A PRACTICAL & EXAMPLE-BASED GUIDE TO MASTER AWS ESSENTIAL SERVICES AND BUILD & DEPLOY REAL WORLD APPLICATIONS USING AWS CLOUD.

PRACTICAL AWS

** Practical: Concerned With The Actual Use Of AWS Rather Than With Theory & Ideas. No Bullshit !*

BY AYMEN EL AMRI

@eon01

Your product license key: 0a20e2e44d9e72ff02fede2667a83f0f

Lesson 0 - Preface

Each day a new blog post, video, tutorial is created to help people learn Cloud Technologies like AWS, but have you noticed that a considerable part of this is just a copy/paste of the other?

Most online contents look like introductions and theoretical learning. This is the case even in official documentations.

The best way to learn AWS is by working as an AWS developer, admin, or architect. I know that it's contradictory, if you already have the necessary skills to work as an AWS engineer, you will be doing it without reading me now! But what if you learn from someone who already worked with tens of companies to help them migrate, use, and administer their AWS infrastructures? This is what Practical AWS offers.

I believe in learning with practical examples and use cases taken from the real software industry, and this what I've been doing in my other online courses like Painless Docker or Saltstack For DevOps.

If you are interested in learning AWS with practical examples, this course will help you.

How to Get the Most of Practical AWS

During this course, we are going to discover different concepts, features, and services from the AWS cloud. Whether you are a beginner or have an intermediate level, the course will guide you step by step.

It is normal that some notions at the beginning will be blurred, but I advise you to read on, because some concepts are first used, then explained in detail. During your learning path, take notes of the important concepts you learn and the new things you discover.

The knowledge you will acquire during this course is rather practical, but there are the theoretical background definitions necessary to understand the essentials. If you want to dive into more theoretical details, the official AWS documentation will be a good companion to this course.

This course is not documentation but a practical way to learn quickly and apply your knowledge to create stable and scalable systems using AWS. The most used language is Python; this choice is motivated by the fact that it is a readable language and, above all, very easy to understand. If you don't like using Python, you can very easily understand the code (given the self-explanatory nature of Python code) and reproduce the same code in your favorite language (Node.js, Java, Ruby..etc.).

Note: Some chapters in this ebook are videos that you download here. If you want to download the code, it is available in this GIT repository.

About The Author

Aymen is a cloud architect, entrepreneur, author of best selling trainings and books, CEO of eralabs (an award winning DevOps & cloud consulting & training company), and Founder of Faun Community.

He actually lives in Paris and helps companies and startups from everywhere (Europe, US ..) develop modern applications, builds multi-tenant cloud infrastructures, scalable applications, highly stable production environments, distributed systems and service-oriented architectures (microservices & PaaS).

You can find Aymen on Twitter and Linkedin.

Lesson 1 - Introduction to Cloud Computing

The Cloud is Just Another Name for the Internet

Cloud Computing is the on-demand delivery of IT resources like computing, applications, and databases via the Internet. It offers storing and accessing IT resources, data, applications, and configurations over the Internet.

Tools like Google Drive, Dropbox, and other Software as Service tools are also considered as cloud technologies, but what interests us in this course are public cloud providers like Google Cloud Platform, Microsoft Azure, and Amazon Web Services.

In order to remove possible confusion and misapprehension, we are going to define some cloud computing models.

Software as a Service - SaaS

SaaS providers like Dropbox use the web to deliver their applications and online tools. Most SaaS run using a web interface. While the software runs on the server-side, usually, the SaaS user doesn't need to download any third-party client. Some known SaaS is customer relationship management (CRM) like Zoho or Salesforce, document management tools like Google Drive or Microsoft Office 365, enterprise resource planning (ERP) lie Sage.

Platform as a Service - PaaS

PaaS providers like Heroku, allow customers to develop, build and run applications without needing to maintain infrastructure or buy a software license. It allows developers to use on-demand infrastructure without the need for being an infrastructure expert. With the ease of manipulating PaaS, customers could find themselves in vendor lock-in. Heroku, AWS Elastic Beanstalk and Google App Engine are PaaS providers.

Infrastructure as a Service - IaaS

In the IaaS model, the vendor manages the Data Center, Networking, Storage, Virtualization, and a part of the Operating System (OS). On the other hand, the customer manages an important part of the OS, the middleware part if it exists and the application. Amazon EC2 is a common IaaS example: When you enroll an EC2 virtual machine, you do not need to worry about the hardware, the network, or provisioning the machine with an operating system. You'll just need to install your middleware and run your application.

DaaS, CaaS, and other XaaS

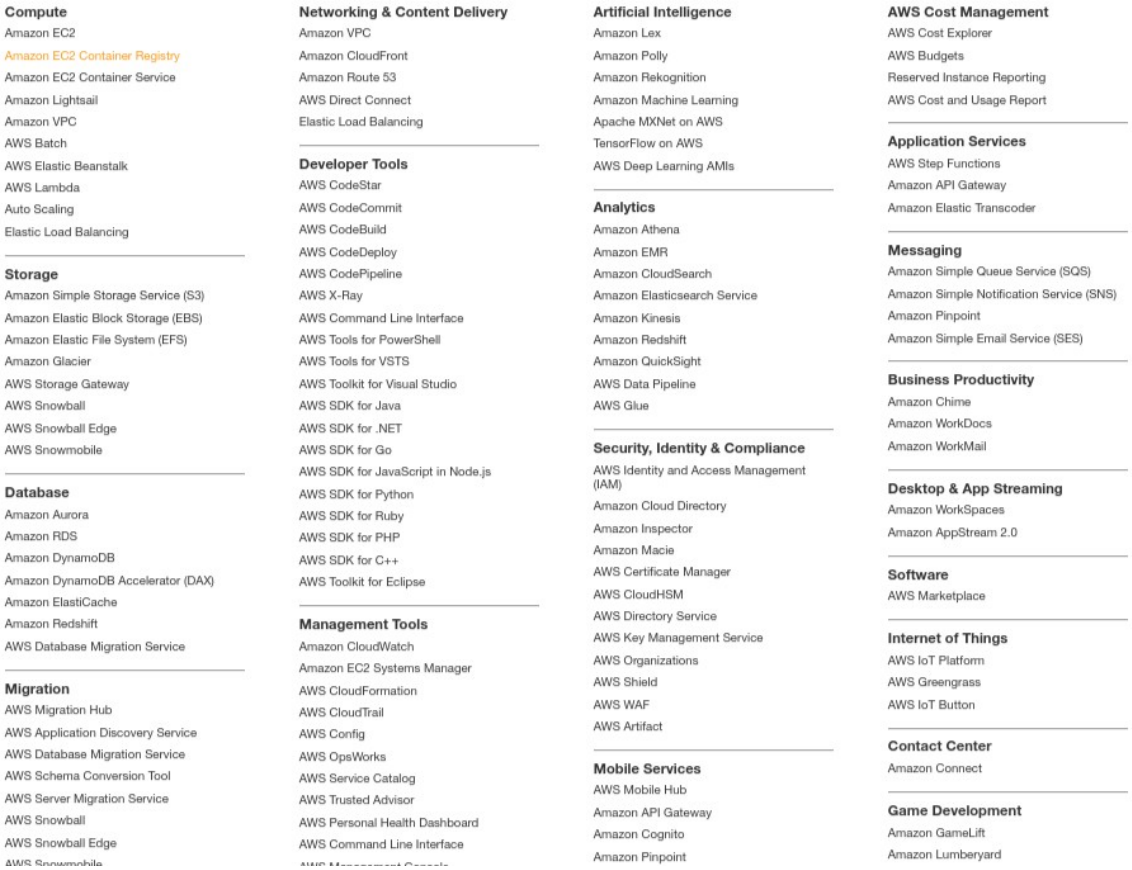
You may find other XaaS variants like CaaS (Containers as a Service), DaaS (Database as a Service), and other XaaS. Most of these fall somewhere between SaaS, PaaS, and IaaS. So, for example, CaaS is a platform that allows the developer to have access to managed containers, in reality, instead of a VM, the cloud provider gives you managed VMs that are invisible to you, that runs containers. This is a subset of IaaS, with the container being a basic resource instead of the VM.

Most of these names were not created to achieve a technical definition but a marketing purpose.

A Quick Introduction to AWS

Amazon Web Services is mainly an IaaS provider but has some IaaS services (e.g., AWS Elastic Beanstalk is a PaaS). AWS is held by Amazon.com, and it is one of the leading cloud computing providers. Probably one of the most known services that AWS offers is EC2, but there are tens of other services: VPC, SES, SQS, S3, CloudFront .. etc

These services are categorized by Amazon as follows:



Compute
Amazon EC2
Amazon EC2 Container Registry
Amazon EC2 Container Service
Amazon Lightsail
Amazon VPC
AWS Batch
AWS Elastic Beanstalk
AWS Lambda
Auto Scaling
Elastic Load Balancing

Storage
Amazon Simple Storage Service (S3)
Amazon Elastic Block Storage (EBS)
Amazon Elastic File System (EFS)
Amazon Glacier
AWS Storage Gateway
AWS Snowball
AWS Snowball Edge
AWS Snowmobile

Database
Amazon Aurora
Amazon RDS
Amazon DynamoDB
Amazon DynamoDB Accelerator (DAX)
Amazon ElastiCache
Amazon Redshift
AWS Database Migration Service

Migration
AWS Migration Hub
AWS Application Discovery Service
AWS Database Migration Service
AWS Schema Conversion Tool
AWS Server Migration Service
AWS Snowball
AWS Snowball Edge

Networking & Content Delivery
Amazon VPC
Amazon CloudFront
Amazon Route 53
AWS Direct Connect
Elastic Load Balancing

Developer Tools
AWS CodeStar
AWS CodeCommit
AWS CodeBuild
AWS CodeDeploy
AWS CodePipeline
AWS X-Ray
AWS Command Line Interface
AWS Tools for PowerShell
AWS Tools for VSTS
AWS Toolkit for Visual Studio
AWS SDK for Java
AWS SDK for .NET
AWS SDK for Go
AWS SDK for JavaScript in Node.js
AWS SDK for Python
AWS SDK for Ruby
AWS SDK for PHP
AWS SDK for C++
AWS Toolkit for Eclipse

Management Tools
Amazon CloudWatch
Amazon EC2 Systems Manager
AWS CloudFormation
AWS CloudTrail
AWS Config
AWS OpsWorks
AWS Service Catalog
AWS Trusted Advisor
AWS Personal Health Dashboard
AWS Command Line Interface

Artificial Intelligence
Amazon Lex
Amazon Polly
Amazon Rekognition
Amazon Machine Learning
Apache MXNet on AWS
TensorFlow on AWS
AWS Deep Learning AMIs

Analytics
Amazon Athena
Amazon EMR
Amazon CloudSearch
Amazon Elasticsearch Service
Amazon Kinesis
Amazon Redshift
Amazon QuickSight
AWS Data Pipeline
AWS Glue

Security, Identity & Compliance
AWS Identity and Access Management (IAM)
Amazon Cloud Directory
Amazon Inspector
Amazon Macie
AWS Certificate Manager
AWS CloudHSM
AWS Directory Service
AWS Key Management Service
AWS Organizations
AWS Shield
AWS WAF
AWS Artifact

Mobile Services
AWS Mobile Hub
Amazon API Gateway
Amazon Cognito
Amazon Pinpoint

AWS Cost Management
AWS Cost Explorer
AWS Budgets
Reserved Instance Reporting
AWS Cost and Usage Report

Application Services
AWS Step Functions
Amazon API Gateway
Amazon Elastic Transcoder

Messaging
Amazon Simple Queue Service (SQS)
Amazon Simple Notification Service (SNS)
Amazon Pinpoint
Amazon Simple Email Service (SES)

Business Productivity
Amazon Chime
Amazon WorkDocs
Amazon WorkMail

Desktop & App Streaming
Amazon WorkSpaces
Amazon AppStream 2.0

Software
AWS Marketplace

Internet of Things
AWS IoT Platform
AWS Greengrass
AWS IoT Button

Contact Center
Amazon Connect

Game Development
Amazon GameLift
Amazon Lumberyard

I've been working with AWS during the latest years and helping companies from all sizes move and use this cloud provider, but I have never seen a company using all of its services. So, our goal here is not to go through all the services, but to understand how the common services work. Once you acquire the knowledge about this, you will have enough knowledge about AWS computing, networking, storage, and security to discover other services.

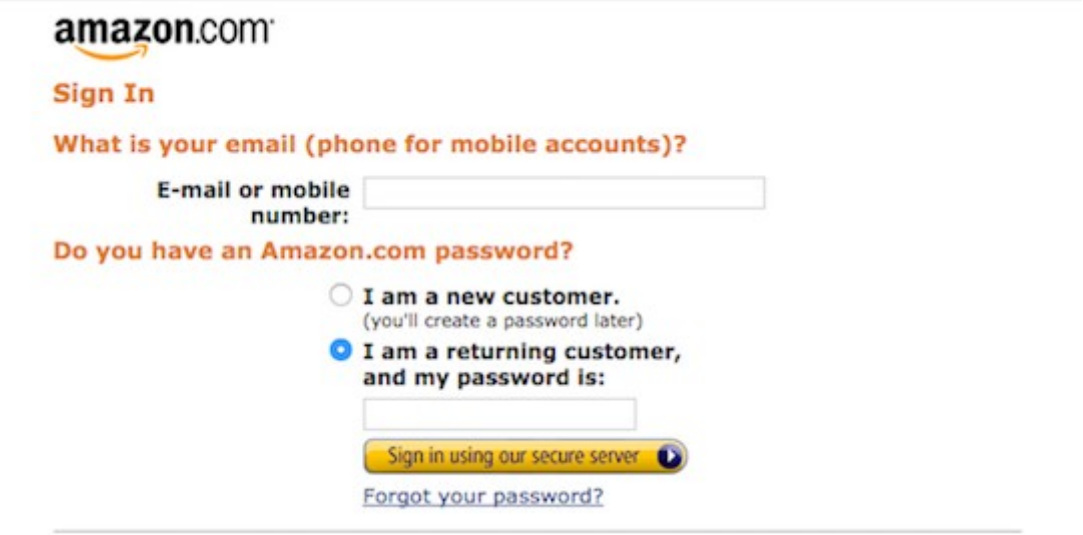
In the next step, we are going to create an AWS account, but let's see how to get the best out of AWS services without spending money. This is possible using the AWS Free Tier. This tier includes offers that expire 12 months following sign up and others that never expire.

e.g., When creating EC2 machines, AWS Free Tier includes 750 hours per month of Linux, RHEL, or SLES t2.micro instance usage and 750 hours per month of Windows t2.micro instance usage.

Lesson 2 - Learn AWS Basics

Creating your AWS Account

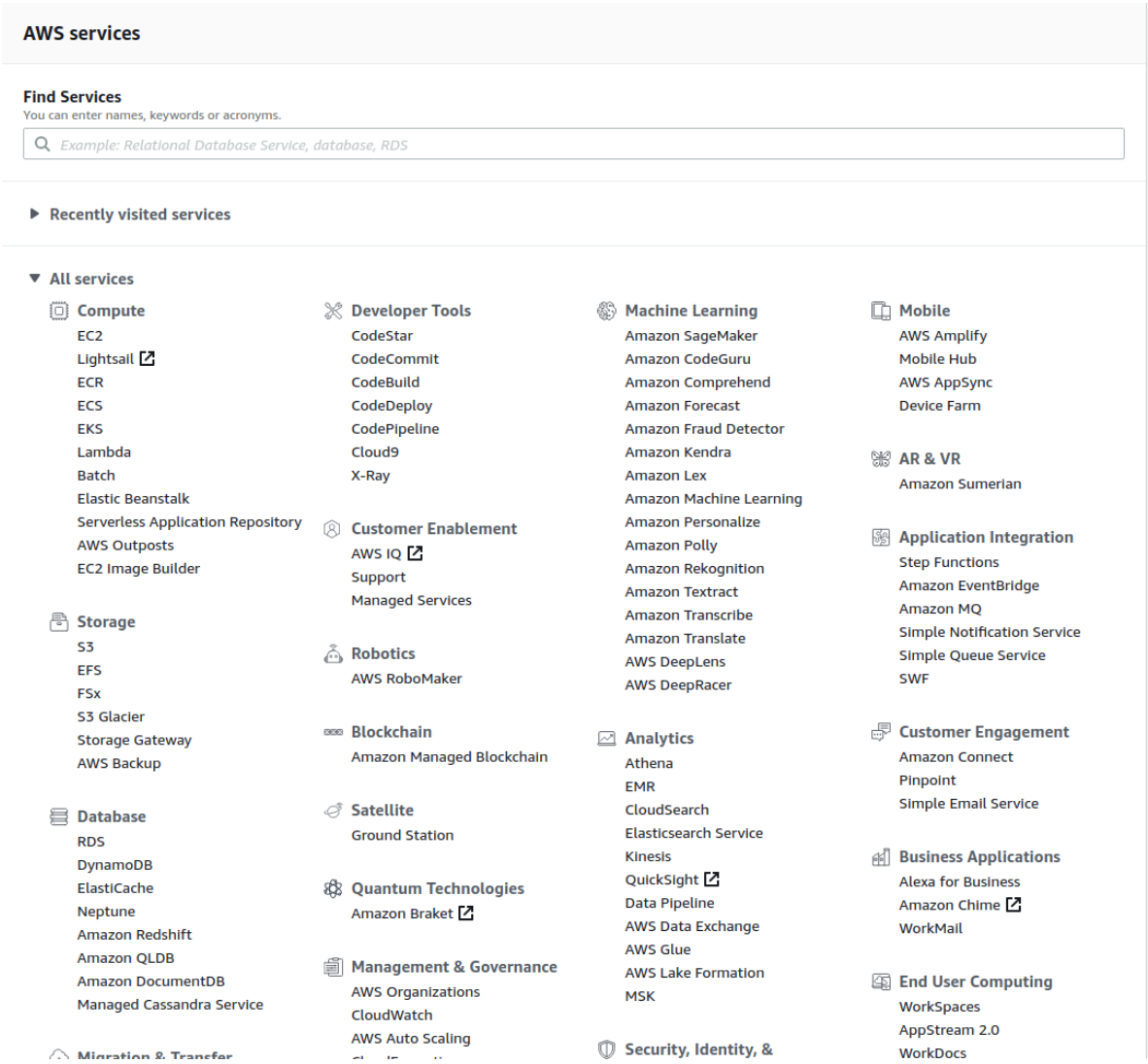
To start learning, you'll need an account. Go to AWS website then create an account using your email.



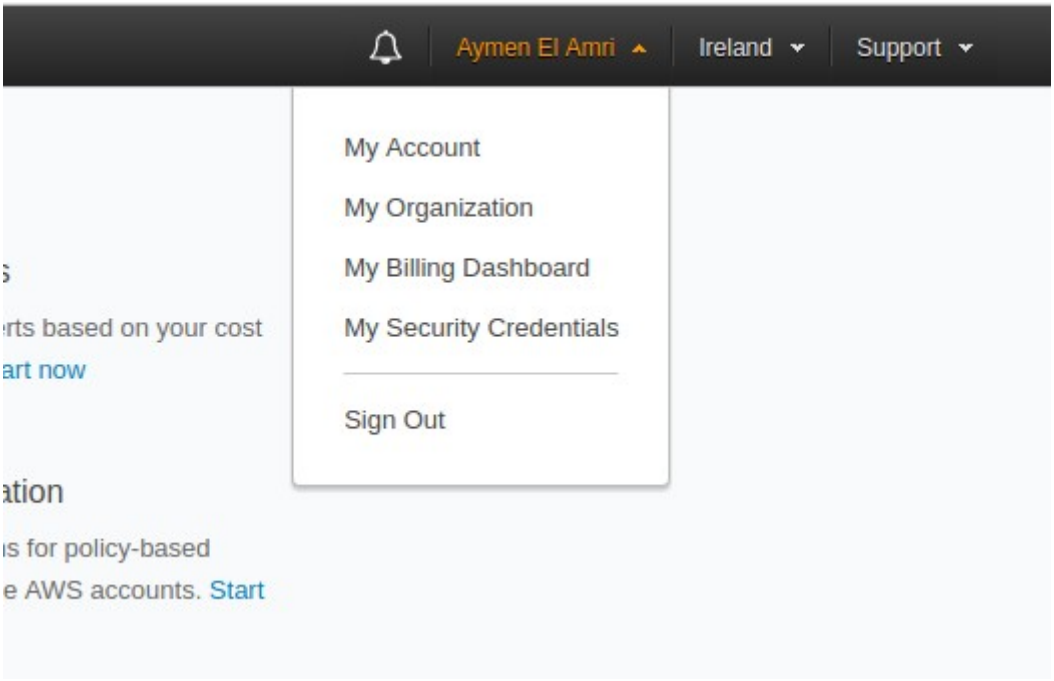
AWS Console

AWS console is a web-based management tool that allows users to access and manage AWS resources. After creating an account, log in using the console URL: aws.amazon.com.

Click on "Services" in order to view the different services that AWS offers.



In order to access your billing dashboard, click on your name on the top right corner of the screen, then click on "My Billing Dashboard".



AWS has a pay-as-you-go model, so you'll be charged only for what you use. This is why, unlike traditional computing services, there are no installation or termination fees.

This does not mean that AWS or cloud computing, in general, is cheaper than traditional computing.

You may pay more in some cases. Cloud computing is neither cheaper nor more expensive than X or Y solution. This is not a good question to ask.

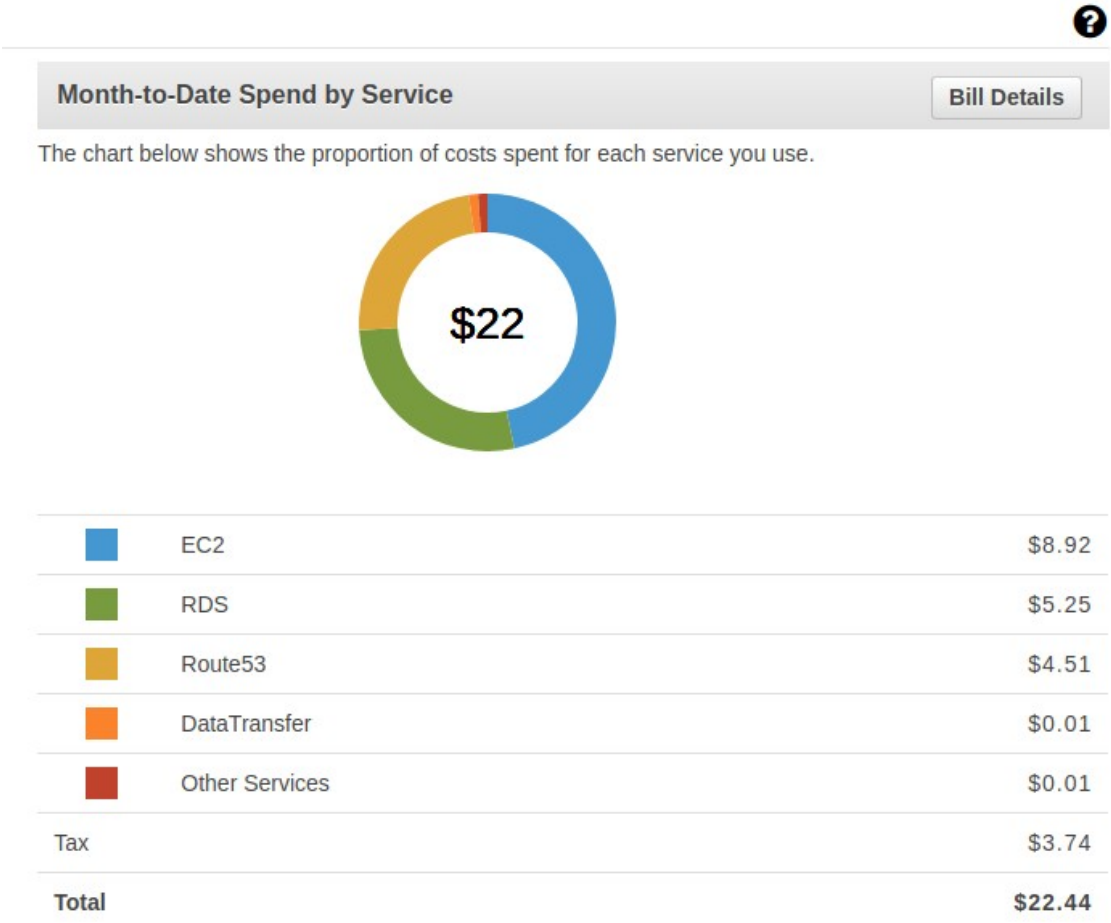
> For every complex problem there is an answer that is clear, simple, and wrong. ~ H. L. Mencken

The question here is whether you need the power of cloud computing and everything that comes with like IaaC (Infrastructure as a Code) or not.

If yes, the best question to ask here is what are the best practices to reduce my costs. If you are changing your infrastructure and using cloud infrastructure instead, but without changing your production and management models, you will certainly spend more than a cloud-ready application/team.

Let's get back to our main topic.

You can follow the usage of resources and the costs allocated to each service separately.



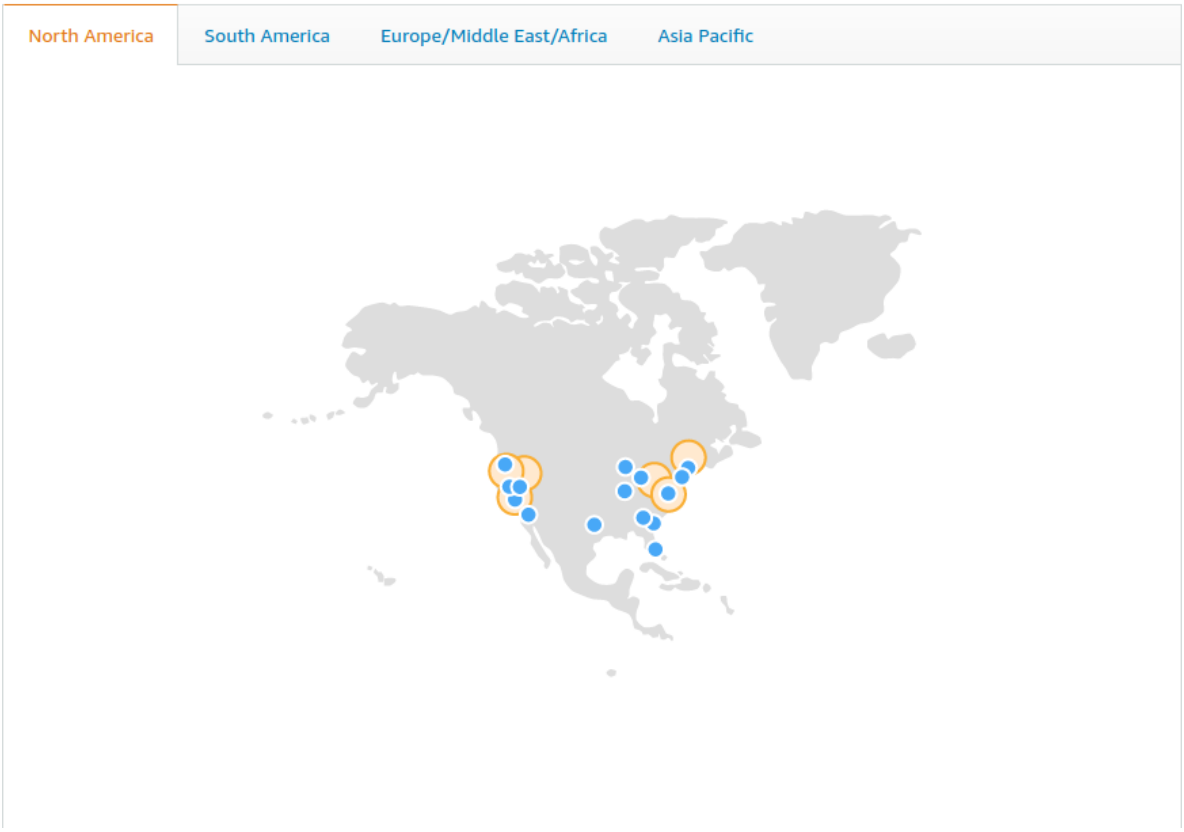
Using the same menu, you can access your account, organization, service quotas, orders and invoices, and your security credentials. Other useful links like "Support Center" and "Documentation" are available when you click on "Support".

AWS Regions

Let's imagine a scenario: You run an application for your customers based in China and Canada while your servers are installed in the UK. Even if your application is fast, you can not have control over the transport of your data packets. Customers from both countries will have delays in sending ingoing and receiving outgoing requests.

Cloud providers generally and AWS specifically offers the possibility of using physical resources in different data centers in the world. These are called regions.

Region Maps and Edge Networks



source aws.amazon.com

The orange points on the map are regions.

