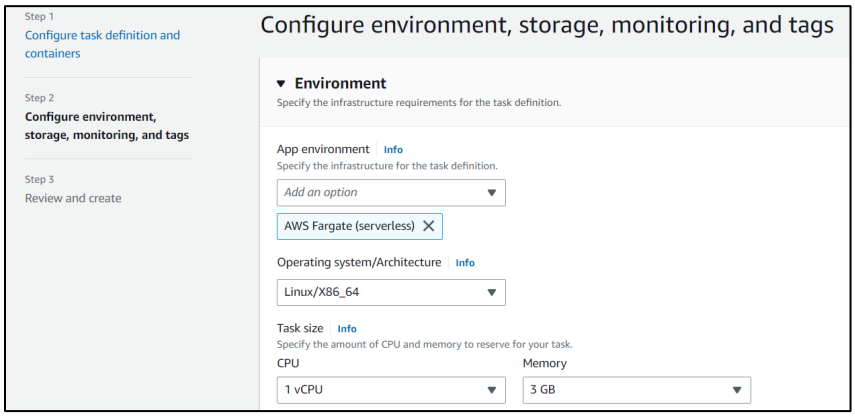


The URI name is supplied as a pointer to the location of the container image. The protocol HTTP with its mapped port 80 is also included here.

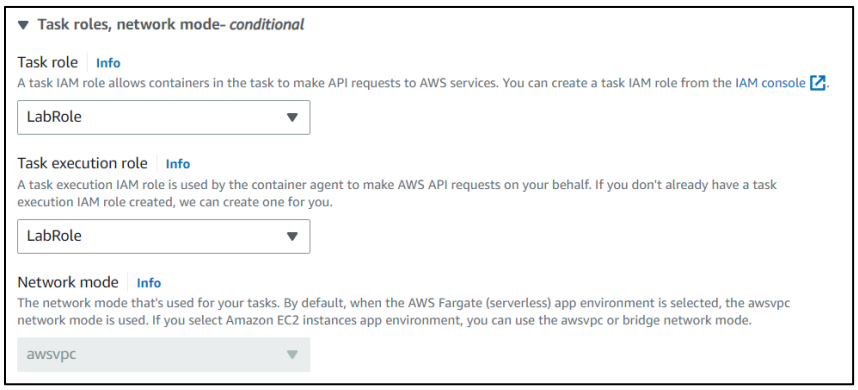


The next step configures the settings for the **environment**. This is where the compute resources to run the application are requested. In this case, the application will be running on a platform managed by AWS Fargate. This is an AWS service defined as “serverless” which is a bit of a misnomer because there are actual servers hosting the application; however, the user does not manage those servers but rather the AWS Fargate service. This make the details of the supporting infrastructure to be opaque to the customer.

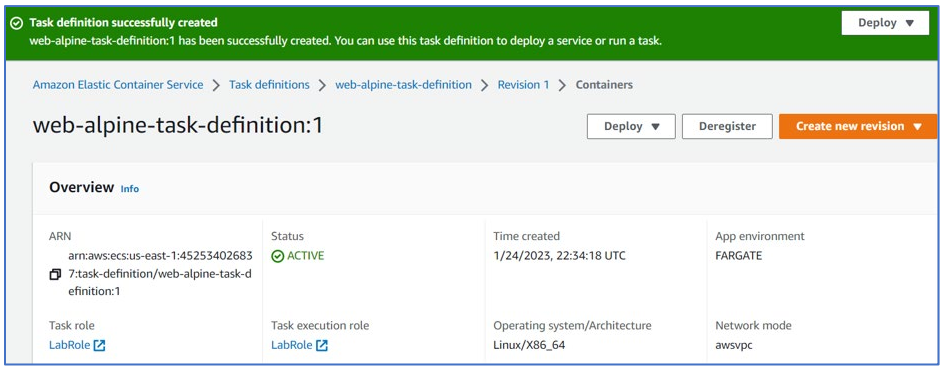
The task definition includes the Unique Resource Identifier, the transport protocol, the application port number, the host operating system, and the amount of resources such as CPU and memory.



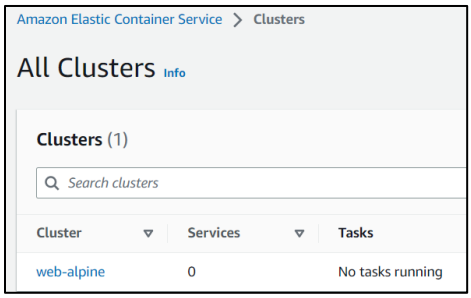
Normally, an IAM Role is associated to assign permissions to access resources. However, since this example is done in AWS Academy, the LabRole is the only option available.



Finally, the task definition is complete.

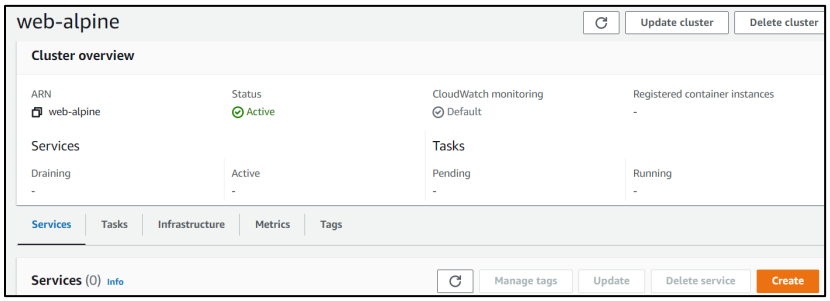


The next step is the configuration of either a **task** or a **service**.

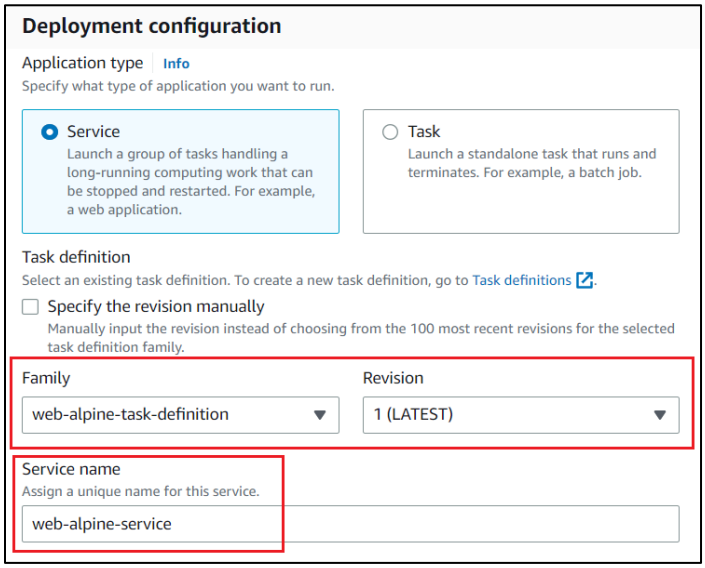


Fundamentally, a service is an automatic manager of the applications. A **service** is defined as an abstraction that specifies how a task or a group of tasks will be running. The configuration states the **desired state** of the service. This includes the number of tasks, the placement specifications for the containers, and the access rules if there is a load balancer. A **scheduler** process works continuously to maintain the desired state of the service. If a task failed, the scheduler would notice it and initiate its replacement. Also, the scheduler can scale up the number of tasks if the load of the service increases.

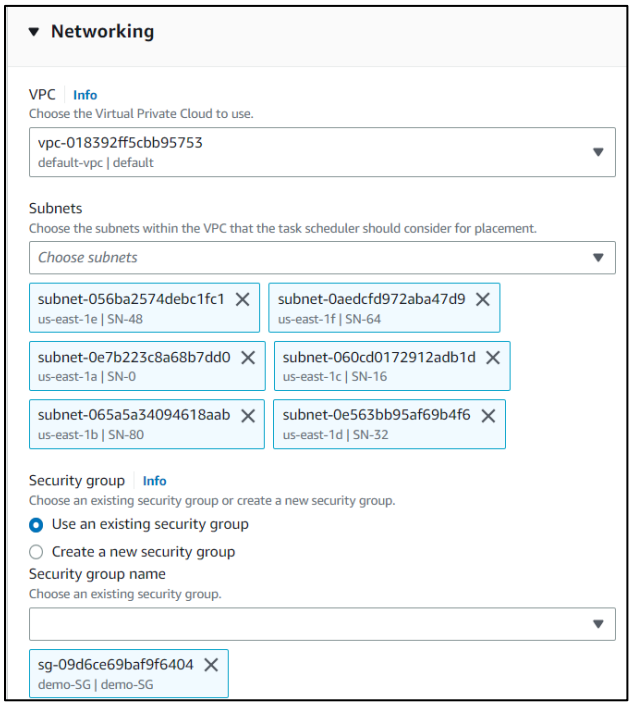
- Create a service under the cluster's services tab.



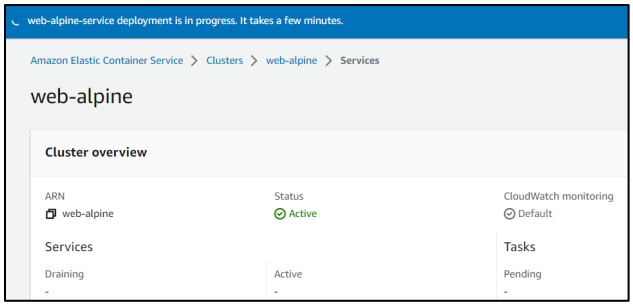
The deployment configuration differentiates two application types between service and task. In this example, the container is a website that will be running continuously. This type of application is a service. On the other hand, a task is a job that just run, does something and then ends.



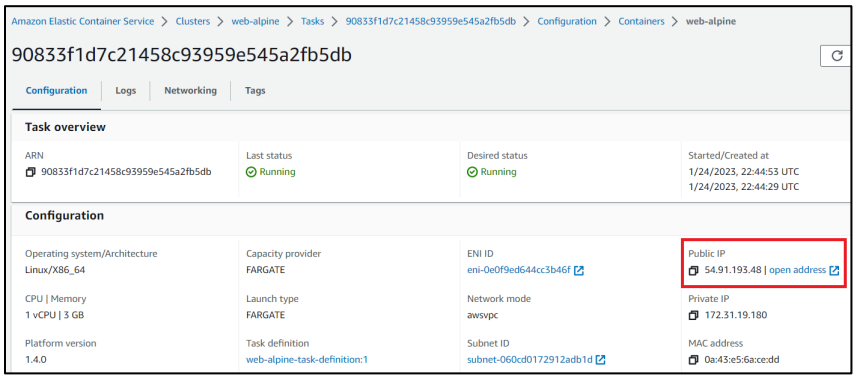
Even though the environment is managed by AWS Fargate, some details regarding the cloud infrastructure need to be supplied by the customer. For example, the account might have several VPCs, so one of them must be selected. Since the task scheduler is continuously invigilating the desired state, it also needs to know the subnets where resources can be provisioned.



The service with the given name web-alpine starts its deployment at this point.



A public IPv4 address is assigned to the service once the deployment is complete.



The web application is reachable now. This webserver is running on a Docker container on an infrastructure maintained by AWS Fargate as defined in the Elastic Container Service ECS.

Thus, deploying containers on AWS Elastic Container Service requires the conformation of several functions and components:

- The **infrastructure** supported by AWS Fargate in this demonstration.
- A **cluster** which is a group of tasks and services that run on the infrastructure.
- A **task definition** which is the list of parameters required to run containers.
- A **running task** which is the template used to format the tasks and services
- A **service** which is the executioner of the task definition.

Chapter Conclusion

This chapter started with the exploration of Docker as a hosting layer for containerized applications. The administration of images and containers was explored by using pre-existing images in the public Docker repository. Once that the fundamentals of Docker container were outlined, the next step was the creation of basic images by writing Dockerfiles. Cloud services support container services. In the case of AWS specifically, Elastic Container Registry is used to store images. These images are retrieved from the Elastic Container Service to deploy containers. The supporting infrastructure can be customer managed or cloud provider managed as in the case of Amazon Fargate.

The container is a critical component of modern applications such as microservices where every functional unit resides in a container. Because containers are devoid of overhead, they have small footprint and they are very efficient. Additionally, the container nature of being based off images enable continuous development and deployment. Whenever an application is improved, a newer image version is created, and the latest version of the application is quickly deployed on containers that replace the previous ones.

Chapter Coursework

Create a webserver application that includes the HTML index and other service files. Create an image based on this application in a cloud9 environment. Then register the image in AWS ECR. Finally, deploy using AWS ECS a highly available infrastructure fronted by an application load balancer.

References

[1] Docker containers. 2023. [Online] Available: https://www.docker.com/

[2] Containers at Google. A better way to develop and deploy applications. 2020. [Online] Available: https://cloud.google.com/containers

[3] Install Docker Engine on Ubuntu. 2020. [Online] Available: https://docs.docker.com/engine/install/ubuntu/

[4] Ubuntu Manual. 2020. [Online] Available: http://manpages.ubuntu.com/manpages/cosmic/man1/apt-transport-https.1.html

[5] Ubuntu Server Guide. 2020. [Online] Available: https://guide.ubuntu-fr.org/

[6] How to Install and Use Docker Compose on Ubuntu 20.04. Erika Heidi. (2020). [Online] Available: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-20-04

[7] How To Deploy Your Webpage Using Alpine Image and Lighttpd Server. ShazForiot. 2020. [Online] Available: https://www.youtube.com/watch?v=Q4Vm5IQZ4Vo

[8] Amazon Elastic Container Service Documentation. 2023. [Online] Available: https://docs.aws.amazon.com/ecs/index.html

[9] What is AWS Fargate? [Online] Available: https://docs.aws.amazon.com/AmazonECS/latest/userguide/what-is-fargate.html

[10] Dockerfile request: 2023. [Online]: https://chat.openai.com/chat

Chapter 13

Security Principles

Description

Computer and network security is based on the principles of secrecy, data integrity and access authentication. Each one of these principles is implemented by methods and protocols. Secrecy is implemented using two methods of ciphering: symmetric and asymmetric. Data integrity is provided by fingerprints of the data created by hash algorithms. Secure authentication is critical to gain access to resources deployed in the clouds. This section focuses on the topic of security principals and its methods and protocols and how all of theses apply to the security of the cloud.

Learning Outcomes

- Explain the fundamental principles of computer and network security.
- Analyze the methods and protocols of secure access to cloud virtual machines.
- Comprehend the system of authentication to access cloud resources.
- Describe the use of security certificates to validate webservers legitimacy.

Main concepts

- Computer and network security fundamentals.
- Security principals.
- Asymmetric key ciphering.
- Symmetric key ciphering.
- Authentication.
- Secure Shell Protocol.
- Security Certificates.
- Secure HTTP protocol HTTPS.

Learning Activities

- Authentication Access to EC2 instances.
- The role of the private key file in the authentication process.
- The functioning of the Secure Shell (SSH) protocol.
- Configuration of Public Key Infrastructure with Certificate Authority.

Security Principles

Computer and network security are based on a set of concepts, rules and methods with the overall goal of protecting data whether it is stationed on a server or travelling across networks. In general terms, robust security must provide the following:

- **Confidentiality** which is implemented with **ciphering** methods.
- **Integrity** which is implemented with **checksums** or **hashes**.
- **Authenticity** which is implemented with **digital signatures**.
- **Authorization** which is implemented with identity **authentication** methods.

Confidentiality

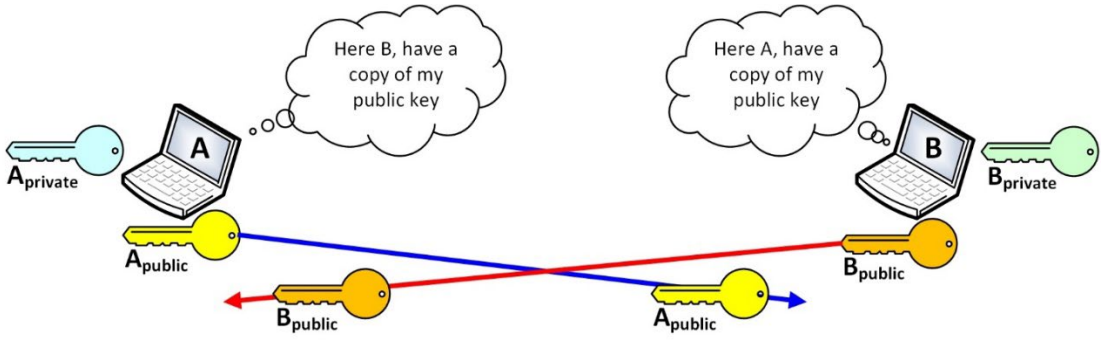
Data confidentiality means secrecy of the data. This is obtained by ciphering the clear text data using methods and algorithms. There are two major ways to cipher data: **asymmetric** key algorithms and **symmetric** key algorithms.

Asymmetric key ciphering

Ciphering with asymmetric keys is the method by which each end-node participating in a communication process owns a pair of keys for the session. One of the keys of the pair is used to cipher clear text data while the other key is used to decipher the ciphered data.

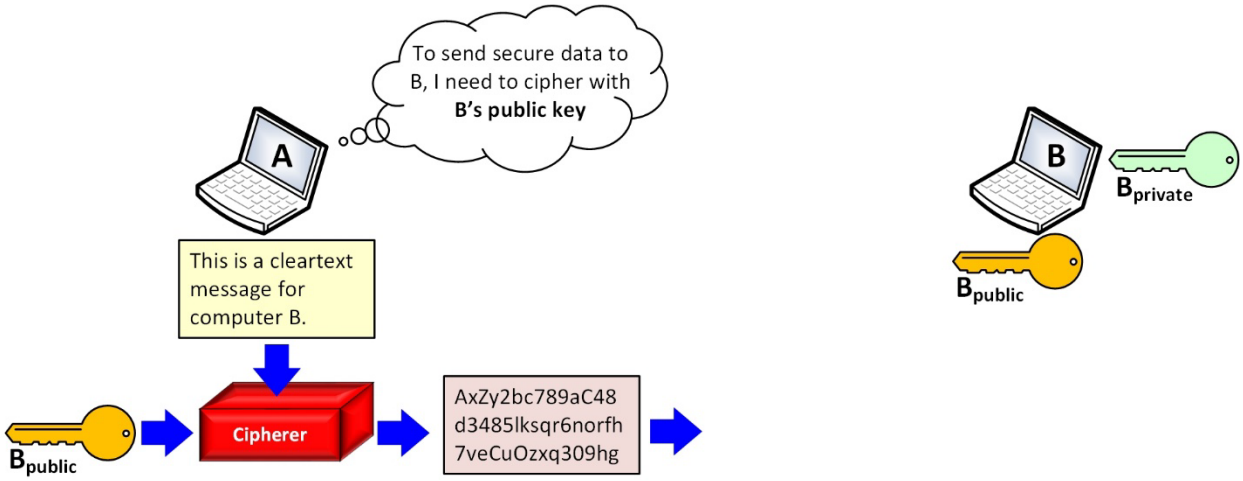
Each node creates its own pair of keys, one is public, the other remains private

One of the keys of the pair is provided to the other node hence such key is **public**. The other key of the pair never leave its owner; thus, it is a **private** key. The computers exchange their public keys only.



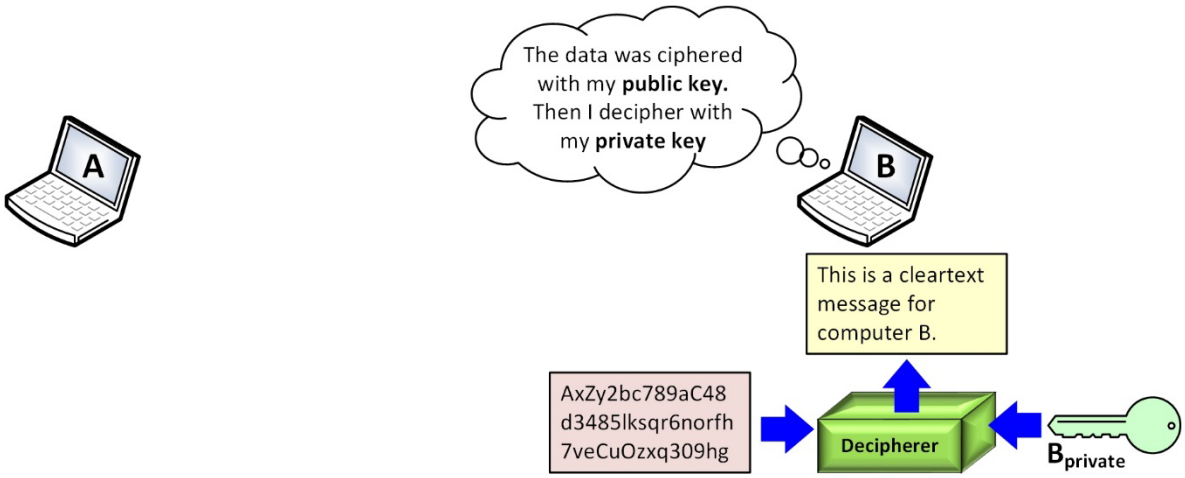
The nodes exchange their public keys

The data ciphered with one key of the pair, can only be deciphered with the other key of the pair. In the following example, computer A sends a ciphered message to computer B. To do so, A ciphers the message with the public key that B already advertised. The only computer that could decipher such cipher-message is computer B.



Computer A sending secure data to computer B ciphered with the public key of B

The clear text data and the public key are supplied to the inputs of a cipherer system (for example Advanced Encryption Standard AES). The output of the cipher block is the cipher-data that is delivered to computer B. When this node receives the data from A, it uses a reversing mechanism.



Computer B deciphers the secure data with its private key

Computers do not have only one key pair, but rather they may have many key pairs for different processes and protocols. The keys are generated according to a complex mathematic procedure, the most common of which is Rivest, Shamir and Adleman (RSA) after the three computer scientists [1] who invented the algorithm in 1977.

The RSA system is based on the principle that for any public key (e) there is a corresponding private key (d) in a modulo number space n. However, knowing one the keys do not offer any clue of the value of the other. In other words, one key can not be guessed by having the other key. That is why, one key is made public while the other is kept private.

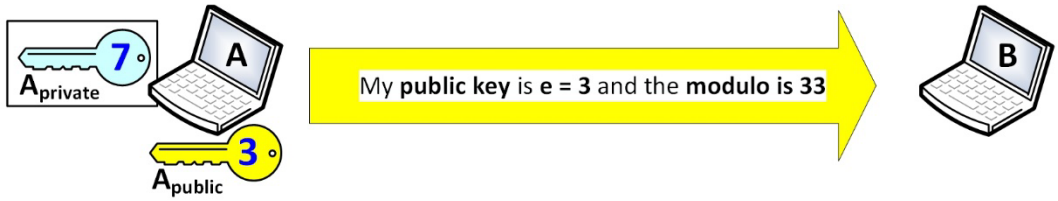
Another fundamental principle of public-private key security is that information ciphered with either one of the keys can only be deciphered with the other key of the pair. The RSA system has two (2) operations to implement **data secrecy**:

1) Cipher with $c = m^e \ mod(n)$
2) Decipher with $m = c^d \ mod \ (n)$

Where:

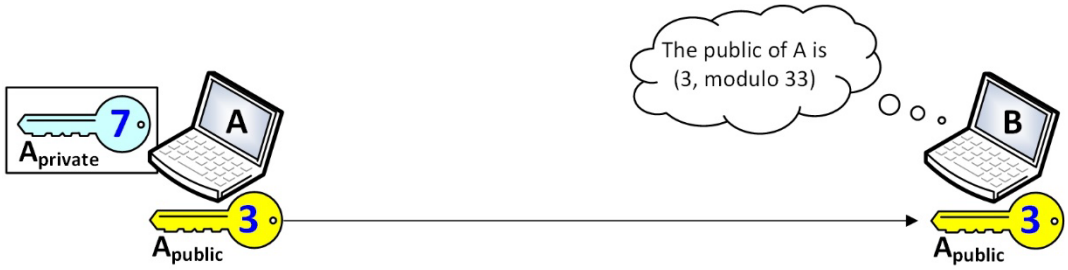
- The letter **c** stands for **cipher-text**.
- The letter **m** is **cleartext message**.
- The letter **e** is for the **public key**.
- The letter **d** is for the **private key**.
- **mod** is math **modulo operation**. (as in addition, subtraction, multiplication, division, modulo).
- The letter **n** is the **value** where the modulo will be calculated.

This is better explained with an example with two small value keys. Computer A has calculated its key pair and found a private key d = 7 and a public key e = 3. The modulo value is 33.



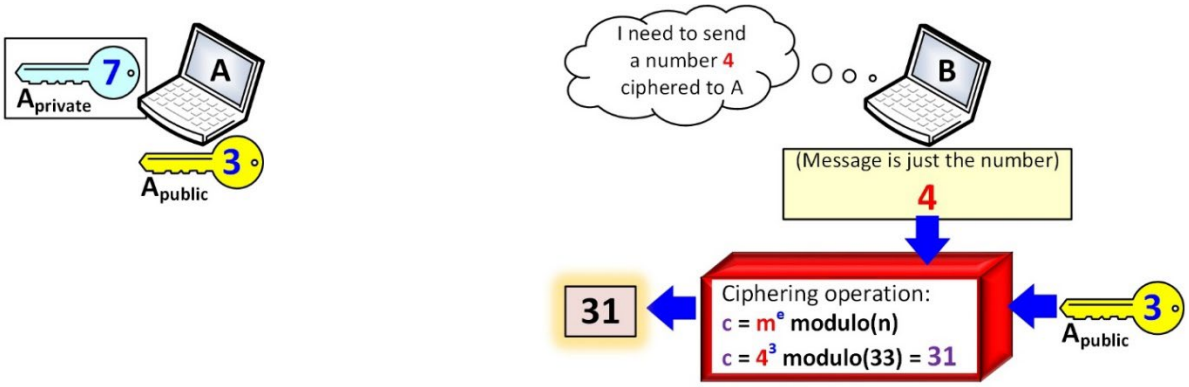
Computer A shares its public key (3,33) to computer B

Computer A sends, literally, its public key to B. This can be done for a one-time session or for a long living session. Computer B stores the public key of PC-A. This key will be used for all the secret conversations in the direction from B to A. Conversely, computer B must share its public key with A, so the conversation from the other direction is also ciphered. However, for the sake of simplicity, this explanation covers only the messages going from B to A.



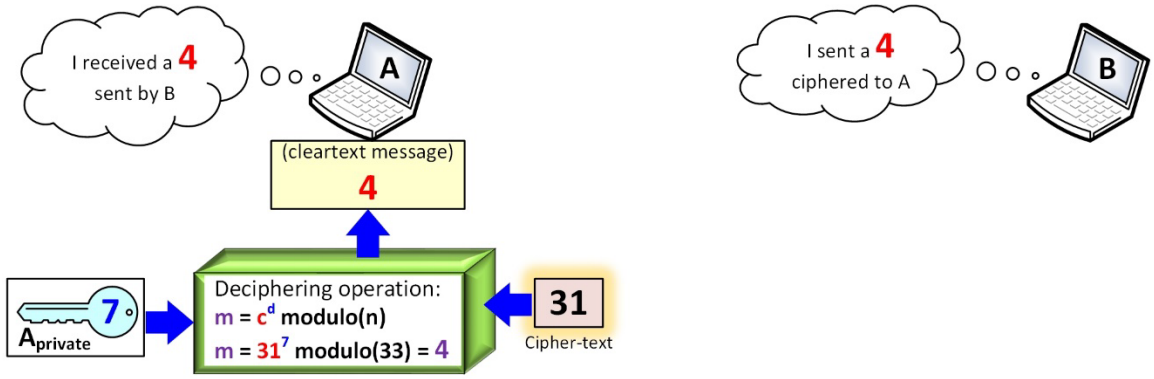
Computer B knows A's public key (3,33)

Let's assume that PC-B needs to send a message to A and the message is just the number 4. Computer B delivers the 4 to its cipher system. The public key (3) of A is also supplied to the cipher which performs the math operation $c = m^e \ mod(n)$ yielding $c = 4^3 \ mod(33) = 64 \ mod(33) = 31$.



Computer B ciphers the message value 4 and it obtains a cipher-text valued 31

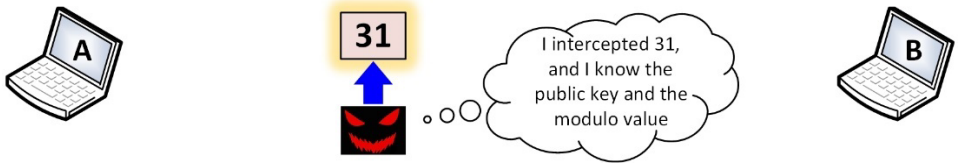
Computer B sends the ciphered message 31 to computer A which performs the reverse operation as illustrated in the figure below.



Computer A deciphers 31 and obtains the message 4

Since B ciphered with the public key of A, only A can decipher that because A is the only node in possession of the private key. Computer A performs the math operation $m = c^d \ mod(n)$ yielding $m = 31^7 \ mod(33) = 27{,}512{,}614{,}111 \ mod(33) = 4$ which is the original clear text message from B.

A valid question at this point is how solid this secrecy system of public and private keys might be. Would it not be easy to find the private key by having the public key and the modulo value? Let's try an exercise to demonstrate this point. A malicious third party has somehow got a hold of the ciphered message, the public key and the modulo.



A malicious party attempting to break the public-private key system

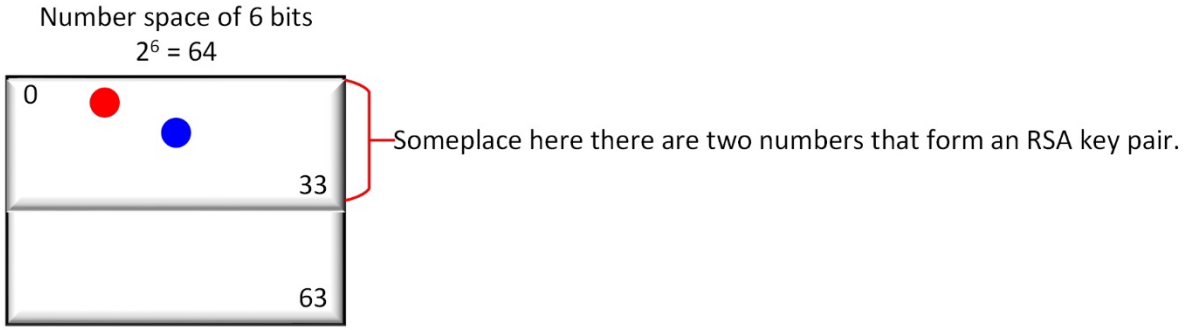
The RSA algorithm is public and well known. The malicious party has all the components of the system except the private key. One of the principles of the system is that the two keys reside in the number space of the modulo. Thus, somewhere between 1 and 33, the other key of the pair exists. Knowing this, the malicious party can start guessing a private key, number by number. For example:

Attempted (private key) value	Replace values in $c = m^e \ mod(n)$	Result
1	$c = 31^1 \ mod(33)$	31
2	$c = 31^2 \ mod(33)$	4
3	$c = 31^3 \ mod(33)$	25
4	$c = 31^4 \ mod(33)$	16
5	$c = 31^5 \ mod(33)$	1
6	$c = 31^6 \ mod(33)$	31
7	$c = 31^7 \ mod(33)$	4
8	$c = 31^8 \ mod(33)$	25
9	$c = 31^9 \ mod(33)$	16
10	$c = 31^{10} \ mod(33)$	1
11	$c = 31^{11} \ mod(33)$	31
12	$c = 31^{12} \ mod(33)$	4
13	$c = 31^{13} \ mod(33)$	25
14	$c = 31^{14} \ mod(33)$	16
Keep going until 33	$c = 31^{33} \ mod(33)$	25

Table of possible values of a deciphering operation with different private keys

The result seems inconclusive, even confusing and that is precisely a security concept, **confusion**. Cipher data should appear confusing to uninvited observers. The number 4 appears several times, but only one is correct, the one obtained with the private key valued 7. For the hacker to draw a conclusion, one sniffed message is not enough, but rather, the hacker would have to capture a large number of ciphered messages to be able to sort out the correct private key. This brings up another security concept, **opportunity**. If there is no chance of obtaining a larger volume of cipher data, breaking the system is less likely.

This example gives an idea of the power of public-private key system even if this was done with two small numbers inside a small number space. Notice, that the number 33 fits in a binary number space of at least 6 bits. In such small number range, 3 and 7 are the only numbers that work as a public-private key pair.



A number space of 6 binary bits contains the number 33 used as a modulo

Conceptually, that is a weakness of the system because it is known that the two numbers exist in such number space. In principle, it is a matter of trying all the possibilities, assuming that the data is available,

and testing until finding the right combinations. But this is an example on a small number space for educational purposes. What if the number space is made bigger, let's say 512 bits. How many numbers fit in a number space of 512 binary bits?

$$total\ numbers\ in\ 512\ bit\ space = 2^{512} = 1.340780792994259709957402499820 6 \times 10^{154}$$

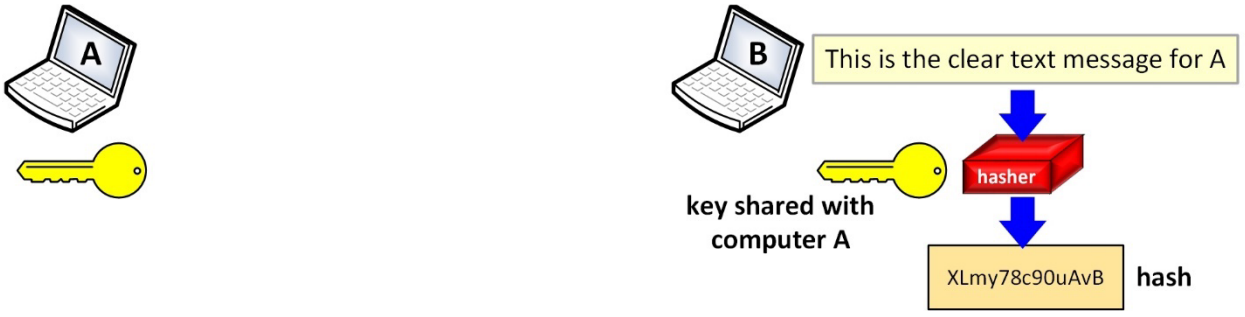
That is a number with 154 decimal positions which is an unfathomable quantity. In such vast number space, there are billions of possible key-pair combinations, so if one public key is known, where in the space of 1.34 x 10^{154} is the private key of the pair? Even for a powerful computer, it would take a long time to test all the combinations. The difficulty of guessing the other key of the pair is what makes the RSA system so strong. However, key spaces of 512 bits are considered not strong enough because supercomputers can run over all the cases in a feasible time. For that reason, the key length has been increased to 1024 bits, then 2048 bits and even 4096 yielding these gigantic number spaces:

- $total\ numbers\ in\ 1024\ bits\ space = 2^{1024} = 1.797693134862315907729305190789 \times 10^{308}$
- $total\ numbers\ in\ 2048\ bits\ space = 2^{2048} = 3.231700607131100730071487668867 \times 10^{616}$
- $total\ numbers\ in\ 4096\ bits\ space = 2^{4096} = 1.044388881413152506691752710716 6 \times 10^{1233}$

The larger the key space, the strongest the security. It is extremely difficult to find the other key of the pair even though the public key, the modulo and the algorithm are known. This is why the RSA public-private key system works.

Integrity

Integrity is the security concept that, when implemented, provides confidence that the data is accurate. In principle, data could be tampered by a malicious party or the data could contain errors. To counter these risks, methods that provide data integrity are needed. The fundamental method of data integrity is the **hash** which is a **checksum** or **digest** derived from the original clear text.



Computer B calculates a hash out of the clear text message

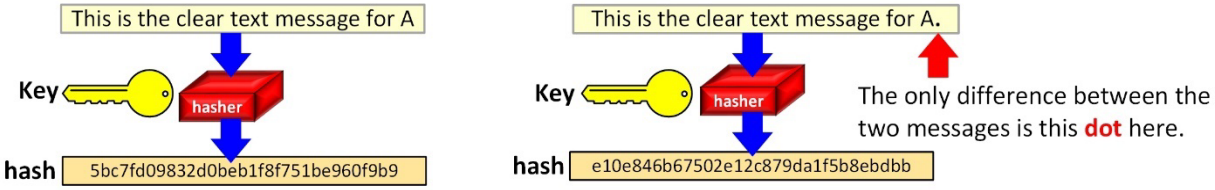
The clear data is supplied to a hashing method (for example **S**ecure **H**ashing **A**lgorithm SHA) together with an input key. The same algorithm and key must be available at the receiving end. The hasher "digests" the message and the key to produce a string of data with these properties.

- Hashes are not reversible. That means there is nothing to decipher.
- Hashes have fixed length regardless of the size of the data being hashed.
- Hashes are just digest fingerprints of the data.
- Hashes prove integrity of the data.

It is worth to note that hashing does not always require the use of a key. Data can be hashed without it. However, when a key is used in the hashing process, it introduces a secret element that reinforces the process of integrity checking. In this case, the key receives the name of “salt”. The only way that the same hash can be obtained is that the key is available.