Most of AWS services can be used from different regions of the world. Each service has a regional endpoint.

e.g. EC2 service has 19 endpoints. These are some of them:

Region Name	Region	Endpoint	Protocol
US East (Ohio)	us-east-2	ec2.us-east-2.amazonaws.com	HTTP and HTTPS
US East (N. Virginia)	us-east-1	ec2.us-east-1.amazonaws.com	HTTP and HTTPS
US West (N. California)	us-west-1	ec2.us-west-1.amazonaws.com	HTTP and HTTPS
US West (Oregon)	us-west-2	ec2.us-west-2.amazonaws.com	HTTP and HTTPS
Canada (Central)	ca-central-1	ec2.ca-central-1.amazonaws.com	HTTP and HTTPS
Asia Pacific (Mumbai)	ap-south-1	ec2.ap-south-1.amazonaws.com	HTTP and HTTPS
Asia Pacific (Seoul)	ap-northeast-2	ec2.ap-northeast-2.amazonaws.com	HTTP and HTTPS
Asia Pacific (Singapore)	ap-southeast-1	ec2.ap-southeast-1.amazonaws.com	HTTP and HTTPS
Asia Pacific (Sydney)	ap-southeast-2	ec2.ap-southeast-2.amazonaws.com	HTTP and HTTPS
Asia Pacific (Tokyo)	ap-northeast-1	ec2.ap-northeast-1.amazonaws.com	HTTP and HTTPS
EU (Frankfurt)	eu-central-1	ec2.eu-central-1.amazonaws.com	HTTP and HTTPS
EU (Ireland)	eu-west-1	ec2.eu-west-1.amazonaws.com	HTTP and HTTPS

Region Name	Region	Endpoint	Protocol
EU (London)	eu-west-2	ec2.eu-west-2.amazonaws.com	HTTP and HTTPS
South America (São Paulo)	sa-east-1	ec2.sa-east-1.amazonaws.com	HTTP and HTTPS

You can check the long list here.

You can't access additional regions from an AWS account, such as AWS GovCloud (only US-West is available) or the China regions (only Chinese regions like Beijing and Ningxia are available).

Other regional endpoints are available for other services.

e.g. DynamoDB endpoint for Oregon (us-west-2) region is "https://dynamodb.us-west-2.amazonaws.com"

If a service supports regions, the resources in each region are independent: For example, if you create an Amazon EC2 instance or an Amazon SQS queue in one region, the instance or queue is totally independent of instances or queues in another region. You can't use the resources of one region in another one.

AWS Availability Zones

A region is made out of zones.

We have seen that AWS has regions and that each region is completely independent.

This is not the case for zones. A zone is isolated but still have a through low-latency link with other zones in the same region. To build an efficient infrastructure for your customers in a given region, having your application running in multiple AZs (availability zones) is a good practice. Your infrastructure has backups of computing, storage, and other types of resources in other AZs.

When a problem occurs in an AZ, other AZs will support the continuity of your production.

For example, "us-east-1" region has 3 AZs:

- us-east-1a
- us-east-1b
- us-east-1c

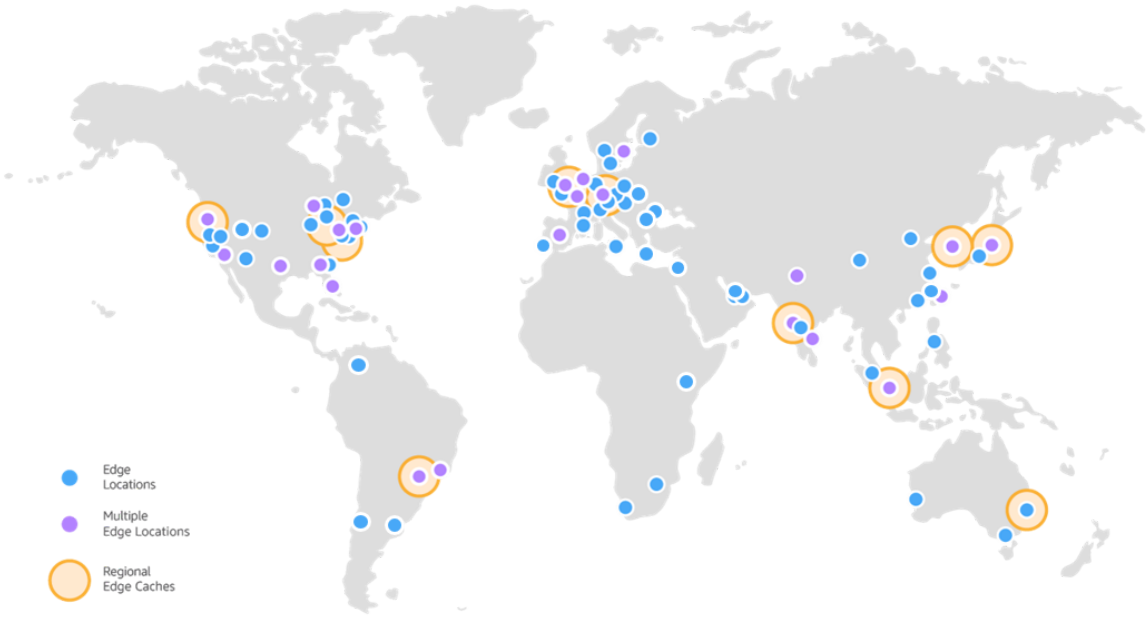
AWS Edges

Some services like AWS CloudFront use edge locations instead of zones and regions.

CloudFront is the CDN of Amazon Web Services with advanced features like authentication and programmatic access.

Amazon CloudFront has a global edge network of more than 200 points of presence in more than 80 cities. Some of these edges are considered as "edge locations", and others are considered as "regional edge caches".

Cache edges are a way to accelerate delivering content using CloudFront and reduce the workload on origins to help you increase the availability of your applications.



source: aws.amazon.com

Lesson 3 - Create your First EC2 Virtual Machine

What is a Virtual Machine?

Let's say you have a computer with a good hardware configuration: (e.g., 16GB DDR3, SSD drives, and a very good). You want to use this machine as a server, so you connect it to the Internet, you already have a static IP address, and you will host your customer small business' websites using the LAMP stack (Linux, Apache, MySQL, PHP).

This server, with its actual configuration (16GB DDR3 memory ..etc.), could probably host hundreds if not thousands of small websites; you can actually use it for many customers. However, each customer would love to have an isolated environment with his own LAMP stack, files, and configurations. This is when virtualization helps in the pooling of physical resources.

Virtualization is a way to partitioning one physical server into several virtual servers, or machines. These machines are called guests and will share the same hardware as the host server.

Every virtual machine acts like a "normal" machine; it can be accessed separately, it can have an IP address ..etc.

Using a hypervisor, a system engineer could create multiple machines using a single physical one.

- VMware Workstation
- VMware Player
- VirtualBox
- Parallels Desktop for Mac,
- QEM,
- Xen,
- Oracle VM Server for SPARC,
- Oracle VM Server for x86,
- Microsoft Hyper-V
- VMware ESX/ESXi

All of the above are examples of hypervisor technologies used in the IT industry. Virtualization saves money and resources and makes software installation and system administration easier.. and if you care about the environment, it can be considered an eco-friendly technology.

Virtualization in AWS

AWS has data centers around the globe, working together to deliver a global cloud computing service. The data centers are no more than physical machines. To create VMs, Amazon uses Xen virtualization technology.

To quote the AWS Security Whitepaper, Amazon EC2 currently utilizes a highly customized version of the Xen hypervisor, taking advantage of paravirtualization (in the case of Linux guests).

Amazon Machine Images (AMI) has two virtualization types; HVM and PV.

HVM is for "Hardware Virtual Machine", and PV is for "Paravirtualization".

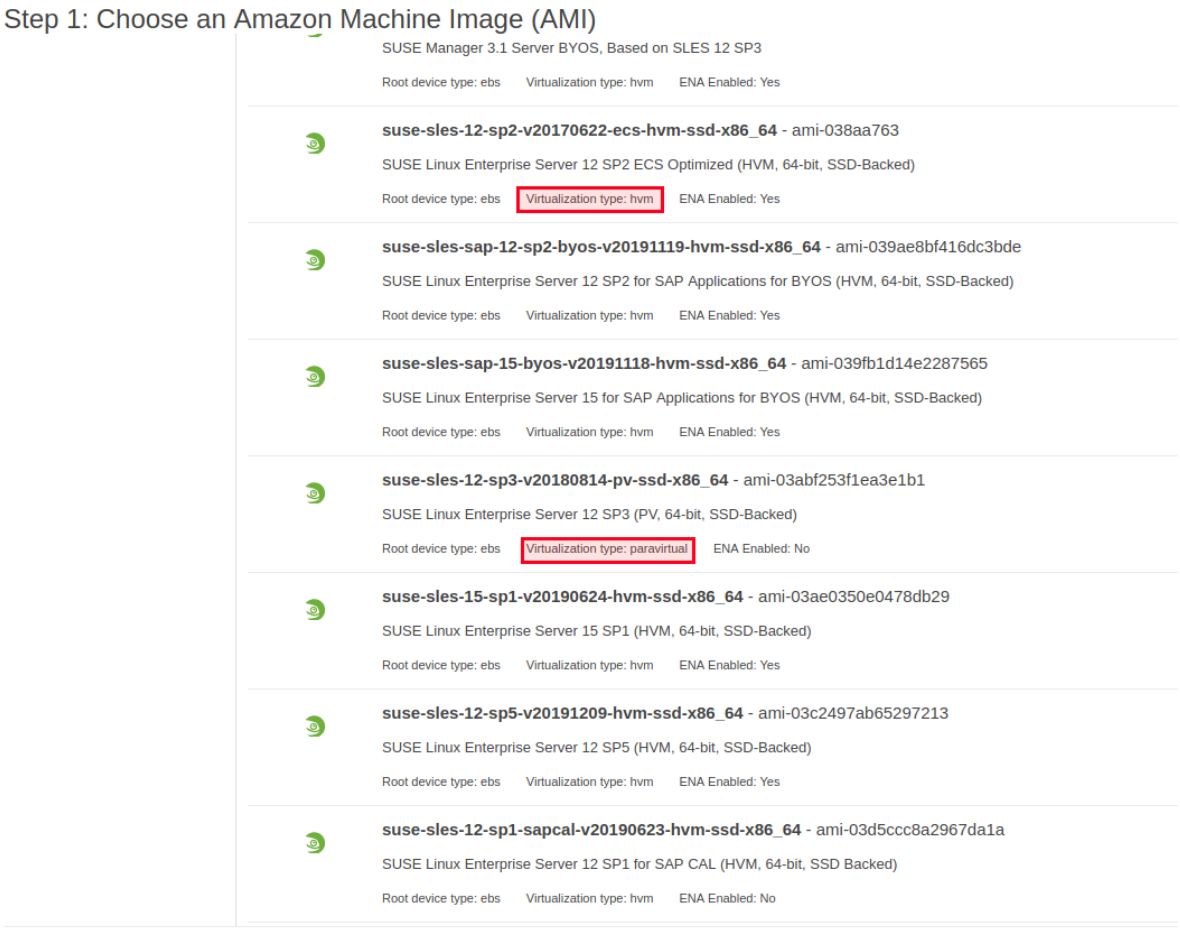
HVM is a full virtualization type, which means that all VMs running on top of the hypervisors are not aware that they are sharing the same hardware.

Paravirtualization, on the other hand, is lighter than HVM. In the case of paravirtualization, the guest machine requires modifications to run properly.

Traditionally, HVM guests had to translate I/O instructions to emulated hardware, which create another overhead.

Historically, PV guests had better performance than HVM guests and performed better with storage and network operations than HVM guests because they could leverage special drivers for I/O. This is no longer true. HVM virtualization has seen many enhancements, such as the availability of PV drivers for HVM AMIs (Amazon Machine Images).

So, Amazon Machine Image (AMI), which are the images that provide the information required to launch a virtual machine on AWS cloud, can use either HVM or PV. When creating a virtual machine on AWS, I recommend using HVM machines even if you can find some PV AMIs.

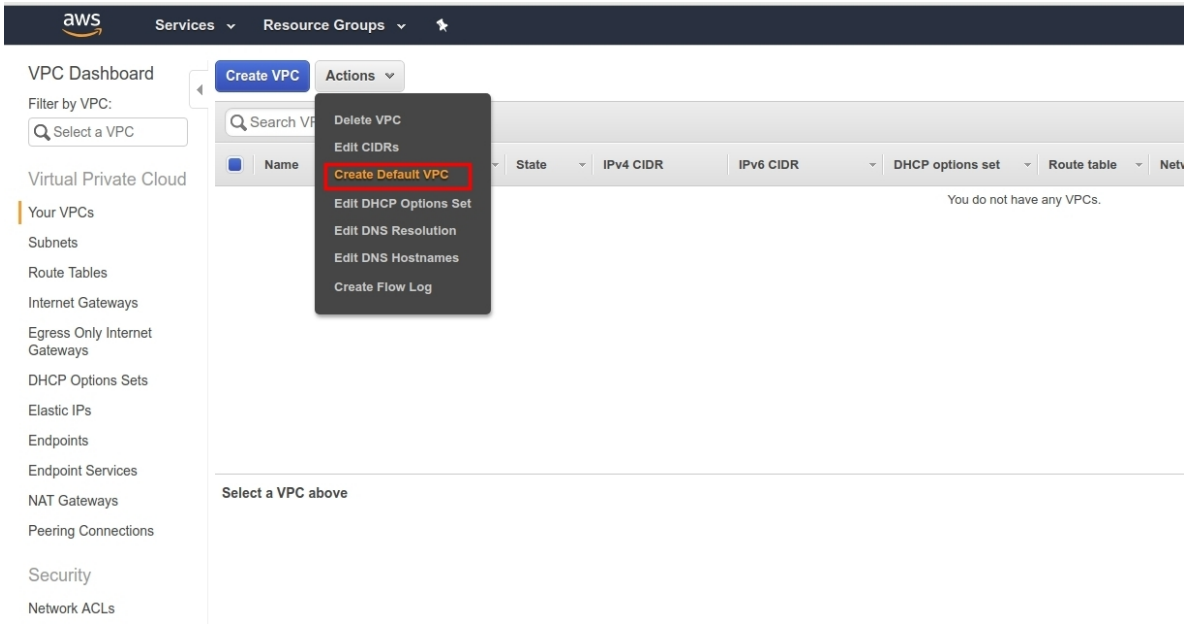


Creating Virtual Machines Using AWS EC2 Console

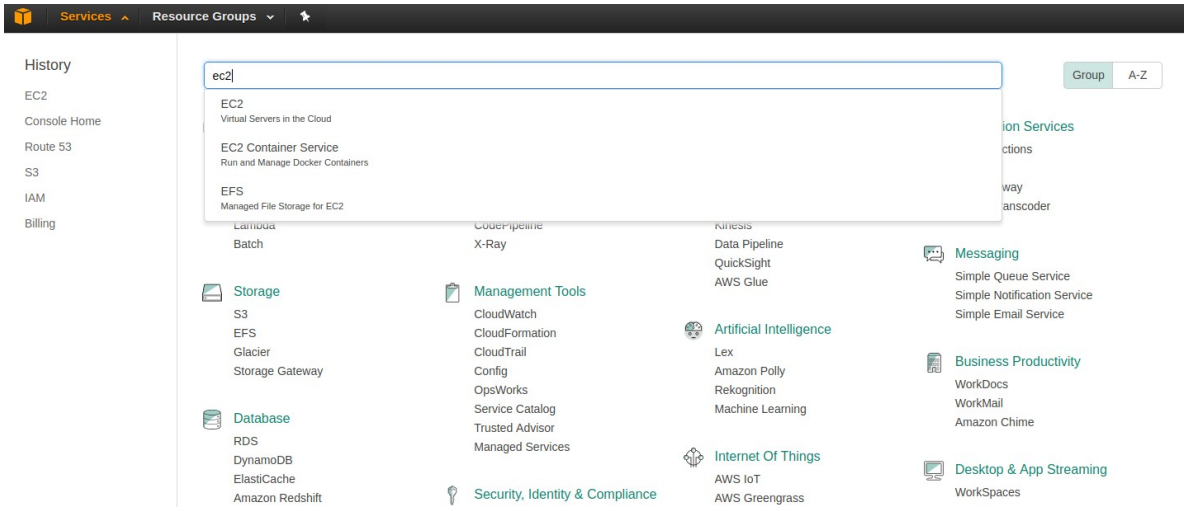
EC2 or Elastic Compute Cloud is the commercial name for the virtual machine you can use in AWS cloud service. We are going to create a machine using the web console.

Our goal is to create a web server using NGINX. This is a quick start; some concepts will be detailed in the next sections. You need to go to aws.amazon.com and type your login and password or create a new account if it is not already done.

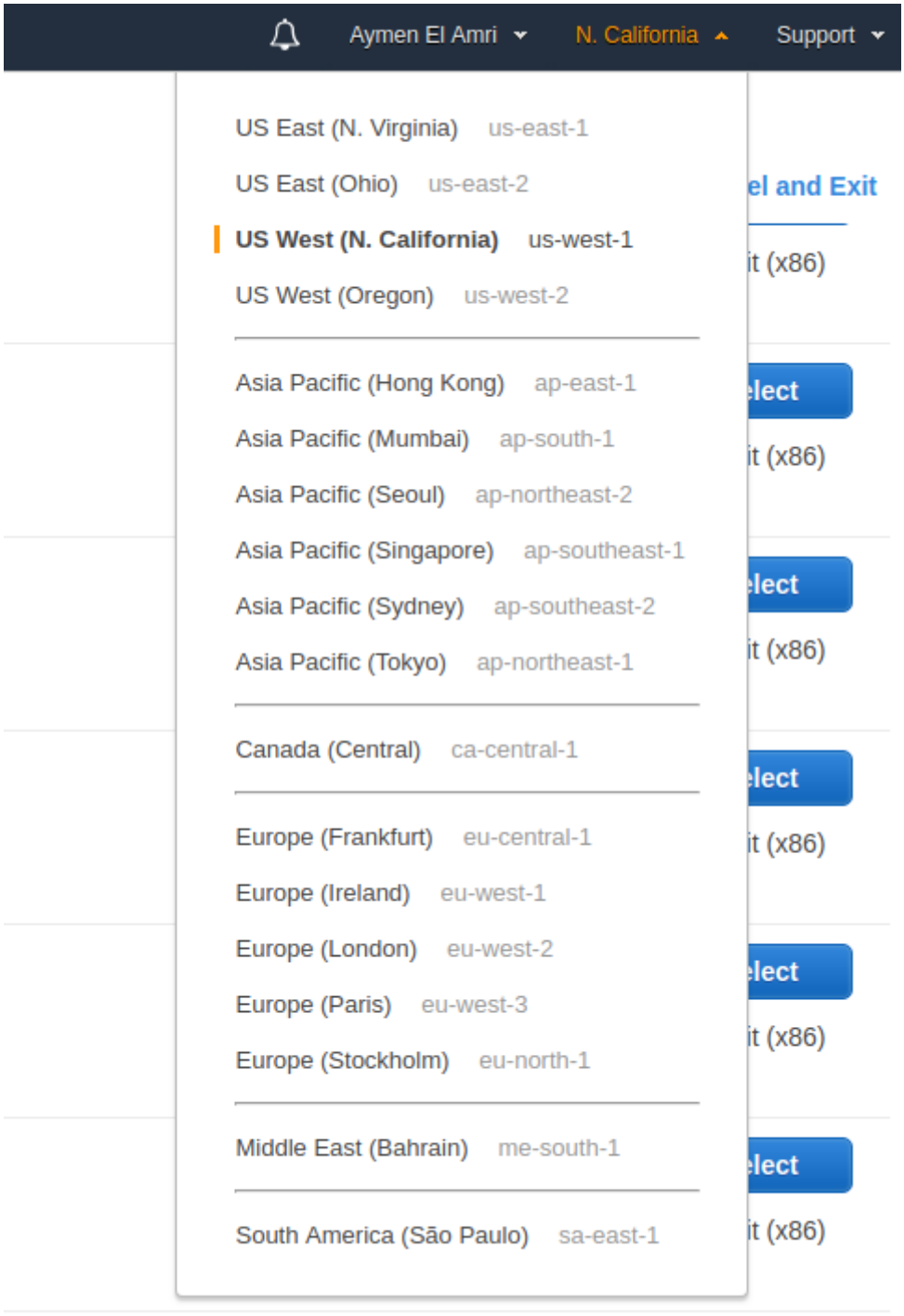
For the sake of simplicity, I will be using the AWS default VPC. We are going to learn more about VPC in this course. If you already deleted it, you can restore it by going to your VPC configuration, then create a new one:



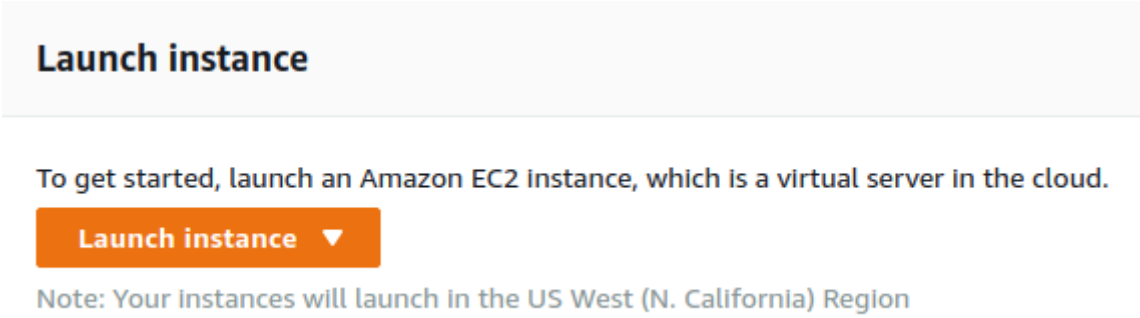
You can find the link to access EC2 under the Compute menu, or you can search for it by simply typing "ec2".



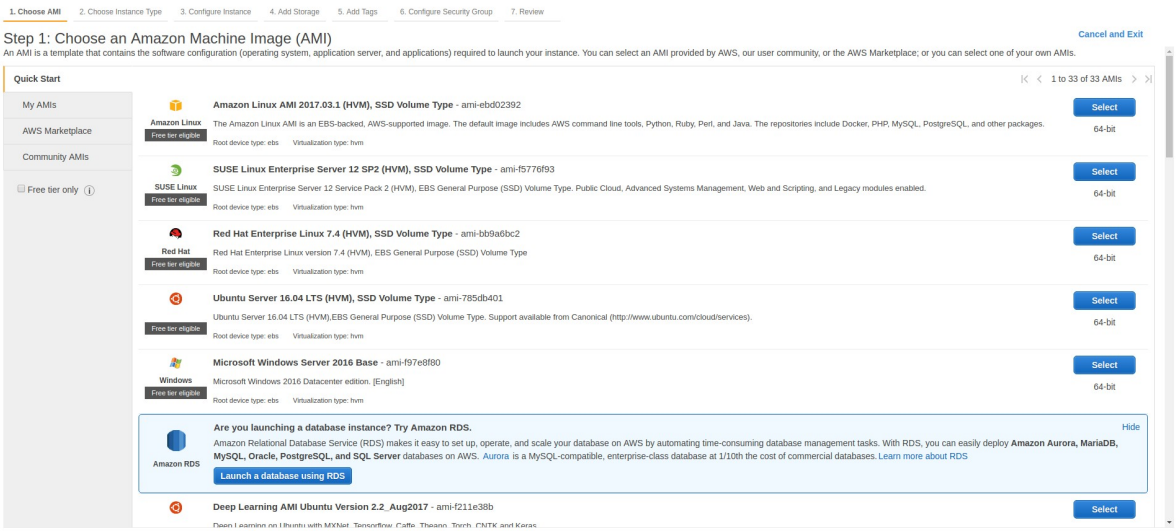
Now you should choose the region you want to use for your EC2 virtual machine. I used to use Ireland as a region since it is the cheapest nearest region to me. This may not be the case for you, but you can see more details on pricing here.



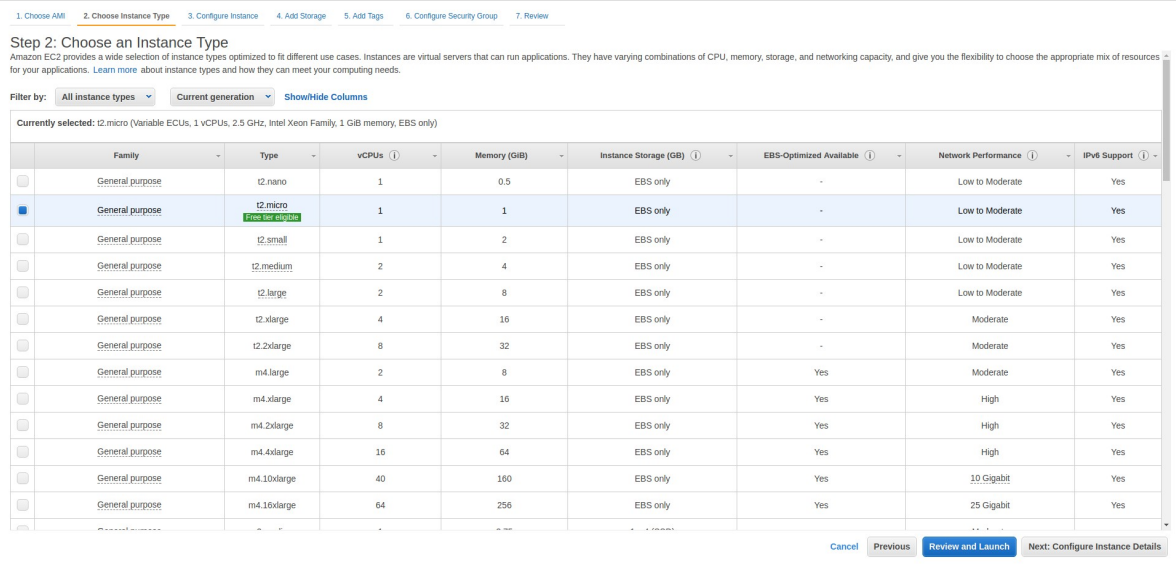
Click on "Launch Instance":



This will take you to this screen where you should choose the operating system to use:

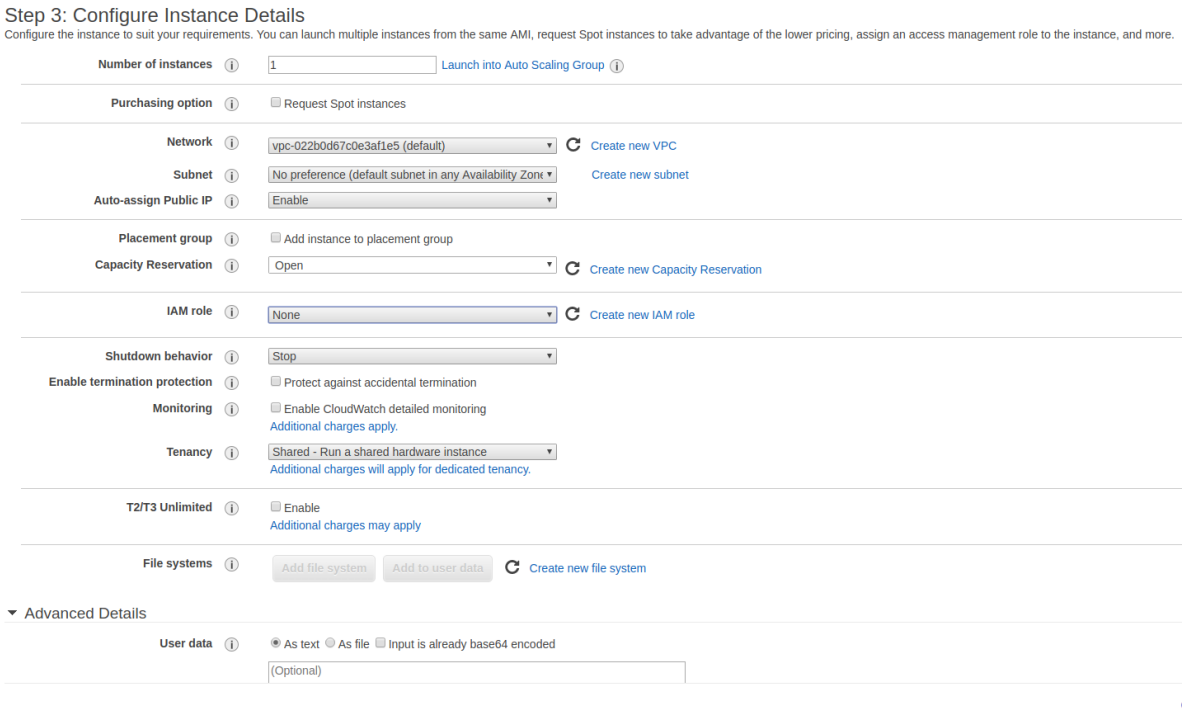


I am going to use Ubuntu, choose your own OS and click on "Select". You will be redirected to the second screen where you should use the machine size.



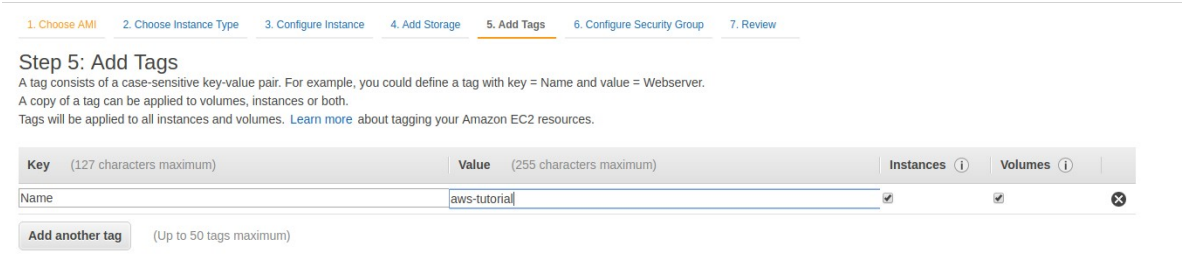
Now you can use the t2.micro instance (Free tier eligible).

After clicking on "Next", you will be asked to provide the configuration for your virtual machine. Keep everything as it is and enable "Auto-assign Public IP".



Click on "Add Storage". Keep everything as it is for the moment and click on "Add Tags".

Click on "click to add a Name tag" and add "aws-tutorial" as "Name".

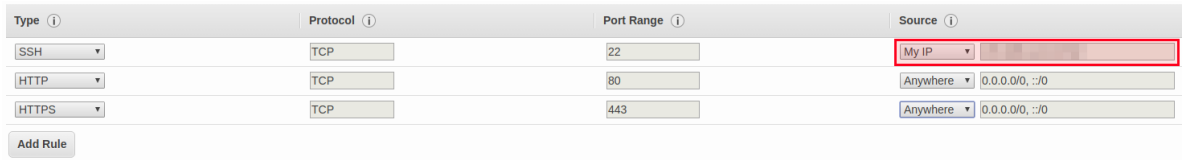


Choose "Next" and click on "Create a new security group", add a security group name (I name it "aws-tutorial-sg") and the description you want (add a description that will help you to remember why this security group was created, if you can delete it later if it is used for a production/testing server .. etc.), then allow SSH, HTTP & HTTPS from "everywhere":

In "source", choose "custom" and add `0.0.0.0/0`.

Adding `0.0.0.0/0` as a source allows all IP addresses to access your EC2 machine. This should be used carefully, especially with ports like SSH.

If you and only you will access the machine using SSH, allow only your IP address:

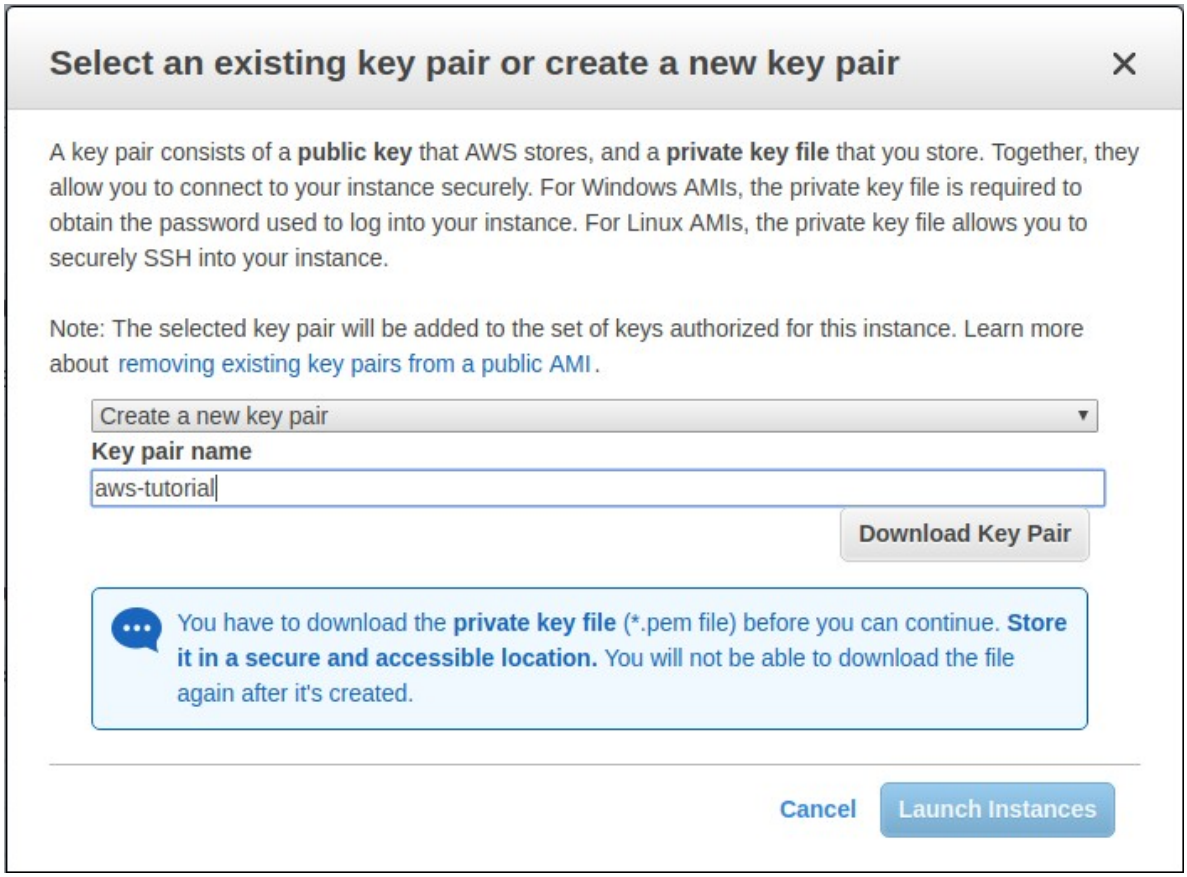


If you want to allow a block of IP addresses. For example, the range from 176.185.143.0 to 176.185.143.255 (256 in all), you can do it using:

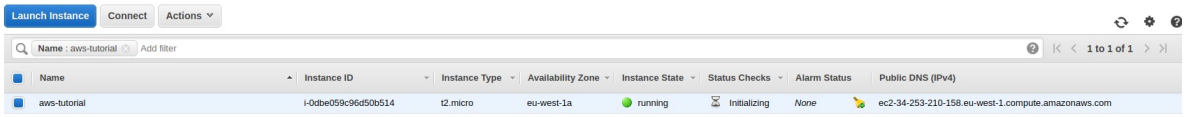
```
176.185.143.0/24
```

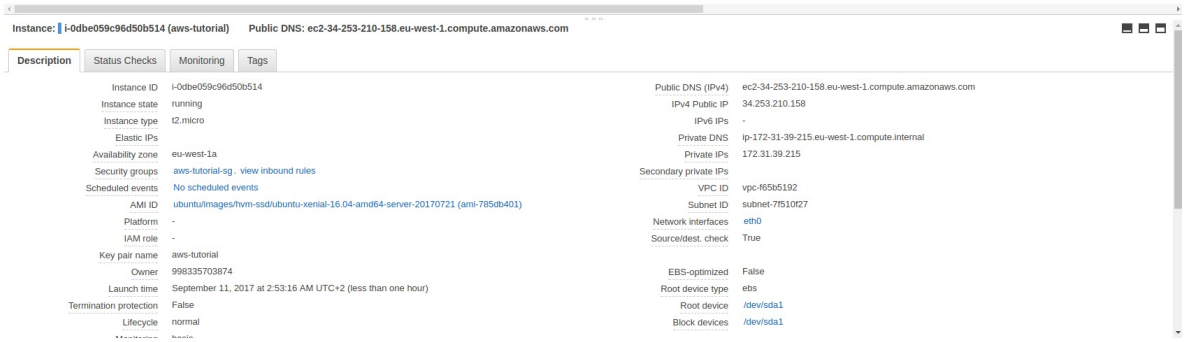
"176.185.143.0/24" is the CIDR, and it uses "255.255.255.0" as a netmask (or subnet mask). You can change your netmask to adapt the allowed addresses to your context. For example, "176.185.143.0/30" only allows "176.185.143.0", "176.185.143.1", "176.185.143.2" and "176.185.143.3". If you are not familiar with CIDR, you should study this in case you need it; you can also use online CIDR calculators.

Click on "Review and Launch" then on "Launch". If this is the first time you use AWS and EC2, you will be asked to create a key pair that will allow us to ssh into the virtual EC2 machine.



Give your key pair a name and download it to keep it in a safe place. I usually move this to my home folder, and this is what I am going to do in this tutorial. When you finish downloading the key pair, you can click on "Launch Instances" then on "View Instances".





Wait for the initialization to finish and right-click on the instance, then click on "Connect". You will see this screen with the SSH command you should use in order to get into the machine.

If you are using Linux, Mac OS, or Ubuntu on Windows, you can use your terminal with the given command.

If you are using Windows, Putty is a known and widely used SSH client that you can use, but we recommend using Bash.

The command used to SSH into the machine looks similar to this:

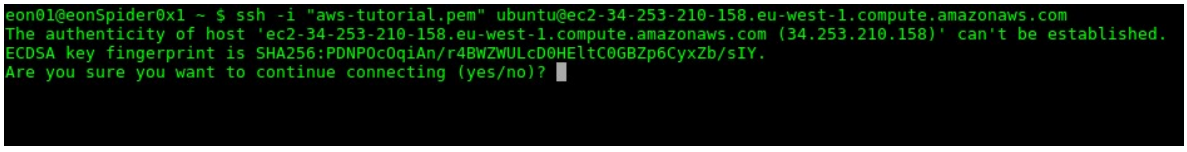
```
ssh -i "aws-tutorial.pem" ubuntu@ec2-34-253-210-158.eu-west-
1.compute.amazonaws.com
```

- "aws-tutorial.pem" is the certificate name or path that allows us to ssh to the created machine
- "ubuntu" is the username
- "ec2-34-253-210-158.eu-west-1.compute.amazonaws.com" is the hostname

Every OS has a default username that should be used. Here is a list of some of them:

Distribution	Ssh Username
Amazon Linux	`ec2-user`
Ubuntu	`ubuntu`
Debian	`admin`
RHEL 6.4 & later	`ec2-user`

I moved the key pair from my "Download" folder to my "Home " folder, and I executed the command given by AWS connection popup.



You can probably get an error message similar to this one:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@         WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions 0644 for 'aws-tutorial.pem' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "aws-tutorial.pem": bad permissions
Permission denied (publickey).
```

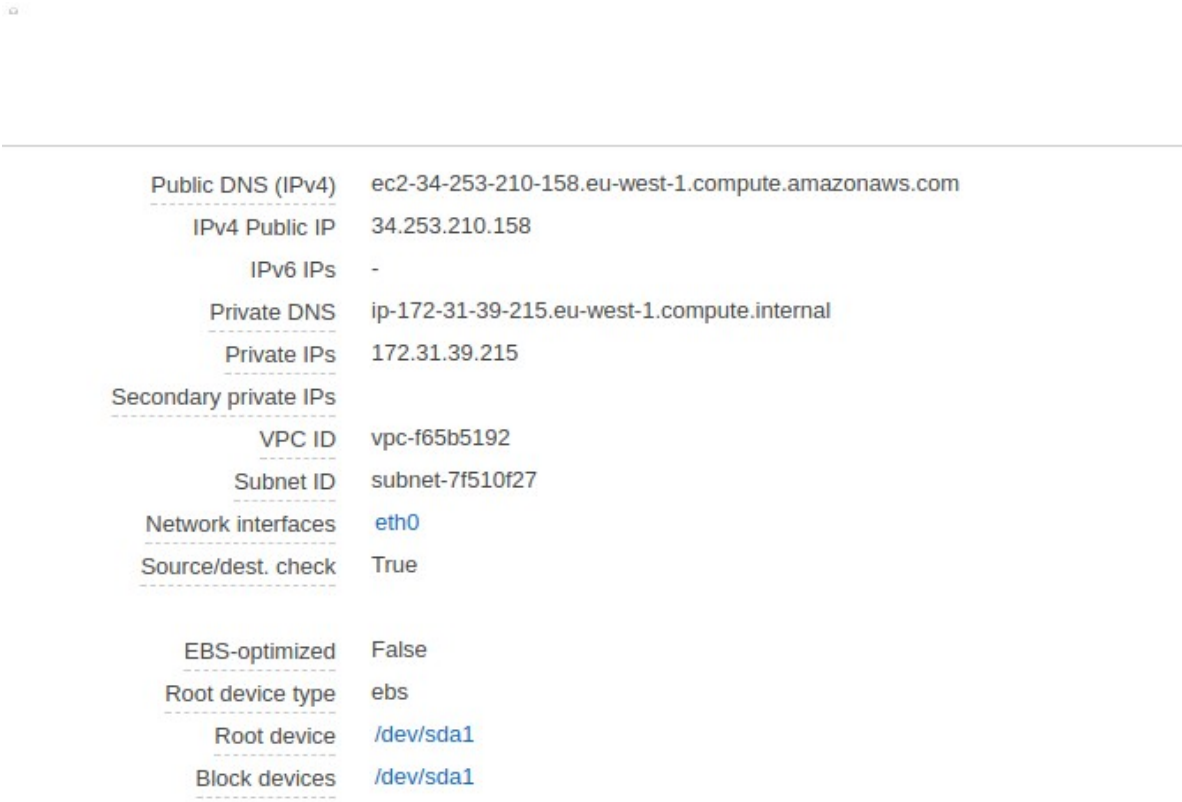
This is a permission problem that could be fixed using chmod:

```
chmod 0400 aws-tutorial.pem
```

Once we made an SSH into the created machine, we can start installing NGINX:

```
sudo apt-get update && sudo apt-get install -y nginx
```

Now let's get back to the console and check the public IP address that our machine is using. We can see this is the web dashboard:



Our public IP is

```
34.253.210.158
```

When typing this in my browser, I can see that NGINX was successfully installed:

You can do the same using the IP address that AWS assigned to your virtual machine and check if everything is working fine!

Troubleshooting

As this is one of the introductions to use AWS, you may find some problems in creating the first NGINX EC2 machine. Let's understand why:

SSH Issues

Sometimes, you may have problems accessing your machine using SSH. The first step to do here is activating the debug on your SSH command using `-vvv` :

```
ssh -vvv -i "aws-tutorial.pem" ubuntu@ec2-34-253-210-158.eu-west-
1.compute.amazonaws.com
```

Your problem may come from the fact that the key file (in our case, the aws-tutorial.pem file doesn't have the right permissions). Make sure to execute:

```
chmod 400 ~/.ssh/aws-tutorial.pem
```

This should be done after copying the downloaded pem file to the above folder:

```
cp $HOME/Download/aws-tutorial.pem $HOME/.ssh/
```

Also, your machine may not be accessible from the outside, and in this case, whether you set the right or the wrong permissions on your key, it will remain inaccessible.

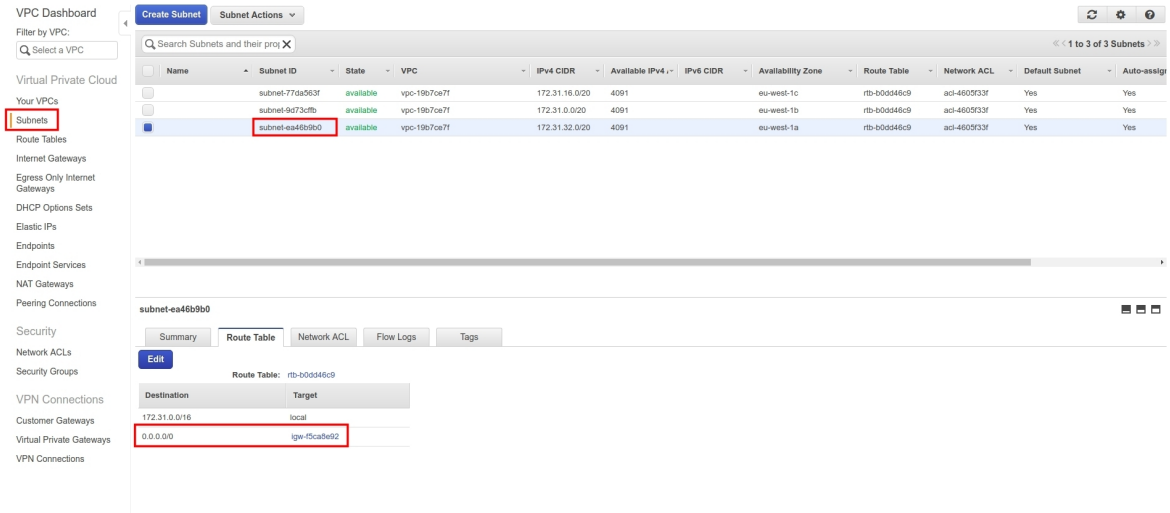
Your Machine is not Accessible From the Outside

To use NGINX on a VM, make sure that it is accessible from the outside. In the first example, we used the default VPC. Normally, when attaching a public IP to the EC2 machine, it should be accessible from the outside, but you had probably changed some of your default VPC configurations.

What is important to us is the networking part. The subnet that you used to create the EC2 machine must have a valid routing table.

Say we made a choice to use the subnet with the id `subnet-ea46b9b0` when we created our EC2 instance. Make sure that the selected subnet has access to the Internet by checking its Route Table. The Route Table should be at least using one Internet Gateway to reach the Internet.

Make sure that the destination is `0.0.0.0/0` and that the target is a valid Internet Gateway:



Later in this course, we are going to see in detail some of the concepts introduced in this part.

Your Security Groups

Everything is good, but you can't ssh to your machine? You can use SSH and install NGINX, but you don't see the NGINX welcome page?

Our NGINX VM should be accessible on port 80 (HTTP) and port 443 (HTTPS). Our SSH port (22), should be at least accessible from your IP address (the public IP address of your home or work).

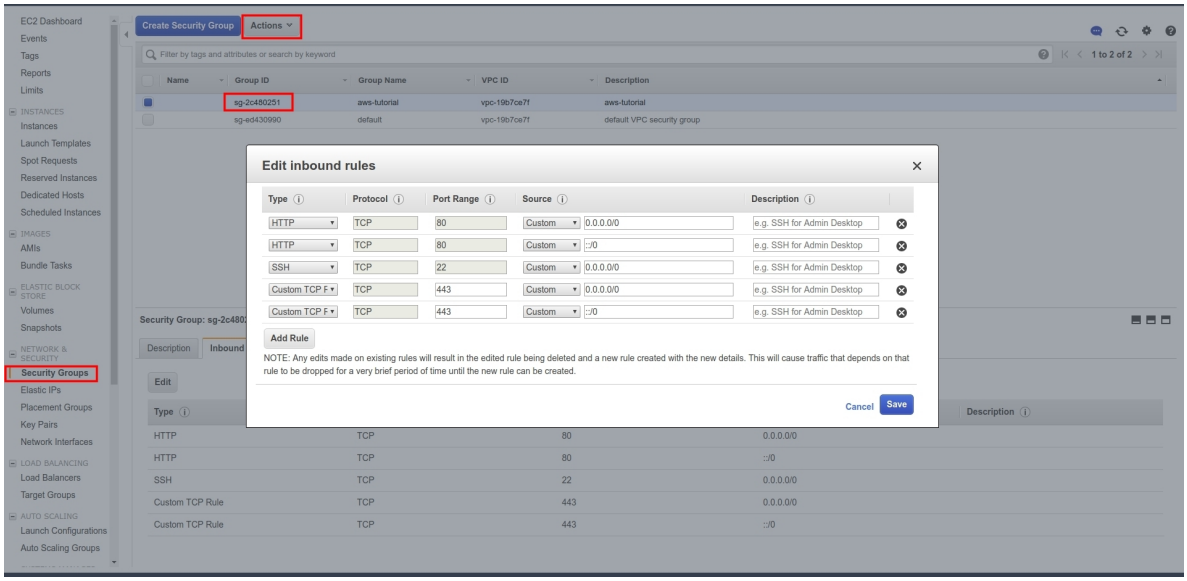
For the sake of simplicity, we allowed SSH from everywhere (0.0.0.0/0 for the IPV4 and ::/0 for the IPV6).

(Note that, even if we did it on a temporary machine we use to learn, allowing SSH from everywhere is not a good security practice)

To troubleshoot this, go back to the used Security Groups and make sure that you allowed:

- HTTP from everywhere
- HTTPS from everywhere
- SSH from your IP address or from everywhere (or your IP address range)

Go to your EC2 dashboard, select "Security Groups", click on the created one, and click on "Actions" then "Edit" the inbound rules.


EC2 Dashboard
Events
Tags
Reports
Limits
INSTANCES
Instances
Launch Templates
Spot Requests
Reserved Instances
Dedicated Hosts
Scheduled Instances
IMAGES
AMIs
Bundle Tasks
ELASTIC BLOCK STORE
Volumes
Snapshots
NETWORK & SECURITY
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
LOAD BALANCING
Load Balancers
Target Groups
AUTO SCALING
Launch Configurations
Auto Scaling Groups
Create Security Group
Actions
Filter by tags and attributes or search by keyword
1 to 2 of 2
Name
Group ID
Group Name
VPC ID
Description
sg-2c480251
aws-tutorial
vpc-19b7ce7f
aws-tutorial
sg-ed430990
default
vpc-19b7ce7f
default VPC security group
Edit inbound rules
Type
Protocol
Port Range
Source
Description
HTTP
TCP
80
Custom
0.0.0.0/0
e.g. SSH for Admin Desktop
HTTP
TCP
80
Custom
::/0
e.g. SSH for Admin Desktop
SSH
TCP
22
Custom
0.0.0.0/0
e.g. SSH for Admin Desktop
Custom TCP F
TCP
443
Custom
0.0.0.0/0
e.g. SSH for Admin Desktop
Custom TCP F
TCP
443
Custom
::/0
e.g. SSH for Admin Desktop
Add Rule
NOTE: Any edits made on existing rules will result in the edited rule being deleted and a new rule created with the new details. This will cause traffic that depends on that rule to be dropped for a very brief period of time until the new rule can be created.
Cancel
Save
Security Group: sg-2c480
Description
Inbound
Edit
Type
Protocol
Port Range
Source
Description
HTTP
TCP
80
0.0.0.0/0
HTTP
TCP
80
::/0
SSH
TCP
22
0.0.0.0/0
Custom TCP Rule
TCP
443
0.0.0.0/0
Custom TCP Rule
TCP
443
::/0

Lesson 4 - Back to The Basics

Introduction

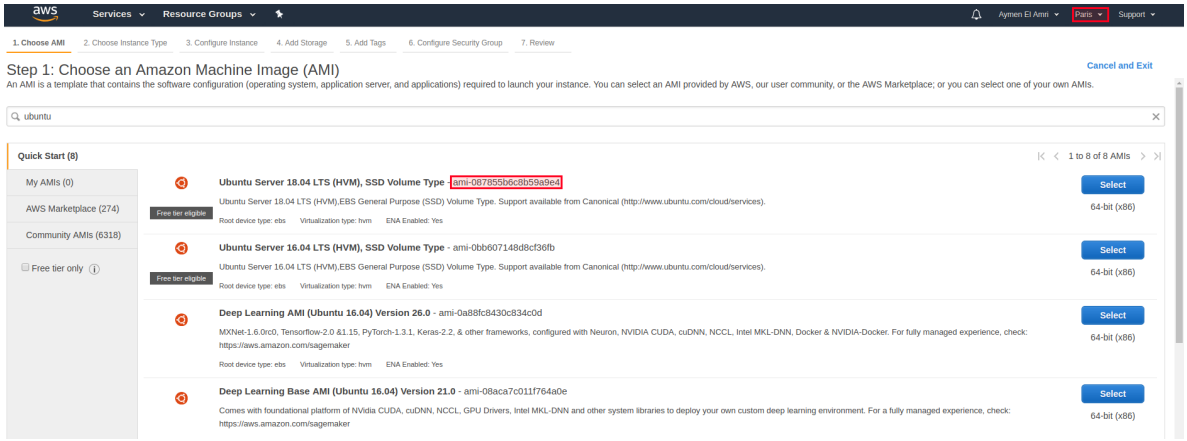
We have seen together how to create an EC2 machine using the web UI, and we followed some steps during which we chose the AMI, instance type, tags, security groups ..etc.

In this part, we are going to explain the concepts which we covered in a hurry.

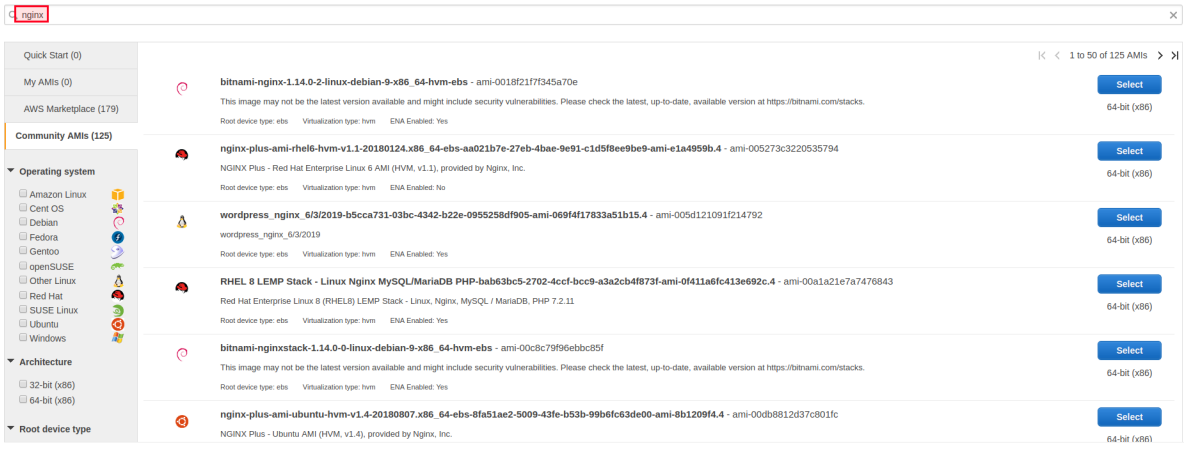
AMI: Amazon Machine Images

Amazon Machine Image or AMI is a virtual appliance used by AWS to launch a machine. An AMI is a read-only system that provides some required information to start an EC2 instance. It includes a template defining the operating system to use, and in addition to that, it can define an application.

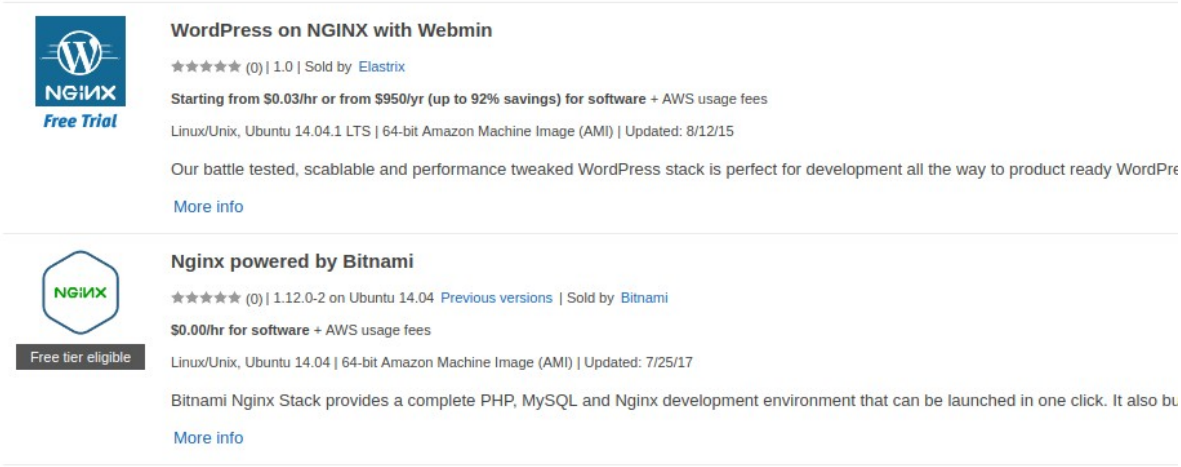
Let's say we would like to use the official 64-bit (x86) Ubuntu 18.04 image on a virtual machine in Paris. In this case, the AMI we should use is identified by "ami-087855b6c8b59a9e4".



Previously, we installed Nginx on an EC2 using the official Ubuntu AMI, but we can find some AMIs where Nginx is already installed. We just need to search for it at the moment of choosing the AMI (first step of creating an EC2 machine):



Some AMIs are paid, and some are free.



If you want to use a free Nginx AMI, make sure that there are no additional fees. In the screenshot above, using the first AMI will cost you AWS fees + the software fee ($0.03 USD per hour).

Instance Types

AWS offers 5 types of EC2 instances:

- General Purpose
- Compute Optimized
- Memory Optimized
- Accelerated Computing
- Storage Optimized

You should choose the best for your use case.

General Purpose:

e.g. A1, T3, T3a, T2, M6g, M5, M5a, M5n, M4

For mainstream applications, these instances are the best choice since they offer a balance of compute, memory, network capabilities, and cost.

T2 instances are optimized for mid-size websites, web applications, artifact/development/build/test/staging environments, and microservices.

M4 instances could be used for mid-size databases and data processing applications.

Compute Optimized:

e.g C5, C5n, C4

Compute optimized instances offers the highest performing processors and the lowest price per compute performance in EC2.

Compute optimized instances are a good fit for high computational scientific operations, compute-intensive applications, MMO gaming, batch processing, distributed analytics, high traffic websites, and video-encoding.

Memory Optimized

e.g., R5, R5a, R5n, R4, X1e, X1, High Memory, z1d

Memory optimized instances are designed to deliver fast performance for workloads that process large data sets in memory.

While X1 instances are a good fit for in-memory databases and have the lowest price per GiB of RAM among Amazon EC2 instance types, R4 instances are a good fit for high performance/realtime databases, data mining applications, large scale in-memory caches and Hadoop/Spark clusters.

R4 instances offer better prices per GiB of RAM than R3.

Accelerated Computing :

e.g P3, P2, Inf1, G4, G3, F1

This type of instance has parallel processing capabilities that give it the ability to host high-performance databases, large-scale machine learning, and different types of server-side GPU compute workloads.

While P2 instances are a good fit for general-purpose GPU compute applications, G3 instances are more optimized for graphics-intensive applications like 3D visualizations, and F1 instances offer customizable hardware acceleration with field-programmable gate arrays (FPGA) and works better for real-time video processing, Genomics, analytics and other complex types of computing.

Storage Optimized

e.g., I3, I3en, D2, H1

I3 and D2 instances are designed for storing and processing data. They offer low-cost storage.

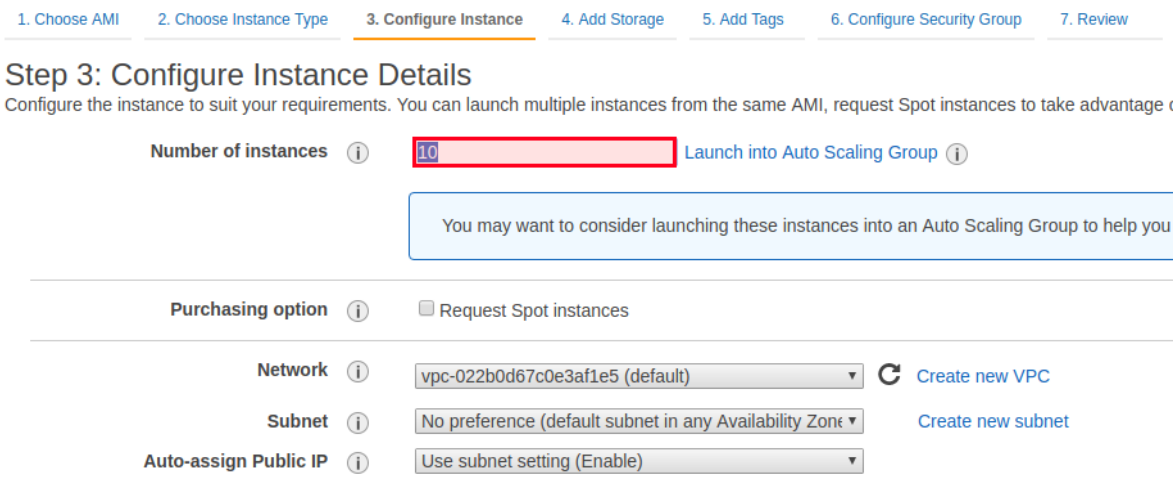
I3 instances have a high I/O performance and are intended to run large NoSQL databases, Elasticsearch, and in-memory databases.

D1 instances are dense-storage instances and are intended to run MapReduce and Hadoop distributed computing, distributed file systems, network file systems, or log or data-processing applications.

Instance Details

Number of Instances

Using the step 3 of EC2 creation, you can choose more than 1 instance. If you create 2 or more instances, they will share things like the network, subnet, IAM role ..etc.



You can request an ordinary instance or a spot one.

Purchasing Option

EC2 Spot instance allows you to bid on spare Amazon EC2 computing capacity in order to reduce the cost of your running applications using the same budget. This can save you up to 80-90% of costs.

This type of instance runs when your bid price exceeds the spot price.

When to Use Spot Instances

So EC2 spot instance is cheap, but they are not always suitable to use in a production environment because there are no SLA guaranteed by AWS. This means that if something happens to a spot instance, it is your responsibility.

However, in some use cases, it is possible to benefit from the advantages that spot instances offer. Some applications can be fault-tolerant and flexible, like CI/CD, data processing, and computing back-ends that save data constantly to persistent external storage.

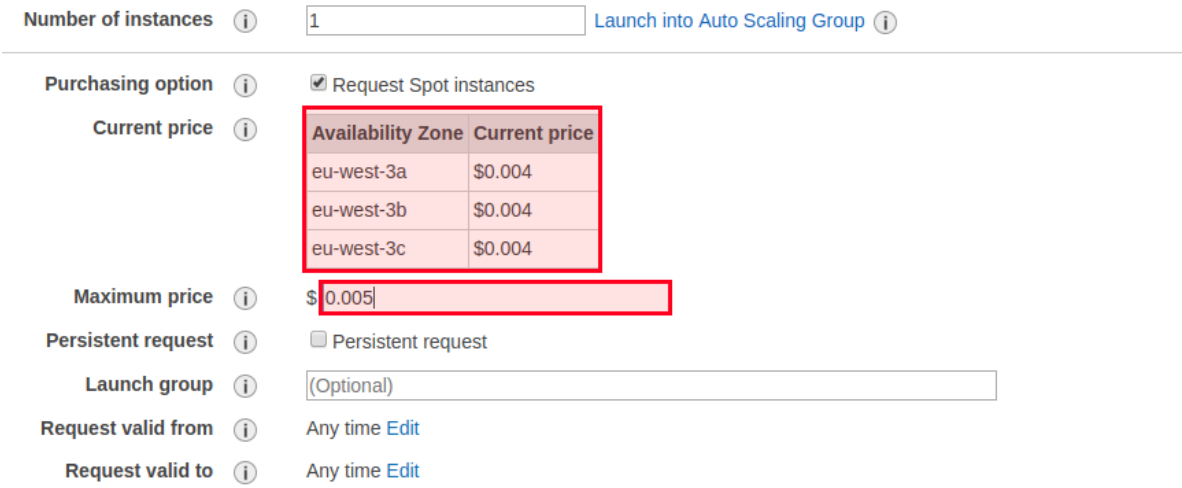
In other words, if you plan to use spot instances, you must have a flexible production architecture with a fault-tolerant system, which is usually another spot instance, in the same cluster, that treat the workload when the first one is not responding.

Stateless applications are a good use case. REST APIs are an example of an application that can be built in a stateless way. Image processing, parallel computing, analytics processing, and big data are also other suitable uses cases.

Note: An application is stateless when the server side does not store any state about the client session. Instead, the session data is stored on the client-side and passed to the server when needed.

Spot instance is not good to run sensitive workloads or databases, but you can use them to follow this course.

When creating an EC2 instance, you will be able to see the actual spot prices. When the maximum price per hour for your request exceeds the spot price, Amazon EC2 fulfills your request if capacity is available.



Note that the prices fluctuate depending on supply and demand, and can vary by region and availability zone.

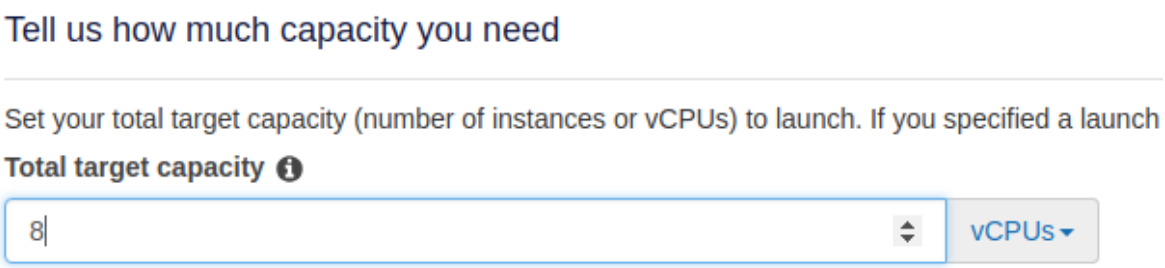
Spot Fleet

A spot fleet is a collection of spot instances. To create one, use: https://console.aws.amazon.com/ec2sp/

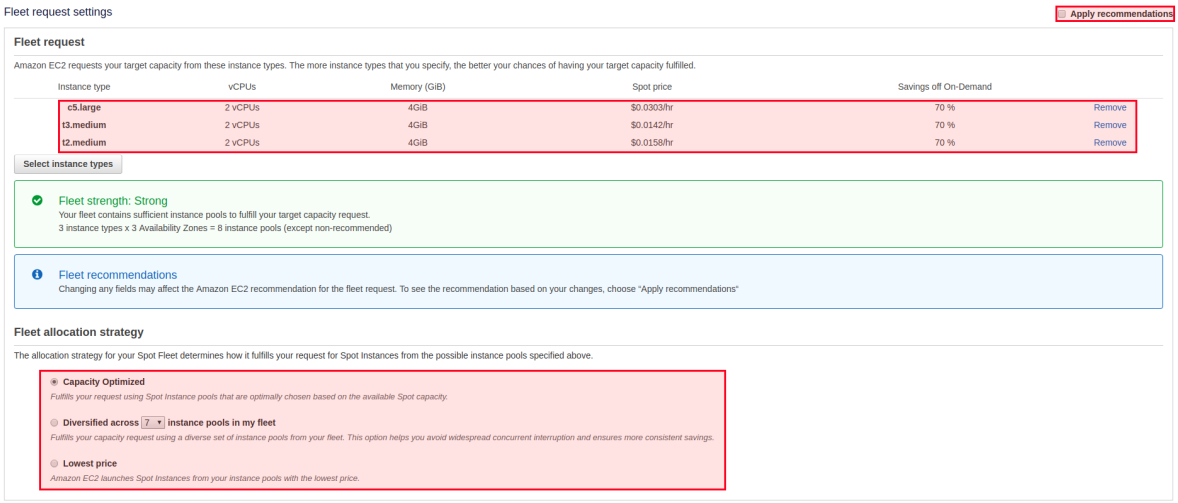
It allows to launch a number of spot instance and maintain the target capacity that you specify in advance. So when a spot instance is interrupted, the fleet will try to launch a replacement.

The spot fleet allows you to decide on certain behaviors of your spot instances. You can, for instance, decide the default action to do when your maximum price has been exceeded. The default is to terminate the instance. You can decide just to stop your instance. This will incur more charges because your spot instances states are saved, but it could be useful in many cases to choose this behavior over termination. Note that you need to use an EBS-backed AMI to be able to stop spot instances.

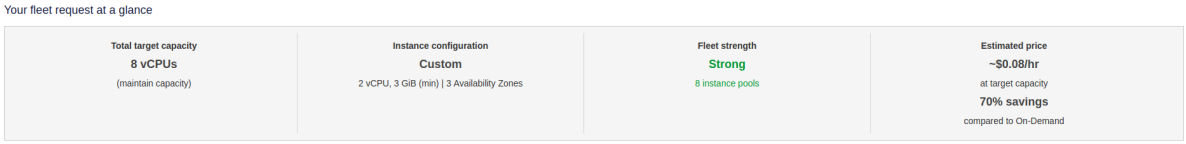
It is also possible to choose other configurations for your fleet like the target capacity to maintain in term of machines of vCPUs:



It is also possible to decide which instances types, your Amazon EC2 requests your target capacity from. A good practice here, if you don't have special requirements, is to apply Amazon recommendations. According to AWS, the more instance types that you specify, the better your chances of having your target capacity fulfilled.



In the end, you will be able to evaluate your fleet and check its availability and cost:



Network

When creating an EC2 instance, you can choose which network to use. When you start a VM, you should choose your Amazon Virtual Private Cloud (VPC). This VPC defines your machines' IP addresses range, subnets, route tables, network gateways, and other networking configurations.

If you don't choose a VPC, your AWS account comes with a default VPC that will be chosen automatically.

Subnet

After choosing your VPC, you will get a list of the available subnets within your VPC, and you can choose one of them.

A subnet is the range of IP addresses in your VPC that can be used to isolate EC2 resources from each other or from an external network (e.g., Internet).

Each subnet resides in an availability zone, and an availability zone can not have multiple subnets. By default, your machine can be in a single subnet, which is the most common use case: One network interface, one IP address.

A subnet can be public (accessible by anyone) or private (accessible by only those who are authorized).

An EC2 Instance / Multiple Subnets

An EC2 machine may have multiple network interfaces or what we call ENI (Elastic Network Interface).

However, it is not possible all the time. Let's see some examples. Start with this scenario:

We have a VPC with 3 availability zones, which systematically means that you have 3 subnets and that each subnet has an IPv4 CIDR.

Example:

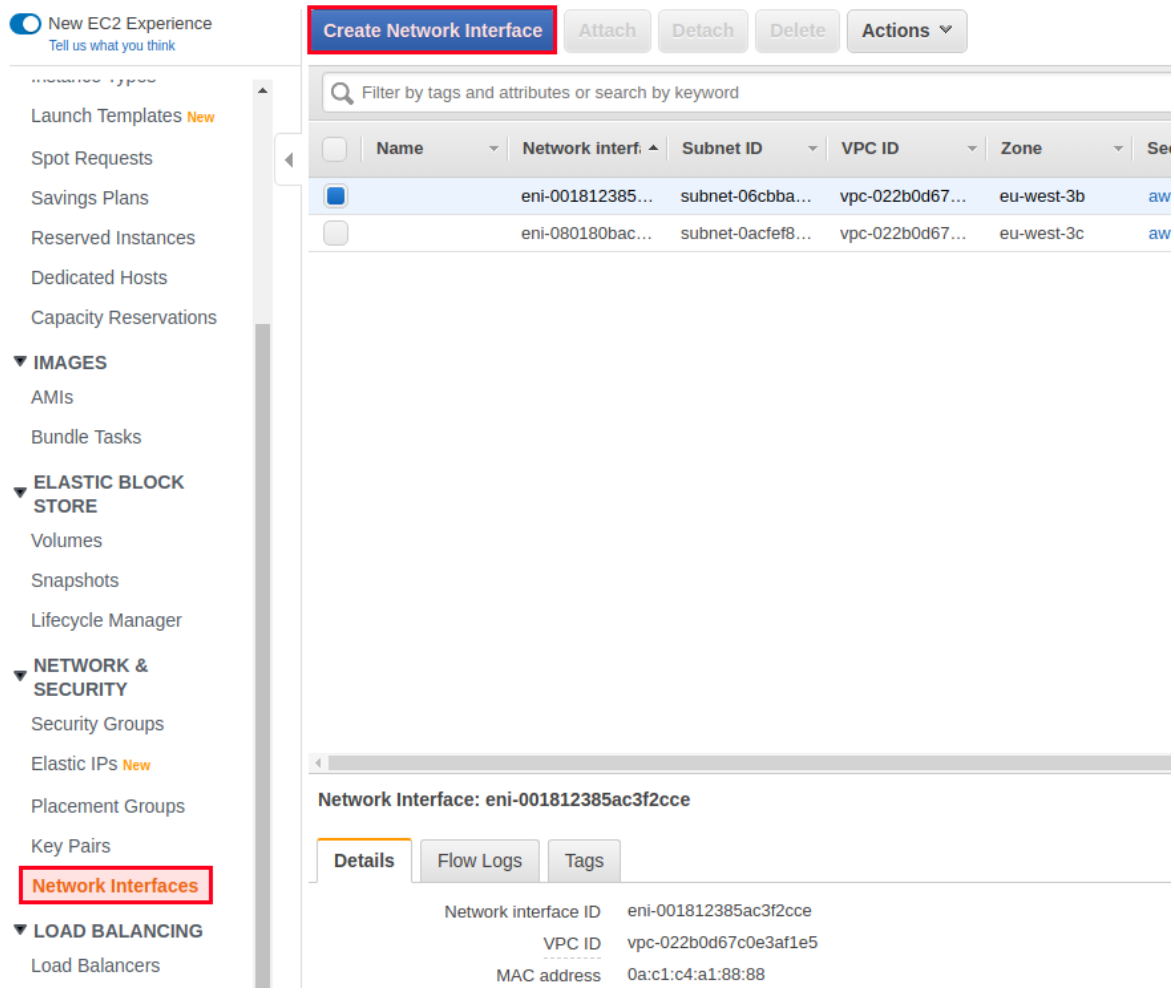
- 172.31.0.0/20 subnet on eu-west-3a AZ
- 172.31.16.0/20 subnet on eu-west-3b AZ
- 172.31.32.0/20 subnet on eu-west-3x AZ

AZ: Availability Zone

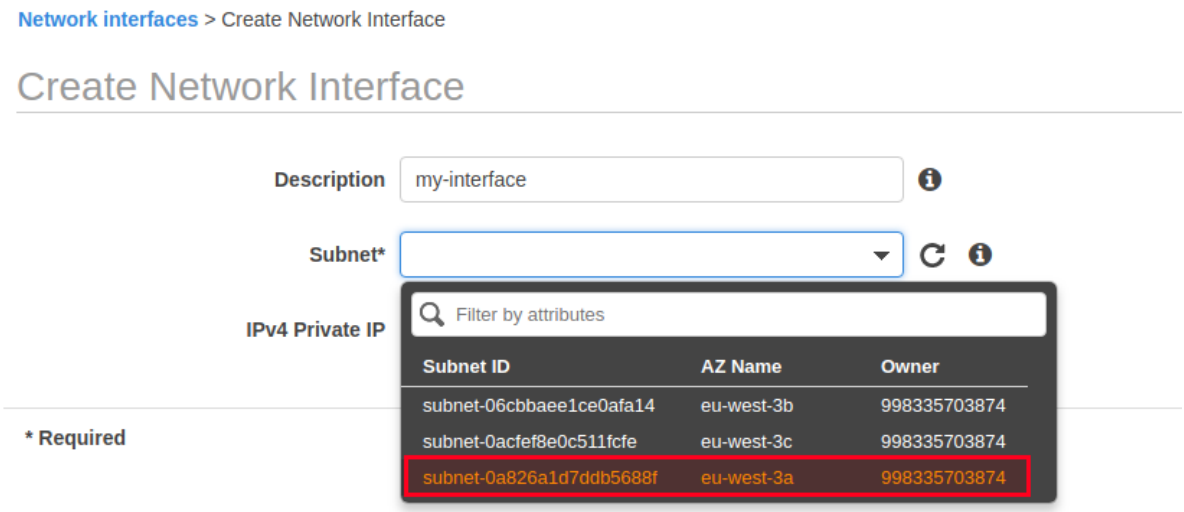
If you create an EC2 machine on the subnet 172.31.0.0/20, the IP address (private or public IP) of this machine will be attached to a single subnet and a single AZ: "172.31.0.0/20 subnet on eu-west-3a AZ".

To add a new network interface to the same machine, you should create an ENI, then attach it to the EC2 machine. If this ENI is on another AZ, it will not be possible to attach it.

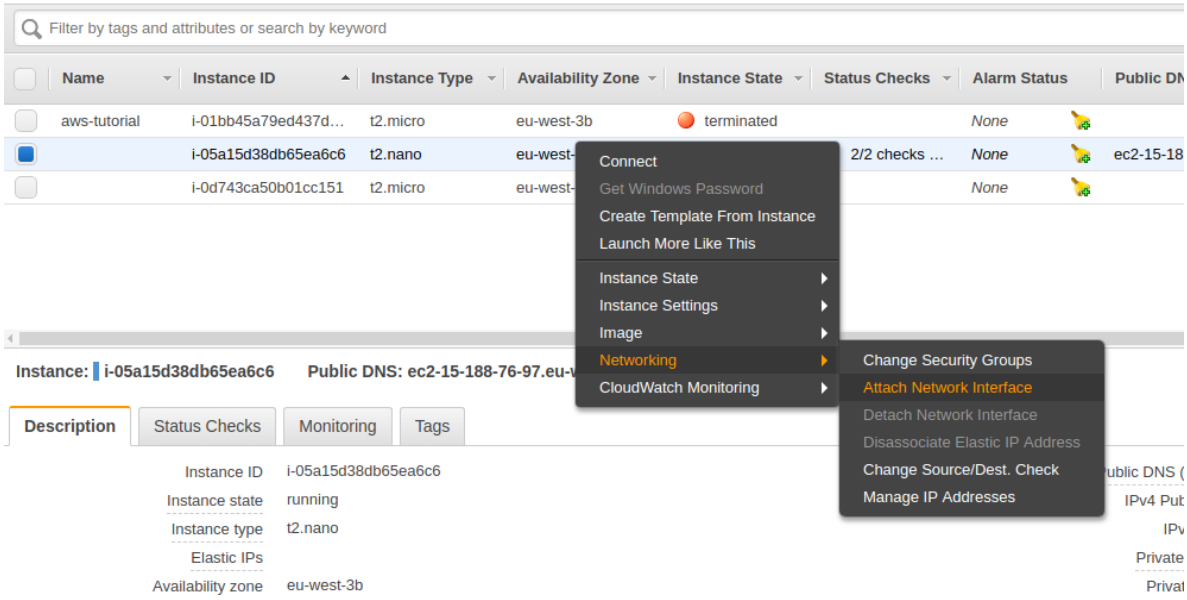
Go to your EC2 console and create a new ENI.



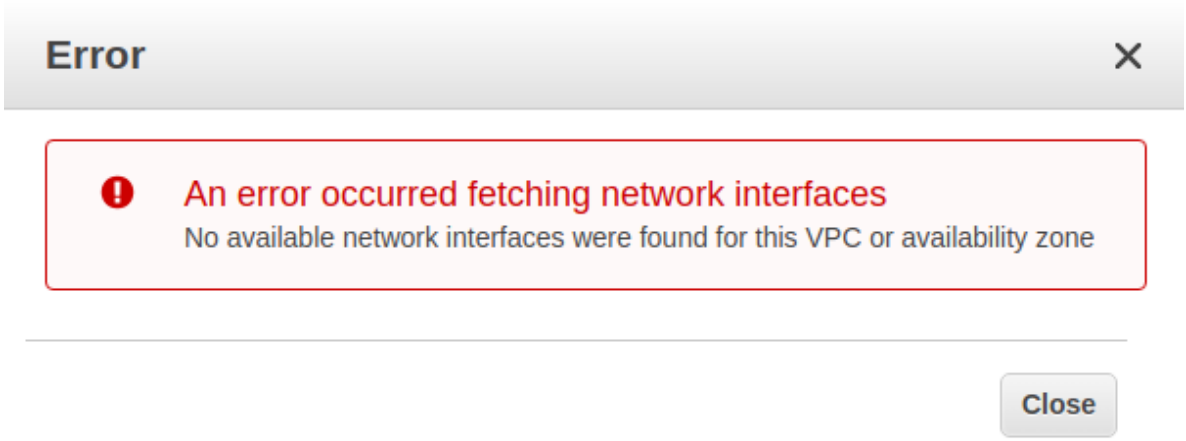
Make sure to choose the right subnet:



Create the IP and go back to the list of your EC2 instances to attach a new ENI to the instance:



Select the IP address. Congratulations, your EC2 machine has two interfaces and two IP addresses. If you see an error dialog like the following one, make sure that you created an ENI in the same AZ of the instance.



Auto-assign Public IP & Amazon Elastic IP

In order to make your instance accessible from the Internet, you should have a public IP. An instance in a subnet can not access the Internet unless it has a public IP, and we are talking about evidence here. Using this option ("auto-assign public IP"), you can request a public IP address from Amazon's public IP address pool.

The public IP address, in most cases, is attached to the instance until it's stopped or terminated, after which it's no longer available for you to use. So if you create a web server to host your website and accidentally deleted the instance, you should recreate it, and it will have a new IP address. You should also reconfigure your domain NS and/or A zones.

This is when EIP (Elastic IP) is helpful. If you need a static IP address that you will always have even when you terminate or stop your EC2 instance, you should create an instance (without a default public IP), then create a new EIP, and finally associate the IP to the instance.

The auto-assigned IP can be assigned to only one instance at a time.

If you have an instance in a private subnet and don't want to attach a public IP address to it, you should add a NAT and assign EIP to it, then modify route tables to allow 0.0.0.0/0 from the private subnet to go to this NAT. We are going to see this in detail later in this course.

Adding a Storage

By default, your AWS machine comes with the root volume, usually with `/dev/sda1` as the device name for systems like Ubuntu and `/dev/xvda` for a system like Amazon Linux...etc

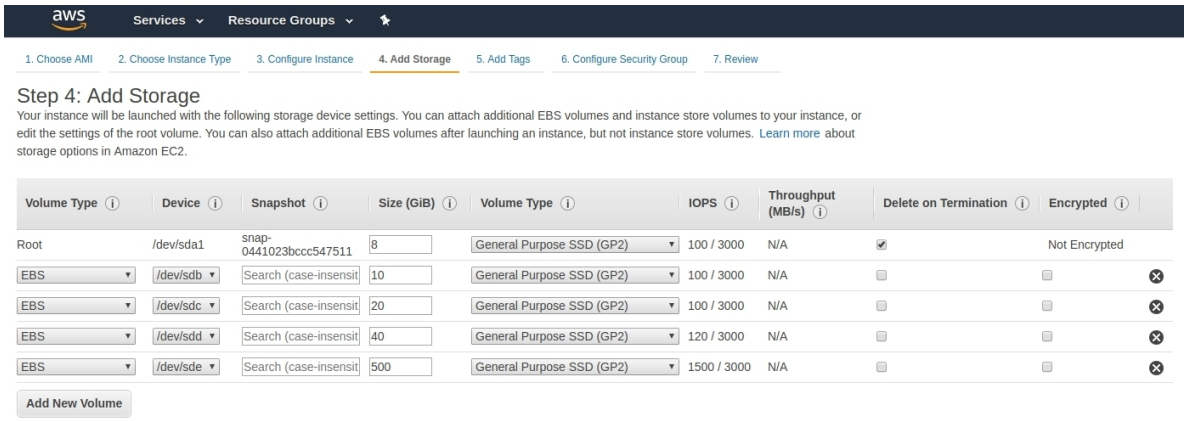
This root volume has 8 GiB, and it's a "General Purpose SSD (GP2)" volume, it comes with 100 / 3000 IOPS, and it will be deleted on the termination of the machine.

Note: IOPS defines the number of I/O operations per second that the volume can support. For Provisioned IOPS (SSD) volumes, you can provision up to 50 IOPS per GiB. For General Purpose (SSD) volumes, baseline performance is 3 IOPS per GiB, with a minimum of 100 IOPS and a maximum of 10000 IOPS. General Purpose (SSD) volumes under 1000 GiB can burst up to 3000 IOPS.

These are the default configuration, but you can change it. For a general-purpose machine, you can keep these default configurations.

We can also add another volume (EBS or Elastic Block Store) using the block device-mapper, and it's actually a good practice: Keep the default volume for your operating system and create new volumes in accordance with what you need.

Let's say we are building an EC2 machine for CI/CD, and we are going to use Docker. Docker users know that `/var/lib/docker` can grow fast in size; at the same time, we want to have a backup of our images. A solution here is creating a second volume; it can be an SSD if you need more speed, then mounting it to `/var/lib/docker`.



Using AWS EBS Volume on Linux

Amazon EBS is a block-level storage volume. It persists independently from the lifetime of an EC2 instance, which allows you to stop and restart your instance at a later time. The block device mapping will include your device name (e.g. `/dev/sdb`, `/dev/sdc` ..etc.) and the volume that maps to it. Each entry in a block device mapping includes a device name and the volume that it maps to.

The default block device mapping is specified by the AMI you use. Alternatively, you can specify a block device mapping for the instance when you launch it.

When you ssh to the machine, type `sudo lsblk -o NAME,FSTYPE,SIZE,MOUNTPOINT` and you will be able to see the attached volume:

```
loop0   squashfs  89M /snap/core/7713
loop1   squashfs  18M /snap/amazon-ssm-agent/1480
xvda               8G
└─xvda1 ext4       8G /
xvdb               8G
```

In an Ubuntu EC2 machine, there are two squashfs partitions, an ext4 system partition `xvda1` for Ubuntu and another partition without any filesystem. We are going to use the latter to create an XFS partition.

To make sure that that this is the EBS disk we attached, we can use the `file -s` command to get information about it, and if we see `data` in the output, we can be sure that there is no filesystem in the device. We can create the FS using:

```
sudo mkfs -t xfs /dev/xvdb



---
meta-data=/dev/xvdb              isize=512    agcount=4, agsize=524288 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=0, rmapbt=0,
reflink=0
data     =                       bsize=4096   blocks=2097152, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=1
log      =internal log           bsize=4096   blocks=2560, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

Note: You can use ext4 if you wish, you should use `sudo mkfs -t ext4 /dev/xvdb` instead of `sudo mkfs -t xfs /dev/xvdb`

Now create a mount point directory for our disk. For example, `/data/`:

```
mkdir /data
```

Mount the volume to the directory we created.

```
sudo mount /dev/xvdb /data
```

Now check if the EBS disk is mounted:

```
sudo lsblk -o NAME,FSTYPE,SIZE,MOUNTPOINT



---
NAME    FSTYPE   SIZE MOUNTPOINT
loop0   squashfs  89M /snap/core/7713
loop1   squashfs  18M /snap/amazon-ssm-agent/1480
xvda               8G
└─xvda1 ext4       8G /
xvdb    xfs        8G /data
```

Automating EBS Disk Mount

The steps we followed previously helped us to mount the EBS device we created on `data`. However, if you try to stop then start the machine, you will discover that the mount is not permanent since it will disappear.