Hash methods always yield the same hash length regardless of the size of the data being a few bytes or many bytes. For example, SHA-512 always produces hashes of 512 bits long while the older, and weaker, MD-5 always produces hashes of 128 bits long.

The following diagram shows the result of two messages done with an online hash calculator [1]. The hash algorithm chosen is MD-5 and there is an embedded fixed key (not shown in the calculator by default. Notice that the only difference between the two messages is a dot at the end of the right-side message.



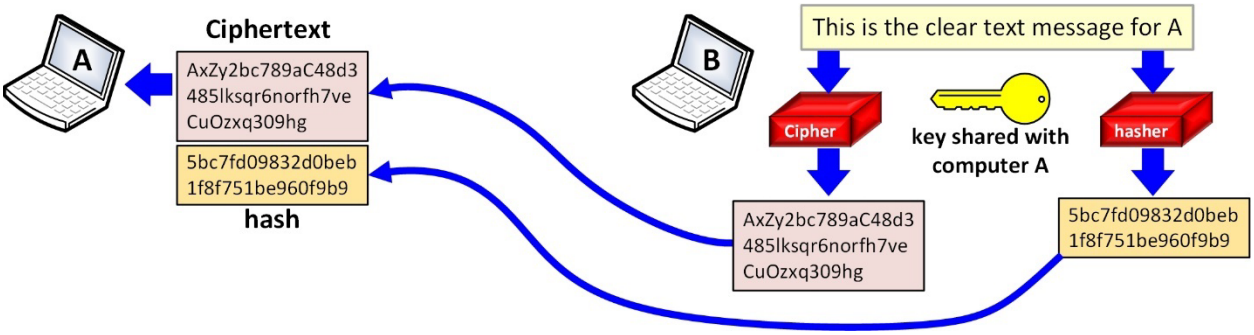
Hash results from similar messages

A desired property of a good hash algorithm is that it spreads and mixes the data so there are no discernible patterns. Even though the difference between the two messages above is a mere dot, the resulting hashes are so completely different as it can be seen by comparing them below.

5	b	c	7	f	d	0	9	8	3	2	d	0	b	e	b	1	f	8	f	7	5	1	b	e	9	6	0	f	9	b	9
e	1	0	e	8	4	6	b	6	7	5	0	2	e	1	2	c	8	7	9	d	a	1	f	5	b	8	e	b	d	b	b

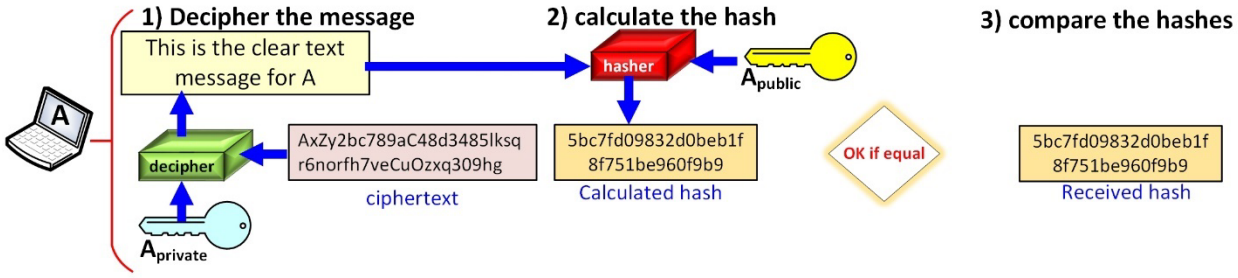
These two hashes have the same fixed length of 128 bits (32 hex characters times 4). Longer hash algorithms are stronger than shorter hash algorithms. For instance, SHA-512 is stronger than SHA-384.

If computer B sends a ciphertext to computer A plus the hash of the clear-text to validate its integrity, it is implicit that computer A is using the same algorithm, the same key and the same data to obtain the same hash result. Let’s assume that A’s public key is used for both ciphering and hashing.



Computer B sending a ciphertext and the hash of the cleartext

In the receiving side, computer A obtains the cleartext by deciphering. Afterwards, it runs the same process than computer B did to calculate the hash. Then it compares the two hashes and if they have the same value, then it concludes that the data has not been modified.



Computer A validates the hashes

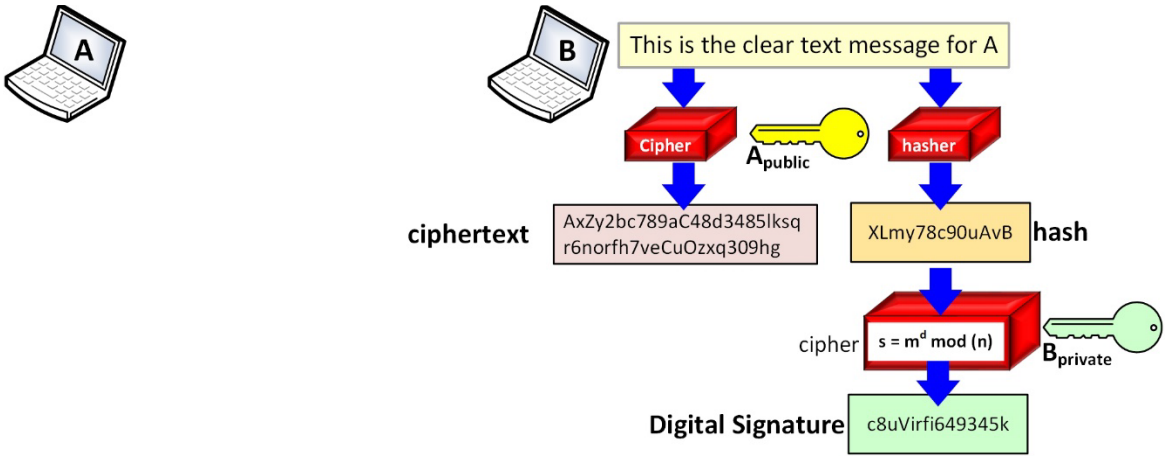
The procedure portrayed above provides secrecy and integrity however it has a weakness. If a malicious party obtains the hash in transit, it could try an attack. In principle, the hacker might have possession of the public key of A and since the hash algorithm is well-known, then there are too many pieces available to attempt a brute force attack. Even though for data hashed with a strong algorithm, it will take an unrealistic number of tries of combinations of data to arrive to the same hash. The next section expands on the security procedure to add authenticity.

Authenticity

Authenticity is the act of proving that the **data origin** is **legitimate**. The method used to provide authenticity is the **digital signature**. This is another application of the public-private key system. The sending party of the communication session accompanies the ciphered data with a string of data that is also ciphered but with the private key of the sender. The signature is created with the operation:

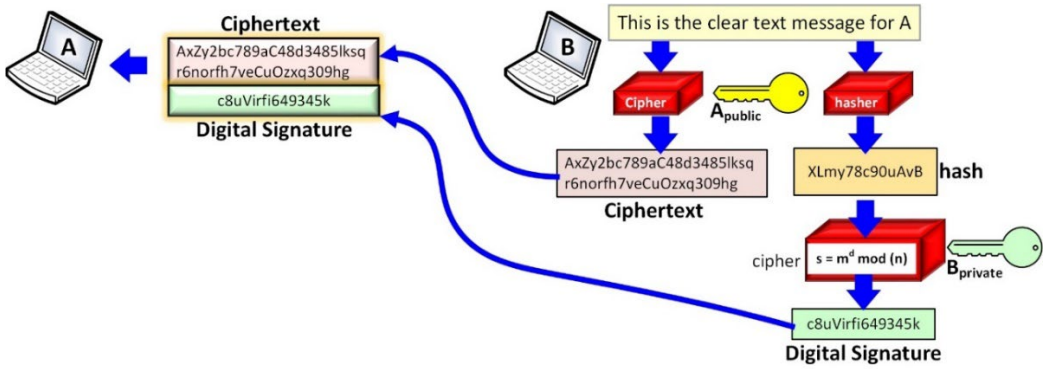
$$s = m^d \ mod \ n$$

Where s is the signature, m is a string of data, d is the private key of the sender (B in this case) and n is the modulo value. Let's start the explanation with an illustration.



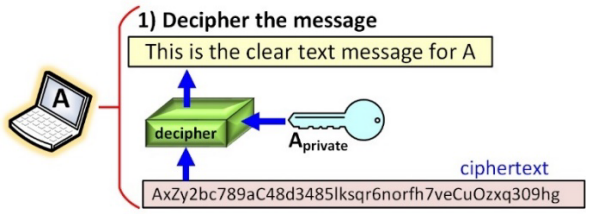
The process of signing the data digitally

In this example, computer B is sending a message to computer A. Therefore, it processes the message with the public key of A to obtain a ciphered message. Computer A will be able to decipher such ciphertext with its own private key. The difference now is that the same message is also sent to a **hash** processor. The result of the hash is a **checksum value** of the message that validates its **integrity**. If even one bit of the message were to change for any reason, the hash will not match ay the destination, thus declaring the message invalid. The checksum is passed thru another module that ciphers it with the private key of the sender (B). That might sound odd because anyone with the public key of B will be able to open it, but what this process is meant to prove is **authenticity** or **legitimacy** of origin. If anyone can decipher the hash with the public key of B, then it only proves that B sent the message for only B could have **signed** it with its private key. Once that the process is complete, B sends the ciphertext and the **digital signature** to A.



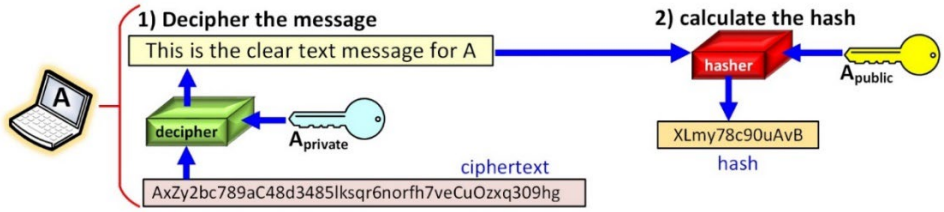
Computer B sends the data ciphered and the digital signature to computer A

Computer A receives all these data and it proceeds first to decipher the cipher-text using its private key since the data was created by B using A's public key. The result is the clear text message.



Computer A deciphers the cipher-message

Once that the cleartext is obtained, computer A needs to **validate** that the message is **legit** and **correct**. Hence, it proceeds to calculate a hash of the message. It feeds the hasher protocol with the clear-text message and its own public key (A) to create a digest value of the message. This hash should be identical to the hash previously calculated by computer B because it is the same message and the same key.

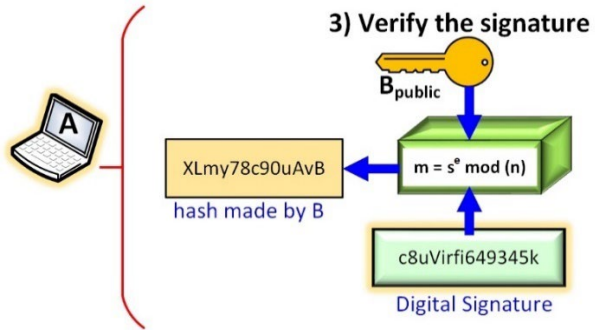


Computer A calculates a hash of the clear text message

Now, computer A needs to compare the hashes so it deciphers the digital signature of the hash created by computer B. Since this computer signed the hash with its private key, computer A can open it with the public key of B. This proves that the signature was made by computer B because only the public key of B can be used to decipher something ciphered with the private key of B. This proves **legitimacy** that B signed the hash. The operation to verify the signature is:

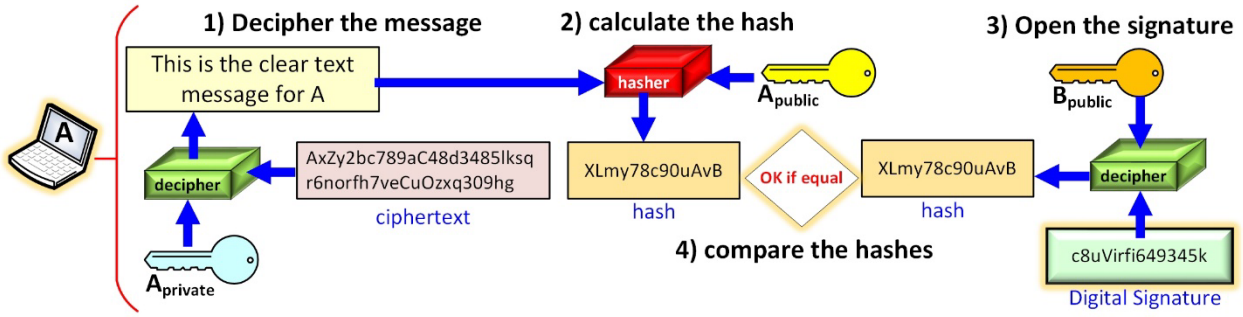
$$m = s^e \ mod \ n$$

Where m is the deciphered original data, s is the digital signature, e is the public key of the sender (B in this case) and n is the value of the modulo. The process is shown in the figure below:



Computer A verifies the digital signature of the hash

Now, it remains one more step of the process. Computer A needs to verify that the message was not altered in transit so it compares the hash obtained from the deciphered message with the hash that came ciphered inside the digital signature. If the hashes are equal that means that the message is correct, this proves **integrity**. But if the message differs, by even one bit, the message must be discarded as it was probably tampered or maybe it has an error.



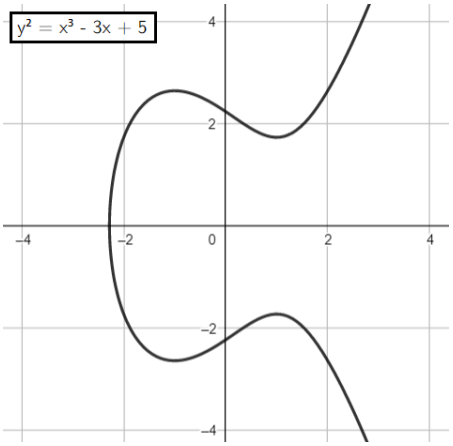
Computer A deciphers the digital signature of the hash

This is a robust security system that ciphers the messages, validates the authorship of the message and that the message is correct. In short, the system implements the **security principles** of **secrecy**, **authenticity** and **integrity**.

Elliptic Curve Cryptography (EC)

The confidence on the RSA security system rests chiefly on the key size. As computer power has continued to increase, the potential for breaking the RSA system has also increased. The response to this challenge has been to keep increasing the key size. But this is a problem for low power equipment such as mobile devices since the processing of the keys consume resources.

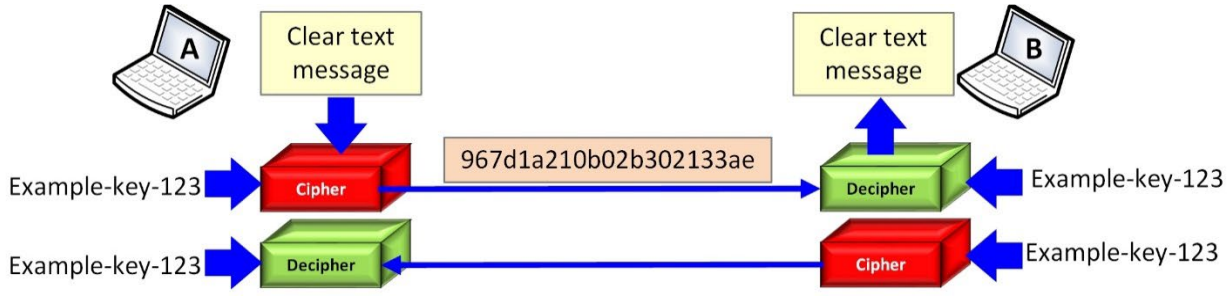
Elliptic curve cryptography is an alternative to RSA that can offer the same or greater level of protection but with smaller size keys than RSA. This system is based on the calculation of the public and private key using graph mathematics over elliptic curves. The computer choose one out of the many elliptic curve equations and then it starts a series of iterations easy to calculate (for a computer, not for a human) but extremely hard to reverse. At the moment of this writing, elliptic curve algorithm are being adopted at a fast pace by most security protocols and it is also the algorithm being used in blockchain technologies.



Sample elliptic curve

Symmetric key ciphering

Symmetric key systems are based on the principle that the participating nodes **share the same key** to cipher the communications. In contrast, asymmetric systems require four different keys for every pair of communicating nodes. In fact, public-private systems are computing intensive as they consume resources to do all the calculations, keep track of the keys, and the ciphering and deciphering of the data. Symmetric key system require less resources than asymmetric systems, so that makes them attractive to use. The same key is supplied to the cipher and decipher mechanisms.



The symmetric key systems use the same key to cipher and to decipher

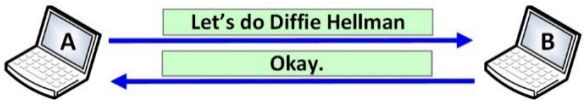
An inherent problem with symmetric key systems is the distribution of the key. For two nodes A and B to cipher their communications, the same key must be installed in both machines but doing that manually is out of the question. However, the key can be distributed in a dynamic way. For example, the key could be exchanged ciphered using a public-private system. Once that the key is in place, the communications session switches to the symmetric key system which is less demanding for the computers involved.

There is another way to use symmetric keys which does not distribute the key at all; instead, both communication partners calculate the same **session key**. Such a system is explained next.

Diffie-Hellman (DH)

Diffie Hellman (DH) is a process to generate a **secret session key** via the exchange of messages over an unsecured network between two computers. This brilliant system allows the two communicating parties to calculate the same session key that will be used for both ciphering and deciphering. Diffie Hellman is not a ciphering algorithm, but a process to calculate the symmetric key in a dynamic exchange of data. Furthermore, the key resulting from the DH process is passed onto a cipher system (such as Advanced Encryption System AES, for example).

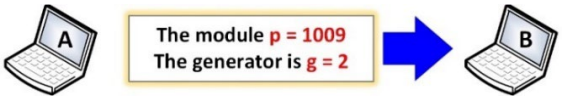
The Diffie-Hellman process is found at the start of a secure protocol transaction (for example, the SSH protocol uses DH). One of the nodes tells the other to start a Diffie Hellman process.



Computer A asks computer B to initiate a Diffie Hellman process

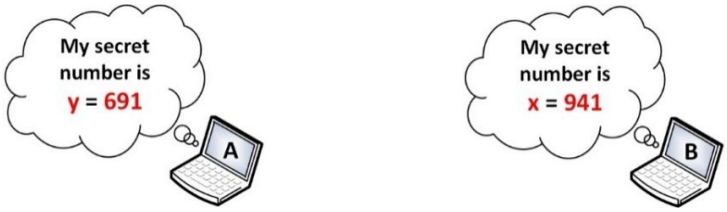
Note: In the following example, small numbers are used to make the explanation possible, but in real life, the numbers used as extremely big.