To automatically mount the disk at the system startup, you should find out the UUID of the disk:

```
sudo blkid



---
/dev/xvda1: LABEL="cloudimg-rootfs" UUID="651cda91-e465-4685-b697-67aa07181279"
TYPE="ext4" PARTUUID="eeaf5908-01"
/dev/loop0: TYPE="squashfs"
/dev/loop1: TYPE="squashfs"
/dev/xvdb: UUID="903a046b-ae16-4f79-9b38-c0f92c597ae9" TYPE="xfs"
```

The UUID is `903a046b-ae16-4f79-9b38-c0f92c597ae9`. We should add this line to `/etc/fstab`:

```
UUID=903a046b-ae16-4f79-9b38-c0f92c597ae9  /data  xfs  defaults,nofail  0  2
```

`defaults` is the default mount settings (equivalent to rw,suid,dev,exec,auto,nouser,async).

Other possible options are:

- **auto** - file system will mount automatically at boot, or when the command 'mount -a' is issued.
- **noauto** - the filesystem is mounted only when you tell it to.
- **exec** - allow the execution binaries that are on that partition (default).
- **noexec** - do not allow binaries to be executed on the filesystem.
- **ro** - mount the filesystem read only.
- **rw** - mount the filesystem read-write.
- **sync** - I/O should be done synchronously.
- **async** - I/O should be done asynchronously.
- **flush** - specific option for FAT to flush data more often, thus making copy dialogs or progress bars to stays up until things are on the disk.
- **user** - permit any user to mount the filesystem (implies noexec,nosuid,nodev unless overridden).
- **nouser** - only allow root to mount the filesystem (default).

- **suid** - allow the operation of suid, and sgid bits. They are mostly used to allow users on a computer system to execute binary executables with temporarily elevated privileges in order to perform a specific task.
- **nosuid** - block the operation of suid, and sgid bits.
- **noatime** - do not update inode access times on the filesystem. Can help performance.
- **nodiratime** - do not update directory inode access times on the filesystem. Can help performance. You do not need to enable this flag if you have already enabled **noatime**.
- **relatime** - update inode access times relative to modify or change time. Access time is only updated if the previous access time was earlier than the current modify or change time (similar to noatime, but doesn't break mutt or other applications that need to know if a file has been read since the last time it was modified). Can help performance.

source: wiki.debian.org

`nofail` tells the system to not report errors for this device if it does not exist.

`0` is the dump value, and it is used by the dump utility to decide when to make a backup. When installed, dump checks the entry and uses the number to decide if a file system should be backed up. Possible entries are 0 and 1. If 0, dump will ignore the file system; if 1, dump will make a backup. Most users will not have dump installed, so they should put 0 for the dump entry.

`2` is the pass. Fsck reads this number and determines in which order the file systems should be checked. Possible entries are 0, 1, and 2. The root file system should have the highest priority, 1, all other file systems you want to have checked should get a 2. File systems with a pass value 0 will not be checked by the fsck utility.

This is the final fstab file:

```
sudo cat /etc/fstab



---
LABEL=cloudimg-rootfs    /     ext4    defaults,discard    0 0
UUID=903a046b-ae16-4f79-9b38-c0f92c597ae9  /data  xfs  defaults,nofail  0  2
```

Configuring Tags

A tag is a key-value pair; for instance, "name": "my_server".

Using tags, you can create different categories of your EC2 machines. Let's imagine a scenario where you have 100 production machines and 100 development machine, you will see in your dashboard 200 machines.

Let's say that you shut down all of your development machines at the end of the day and start them tomorrow.

Without tags, you will probably stop the development machines manually, one by one, but if you want to do things faster, you can tag all of your development machines with "env": "dev" for example. After that, all that you have to do is:

```
aws ec2 stop-instances \
--instance-ids \
`aws ec2 describe-instances --filters "Name=tag:env,Values=dev" --query
'Reservations[].Instances[].InstanceId' --output text`
```

Note: The below command is using AWS CLI. The CLI should be installed and configured before using it.

Let's examine each command apart. The embedded command will list the instance ids having the tag "env": "dev":

```
aws ec2 describe-instances --filters "Name=tag:env,Values=dev" --query
'Reservations[].Instances[].InstanceId' --output text
```

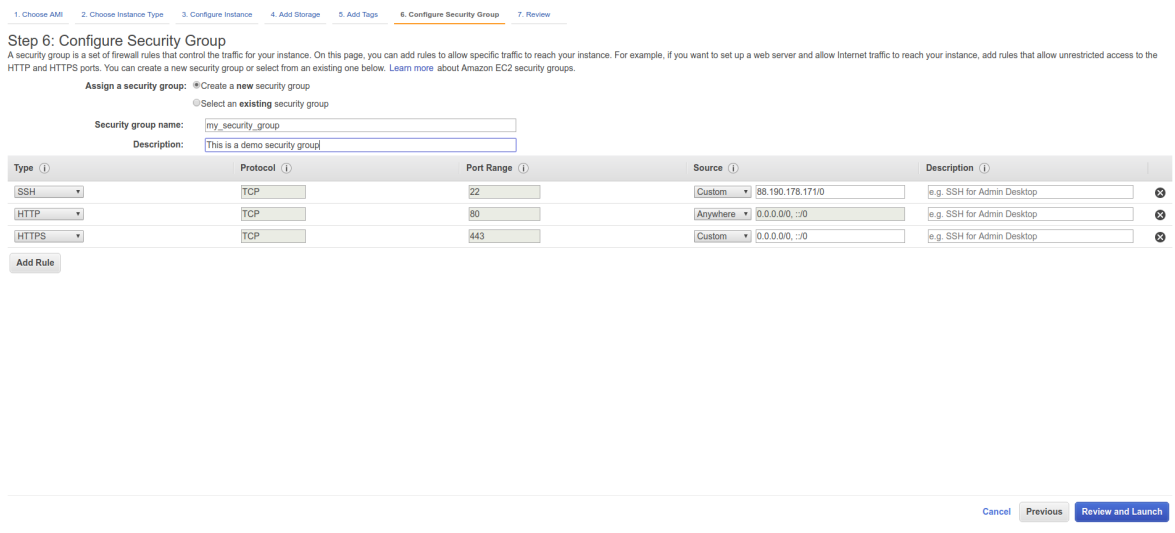
The second one will simply kill these machines:

```
aws ec2 stop-instances --instance-ids <instance_id_1> <instance_id_2> ..
<instance_id_n>
```

Adding Security Groups

A security group acts as a firewall. Let's imagine this scenario:

I want to create a web server, I need the port 80 as well as the HTTPS port 443 open to everyone outside, but I don't need someone else except me access the machine using its SSH port. Since I have a static IP address, I will allow SSH traffic only from my IP:



The screenshot above shows how to do this. I called my security group "my_security_group", I gave it a description (optional), and all that I should do to find it later for future use is clicking on "Review and Launch".

Lesson 5 - Installing And Configuring AWS CLI

Install the CLI

If you are using Linux or macOS, aws-cli could be installed using pip. You should install Python3 if it is not already done.

```
sudo apt-get install python-pip
sudo pip install awscli
```

If you want to upgrade aws-cli, run:

```
sudo pip install --upgrade awscli
```

For Mac users, you can follow the same instructions or just download the tarball, unarchive it and execute:

```
cd <path_to_awscli>
python setup.py install
```

Windows users should use the appropriate MSI installer:

- AWS CLI MSI installer for Windows (64-bit) : https://s3.amazonaws.com/aws-cli/AWSCLI64.msi
- AWS CLI MSI installer for Windows (32-bit) : https://s3.amazonaws.com/aws-cli/AWSCLI32.msi

Open the installer and follow the instructions.

Amazon Identity and Access Management

Before moving to configure the CLI, there are some things we should know.

AWS CLI is installed, but you should configure it. Before doing anything, we will need two keys:

- Access key ID
- Secret access key

They are a kind of a login/password that we assign to an IAM user.

Identity and Access Management (IAM) is an Amazon service that enables you to manage access to AWS services and resources securely, create and manage users and groups, and use permissions to allow and deny their access to AWS resources.

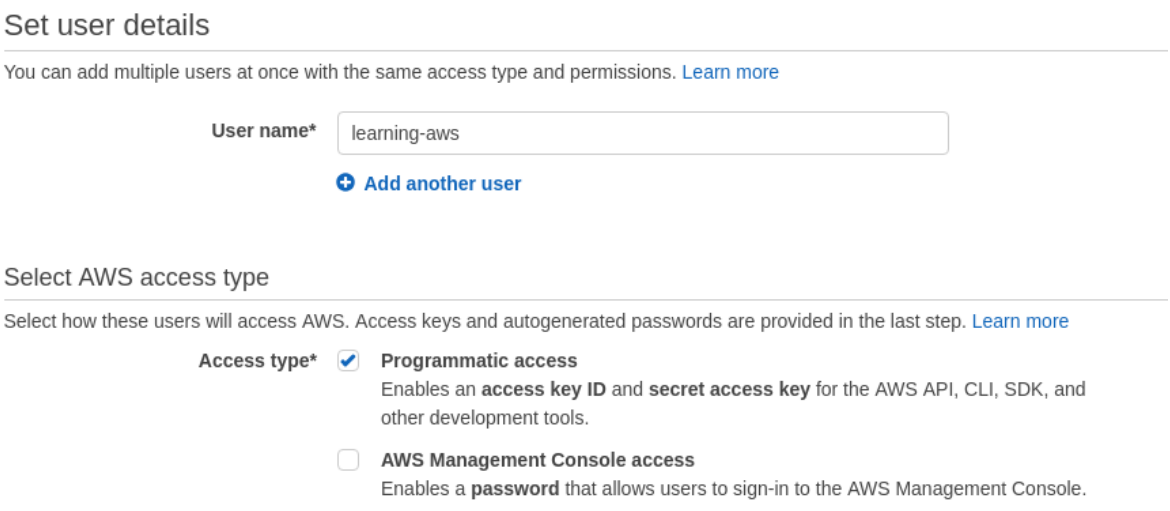
This is a free service, so whether you create 0 users/groups or 10, there are no fees.

To create a user, open the IAM console, click on "Users" then click on "Add user".

When you create a new user, you should choose if you need "Programmatic access" or "AWS Management Console access". It is possible to choose both.

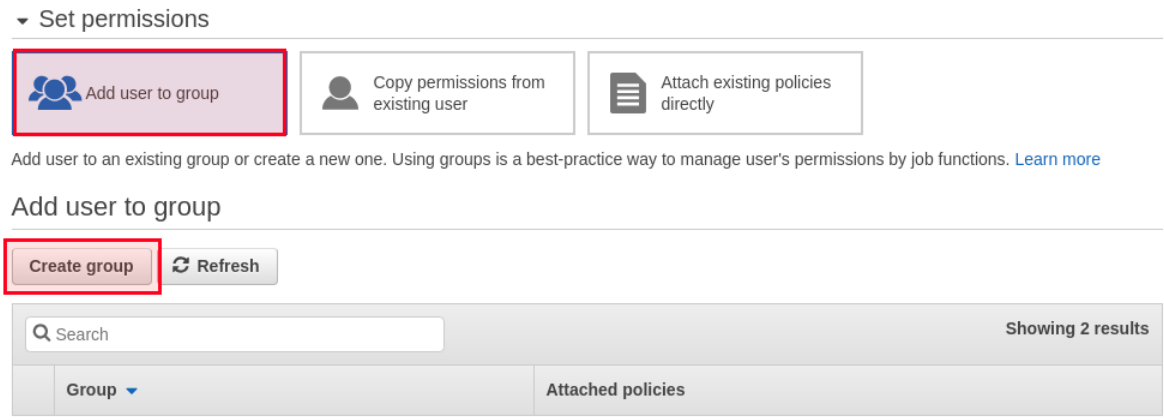
The programmatic access enables the key/secret to use the AWS API, CLI, and SDK as well as the development tools. AWS management console access, on the other hand, allows the user to access the AWS management web console.

Let's create a user "learning-aws" with programmatic access.

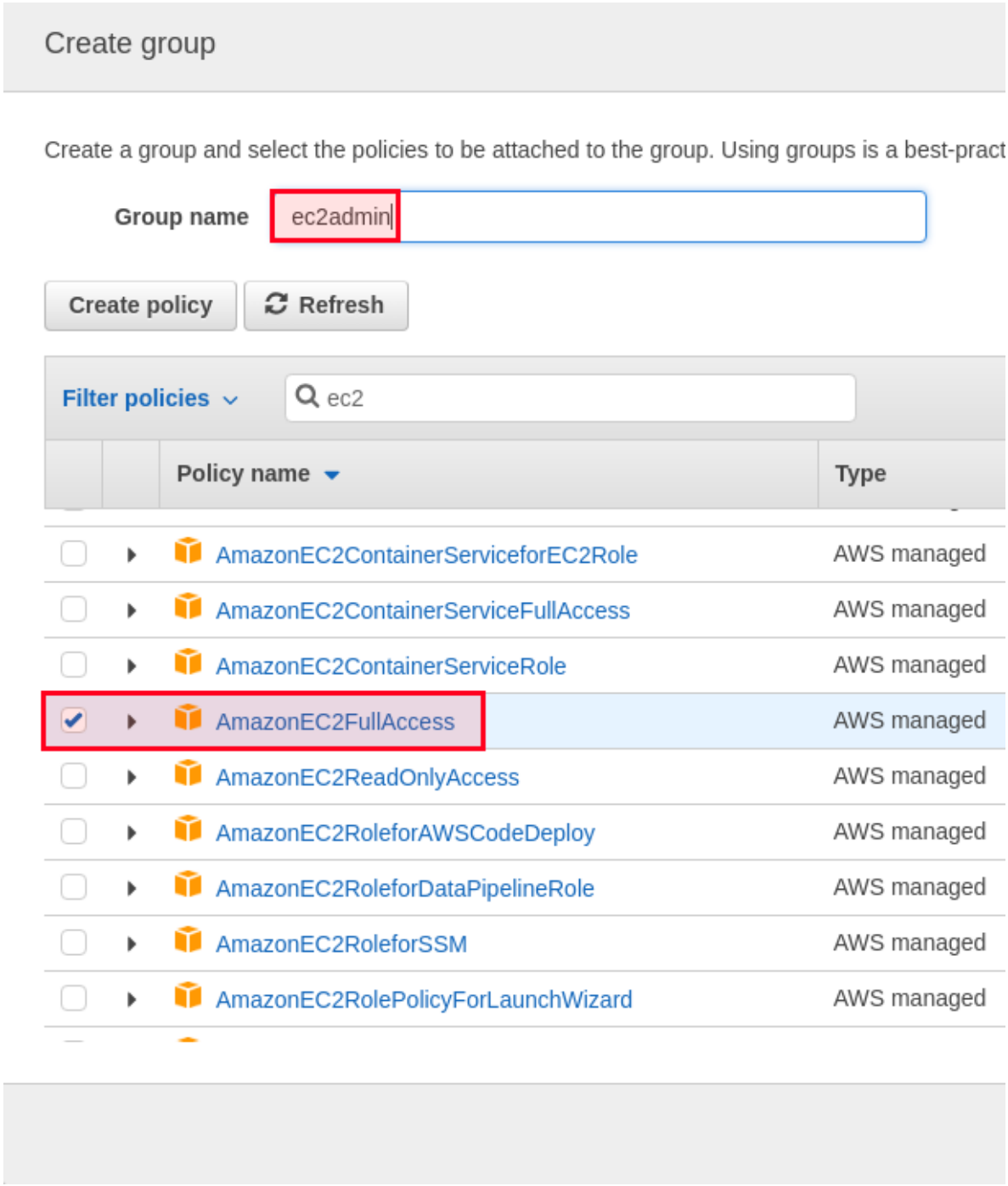


After confirming the access type, you should give this user access to AWS services and permissions or deny it from using them.

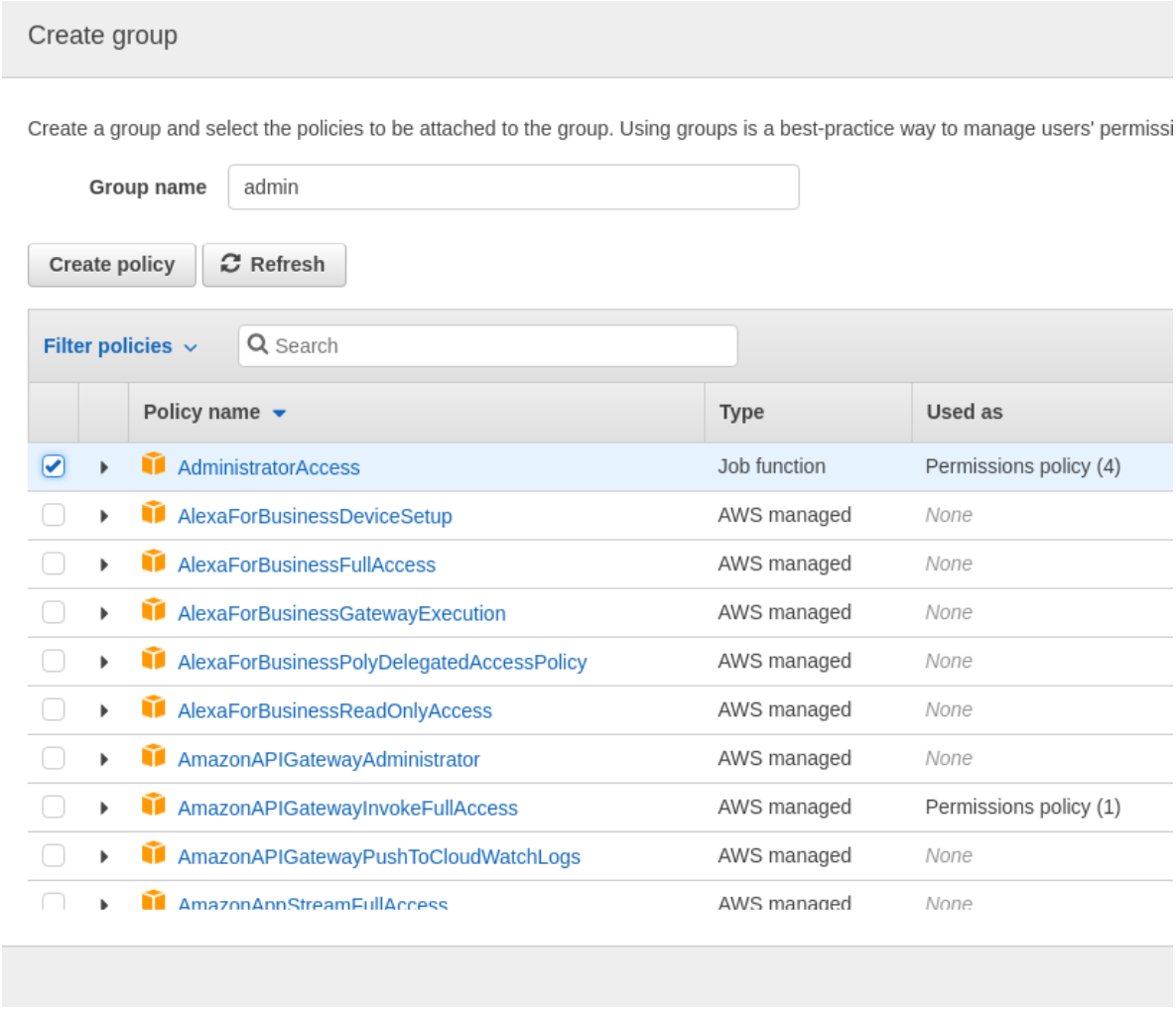
This can be done using AWS policies. By attaching existing AWS policies to the user, you can set the user permissions. It is also possible and better in many cases, to create a new group, assign the p



For instance, if you are going to administer EC2 instances but no other services, you can create the group "ec2-admin" and attach the "AmazonEC2FullAccess" policy.

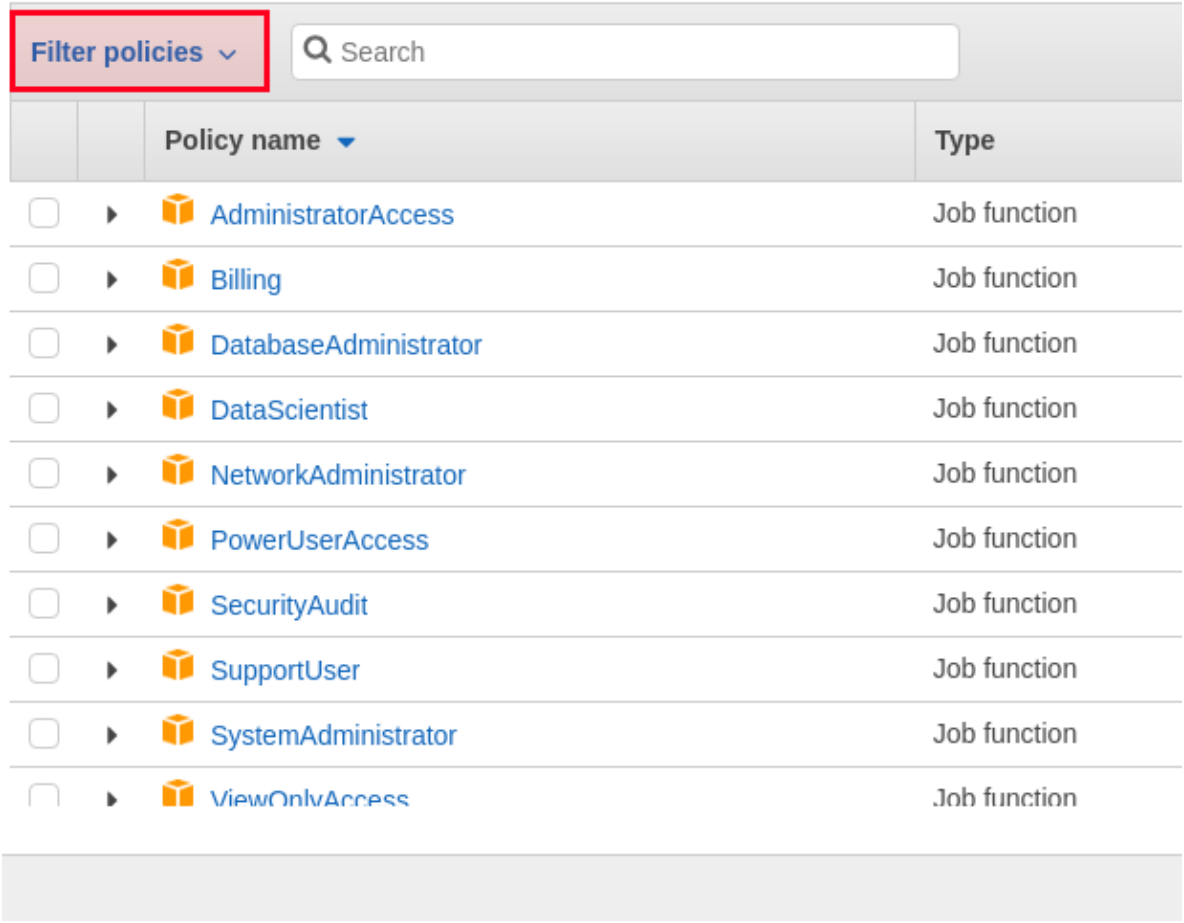


If you need a super administrator user, you can create the admin group and add the "AdministratorAccess" policy.



A policy can be a job type, and this may help in setting up the basic policies then extending them for your team members, for instance: billing team, support team, data scientists.. etc.

You can filter this and choose the job type using the same group creation interface:



Make sure to choose the right permission policies to guarantee a secure environment.

The user will be automatically added to the group you create.

After creating the user, you will be able to view and download its security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Each user with AWS Management Console access can sign-in at

```
https://<user_id>.signin.aws.amazon.com/console
```

Don't forget to download the .csv file and keep it in a safe place. This file contains the "Access Key ID" and the "Secret Access Key".

Configure the CLI

Now that we have both keys "Access Key ID" and "Secret Access Key", we can configure the CLI, so open your terminal and type `aws configure` in order to configure these settings:

- AWS Access Key ID
- AWS Secret Access Key
- Default region-name
- Default output format

For the region name, choose your favorite one. These are some regions.

Code	Name
us-east-1	US East (N. Virginia)
us-east-2	US East (Ohio)
us-west-1	US West (N. California)
us-west-2	US West (Oregon)
ca-central-1	Canada (Central)
eu-west-1	EU (Ireland)
eu-central-1	EU (Frankfurt)
eu-west-2	EU (London)
ap-northeast-1	Asia Pacific (Tokyo)
ap-northeast-2	Asia Pacific (Seoul)
ap-southeast-1	Asia Pacific (Singapore)
ap-southeast-2	Asia Pacific (Sydney)
ap-south-1	Asia Pacific (Mumbai)
sa-east-1	South America (São Paulo)

You can also leave this blank and configure it later.

For the output format, you have the choice between:

- json
- table
- text

Testing the CLI

In order to test if our CLI is well configured, let's type a command.

If you type:

```
aws ec2  describe-regions
```

You should get the list of regions available to use with EC2 instances.

```
{
    "Regions": [
        {
            "Endpoint": "ec2.eu-north-1.amazonaws.com",
            "RegionName": "eu-north-1",
            "OptInStatus": "opt-in-not-required"
        },
        {
            "Endpoint": "ec2.ap-south-1.amazonaws.com",
            "RegionName": "ap-south-1",
```

```
            "OptInStatus": "opt-in-not-required"
        },
        {
            "Endpoint": "ec2.eu-west-3.amazonaws.com",
            "RegionName": "eu-west-3",
            "OptInStatus": "opt-in-not-required"
        },
        {
            "Endpoint": "ec2.eu-west-2.amazonaws.com",
            "RegionName": "eu-west-2",
            "OptInStatus": "opt-in-not-required"
        },
        {
            "Endpoint": "ec2.eu-west-1.amazonaws.com",
            "RegionName": "eu-west-1",
            "OptInStatus": "opt-in-not-required"
        },
        {
            "Endpoint": "ec2.ap-northeast-2.amazonaws.com",
            "RegionName": "ap-northeast-2",
            "OptInStatus": "opt-in-not-required"
        },
        {
            "Endpoint": "ec2.ap-northeast-1.amazonaws.com",
            "RegionName": "ap-northeast-1",
            "OptInStatus": "opt-in-not-required"
        },
        {
            "Endpoint": "ec2.sa-east-1.amazonaws.com",
            "RegionName": "sa-east-1",
            "OptInStatus": "opt-in-not-required"
        },
        {
            "Endpoint": "ec2.ca-central-1.amazonaws.com",
            "RegionName": "ca-central-1",
            "OptInStatus": "opt-in-not-required"
        },
        {
            "Endpoint": "ec2.ap-southeast-1.amazonaws.com",
            "RegionName": "ap-southeast-1",
            "OptInStatus": "opt-in-not-required"
        },
        {
            "Endpoint": "ec2.ap-southeast-2.amazonaws.com",
            "RegionName": "ap-southeast-2",
            "OptInStatus": "opt-in-not-required"
        },
        {
            "Endpoint": "ec2.eu-central-1.amazonaws.com",
            "RegionName": "eu-central-1",
            "OptInStatus": "opt-in-not-required"
        },
        {
            "Endpoint": "ec2.us-east-1.amazonaws.com",
            "RegionName": "us-east-1",
            "OptInStatus": "opt-in-not-required"
        },
        {
```

```
            "Endpoint": "ec2.us-east-2.amazonaws.com",
            "RegionName": "us-east-2",
            "OptInStatus": "opt-in-not-required"
        },
        {
            "Endpoint": "ec2.us-west-1.amazonaws.com",
            "RegionName": "us-west-1",
            "OptInStatus": "opt-in-not-required"
        },
        {
            "Endpoint": "ec2.us-west-2.amazonaws.com",
            "RegionName": "us-west-2",
            "OptInStatus": "opt-in-not-required"
        }
    ]
}
```

AWS CLI is installed and configured without problems. If you didn't configure the region, you could do it now since you have the updated list of regions. Type `aws configure` and leave everything as it is by hitting the enter button, configure the region, then confirm the new settings.

Lesson 6 - AWS Networking

Introduction

In this lesson, we are going to learn how to configure your AWS network. We are going to create :

- A VPC (Virtual Private Cloud)
- Private and Public Subnets
- Route Tables
- An Internet Gateway
- A NAT Gateway

VPC acts as network isolation for your computing, storage, and other resources you use. All resources that belong to a VPC are logically isolated and launched inside an isolated virtual network. In a VPC network, we have sub-networks (subnets).

Subnetting has many benefits. In a large network with no subnets, every server would see broadcast packets from all the servers and users on the network, resulting in the switches having to move all that traffic to the appropriate ports. It also allows for easier administration and growth control. It helps in boosting security; for instance, it is more secure to create a private subnet for the machines that we don't need to expose to the Internet.

Note: A machine in a private subnet, even without being public, can access to the Internet. This is possible using the NAT gateway (Network Address Translation). When a private-addressed machine sends or receive traffic, it will use the NAT gateway without even having a public IP address.

So, each private subnet should have its own NAT gateway in order to allow external access. Even if we don't need access to the Internet in most cases, using APT or YUM package managers to install and upgrade your system packages, need Internet access.