Computer A proposes to B, two components to be used in the operation. The **modulo number p** where the calculation will be done and the **generator g base number**. These values are exchanged in clear text format so an observer of the traffic can see the numbers in the message.



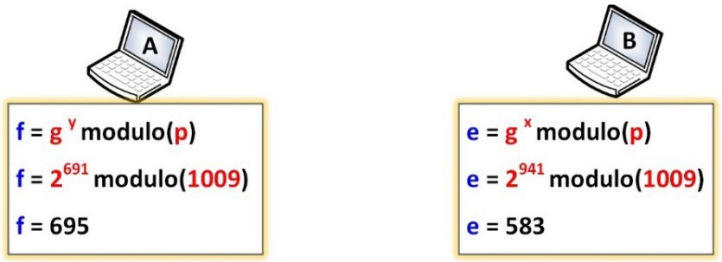
Computer A sends the p and g numbers to B

Both computers find now **large prime numbers** within the large modulo number space. These numbers never leave the respective computers because they are the **secret component** of the operation. (Again, in reality these numbers should be extremely big prime numbers).



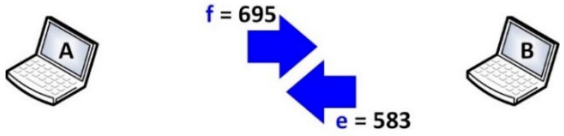
Each computer finds its own secret prime number

The process continues with each computer doing a calculation with the equation $g^{secret} modulo(n)$. Notice that the equation, the generator and the modulo value are public, the only hidden components are the secret values on each side of the operation.



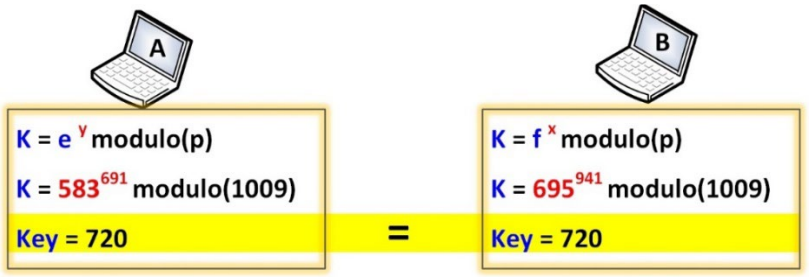
Each computer performs the same operation

Now, the two computers exchange the values found in the previous step and again all of this happens in a clear-text exchange of data between A and B.



The two computers exchange their $g^{secret} modulo(n)$ values

Finally, the two computers do a calculation involving the exchanged values and the secret components. Amazingly, the result is the same key value (K = 720) on each side. The key itself is never exchanged and it only last for one communication session. Every new session involves a new key calculation. Diffie Hellman only calculates keys, it is not a cipherer. Once that the key is calculated, it is supplied to a cipher system block, AES for example, to start the ciphering of data.

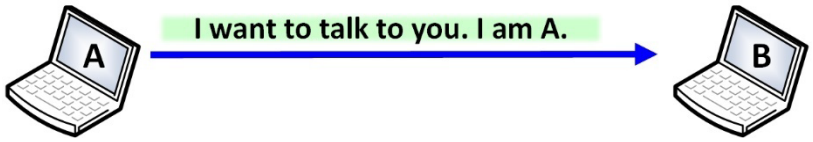


The computers calculate the secret session key

Diffie-Hellman is a remarkable invention dating from 1976 [2] and it is used in secure protocols such as SSH, IPSec, and SSL/TLS. Even though the message exchange is done in the open, it is virtually impossible to guess the secret component and the session key if the number space is very large (2048, 4096 bits). Since the ciphering and deciphering is done with the same key, the processing cost is lower than using the public-private RSA system. Another advantage is that the key is only used for one session, so there is nothing to be stored that can be potentially stolen. Moreover, elliptic curve cryptography has been incorporated into the DH process to create the components of the operation providing an even higher level of security.

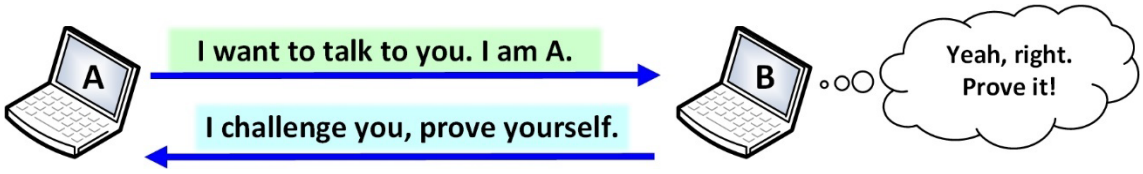
Authentication

The process of authentication consists of proving the legitimacy of the client trying to access a resource in another computer. Only an authenticated client can gain access to a remote computer. The following illustrated transactions represent the general philosophy behind the authentication process when the user of computer A tries to access the remote computer B.



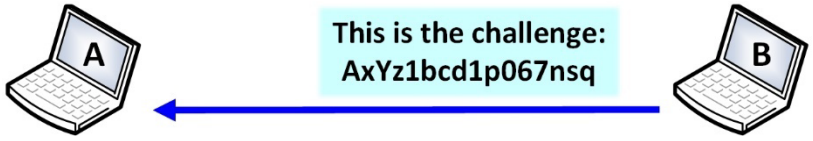
Beginning of the Access Process

Generically, A requests to access B; however, B can not just allow that. First, it will make A to prove its identity. It does so, by supplying a challenge. The access will be allowed only if A passes the challenge.



Computer B challenges A to prove its identity

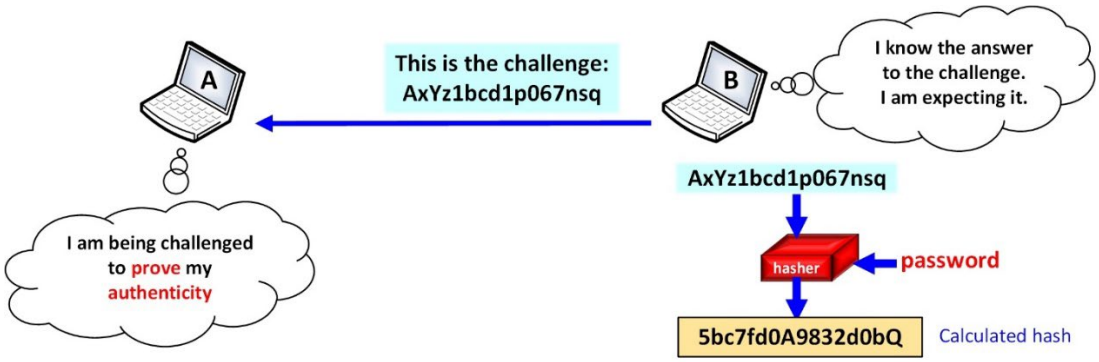
In this example, a string of data is delivered to A. In turn, A has to reply with an answer that only A could possibly create. That means, that A and B must have some secret in common.



Challenge data is delivered to A

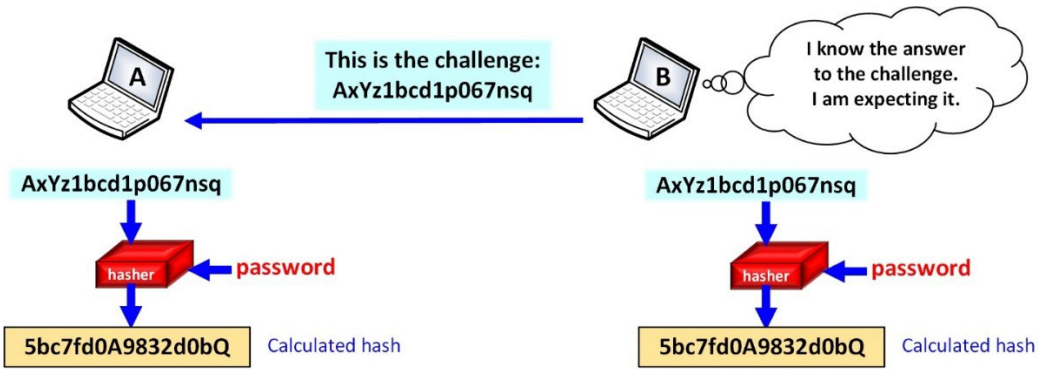
The secret component in this example is a password hence this is a password authentication method. Both A and B share the password. This is a common method present in many communications system. The idea is that B has a list or a database with users and corresponding passwords. When A says “I am A” is not enough, it has to proves its authenticity. That is why B challenges A with a string of data. The answer coming back from A must be something that only A can create. So, indirectly, this proves authenticity or identity.

Computer B sends the challenge data while locally it also creates a hash of it with the password of A. This is the answer that is expected from A. If the answer does not match, then B can not be allowed in. However, if the answer is exactly the same that B has calculated then that indirectly proves that A is A because the only way that the answer can be the same is that it was created with the same password.



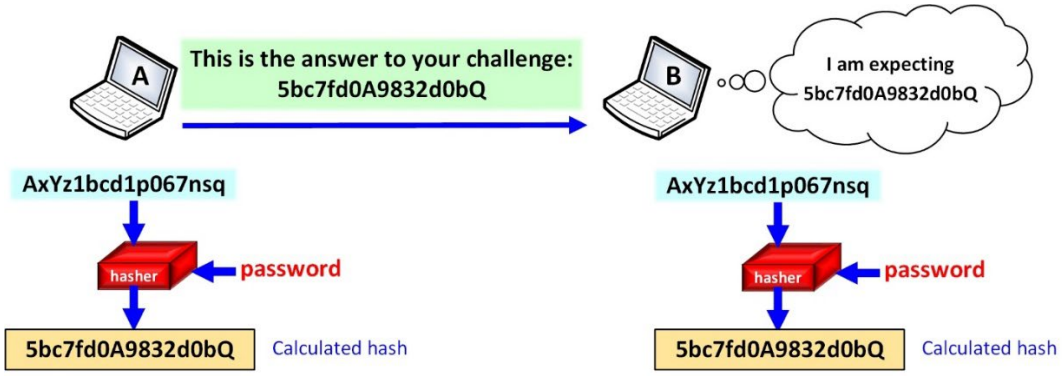
Computer B expects certain answer from A

When A receives the challenge, it does the same process than B has done. It creates a hash of the challenge data with its password.



Computer A creates a hash with the challenge data and its own password

Computer A sends the hash of the challenge data to B. The answer must match to the one being expected.



Computer A replies back to computer B

Now, this procedure has a weakness. Let's assume than an evil hacker has been sniffing the conversation. Consequently, it has obtained the challenge in the way from B to A and the answer to the challenge in the way back from A to B.

The hacker saw the challenge and the answer

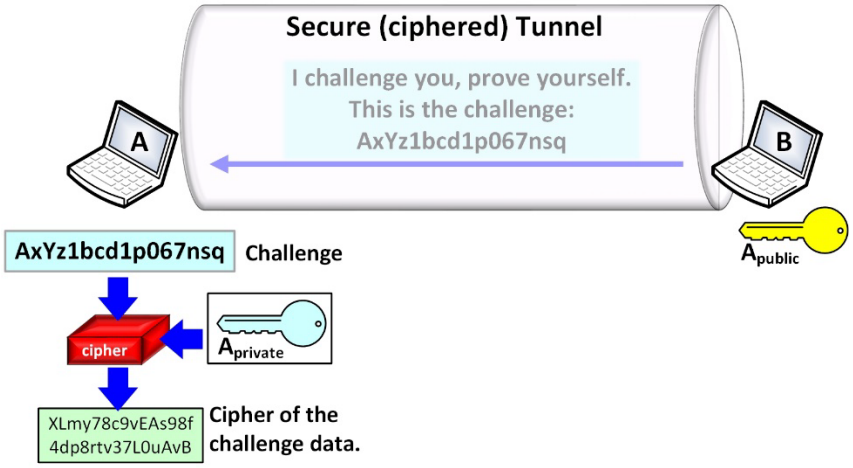
Thus, the hacker has two pieces of information out of three. The only piece missing piece is the password. The hacker can now attempt a brute force dictionary attack. This consists of testing the challenge through the same hash algorithm with the passwords supplied by a huge dictionary with the most commonly used passwords. If that does not give a matching answer, then it tries with all kinds of combinations until a match is found. If the password is weak, it can be obtained and that is obviously bad.

The hacker performs a dictionary attack

Consequently, the transactions can not be allowed to happen in the open. But what if the authentication conversation were conducted within a ciphered tunnel? In such case, the hacker could not get the challenge and the reply. The problem is how to create the tunnel in the first place. There are different ways to do this, but Diffie Hellman is the most common. Before the challenge messages are exchanged, a Diffie Hellman negotiation creates a session symmetric key that is used to cipher the conversation.

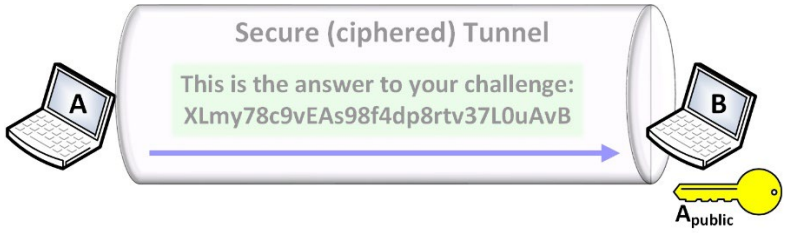
A ciphered tunnel hides the authentication exchange

If the client A does not pass the authentication challenge, the tunnel is broken down and the process is terminated by computer B. Furthermore, the authentication process can be made even better or stronger. Instead of using a password which is considered weak authentication, public-private keys are used.



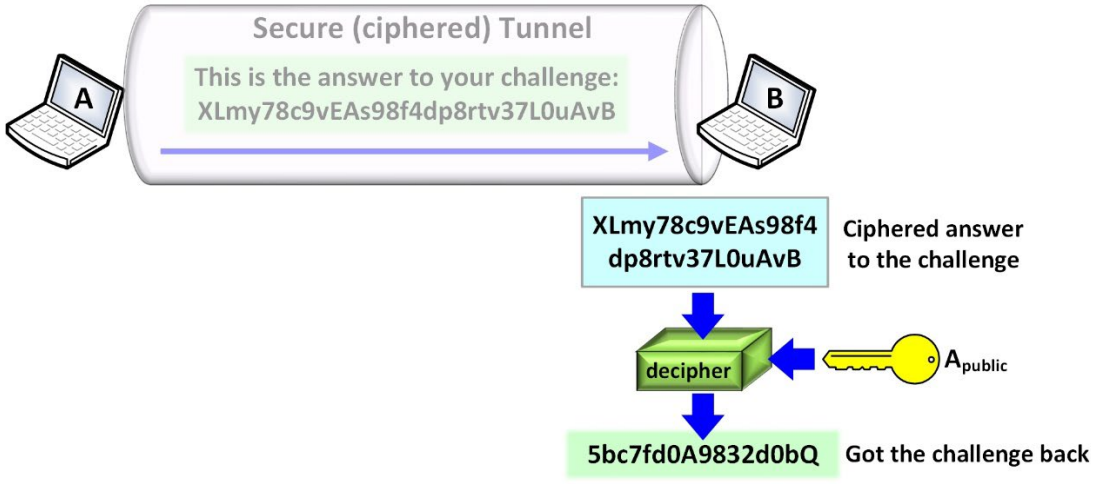
Computer A ciphers the challenge data with its private key

For this system to work, computer B must have A's public key. Computer A ciphers the challenge string of data with its own private key. Only the public key of A can decipher that.



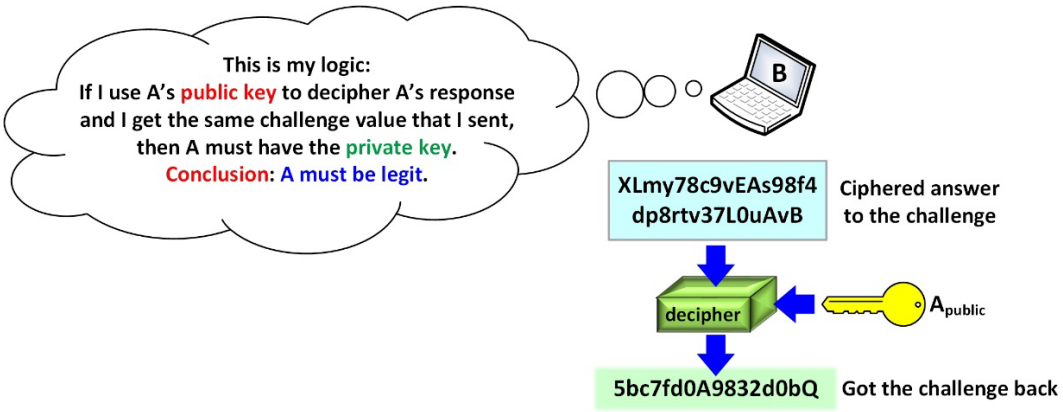
Computer A returns the challenge data ciphered with its private key

Computer B receives the ciphered challenge and it proceeds to decipher it with the public key of A.



Computer B obtains the challenge data back

If the resulting clear text is the same as the challenge, then that proves that it was ciphered with the private key of A and since the only computer that could have such key is A that proves that it is authentic.



Computer B authenticates computer A

Learning Activity

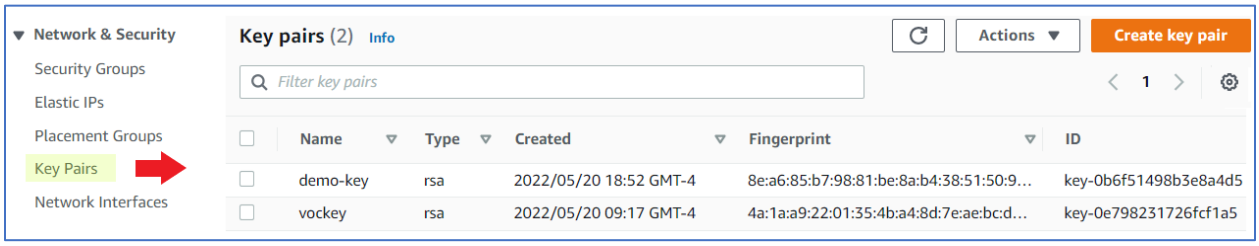
Observing the Security Principles when Accessing Cloud Resources

Secure Shell (SSH) Protocol

SSH is a protocol for securing communications between two ends over an unsecured network. It mode of operation is client-server where the end-node that initiates the SSH exchange is called the SSH client while the end-node that is the target of the session is the SSH server. SSH most common application is to secure remote login procedures. Particularly, it is the protocol used to access AWS EC2 instances in a secure manner. In this learning activity, this specific case is analyzed step by step to observe the application of network security concepts.