Internet Gateway vs. NAT Gateway

Let's try to understand the difference between these gateways. They both allow instances to have external access (Internet), and this may be confusing to understand the real difference.

The first thing and the main difference is that the Internet Gateway (IGW) is attached or allocated to a VPC while a NAT Gateway (NGW) is associated with a subnet in the VPC. A VPC can have multiple subnets, therefore multiple IGW; however, it can not have more than 1 IGW.

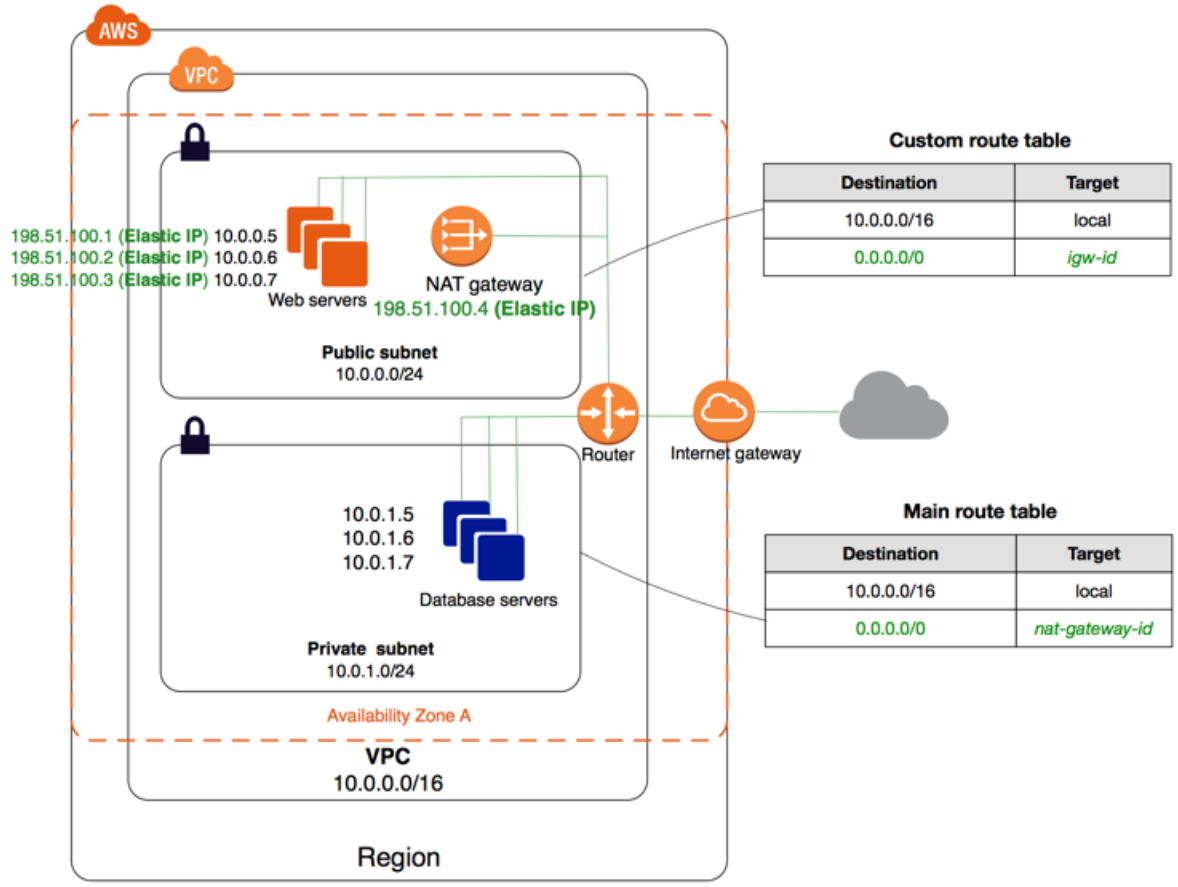
There is also another difference: The IGW gives your VPC external access and the NAT gateway routes to an IGW to access the internet. So even if a subnet has a NGW, it will be unable to reach external hosts if the VPC does not have an IGW.

Another difference resides in the fact that IGW is nothing then a logical link to the Internet. It is like your RJ45 cable, it allows your home to have external access, but there is no control. However, the NGW is a machine; consider it like your home switch that you can control to open/close ports, block web sites ..etc.

Historically, the NGW was not a managed service, you should create your own EC2 machine, but now when you create a NAT Gateway, logically, AWS creates a managed VM for you. In all cases, now the NGW is a managed service. To create a NGW, you should assign an Elastic IP address to it. The routing mechanism of a NGW is done using Route Tables.

Creating a VPC

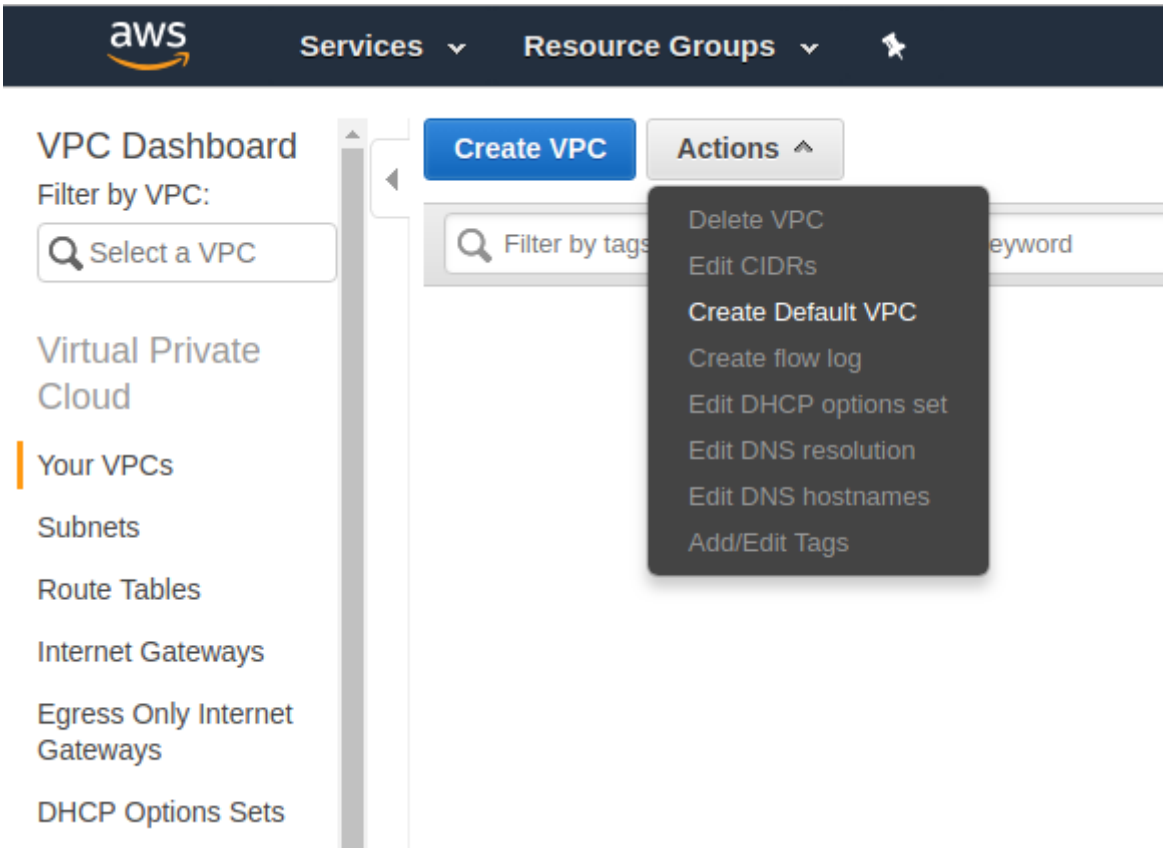
To understand all of this, the best and the most common example I found is the following:



source: aws.amazon.com

Important Note: The network we are building follows the same diagram but may uses different IP addresses VPC, subnets, CIDRs..etc

Let's start by creating a new VPC. You can delete the default VPC and if you want to recreate it later, use the "Create Default VPC" menu.

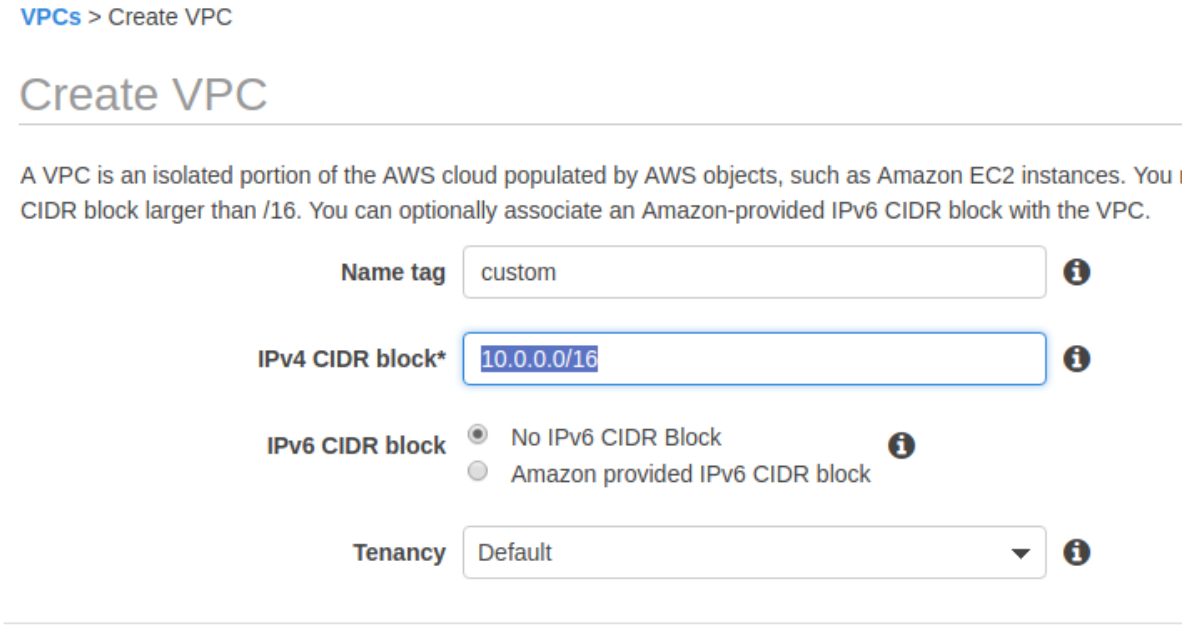


Click on "Create VPC"

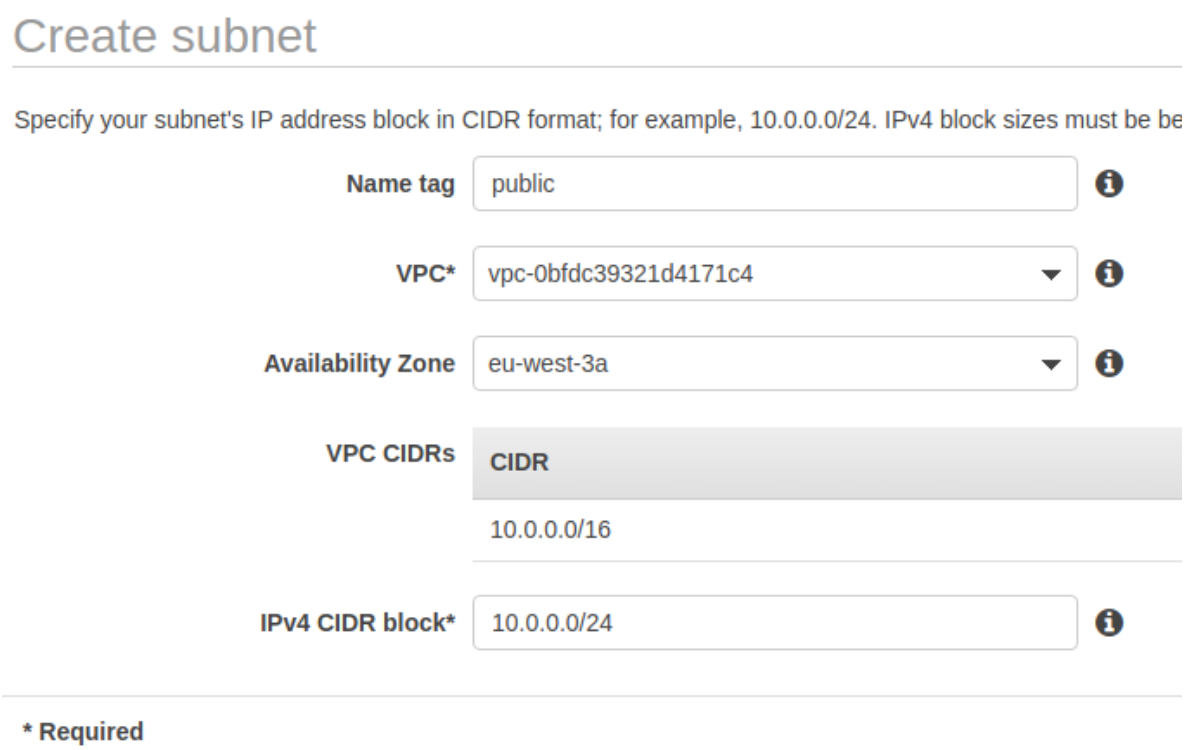
Give the VPC a name. I name it "custom".

Provide the IPv4 CIDR block; I used "10.0.0.0/16" which means that I will have the ability to use all the blocks between 10.0.0.0 and 10.0.255.255, which contains 65536 IP addresses.

I do not need a dedicated VPC; the default tenancy once is good for most use cases.

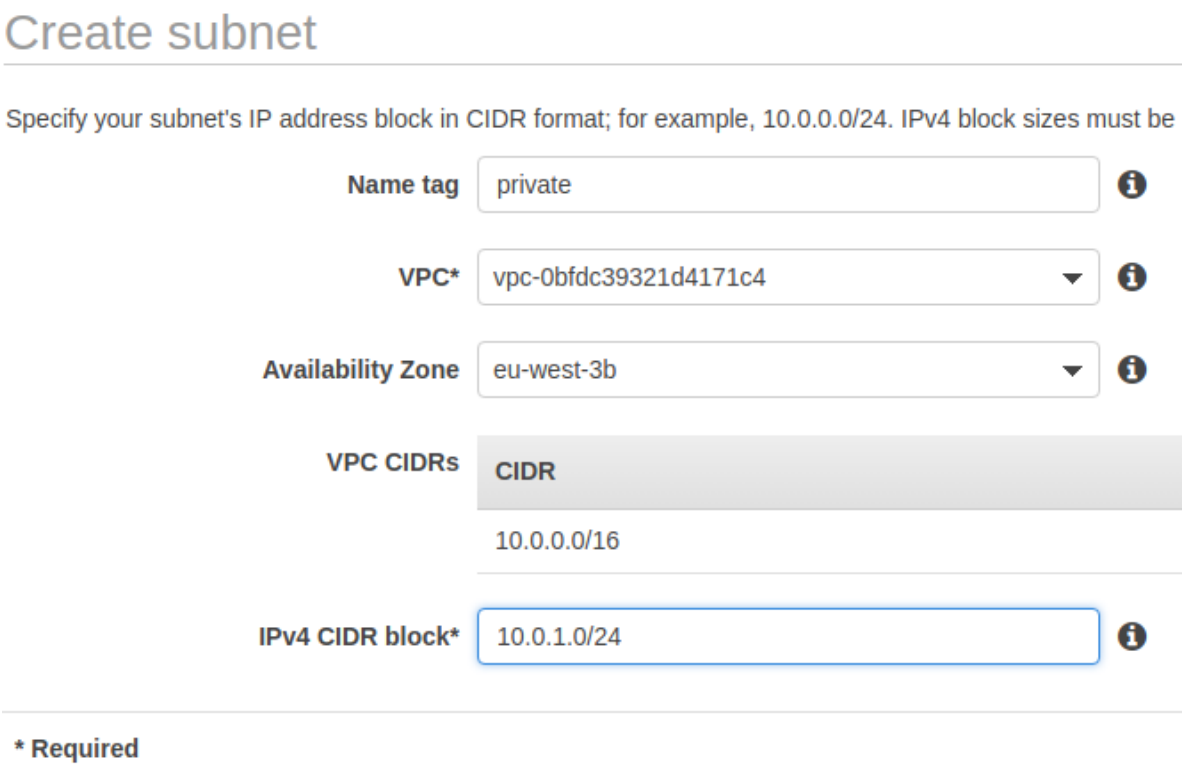


Let's move to create two subnets; one public and one private subnet.



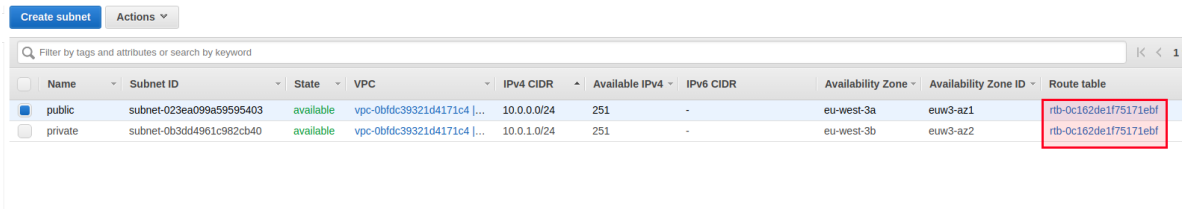
The first public subnet has the name "public"; you can choose another name. I also assign it to the AZ "eu-west-3a". If you are using another region, you can choose a different zone. Finally, for the IP addresses block, it ranges from 10.0.0.0 to 10.0.0.255, which means 256 addresses, and it is enough for most uses cases. Unless you are going to deploy more than 256 machines in this subnet, you can choose another block.

Do the same thing to create another private subnet.

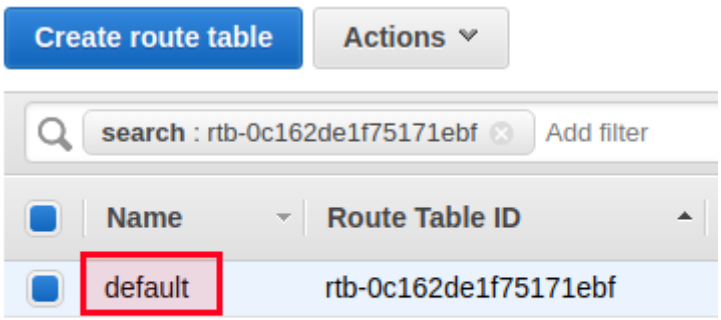


We are using the CIDR 10.0.1.0/24, and this is a random choice, you can choose another CIDR, but since the first subnet has the CIDR 10.0.0.0/24, why not choosing the next CIDR 10.0.1.0/24.

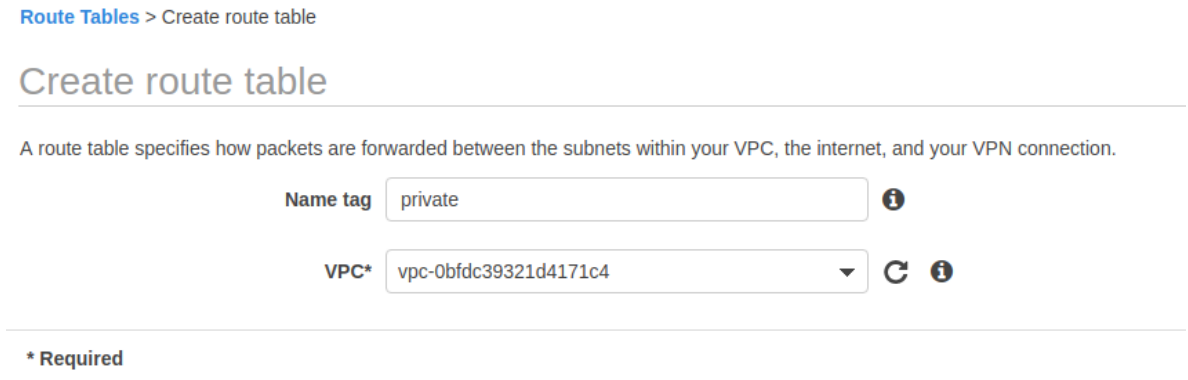
After creating two subnets, we can notice that both are using the same route table, and it is the default route table created by AWS.



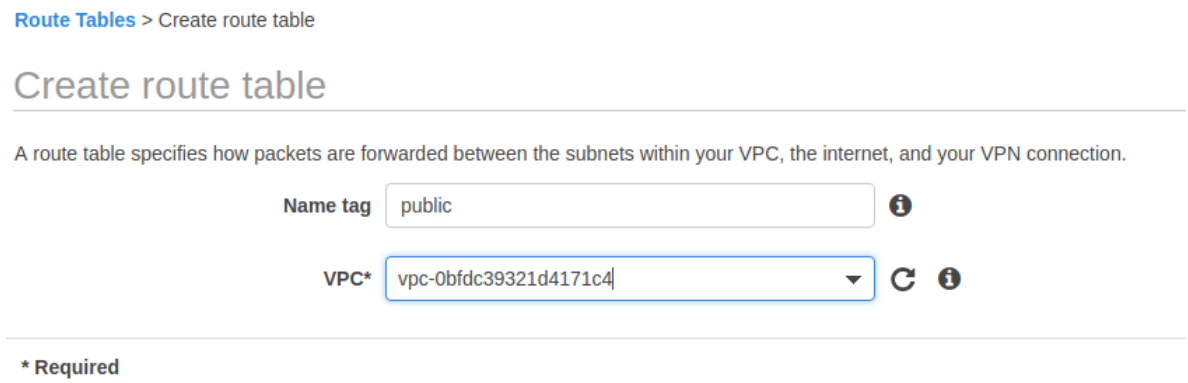
You can click on it and rename it to "default", to avoid any confusion.



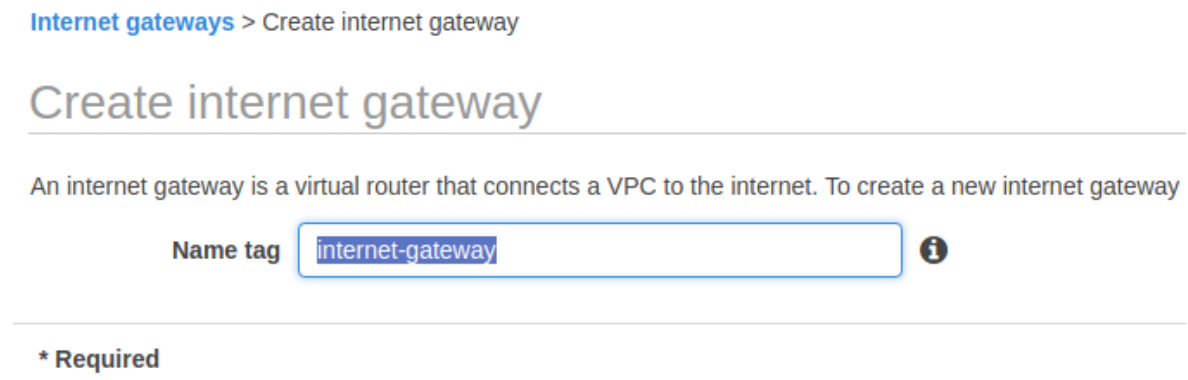
Now, let's create a second route table, which will act as the private route table in our infrastructure.



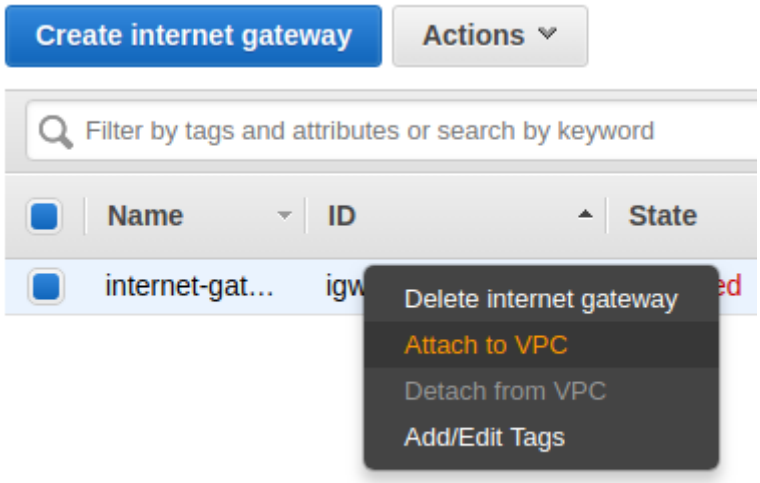
The third route table will be the public one.



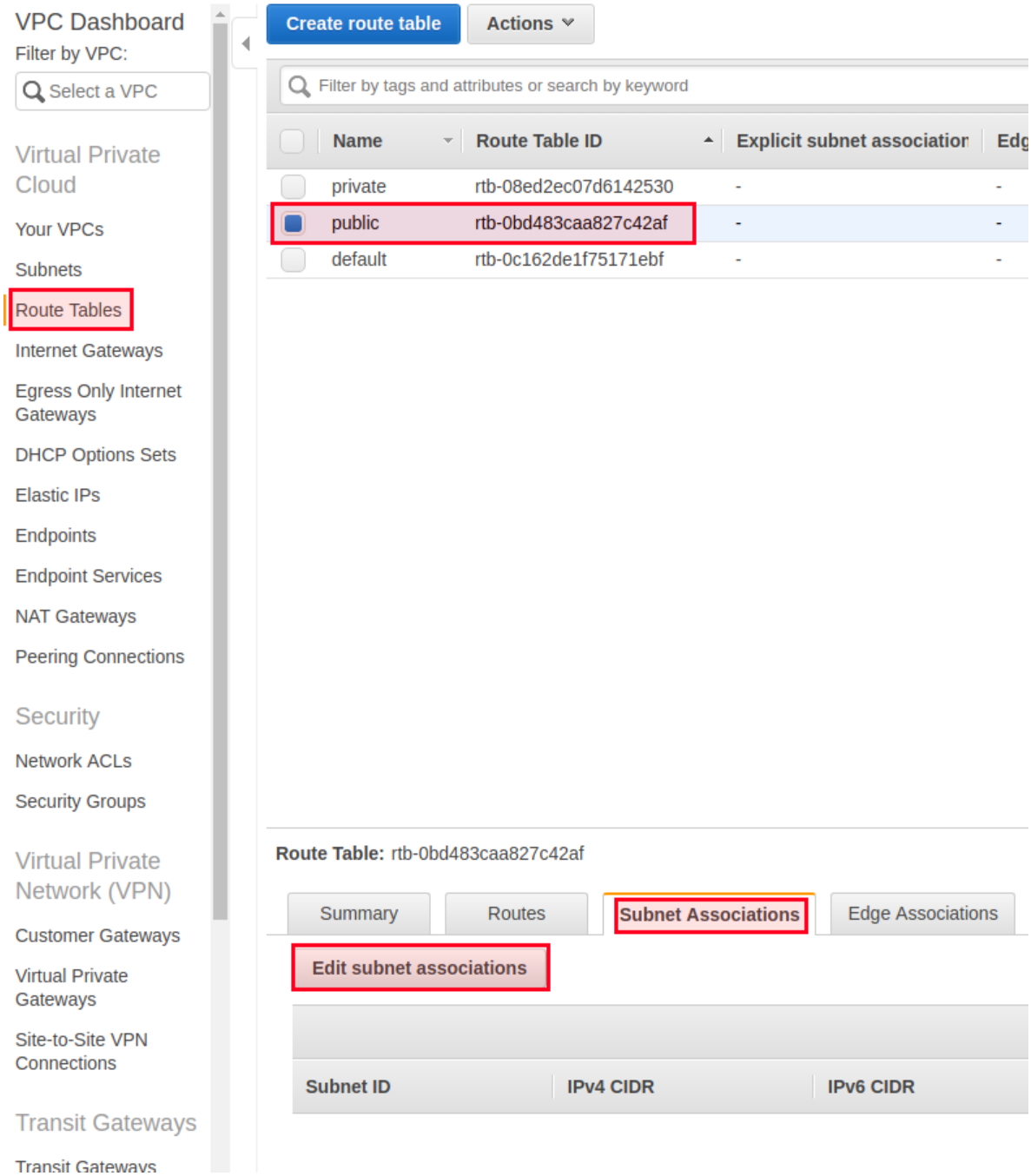
Now, at this step, we have two subnets (1 private and 1 public), two route tables (+ the default one), and we need an Internet Gateway.