Understanding the Security Key Pair

The AWS service Elastic Computer Cloud has a section dedicated to network security where the key pairs are managed.

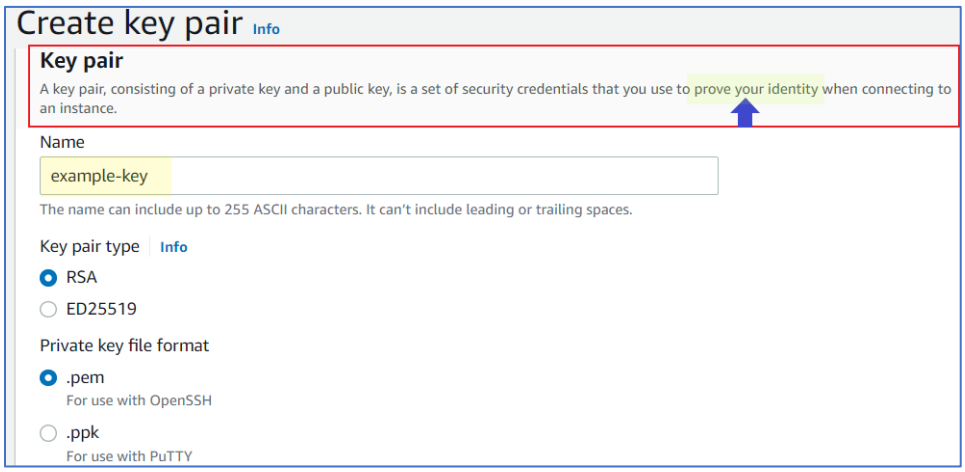


- What are these key pairs?
- What is their function?
- How are they used?

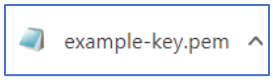
To answer these questions, let's start by creating a new key pair named test-key. There are two key **algorithms** available Rivest, Shamir, and Adleman (RSA) and Edwards-Curve Digital Signature Algorithm with the Elliptic Curve 25519 (ED25519) to create the key pair. RSA is an algorithm that creates public

private keys throughout a calculation over prime numbers. ED25519 also creates a key pair but by using elliptic curves. In this example, RSA is selected.

The key file has two possible **formats** available. Private Enhanced Mail (PEM) originally designed for email security and PuTTY Private Key (PPK) which is the format used by the tool PuTTY. In this example, the file format is left as PEM.



Once that the keys are created, the PEM file is automatically downloaded onto the customer's computer.



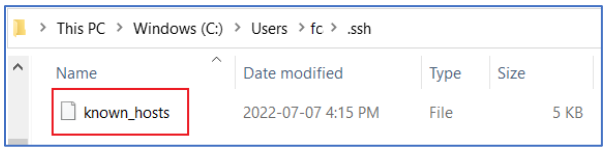

```
-----BEGIN RSA PRIVATE KEY-----
MIIEoQIBAAKCAQEA12DYD8UiZbnUXrfZfSxOofAyxvXiY42GkEw/WTUBBBi8YP9x
1s5RPRLuf3/eXOHHloE6jqVQJMXHXvPuF1mJht9JC+INg8+UqNMnn8kGd//N2pwd
wR2xqvvPZACY3n3Q/uC6BqmjNZeW6NAElCF+vPhptEqtEbpS9qSd42t3PUeLcYPo
zBWKPJ9AKlWVf+KsLuvs6NSdC88rYNIytE8fXbm66sFbzIWw3iHBfZaQChLWoigb
CVZgKW3k0bhCEwbC9f9DssxW+xd+vNKESIVeVxuikw4PPphwhzTnJSB50gYEHdTV
/9I/AYPG3do1pJvaaXPNsWguop2fbsyObG0VcwIDAQABAoIBAQCnqQro09QgDlYE
2eejan9Xzo0olraCOuS6pbBWuE0vLDZQ5+lthzR0b61bcWXfNh9RyQ2Vh5jo11GJ
WChK4veNF3pclr9gkZIvDhj1aN7xQR+1gvwingP1j4p1VYWrBZRRJQ2LLDkR+gyO
9Yfq7FwlrNdUcyFhAIOe3qzOIZU0DHQukllvCPy50aJ2pifmizW4VJezQYxWJ7Ss
sYgwRxS4RFofBxBZy/L7Tem5KID2b1vUskjkL383xBy6hFlqeJv9jnpTJxsS9UVp
yZ8yBR4mHGdM5wAVtLXCnxzdx2OIYP3F2RbKSPq2LhciHaQLUEocPurk3P1b+gS7
3YCqfnXJAoGBAPytomAPfm3glK+S6KUcPD1huIxJTyjunp76uym+KEujamD6sy3X
9bsPGnkviLBn68nVvvTsvvRqkSLefxI8/rubVRTWqD8Ytu+vgGTjEiyz09myfWZa
w9+4weURsHhi/ZVQTLmZ8yJ0HnizVW0buqW4s+eErnI3oGAkipc0hTNPAoGBANo1
rhmq8zsyGsV1hCjFtH3jReS53oYiid5YPUwfXh1b9RY1b/y5nXrIYAyxBEsxcrbP
lNy3jayNBcWQmwMIJTXx4gc66Z4/zKNjsb/gbY7TIjkunM7m/NZWm0ZLmNMvQJpS
/9nIcKL99sx917bFfBJ9klDETEDqqzC9F4lQygKdAn82SReZi7qd4KIRFn7k+iv3
BAeiHB3lrV5myXdsgG4n0fQI4dOdBcjFZ3lIj+3N+IWv1zwvYq2j1UFZPHQQdwpD
RBwDXhrHc4fZ9tqe0ioVcI0TOEW4t8ddSVRe4m4Hbz0L1zSkq26Q+GGTjPPNfSec
cxIzq03PPCQ4g6FC2wXDAoGAAOEGeoP9xtdE1FbM2X0yZO9tcKZvBwtMkuC4Hn5h
2qxsKFK+yL3WrI/yExsDcWoZlDLx3pG+uFs5ErqqZTC7NaRrzKuoFUMch+v241b8
Gf8vvBs79nlnu1VN+tKDIxalo/mBnlb4XS+4tsoq/El/XnOzCzA/PmOznNamKZ9x
sOECgYBL7FrO5szB/CohWYF3N5B77lnW2EUuznuNwnv6PvnVF8CJWF7Eagb1SDIN
EUlU7QfKbA4U2mr/ZJNRd+Jup4nLB/GR1J9SgtnoYQi6K93+RAluqgjFKj5KFpgP
KyjJxDsE3EznXcqI4CnlWy7+tVyKOp5E5giiK0+Lvd+47a+IAg==
-----END RSA PRIVATE KEY-----
```

The downloaded example-key.pem is a Base-64 formatted file with a header and other description fields. Inside the file there is the private key of the pair. This key should be saved into a protected folder and its

access should be restricted. (however, to simplify this explanation, the key file is kept in the same place because it will be deleted once this example is finished).

- Navigate to C:\Users\your_username\.ssh

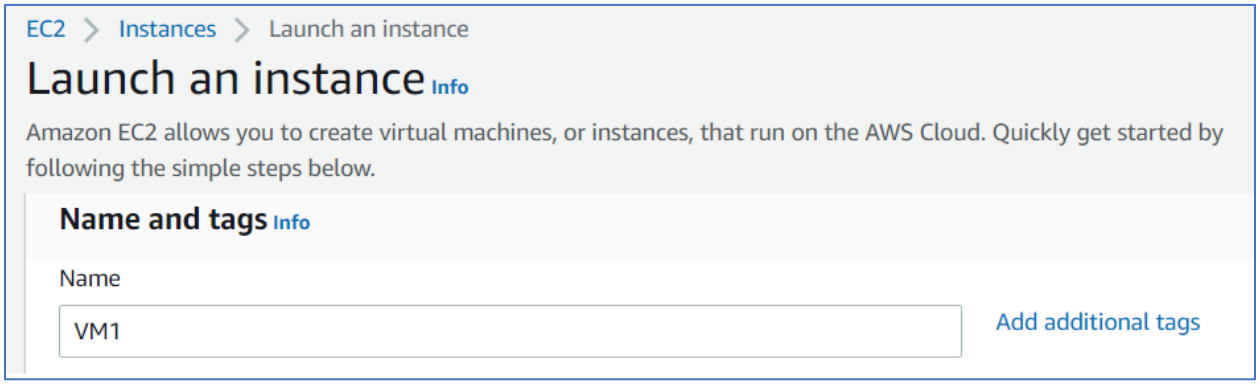
In that location, there must be a text file named **known_hosts**.



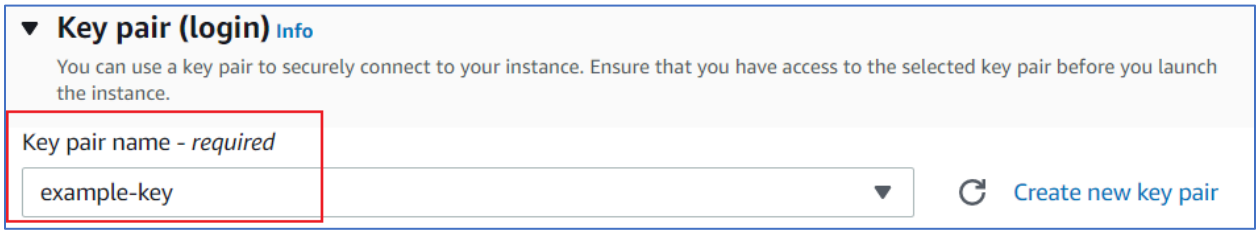
- It contains several entries with the format: IPv4 address, a security algorithm or system, and a string of coded data. These are the known hosts that the laptop has visited. Each host that has been accessed using SSH has left a string inside this file.

```
34.207.90.11 ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBG9BhZkexmsY34+DPj66Tiy2m3fUmvDhcYlrTGu+6EcjFlZ
TBBTdx1IQsPpyjlOY8x5IXKOMkxhv1CKU36Uxf+0=
52.201.217.11 ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBLVnTQA84/fxxreDPJPWZlC7Fe9axcOrhDAxb2rNQmDwA
Iv0oSXEtwZi+wZMEmhE8W+KCA9yP/K2Dg4cHfZluRk=
```

- Launch a new EC2 instance.



- Choose example-key.pem for the key pair.



- SSH into the new VM1.

```
C:\Users\fc>ssh ec2-user@52.90.75.98 -i C:\Users\fc\Downloads\example-key.pem

The authenticity of host '52.90.75.98 (52.90.75.98)' can't be established.
ECDSA key fingerprint is SHA256:6dTwhybKXNSEm4Yz9Z3xTtY52RfEtdIfkuMelK4XHmE.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '52.90.75.98' (ECDSA) to the list of known hosts.

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

[ec2-user@ip-172-31-80-223 ~]$
```

- Go back to C:\Users\your_username\.ssh to see if the file has changed.

```
34.207.90.11 ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBG9BhZkexmsY34+DPj66Tiy2m3fUmvDhcYlrTGu+6E
cjFlZTBBTdx1IQsPpyjlOY8x5IXKOMkxhv1CKU36Uxf+0=
52.201.217.11 ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBLVnTQA84/fxxreDPJPWZlC7Fe9axcOrhDAxb2rNQm
DwAIv0oSXEtwZi+wZMEmhE8W+KCA9yP/K2Dg4cHfZluRk=
52.90.75.98 ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBL0wFxmCQM8zOq6iXZBxmItcA2yNOvXIPhJig0aBq5
bKxdjCjp2TTOXHYADo/sWDHah6IMI/blVwA+Jfankg7XI=
```

It has changed; indeed, a new entry has been added to the file. It begins with the IPv4 address of the VM1, then the cipher information and a string of data. What is that data? That is what will be found out next.

- In the EC2 instance, VM1, navigate to the directory **/etc/ssh**

```
[ec2-user@ip-172-31-80-223 ~]$ cd /etc/ssh
[ec2-user@ip-172-31-80-223 ssh]$ ls -l
(I edited this for simplicity)
```

```
[ec2-user@ip-172-31-80-223 ssh]$ ls

moduli  sshd_config  ssh_config
```

Private keys	Public keys
ssh_host_ecdsa_key	ssh_host_ecdsa_key.pub
ssh_host_ed25519_key	ssh_host_ed25519_key.pub
ssh_host_rsa_key	ssh_host_rsa_key.pub

The directory /etc/ssh contains the configuration and the private and public keys (.pub) that the SSH protocol needs to operate. There are three (3) pairs of keys in the folder. A pair is based on elastic curve DSA, another is Edward curve, and the last one is Rivest, Shamir and Adleman (RSA). The SSH server side negotiates with the client side which one of the pairs will be used for communications sessions.

- Let's take a look at the ECDSA public key.

```
[ec2-user@ip-172-31-80-223 ssh]$ cat ssh_host_ecdsa_key.pub
ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBL0wFxmCQM8zOq6iXZBxmItcA2yNOvXIPhJig0
aBq5bKxdjCjp2TTOXHYADo/sWDHah6IMI/blVwA+Jfankg7XI=
```

- Let's compare that ECDSA public key with the string stored in the laptop's .ssh/known_hosts.

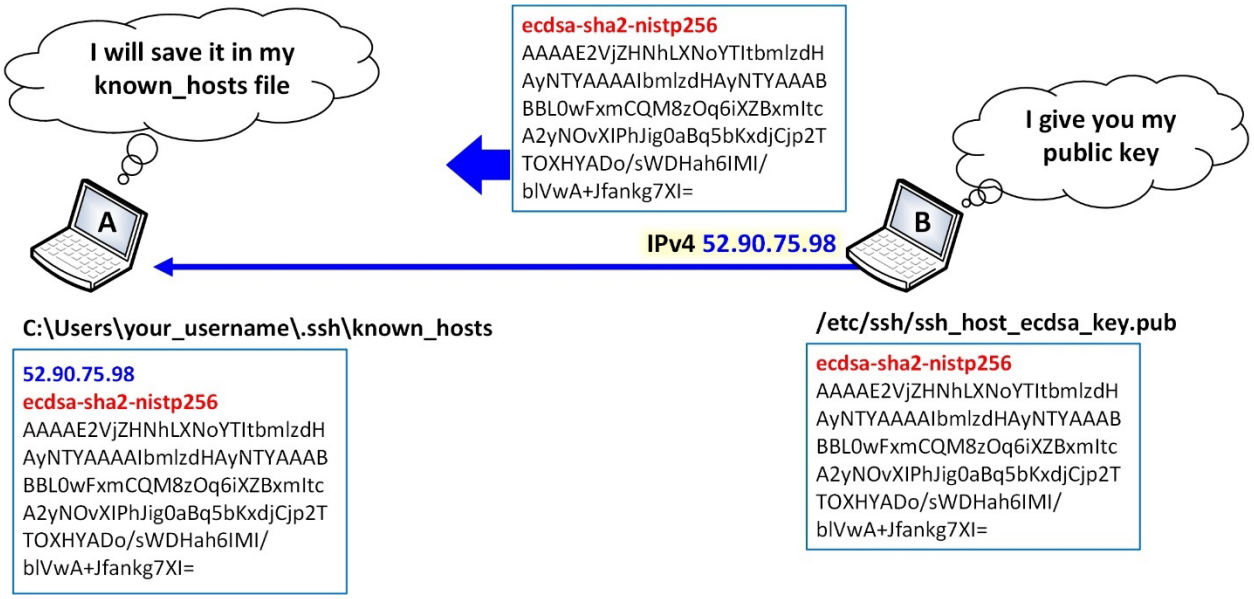
This is on the EC2-instance's /etc/ssh

```
[ec2-user@ip-172-31-80-223 ssh]$ cat ssh_host_ecdsa_key.pub
ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBL0wFxmCQM8zOq6iXZBxmItcA2yNOvXIPhJig0
aBq5bKxdjCjp2TTOXHYADo/sWDHah6IMI/blVwA+Jfankg7XI=
```

This is on the customer laptop's .ssh/known_hosts

```
52.90.75.98
ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBL0wFxmCQM8zOq6iXZBxmItcA2yNOvXIPhJig0
aBq5bKxdjCjp2TTOXHYADo/sWDHah6IMI/blVwA+Jfankg7XI=
```

They are identical. When the client SSH into the EC2 instance, the public key ssh_host_ecdsa_key.pub of the server got added in the known_hosts file of the laptop.



The SSH public key of the server is supplied to the client.

A good question to ask now is: when was the key transferred from B to A?

This message appeared before:

```
The authenticity of host '52.90.75.98 (52.90.75.98)' can't be established.
ECDSA key fingerprint is SHA256:6dTwhybKXNSEm4Yz9Z3xTtY52RfEtdIfkuMelK4XHmE.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '52.90.75.98' (ECDSA) to the list of known hosts.
```

That is when the public key ssh_host_ecdsa_key.pub got transferred to A. The key was offered in the initial steps of the SSH connection. The purpose of this transaction is for future validation of the identity of the server (B). For future sessions, the server will send a fingerprint signed with its private key ssh_host_ecdsa_key. The client will check the fingerprint with the installed public key. If that works, the client concludes that it is the same server than before. If the server gets replaced with another server but using the same IPv4 address, then it will not match and the client gets a message like this:

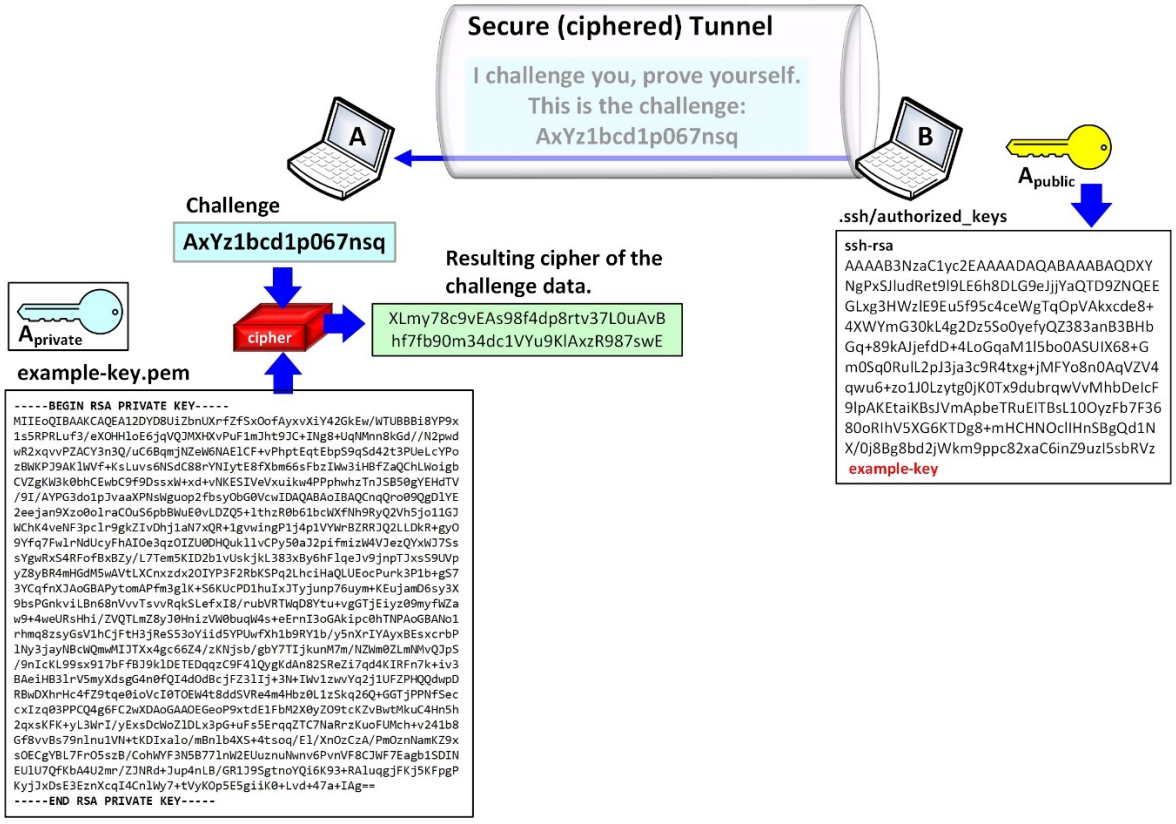
```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
```

Thus, that private-public key pair is used for **authentication of the server side** purpose. What about the other key from the example-key.pem file? Where is it in the EC2-instance and how is it used?