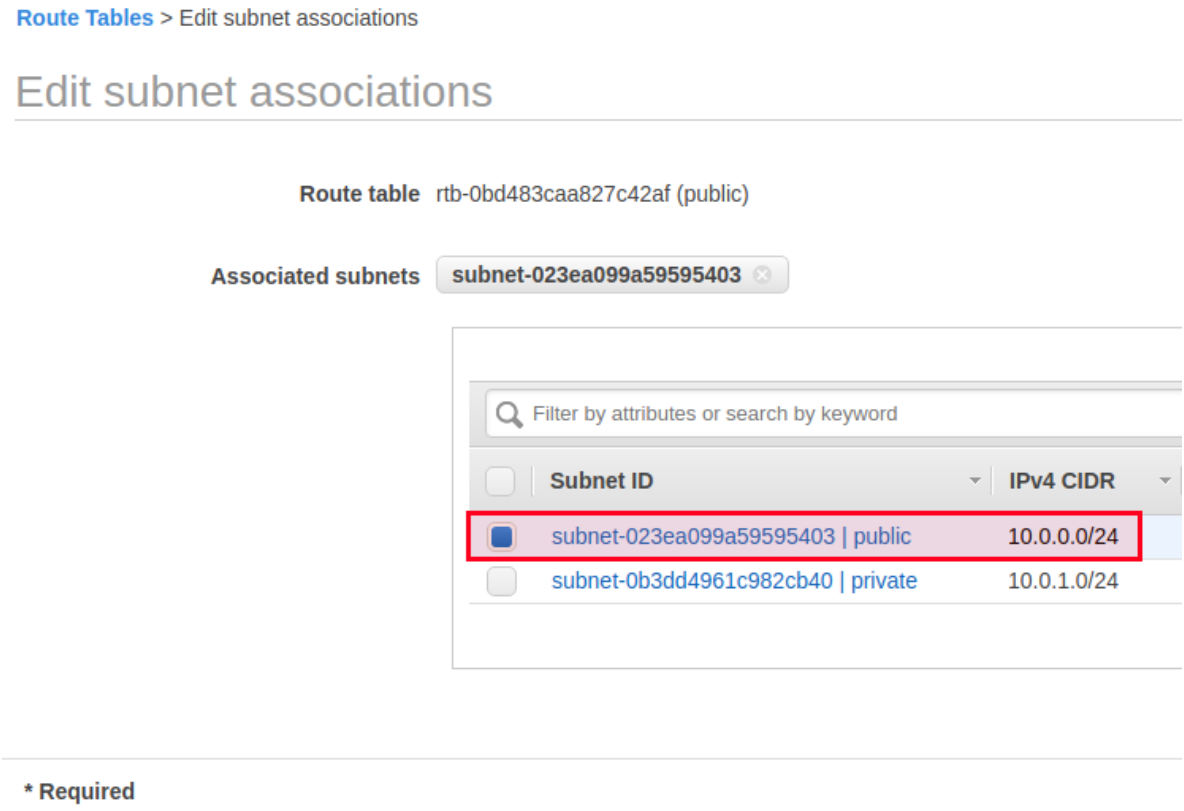
Create one, give a name tag, and attach it to the VPC.



Let's associate the public subnet to the public route table.

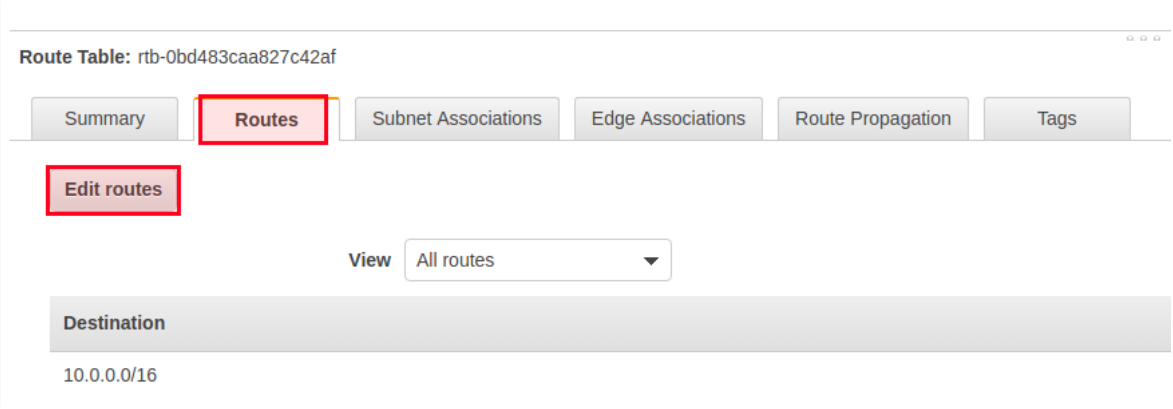


Click on "Edit subnet associations" and choose the public subnet from the list.

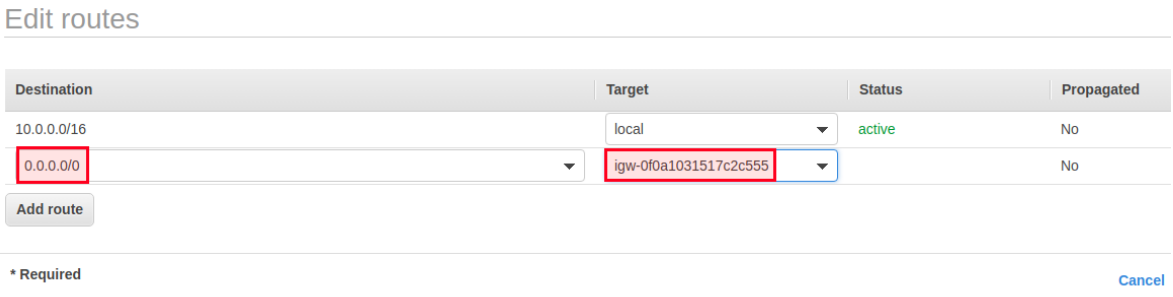


Any EC2 instance in the public subnet should be able to send and receive traffic from the Internet, and this is why the route table should allow the subnet to reach and receive traffic from all hosts (0.0.0.0/0).

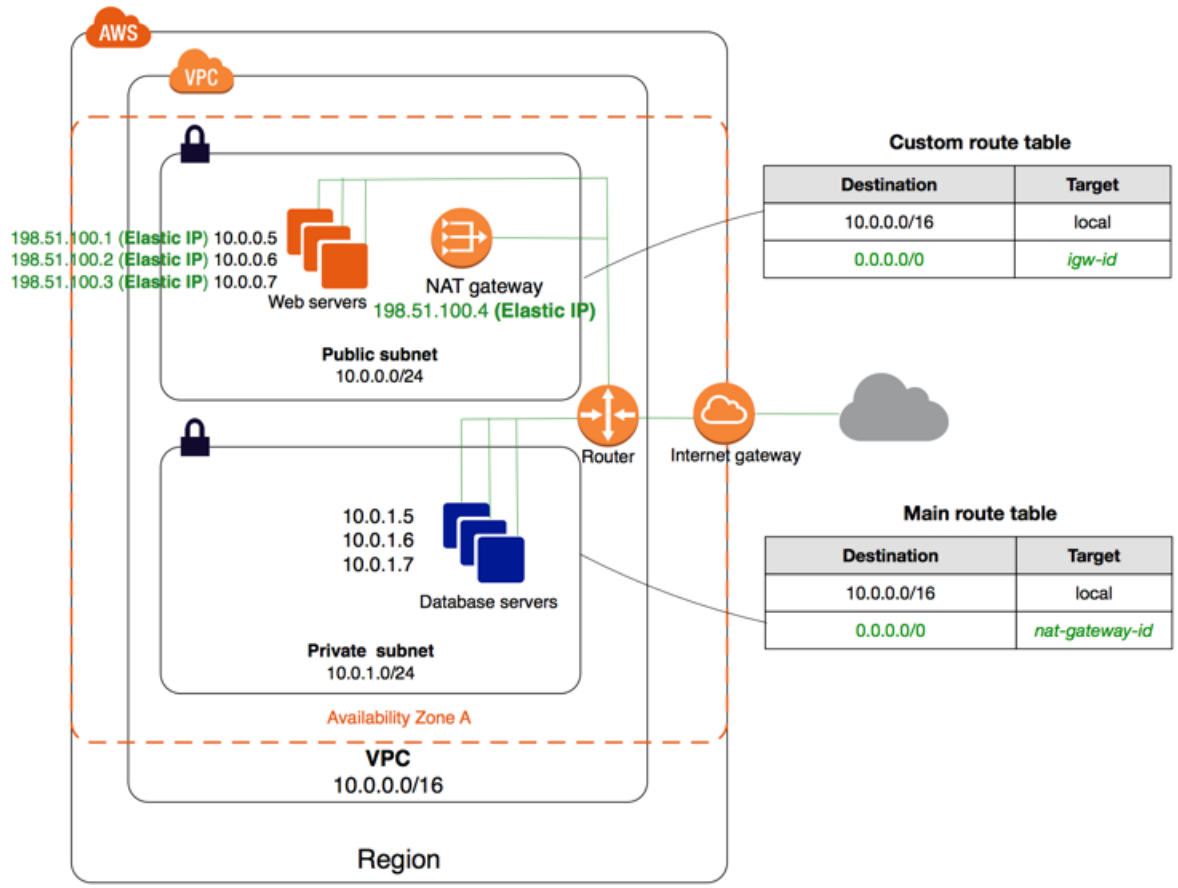
To make this happen, click on "Routes", then "Edit routes".



Add "0.0.0.0/0" as destination and the Internet Gateway we created as the target.

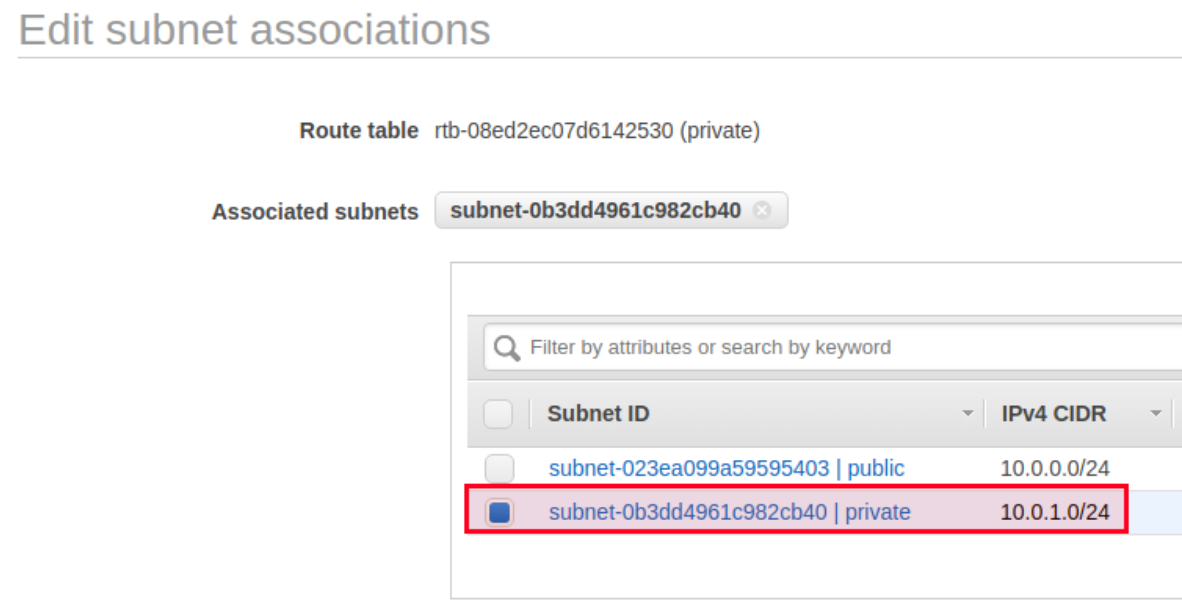


This means that if an EC2 machine in the public subnet wants to ping google, the VPC will check its route table before and find that it can reach the Internet using the line we added to the route table, which means that the traffic will flow through the Internet Gateway. The reverse traffic will also flow through the Internet Gateway, then the router, and finally reach the EC2 machine.

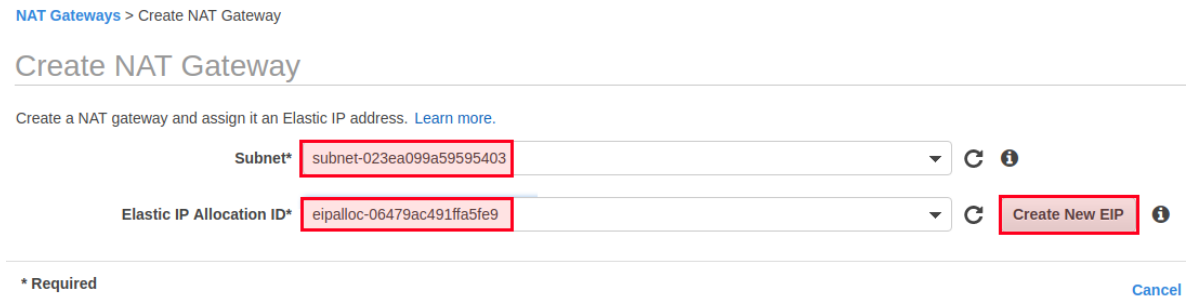


Important Note: The network we are building follows the same diagram but may uses different IP addresses VPC, subnets, CIDRs..etc

Now, go to the private route table and associate the private subnet to it.



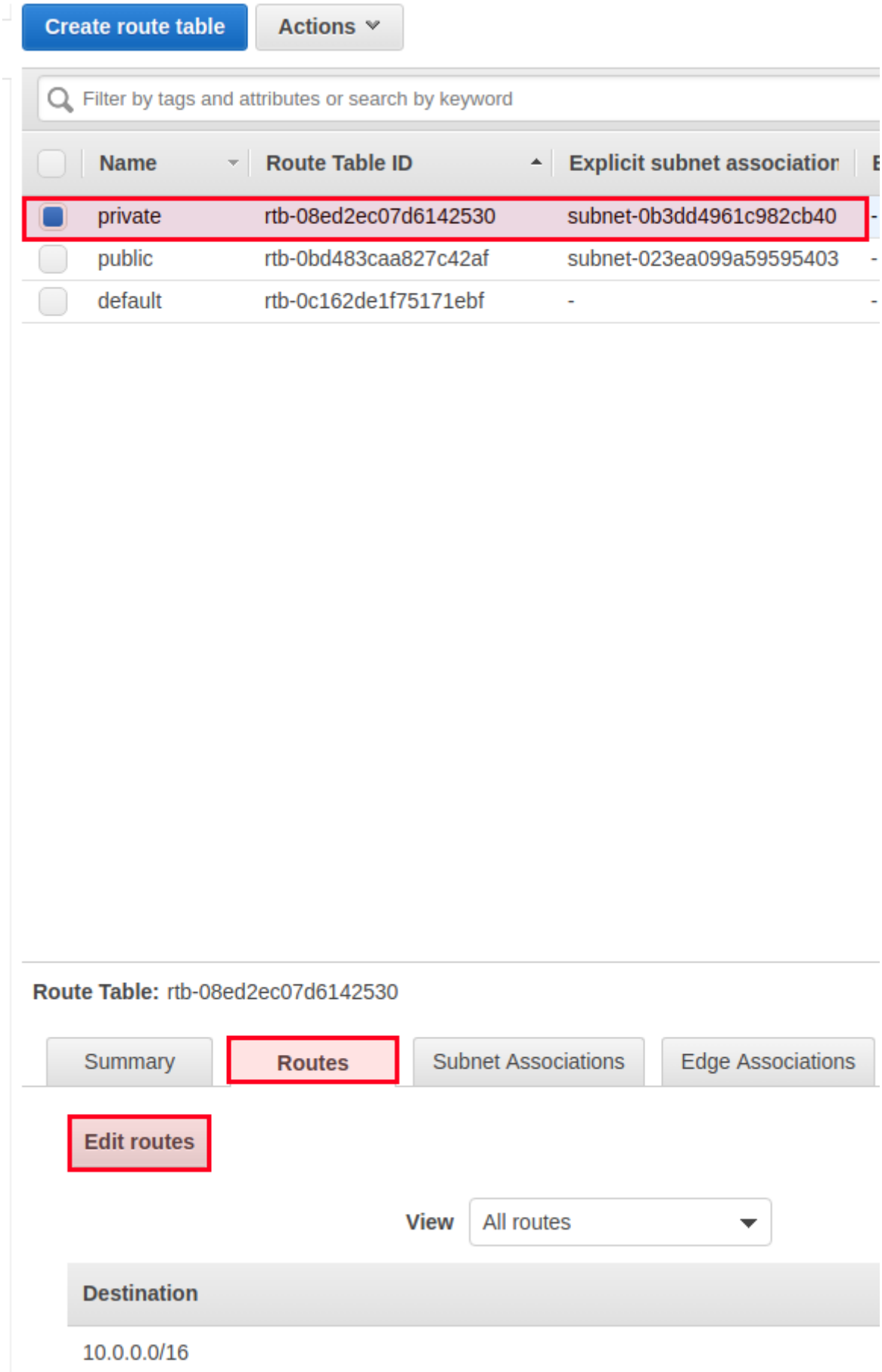
Since the private subnet or the hosts in the private subnet need to access the Internet while keeping them private (not reachable from the outside), we need to create a NAT Gateway.



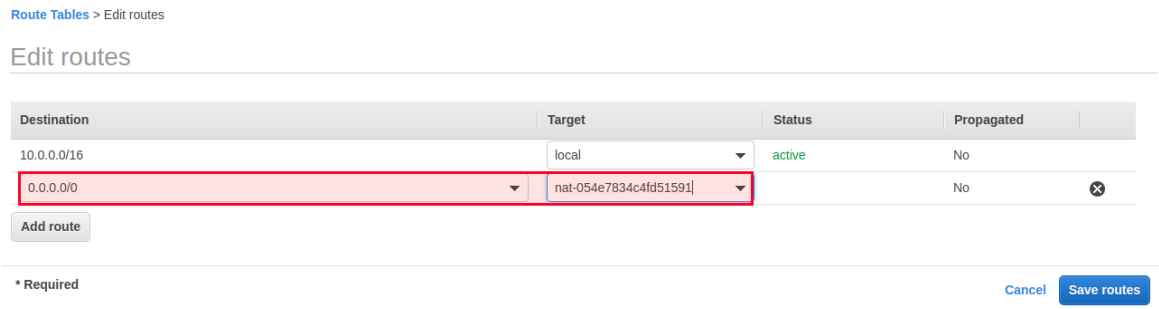
Create a new NAT Gateway, choose the public subnet, and create a new EIP (Elastic IP).

Note: When creating a NAT Gateway, keep in mind two things: It should be on the public subnet, even if we are going to use it with the private subnet. It should have an EIP.

No go back to the route tables and edit the private subnet. We need to tell the VPC (more exactly the private subnet route table), that any outgoing traffic requested by an instance in the private subnet and going out to the Internet should transit through the NAT Gateway.



Edit the routes. As a destination, use "0.0.0.0/0" and, as a target, use the NAT Gateway.



So, any traffic outgoing traffic from the private subnet will check the route table and sees that it should go through the NAT Gateway. The NAT Gateway will also check the route table of the public subnet (because it is in the public subnet) and finds out that in order to send outgoing traffic, it should use the Internet Gateway.

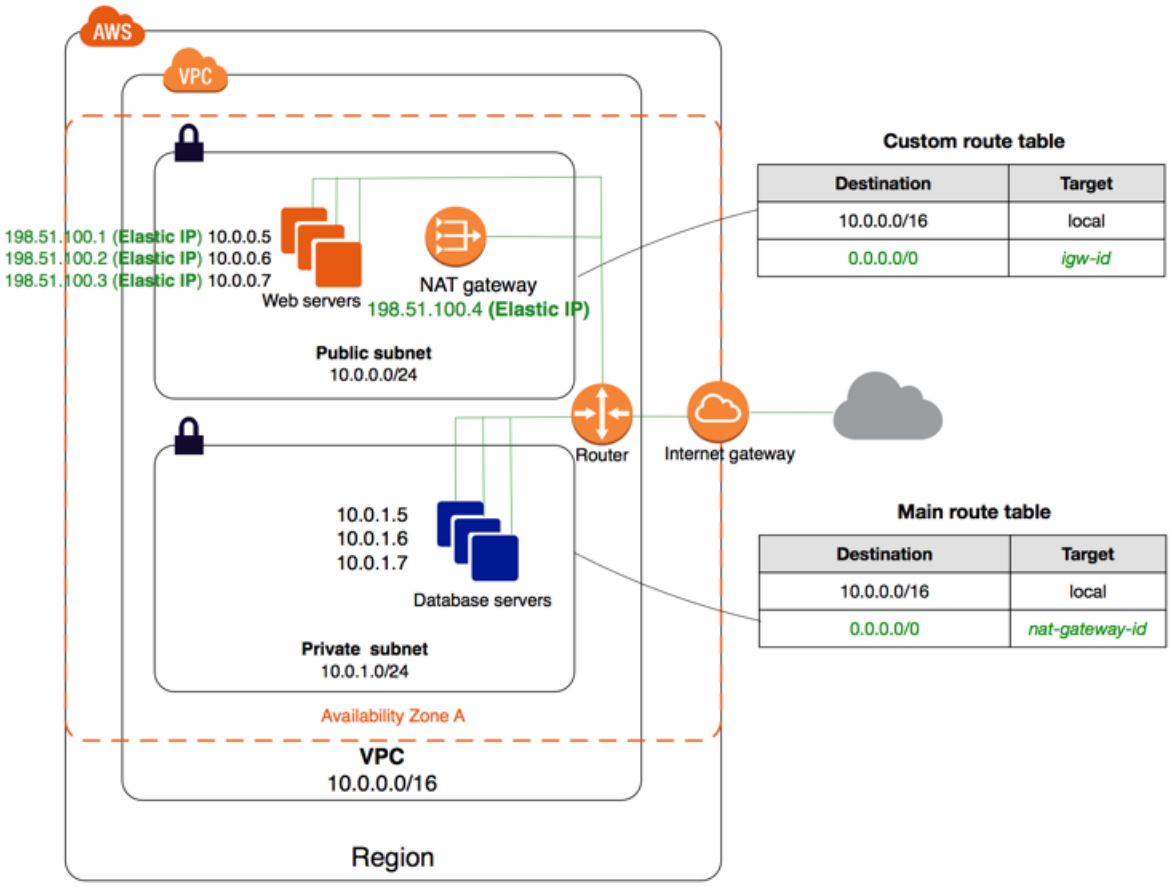
Private EC2 -> Router -> NAT Gateway -> Router -> Internet Gateway.

So, if you were asking this question:

If the NAT Gateway is used with the private subnet and not the public one, why is it in the public subnet?

The answer is:

Exactly, the private subnet uses the NAT Gateway. The public subnet makes no use of the NAT Gateway. However, since the NAT Gateway should have access to the Internet, it should be in the public subnet.



Important Note: The network we are building follows the same diagram but may uses different IP addresses VPC, subnets, CIDRs..etc

Now, if you create an EC2 machine on the public subnet, it will be able to send and receive outbound and inbound traffic. However, the machines in the private subnet will only be able to send outbound traffic. In case we need to ssh into these machines, a solution would be creating a bastion host in the public subnet and use it as an entry point to any machine in the private subnet.

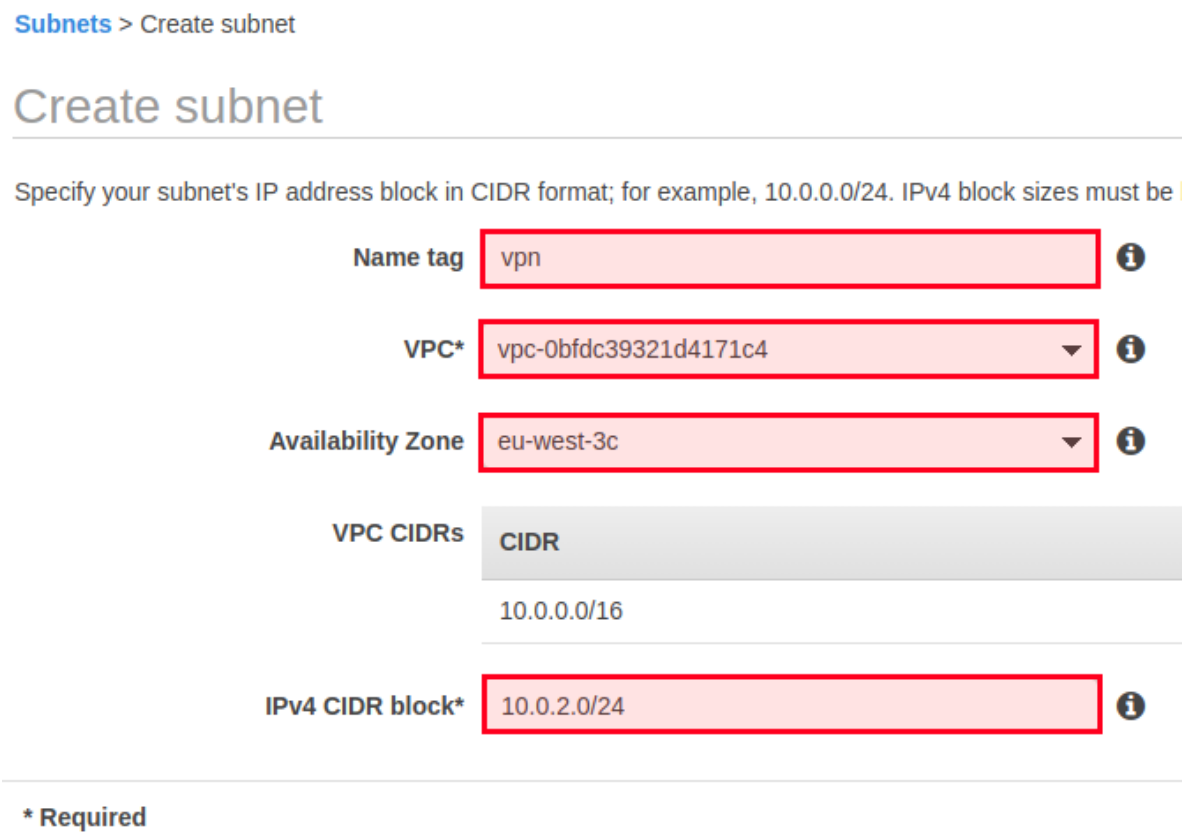
If you want to create a bastion host, you can also deactivate the SSH access to 0.0.0.0/0 in the security group of public hosts and only allow the bastion IP address.

So, the bastion host can be a way to harden security, allowing SSH access to all machines (public and private) only through it.

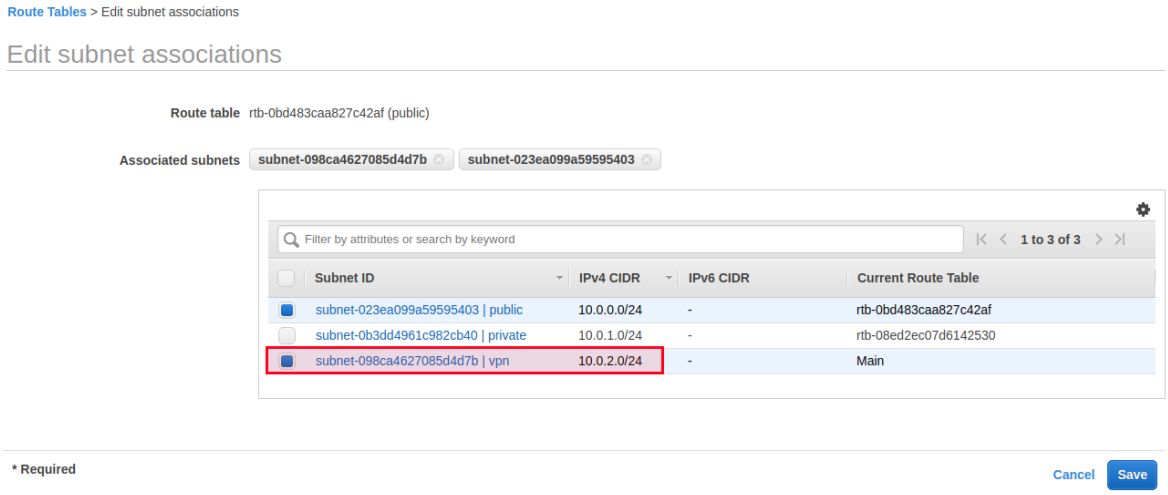
The second solution is creating a VPN.

Using a VPN

Create a new subnet for the VPN machine. We will use a different zone here (eu-west-3c).