- In the EC2 instance, navigate to **.ssh** and look for **authorized_keys**

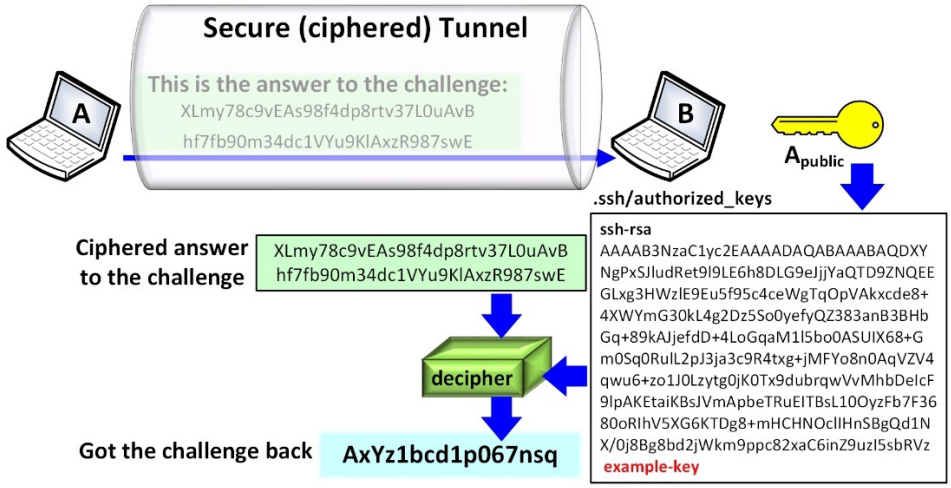
```
[ec2-user@ip-172-31-80-223 ssh]$ cd
[ec2-user@ip-172-31-80-223 ~]$ cd .ssh
[ec2-user@ip-172-31-80-223 .ssh]$ ls
authorized_keys
[ec2-user@ip-172-31-80-223 .ssh]$ cat authorized_keys
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDXYNgPxSJludRet9l9LE6h8DLG9eJjjYaQTD9ZNQEEGLxg/3HWzlE9E
u5/f95c4ceWgTqOpVAkxcde8+4XWYmG30kL4g2Dz5So0yefyQZ3/83anB3BHbGq+89kAJjefdD+4LoGqaM1l5
bo0ASUIX68+Gm0Sq0RulL2pJ3ja3c9R4txg+jMFYo8n0AqVZV/4qwu6+zo1J0Lzytg0jK0Tx9dubrqwVvMhbDeIc
F9lpAKEtaiKBsJVmApbeTRuEITBsL1/0OyzFb7F3680oRIhV5XG6KTDg8+mHCHNOclIHnSBgQd1NX/0j8Bg8bd2j
Wkm9ppc82xaC6inZ9uzI5sbRVz example-key
```

This key is the public key of the example-key pair. The private key of the pair is in the laptop. How is this key used then? The following diagram shows a transaction ciphered inside a tunnel to **authenticate the client side**.



The client ciphers the challenge data with its private key from example-key.pem

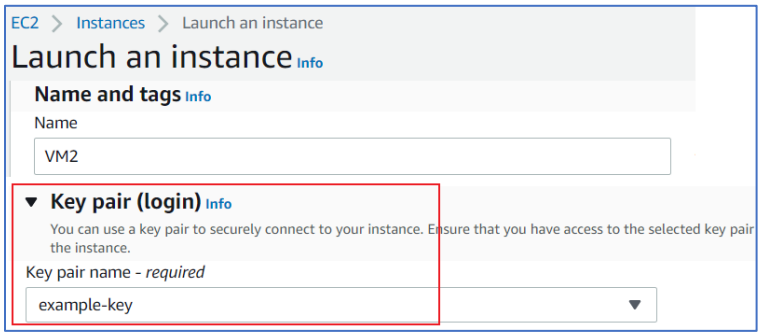
The server side (B) sends the challenge string inside the ciphered tunnel (created after a Diffie-Hellman key calculation process). The client (A) ciphers the challenge with the private key in the file example-key.pem. This is a much stronger system than using passwords. The ciphered challenge is returned to the server to be deciphered with the public key example-key located in .ssh/authorized_keys. If the server can decipher the challenge, then the conclusion is that the client must be legit because it is in possession of the private key.



The server deciphers the challenge cipher-data with the public key from example-key.pem

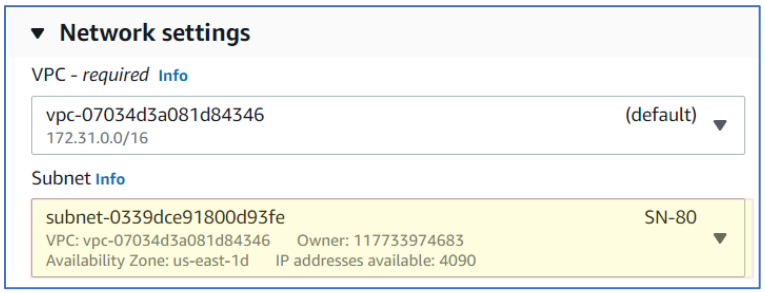
Once that the authentication transactions have been demonstrated between a customer computer and a server EC2 instance in AWS, let's try something else. The SSH session will be from VM1 to another EC2 instance.

- Create a new EC2 instance VM2. Use the same example-key.

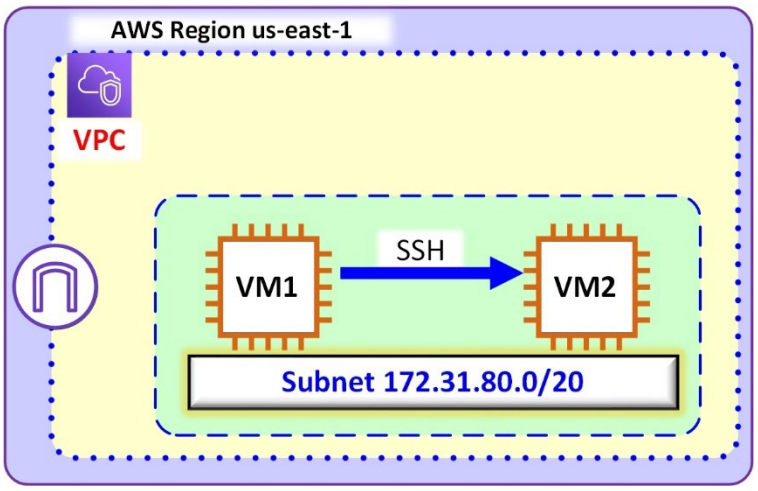


- The existing VM1 is located in subnet 172.31.80.0/20.

- Let's place the new VM2 in the same subnet.



(This step is optional; this is just to simplify the explanation as shown in the scenario below)



Two EC2 instances using the same key pair file.

In this example, the new VM2 got the private IPv4 address 172.31.82.32.

- Now, let's attempt to SSH from VM1 into VM2 using the destination IPv4 address 172.31.82.32.

```
[ec2-user@ip-172-31-80-223 ~]$ cd .ssh
[ec2-user@ip-172-31-80-223 .ssh]$ cat authorized_keys
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDXYNgPxSJludRet9l9LE6h8DLG9eJjjYaQTD9ZNQEEGLxg/3HWzlE9Eu5/f9
5c4ceWgTqOpVAkxcde8+4XWYmG30kL4g2Dz5So0yefyQZ3/83anB3BHbGq+89kAJjefdD+4LoGqaM1l5bo0ASUIX68
+Gm0Sq0RulL2pJ3ja3c9R4txg+jMFYo8n0AqVZV/4qwu6+zo1J0Lzytg0jK0Tx9dubrqwVvMhbDeIcF9lpAKEtaiKB
sJVmApbeTRuEITBsL1/0OyzFb7F3680oRIhV5XG6KTDg8+mHCHNOclIHnSBgQd1NX/0j8Bg8bd2jWkm9ppc82xaC6i
nZ9uzI5sbRVz example-key

[ec2-user@ip-172-31-80-223 .ssh]$ cd
[ec2-user@ip-172-31-80-223 ~]$ ssh ec2-user@172.31.82.32 -i (what key?)
```

There is a problem. What is the key to be used? Because VM1 does not have the private key of the pair. That is located in the customer's laptop. Without that key, there is no way to SSH into VM2. The key needs to be placed in VM1 too.

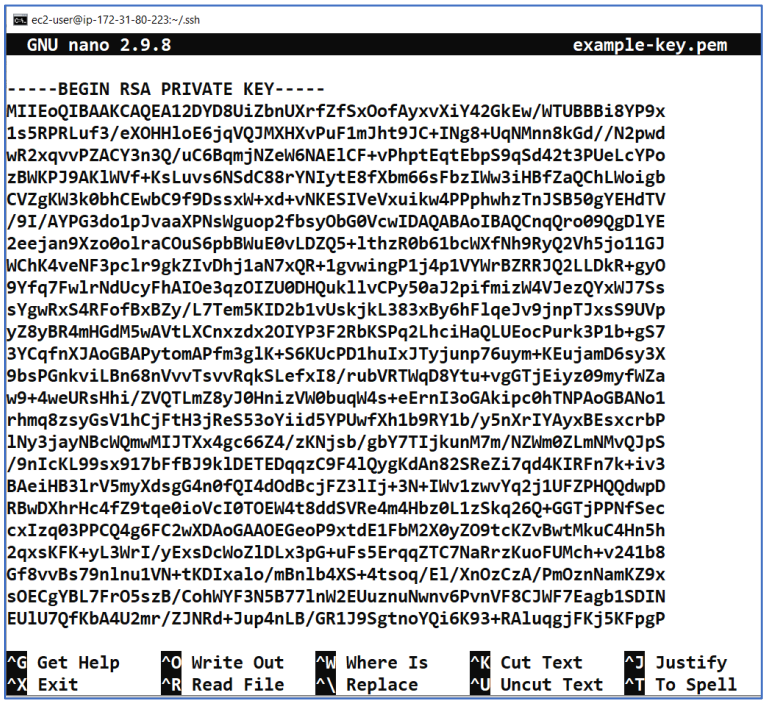
- Thus, proceed to copy the example-key.pem contents.

```
-----BEGIN RSA PRIVATE KEY-----
MIIEoQIBAAKCAQEA12DYD8UiZbnUXrfZfSxOofAyxvXiY42GkEw/WTUBBBi8YP9x
1s5RPRLuf3/eXOHHloE6jqVQJMXHXvPuF1mJht9JC+INg8+UqNMnn8kGd//N2pwd
wR2xqvvPZACY3n3Q/uC6BqmjNZeW6NAElCF+vPhptEqtEbpS9qSd42t3PUeLcYPo
zBWKPJ9AKlWVf+KsLuvs6NSdC88rYNIytE8fXbm66sFbzIWw3iHBfZaQChLWoigb
CVZgKW3k0bhCEwbC9f9DssxW+xd+vNKESIVeVxuikw4PPphwhzTnJSB50gYEHdTV
/9I/AYPG3do1pJvaaXPNsWguop2fbsyObG0VcwIDAQABAoIBAQCnqQro09QgDlYE
2eejan9Xzo0olraCOuS6pbBWuE0vLDZQ5+lthzR0b61bcWXfNh9RyQ2Vh5jo11GJ
WChK4veNF3pclr9gkZIvDhj1aN7xQR+1gvwingP1j4p1VYWrBZRRJQ2LLDkR+gyO
9Yfq7FwlrNdUcyFhAIOe3qzOIZU0DHQukllvCPy50aJ2pifmizW4VJezQYxWJ7Ss
sYgwRxS4RFofBxBZy/L7TemSKID2b1vUskjkL383xBy6hFlqeJv9jnpTJxsS9UVp
yZ8yBR4mHGdM5wAVtLXCnxzdx2OIYP3F2RbKSPq2LhciHaQLUEocPurk3P1b+gS7
3YCqfnXJAoGBAPytomAPfm3glK+S6KUcPD1huIxJTyjunp76uym+KEujamD6sy3X
9bsPGnkviLBn68nVvvTsvvRqkSLefxI8/rubVRTWqD8Ytu+vgGTjEiyz09myfWZa
w9+4weURsHhi/ZVQTLmZ8yJ0HnizVW0buqW4s+eErnI3oGAkipc0hTNPAoGBANo1
rhmq8zsyGsV1hCjFtH3jReS53oYiid5YPUwfXh1b9RY1b/y5nXrIYAyxBEsxcrbP
lNy3jayNBcWQmwMIJTXx4gc66Z4/zKNjsb/gbY7TIjkunM7m/NZWm0ZLmNMvQJpS
/9nIcKL99sx917bFfBJ9klDETEDqqzC9F41QygKdAn82SReZi7qd4KIRFn7k+iv3
BAeiHB3lrV5myXdsgG4n0fQI4dOdBcjFZ31Ij+3N+IWv1zwvYq2j1UFZPHQQdwpD
RBwDXhrHc4fZ9tqe0ioVcI0TOEW4t8ddSVRe4m4Hbz0L1zSkq26Q+GGTjPPNfSec
cxIzq03PPCQ4g6FC2wXDAoGAAOEGeoP9xtdE1FbM2X0yZO9tcKZvBwtMkuC4Hn5h
2qxsKFK+yL3WrI/yExsDcWoZlDLx3pG+uFs5ErqqZTC7NaRrzKuoFUMch+v241b8
Gf8vvBs79nlnu1VN+tKDIxalo/mBnlb4XS+4tsoq/El/XnOzCzA/PmOznNamKZ9x
sOECgYBL7FrO5szB/CohWYF3N5B77lnW2EUuznuNwnv6PvnVF8CJWF7Eagb1SDIN
EUlU7QfKbA4U2mr/ZJNRd+Jup4nLB/GR1J9SgtnoYQi6K93+RAluqgjFKj5KFpgP
KyjJxDsE3EznXcqI4CnlWy7+tVyKOp5E5giiK0+Lvd+47a+IAg==
-----END RSA PRIVATE KEY-----
```

- In VM1, make a new file named example-key.pem using a text editor (nano, vi)

```
[ec2-user@ip-172-31-80-223 ~]$ cd .ssh

[ec2-user@ip-172-31-80-223 .ssh]$ sudo nano example-key.pem
```

- Paste the contents of the private key and save it.




```
[ec2-user@ip-172-31-80-223 .ssh]$ ls
authorized_keys  example-key.pem  known_hosts
```

- Change the rights to the key usage.

```
[ec2-user@ip-172-31-80-223 .ssh]$ sudo chmod 400 example-key.pem
```

- Proceed to SSH into VM2.

```
[ec2-user@ip-172-31-80-223 .ssh]$ sudo ssh ec2-user@172.31.82.32 -i example-key.pem

The authenticity of host '172.31.82.32 (172.31.82.32)' can't be established.
ECDSA key fingerprint is SHA256:05lAvZP8QG26HkxW8wiEY0TpVeGsO1plN447ZhnX4Bs.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.31.82.32' (ECDSA) to the list of known hosts.

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

[ec2-user@ip-172-31-82-32 ~]$
```

VM2 also has a .ssh/authorized_keys that contains the public key of the example-key pair. This proves that when the EC2 instance was created, AWS automatically set this key in this folder. Every EC2 instance created with the same key pair will have such public key installed.

```
[ec2-user@ip-172-31-82-32 ~]$ cd .ssh

[ec2-user@ip-172-31-82-32 .ssh]$ ls

authorized_keys

[ec2-user@ip-172-31-82-32 .ssh]$ cat authorized_keys

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDXYNgPxSJludRet9l9LE6h8DLG9eJjjYaQTD9ZNQEEGLxg/3HWzlE9Eu5/f95c4ceW
gTqOpVAkxcde8+4XWYmG30kL4g2Dz5So0yefyQZ3/83anB3BHbGq+89kAJjefdD+4LoGqaM1l5bo0ASUIX68+Gm0Sq0RulL2
pJ3ja3c9R4txg+jMFYo8n0AqVZV/4qwu6+zo1J0Lzytg0jK0Tx9dubrqwVvMhbDeIcF9lpAKEtaiKBsJVmApbeTRuEITBsL1
/0OyzFb7F3680oRIhV5XG6KTDg8+mHCHNOclIHnSBgQd1NX/0j8Bg8bd2jWkm9ppc82xaC6inZ9uzI5sbRVz example-key
```

- Now, navigate to /etc/ssh and take a look at ssh_host_ecdsa_key.pub.

```
cat ssh_host_ecdsa_key.pub

[ec2-user@ip-172-31-82-32 ssh]$ cat ssh_host_ecdsa_key.pub

ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBIgCNtvKjUCXo218Xg5/GdAIEg//OqRH+6wDnN
erSYhW2J+f3PyYoGZVyj2Bbj8786YDF6OO54fQzoPkGW29FRQ=
```

Since VM1 has accessed via SSH VM2, then the process of server side authentication must have left a key in the known_host file of VM1.

- Exit VM2, go back to VM1.
- Compare the key ssh_host_ecdsa_key.pub in VM2 with the key in VM1's .ssh/known_hosts.

```
[ec2-user@ip-172-31-82-32 .ssh]$

[ec2-user@ip-172-31-82-32 .ssh]$ exit

logout

Connection to 172.31.82.32 closed.

[ec2-user@ip-172-31-80-223 ~]$ cd .ssh

[ec2-user@ip-172-31-80-223 .ssh]$ ls

authorized_keys  known_hosts

[ec2-user@ip-172-31-80-223 .ssh]$ cat known_hosts

172.31.82.32 ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBIgCNtvKjUCXo218Xg5/GdAIEg//OqRH+6wDnN
erSYhW2J+f3PyYoGZVyj2Bbj8786YDF6OO54fQzoPkGW29FRQ=
```

The key ecdsa-sha2-nistp256 in VM2's /etc/ssh is the same than the key ecdsa-sha2-nistp256 in 172.31.82.32 .ssh/known_hosts. This proves that VM2 gave this public key to VM1.

- Exit VM1 and SSH from the laptop into VM2.

```
[ec2-user@ip-172-31-80-223 ~]$

[ec2-user@ip-172-31-80-223 ~]$ exit

logout

Connection to 52.90.75.98 closed.

C:\Users\fc>ssh ec2-user@44.208.162.64 -i C:\Users\fc\Downloads\example-key.pem
```

```
C:\Users\fc>ssh ec2-user@44.208.162.64 -i C:\Users\fc\Downloads\example-key.pem
The authenticity of host '44.208.162.64 (44.208.162.64)' can't be established.
ECDSA key fingerprint is SHA256:05lAvZP8QG26HkxW8wiEY0TpVeGsO1plN447ZhnX4Bs.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '44.208.162.64' (ECDSA) to the list of known hosts.
Last login: Sat Jul 16 15:56:55 2022 from ip-172-31-80-223.ec2.internal

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

[ec2-user@ip-172-31-82-32 ~]$
```

Notice that this is the first time that VM2 is accessed from the laptop, so the warning about the authenticity of the server side appears. When the connection is accepted, VM2 delivers its SSH public key to the laptop. If this reasoning is correct, the key should be in the laptop's **.ssh/known_hosts** file.

Laptop's **C:\Users\fc\.ssh\known_hosts**

52.90.75.98 ecdsa-sha2-nistp256 (This is the SSH public key from VM1)

AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBL0wFxmCQM8zOq6iXZBxmItcA2yNOvXIPhJig0aBq5bKxdjCjp2TTOXHYADo/sWDHah6IMI/blVwA+Jfankg7XI=

44.208.162.64 ecdsa-sha2-nistp256 (This is the SSH public key from VM2)

AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBIgCNtvKjUCXo218Xg5/GdAIEg//OqRH+6wDnNerSYhW2J+f3PyYoGZVyj2Bbj8786YDF6OO54fQzoPkGW29FRQ=

The purpose of the keys observed in this learning activity is to authenticate the interaction between the SSH server and the client. Every time than a virtual machine is created in AWS, the customer chooses the key file that is going to automatically supply the public key to the EC2 instance. The customer must have the key file that contains the private counterpart. These keys are used to authenticate the client when it tries to SSH into the (EC2) server. Furthermore, the first time that the client accesses a server via SSH, it accepts a public key from the server and stores that in the local .ssh\known_hosts file. This public key is used later in the next communication sessions to authenticate that the server is the same. The transactions described here apply to the authentication process for SSH sessions, but none of the keys are used for ciphering the communication, that is a different process.

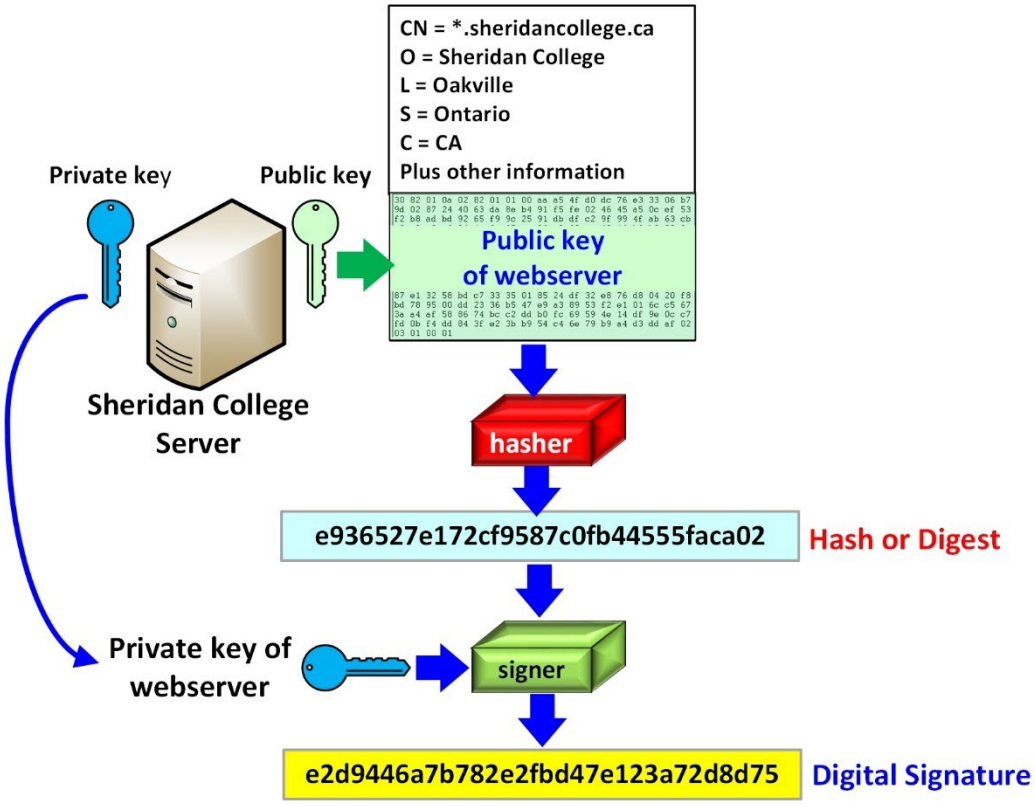
Authentication of a web server with Security Certificates

The process of authentication of a web server is different than the authentication process for SSH sessions. Potentially, a web server takes care of thousands of sessions with different clients, so a system of exchanging public keys with all of these clients will not work. Still, the client needs to know that the web server is authentic and not some bogus site. There must be a way that a webserver can authenticate to so many clients.

The solution to this problem is using a **third part certification authority** that can vouch for the legitimacy of the website. This is done thought the use of a **security certificate** which is guaranteed by the certification authority. When a client tries to access the website using the secure protocol HTTPS, the web server supplies the certificate which must be verified before proceeding any further. The following paragraphs describe the process of creation and usage of security certificates.

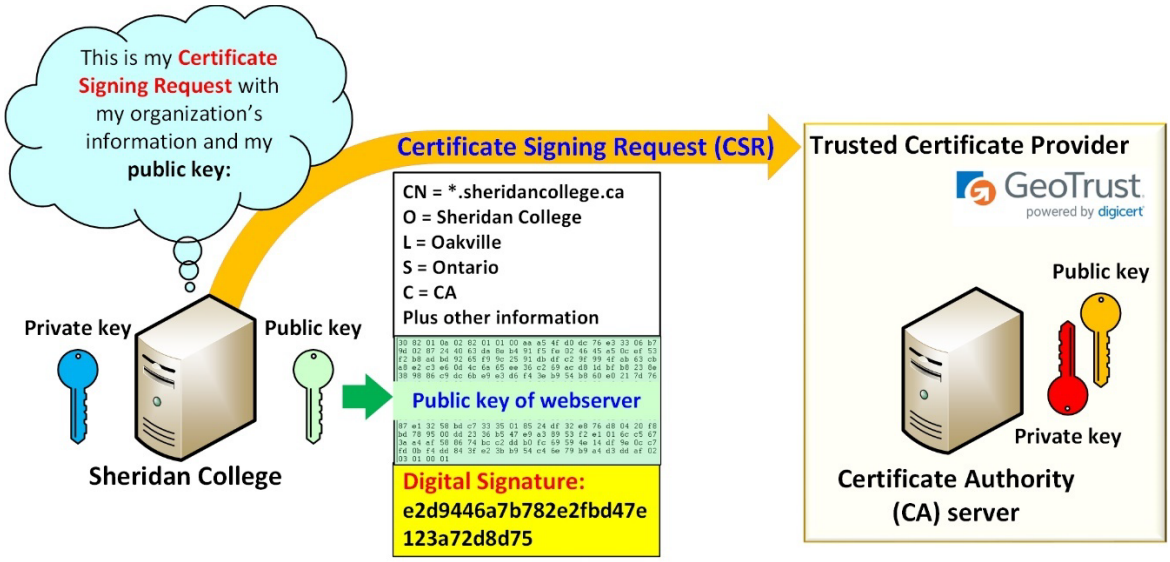
Creation of the Security Certificate in the webserver side.

The first step in securely certifying a web server is to create a Certificate Signing Request (CSR). This is a digital document written in a particular format that contains the information of the organization that owns the website. For example, the country, the location, validity dates, the DNS name, and the public key of the web server. The requester hashes the content of the CSR and ciphers (signs) it with its private key. Only the corresponding public key of the pair can decipher the signature to obtain the digest.



The requester prepares a Certificate Signing Request.

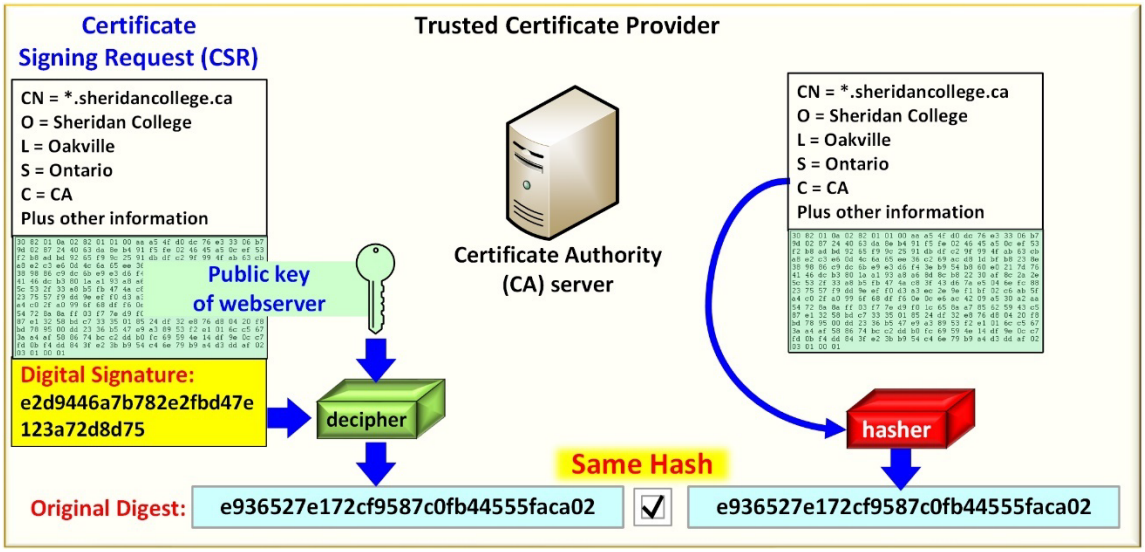
Once that the CSR is ready, it is sent to a Trusted Certificate Provider, GeoTrust in this example.



A Certificate Signing Request (CSR) is sent to a Trusted Certificate Authority.

Certificate providers are companies that specialize in providing this service for a fee. Companies such as Verisign, Digicert, GeoTrust, Entrust, GlobalSign, QuoVadis. Some organizations such as Let's Encrypt is provide certificates for free as long as the requester owns the DNS domain name where the server is deployed. As a manner of example, Sheridan College owns sheridanc.on.ca and sheridancollege.ca.

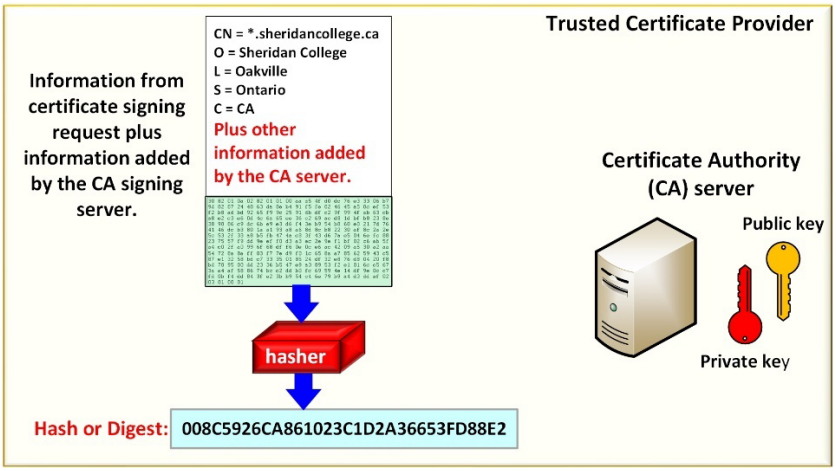
A certificate provider has Certificate Authority (CA) servers with the the only purpose of signing and issuing the security certificates to the requester. In the previous pretend example, Sheridan College has prepared and sent the Certificate Signing Request (CSR) to the provider GeoTrust. In turn, the company GeoTrust will proceed to validate that the information in the CSR is correct and legit.



The CA server validates the integrity of Certificate Signing Request.

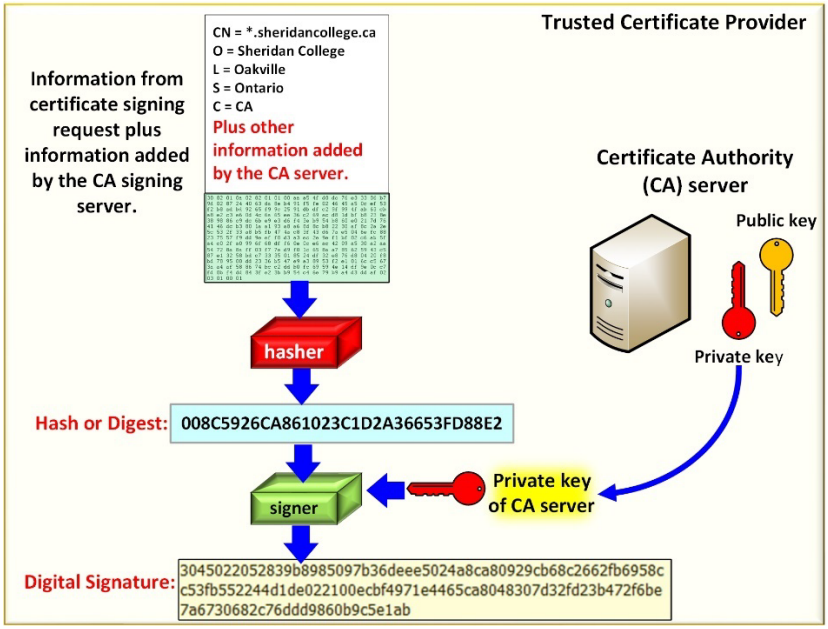
The CA server uses the public key in the signing certificate request to decipher the digital signature. This process renders the hash digest of the data. On the other hand, the CA reproduces the same hash procedure and it should obtain the same result. It compares both and if they match that proves a) that the requester signed the CSR and b) that the data has not been altered.

Once that the legality of the certificate is confirmed, the process of producing the certificate is started. The CA servers adds information such as a serial number, the algorithms being used and the purpose of the key in the certificate. Then the certificate template is passed through a hash algorithm in the CA server. That process yields a unique, non-reversable digest or hash. This process guarantees the integrity of the information for if one bit of the certificate were ever changed, it will never match the same hash value.



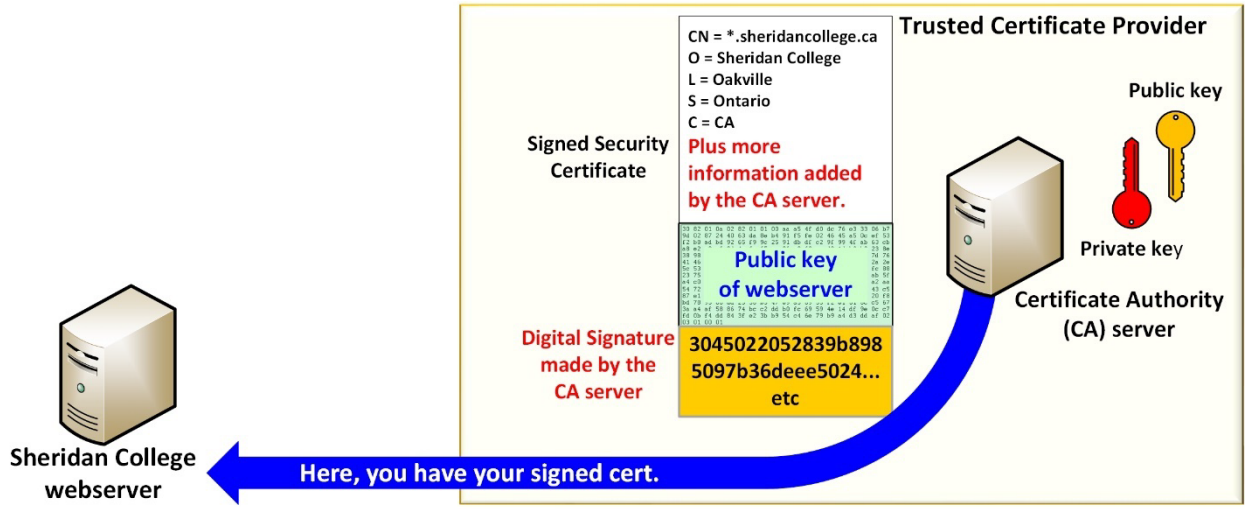
The CA server calculates a hash of the data in the certificate template

Next, the hash is ciphered with the private key of the CA server. This creates a **digital signature** which is the warranty that the Certificate Authority vows for the authenticity of the certificate.



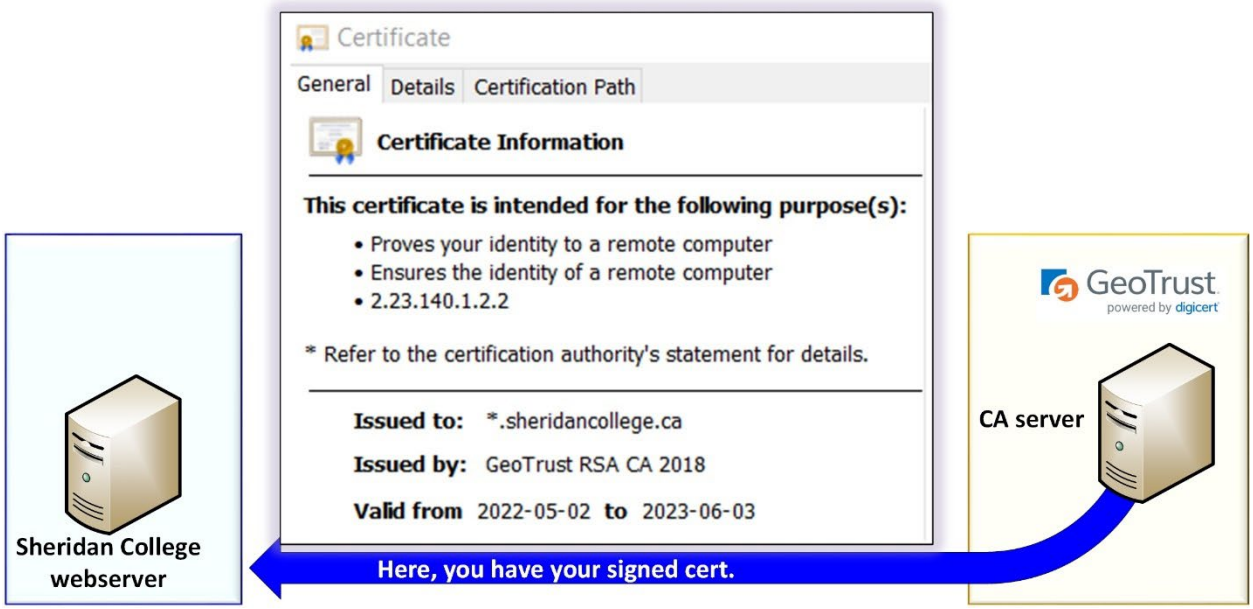
The Certificate Authority server digitally signs the certificate

The digital signature is attached to the body of the certificate. Therefore, the security certificate is ready to be sent to the customer. In this example, GeoTrust delivers the certificate to Sheridan College.



Security Certificate signed and ready to be delivered to the requester.

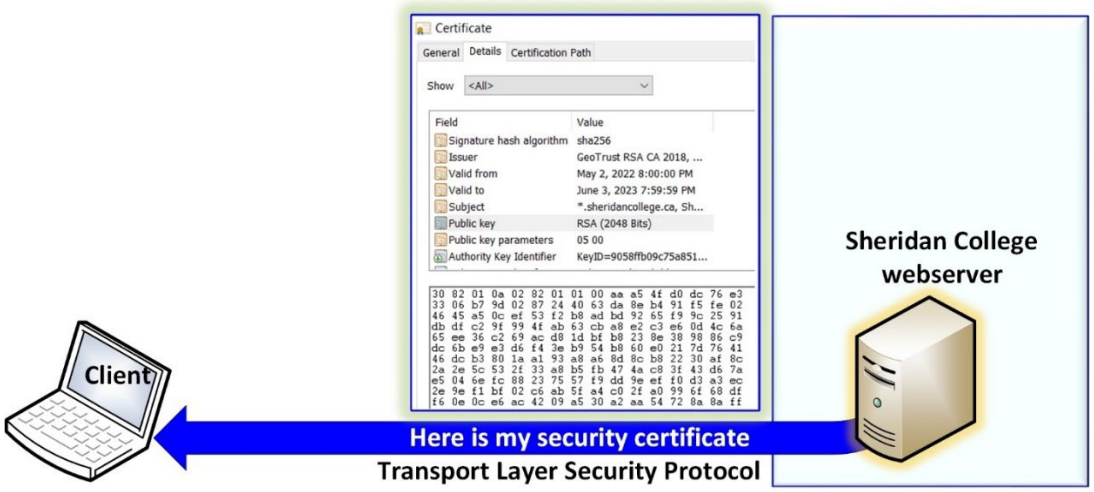
The certificate is finally installed in the webserver. The following snippets shows the real certificate present in Sheridan's webserver in August 2022. The general information states that the purpose of this certificate is to **prove the identity of a remote computer or to a remote computer** which makes sense since the webserver must prove to the clients that it is legitimate and trustworthy. The certificate general information also states that it was issued by GeoTrust CA in 2018 and that it is valid until 2023.



The Certificate Authority delivers the signed certificate to the requester.

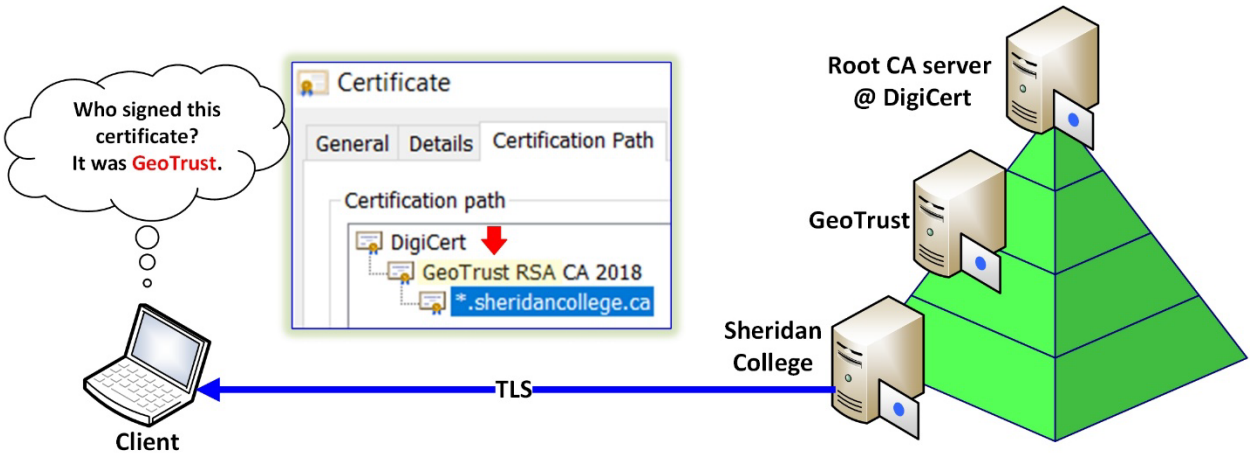
Secure HTTP conversation between the client and the webserver

The configuration of the webserver must have a pointer to the location of the certificate file. The secure protocol HTTPS must be enabled too. When a client requests to access the website, the server and the client carry on a conversation using the protocols TCP, TLS, and HTTPS in that order. During the TLS (Transport Layer Security) session, the server offers its signed security certificate to the client.



The webserver offers its security certificate to the web client.

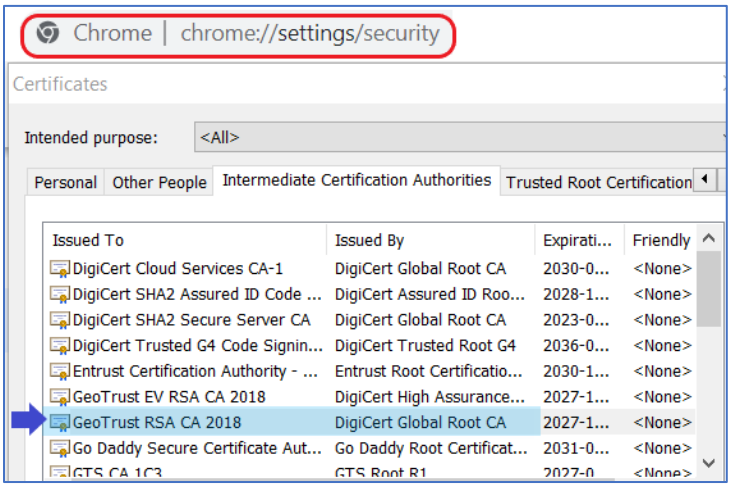
Consequently, the client's web browser starts the evaluation process of the webserver's certificate. First, it looks into the certificate details to find out who was the certificate authority that signed the certificate. The certification path information shows that GeoTrust signed Sheridan's certificate.



The client begins the evaluation of the security certificate.

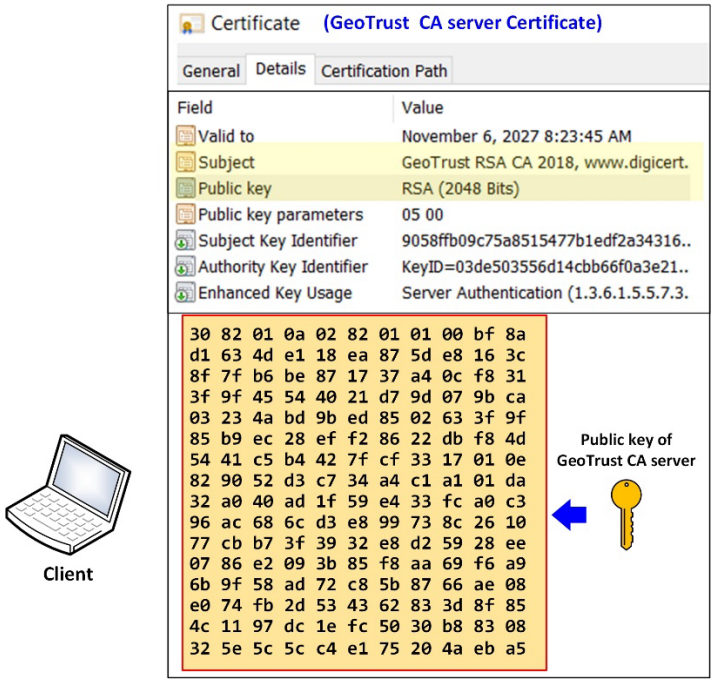
Furthermore, the **chain of validation** indicates that a **root Certificate Authority (CA)** server located in DigiCert issued a certificate to another CA server in GeoTrust. Each certificate is validated by the next level up in the hierarchy. In principle, Sheridan can continue the chain by issuing certificates to its own internal services and devices. The main principle to observe is that each certificate can be verified with the certificate of the issuer up the chain of validation.

The client's web browser (Google Chrome in this example) runs a **Certification Path Validation Algorithm**. This is possible because all web browsers are shipped out with a built-in list of trusted certificate authorities. Accordingly, the client reviews its lists of **Intermediate Certification Authorities** and **Trusted Root Authentication Authorities** under the browser's security settings as shown below.



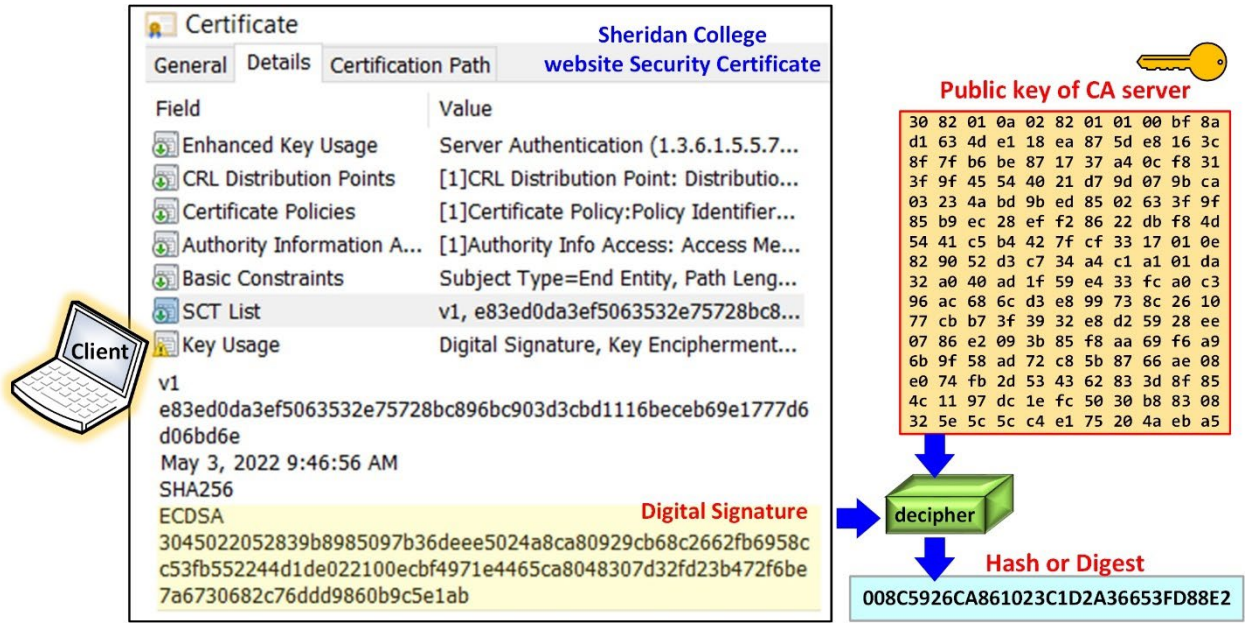
Chrome's web browser privacy settings contain a list of Intermediate Certification Authorities

This is a default repository that contains security certificates from the authority organizations. Right there, the client finds the security certificate of the GeoTrust CA server that signed Sheridan's website certificate. This certificate contains the public key of the GeoTrust CA server.



The client web browsers obtains the public key of the CA server of the Issuing Authority.

The client's web browser uses the public key of the CA server to decipher the digital signature on Sheridan's certificate. This returns the hash calculated by the GeoTrust CA using the certificate's data.

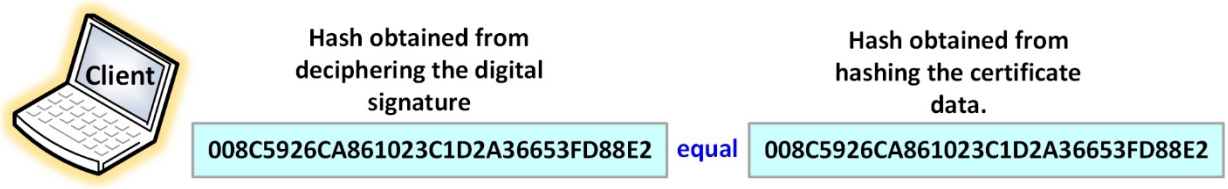


The web browser uses the CA public key to obtain the digest of Sheridan security certificate.

The browser also calculates the hash of the certificate data. The two values are compared and found to be the same. Since the public key of the CA server opens the hash ciphered with the private key of the CA server then the authenticity of the certificate is proven.

The client calculates the hash of the data fields of the certificate

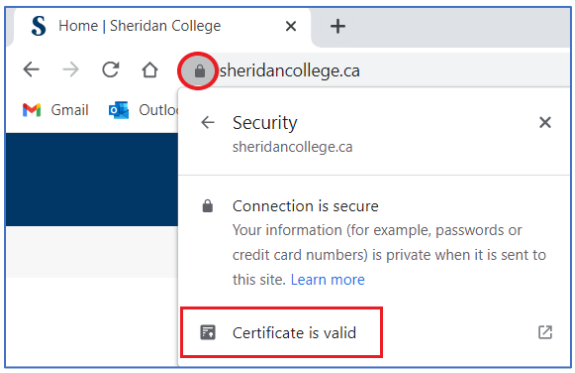
This digest must be identical to the one obtained in the previous step that deciphered the digital signature. If it is, the certificate is accepted as valid.



The client concludes the validation of the security certificate.

The hash obtained from hashing the data locally and the hash digitally signed by the CA server are identical. That concludes the process of authentication of the webserver using a security certificate. Amazingly, all these validation steps are executed in a matter of a second.

Sheridan College's website is a secure server and as such it only accepts HTTPS instead of the insecure HTTP. When the website is accessed, a **lock icon** appears in the address bar (example from Chrome browser below). The details of the website's security certificate can be observed by right-clicking on such lock icon.



Sheridan College website landing page.

It is worth to note that the preceding explanation only dealt with the validation of the authenticity of the webserver using a signed security certificate by an authority. The certificate, in such case, is not used for ciphering the HTTPS communications between the client and the server. There are different applications for certificates. Some certificates are used to prove authenticity, others are used to provide a public key for ciphering.

Conclusion

Computer security is conceived around the concepts of secrecy, authenticity, integrity, and opportunity. The implementation of these principles results in the processes of ciphering, authentication, and data validation. Computer nodes establish secure channels over unsecure networks by negotiating the ciphering and authentication parameters. Keys are part of the security process and there two main systems of keys, the symmetric and the asymmetric. There are multiple uses for key systems, not just in ciphering but also in authentication.

Chapter 12 Coursework

- Start the Wireshark sniffer.
- Set this filter **ip.addr==142.55.7.210** to capture only the traffic that comes or goes to Sheridan webserver.
- On the web browser, open Sheridan College website.
- Click on any link and then close the browser.
- Stop the Wireshark sniffing so no more data is captured.

Analysis of the sniffed conversation

- How does the conversation begin? What is the protocol that opens the conversation?
- What is the destination port (the server)?
- What is the source port (the client)?
- What is the next protocol involved in the conversation?
- Click on the Client Hello message. What is the purpose of this message?
- Click on the Server Hello message. What is the purpose of this message?
- Are these messages in clear-text? Are they readable?
- Can you find the security certificate being exchanged?
- When does the conversation go into ciphered mode?
- Is there a protocol HTTPS anywhere?

References

[1] Cryptographic Communications System and Method, Ronald L. Rivest, Adi Shamir, Leonard M. Adleman. (1983, Sep. 20). 4,405,829 [Online]. Available: https://patentimages.storage.googleapis.com/49/43/9c/b155bf231090f6/US4405829.pdf

[2] Online hash calculator. [Online]. Available: https://www.tools4noobs.com/online_tools/hash

[3] Hellman, Martin. Diffie, Bailey. Merkle, Ralph. Cryptographic apparatus and method. 1977. Patent 4200770. [Online]. Available: https://patents.google.com/patent/US4200770A/en

[4] T. Ylonen, Ed. Lonvick. RFC 4252. The Secure Shell (SSH) Authentication Protocol. 2006. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4252

[5] Amazon EC2 key pairs and Linux instances. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html?icmpid=docs_ec2_console

[6] R. Housley, L. Bassham. RFC-2528. Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. 2002. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3279

[7] R. Housley, et al. RFC-3280. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. 2002. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3280

[8] RFC-5246. T. Dierks, E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. 2008. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5246

[9] AWS Tutorial: Configure SSL/TLS on Amazon Linux 2. 2021. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/SSL-on-amazon-linux-2.html

[10] International Telecommunications Union. Recommendation ITU-T X.509. Public-key and attribute certificate frameworks. 2019. [Online]. Available: https://www.itu.int/rec/T-REC-X.509-201910-I/en