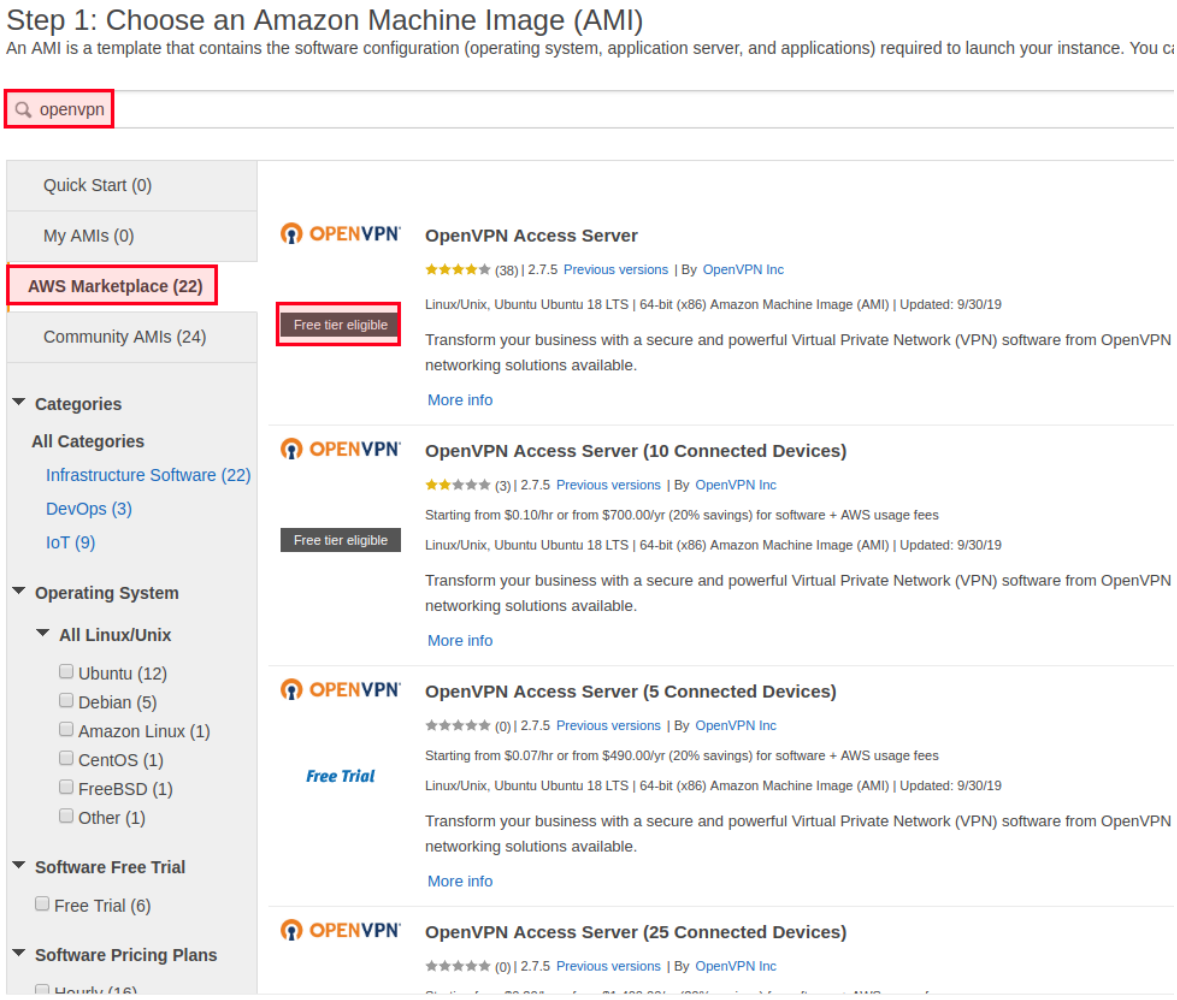
Now, since the VPN should be accessible from our machines, or any other machine, we should associate the public route table to the VPN subnet.



To create a VPN, the most common choice is OpenVPN, let's create an EC2 machine in the VPN subnet and install OpenVPN, then configure it.

You can choose the OpenVPN instance from the AWS Marketplace since we will have a ready-to-use instance of OpenVPN.

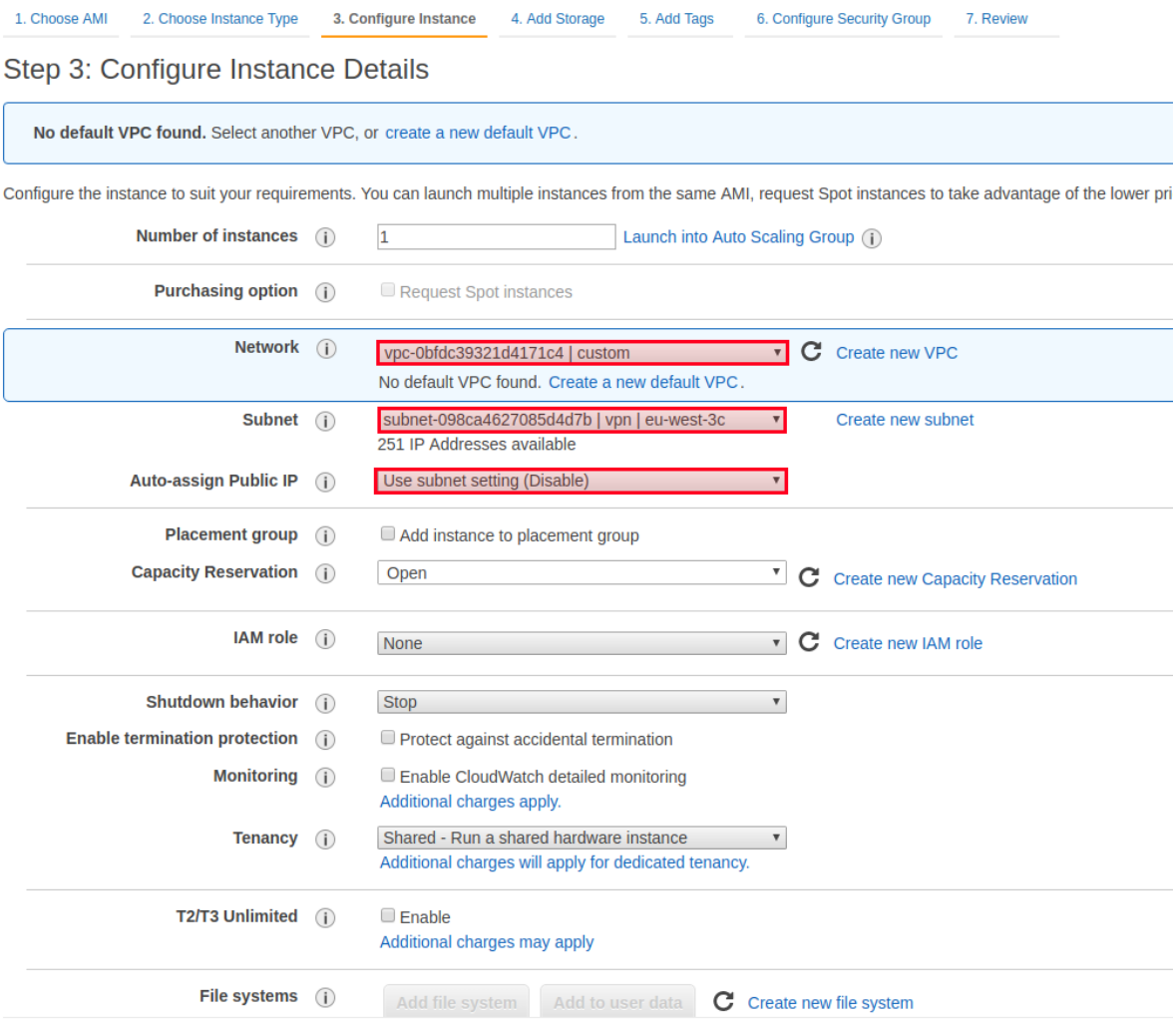
I recommend using the free tier eligible AMI.



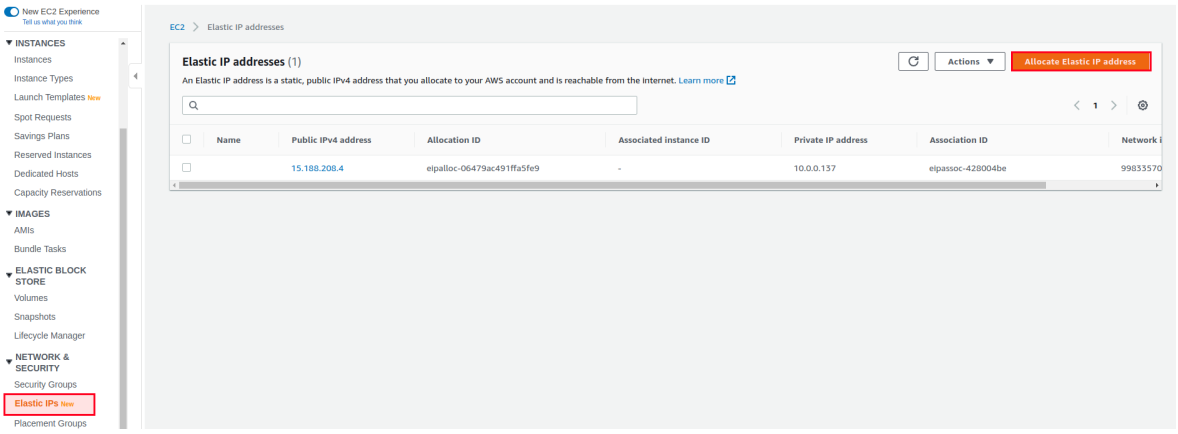
In the instance configuration step, choose the "custom" VPC we created previously, use the "vpn" subnet, and we don't need to assign a public IP at this step.

Public IPs are ephemeral, if we stop or shut down the instance, the IP address will be released, and we will not be able to use it. So if you are using the VPN for a production setup, disable the public IP, we are going to attach an EIP later. Note; As we have seen, EIP is a static IP address. It will not change when we shutdown our VPN instance.

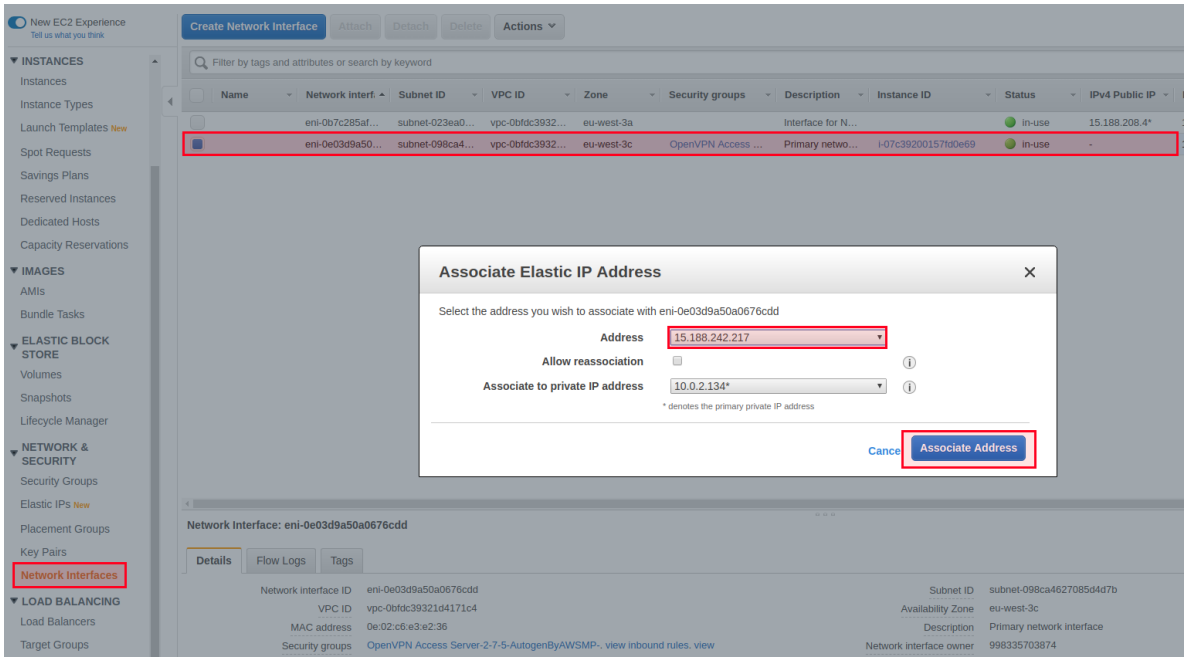
If you are using the VPN to test, choose to auto-assign a public IP.



If you didn't choose to assign a public IP, when the instance is created, go "Elastic IPs" and create a new one.



Now, go to "Network Interfaces" and associate the EIP to the network interface of the VPN machine.



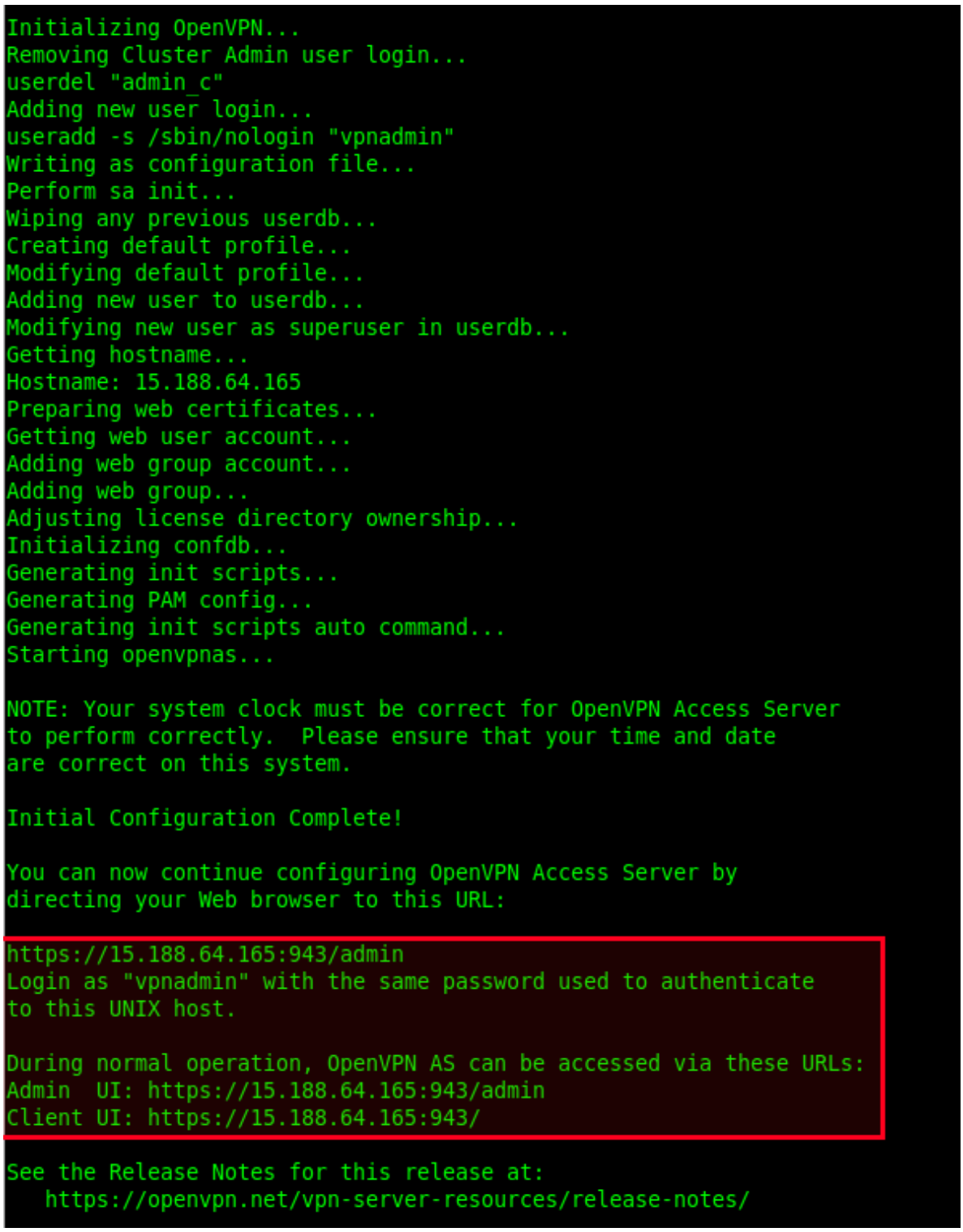
Now, you will be able to SSH into the OpenVPN machine using:

```
ssh -i "aws-tutorial.pem" openvpnas@15.188.64.165:943/admin
```

If you did choose to assign a public IP, use the auto assigned IP.

Make sure to use "openvpnas" as a username and use the EIP with the key you chose previously when creating the machine.

If you created the EC2 machine using the same OpenVPN AMI that I created, when you will ssh into the machine, you would be asked to configure OpenVPN. Choose the default settings, except the username and choose "vpnadmin" as a username or something different from "openvpn" then set the password.

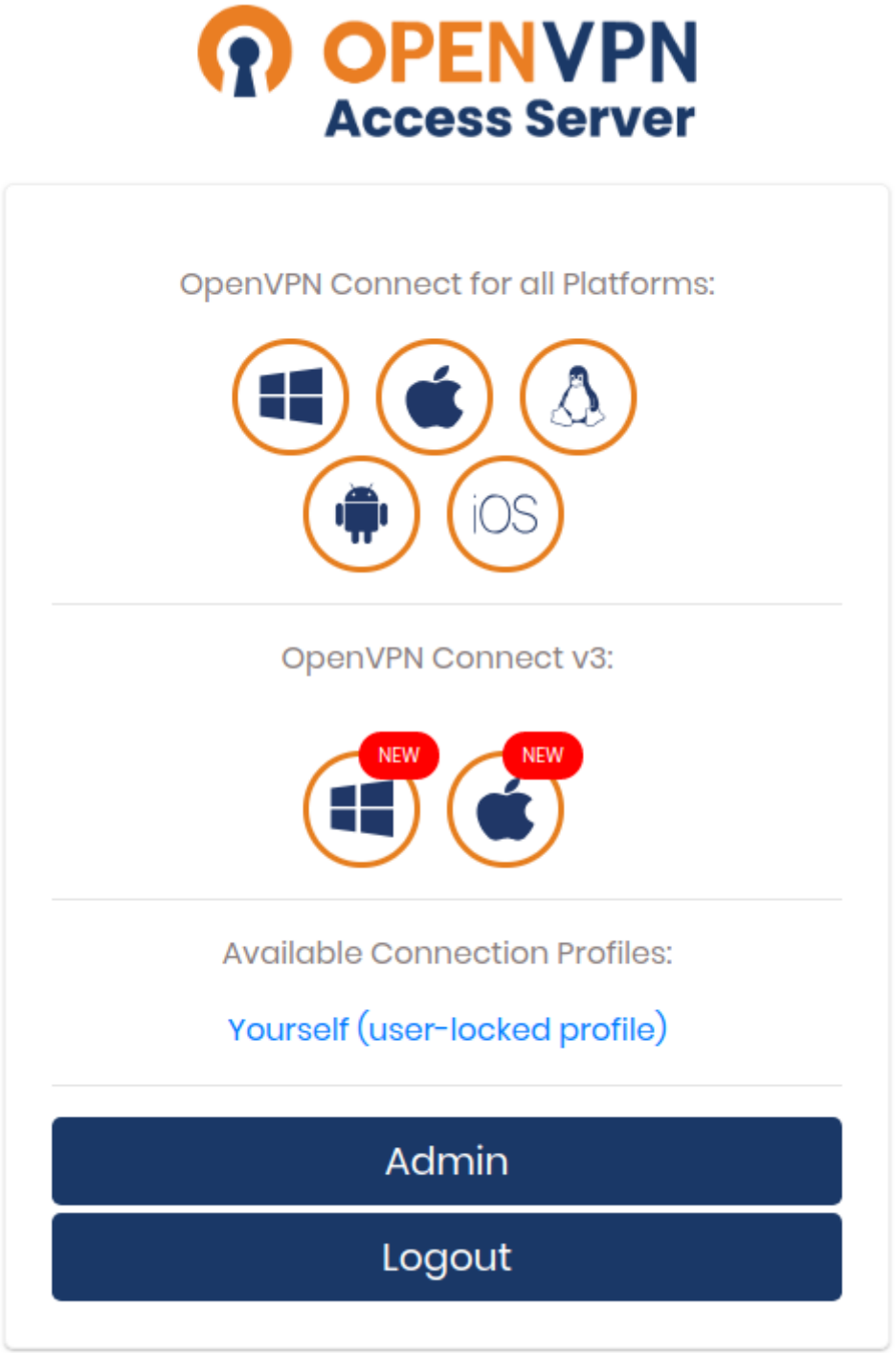


At the end of the configuration, OpenVPN will give you the IP address to access the admin dashboard.

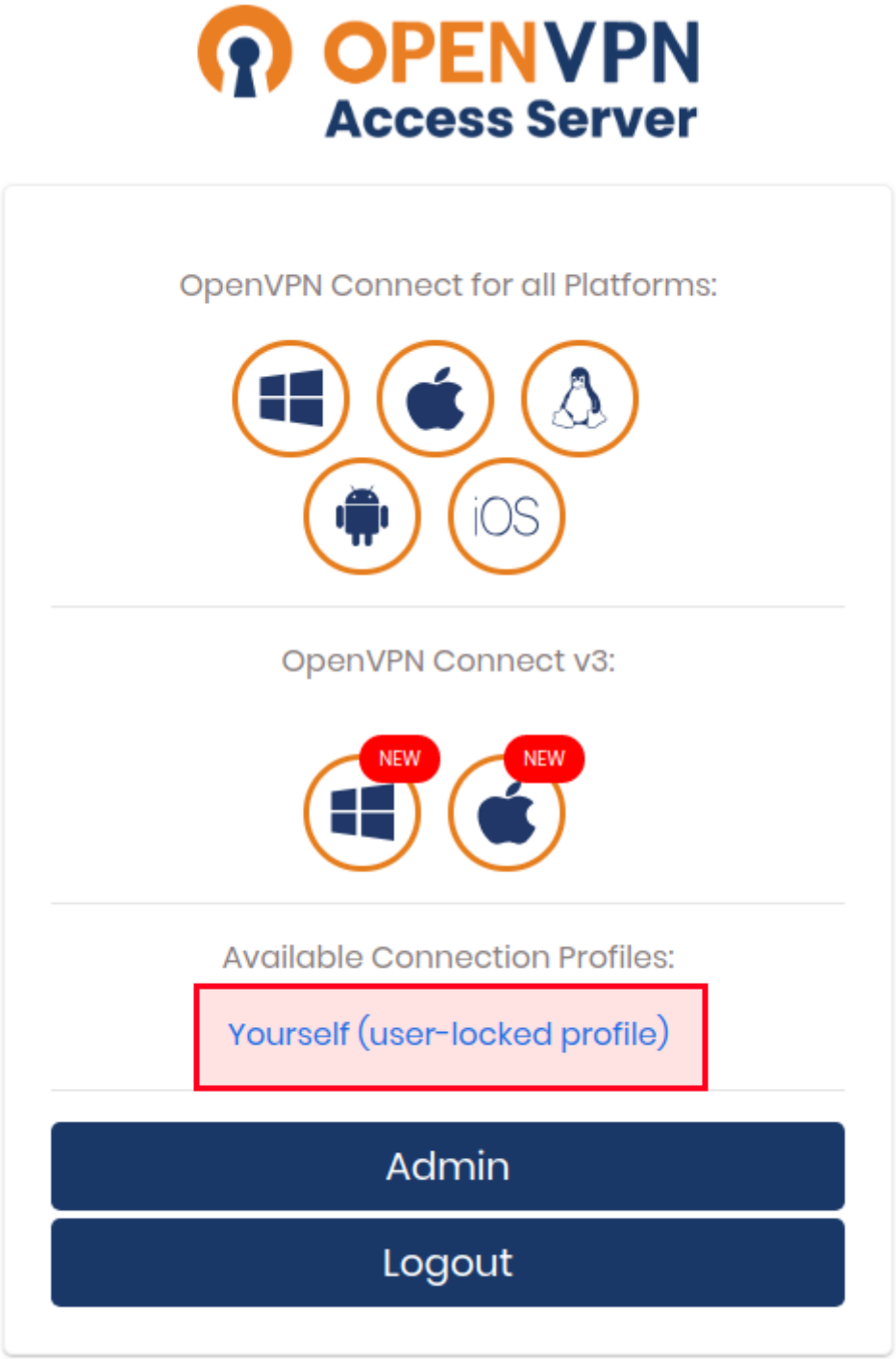
If you want to reconfigure OpenVPN, use:

```
sudo /usr/local/openvpn_as/bin/ovpn-init
```

To download an OpenVPN client, visit the client page using the same address but without "/admin".



Click on your OS, and you will be able to read the official documentation. Download your OpenVPN client, then go back to the same page and to download the "client.ovpn" file, click on your profile:

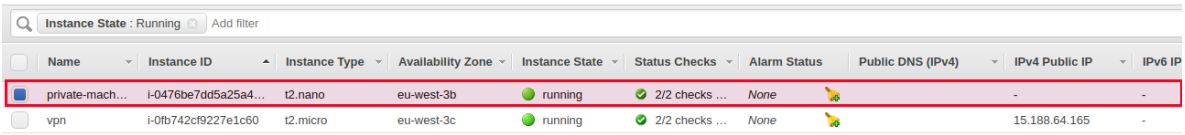


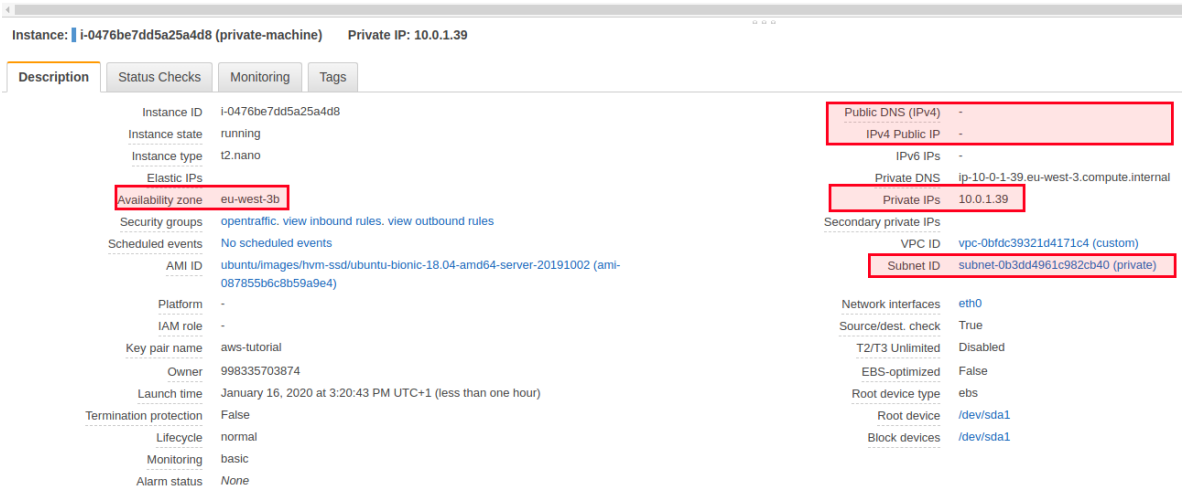
The file we downloaded helps in connecting your laptop to the VPN.

In Ubuntu, for example, install the package "openvpn" then:

```
openvpn --config client.ovpn
```

Now, to test if I can access EC2 private machine from my laptop, I created a machine on the private subnet:





Then using my laptop (after connecting to the VPN), I pinged the private IP of the machine:

```
ping 10.0.1.39



---
PING 10.0.1.39 (10.0.1.39) 56(84) bytes of data.
64 bytes from 10.0.1.39: icmp_seq=1 ttl=63 time=7.00 ms
64 bytes from 10.0.1.39: icmp_seq=2 ttl=63 time=10.8 ms
64 bytes from 10.0.1.39: icmp_seq=3 ttl=63 time=8.88 ms
64 bytes from 10.0.1.39: icmp_seq=4 ttl=63 time=10.7 ms
64 bytes from 10.0.1.39: icmp_seq=5 ttl=63 time=9.26 ms
64 bytes from 10.0.1.39: icmp_seq=6 ttl=63 time=8.14 ms
64 bytes from 10.0.1.39: icmp_seq=7 ttl=63 time=7.13 ms
64 bytes from 10.0.1.39: icmp_seq=8 ttl=63 time=9.38 ms
64 bytes from 10.0.1.39: icmp_seq=9 ttl=63 time=7.84 ms
64 bytes from 10.0.1.39: icmp_seq=10 ttl=63 time=9.87 ms
64 bytes from 10.0.1.39: icmp_seq=11 ttl=63 time=7.52 ms
64 bytes from 10.0.1.39: icmp_seq=12 ttl=63 time=10.4 ms
64 bytes from 10.0.1.39: icmp_seq=13 ttl=63 time=8.75 ms
```

So, if I can ping a machine with a private IP address, I'm sure that my OpenVPN settings work. You can do the same to test.

Lesson 7 - Deploying A Wordpress Website To An EC2 Instance (Video)

Note: Download the videos archive here.

This is a quick overview of using AWS EC2 to host a Wordpress blog. There are more details we are going to see later in order to make this installation highly available.

Notes about the Video

AWS EC2 Key Pairs

EC2 uses public-key cryptography to encrypt and decrypt login information. In the video, we create a key pair that we named "practical-aws" and it's ready to be used with the created machine. At this step, AWS keeps the public key, and you should download the private key. As mentioned previously, you should keep this key in a safe place, since we are going to use it to SSH into our EC2 machine.

Without it, you will not be able to SSH into the machine, and if it's lost, then you should follow some instructions that could take some time.

The private key you will create will be downloaded to your computer and stored under the name `practical-aws.pem`.

PEM is the acronym of "Privacy Enhanced Mail", and it's a Base64 encoded DER certificate. DER is a way to encode data and create a certificate.

The structure of a certificate is described using a standard called ASN.1 (Abstract Syntax Notation One). ASN.1 is a data representation language used in telecommunication, computer networks, and cryptography (our case).

For the sake of simplicity, I stored the PEM key under $HOME directory. In my case, when I would like to SSH into the created machine, I use a similar command to this one:

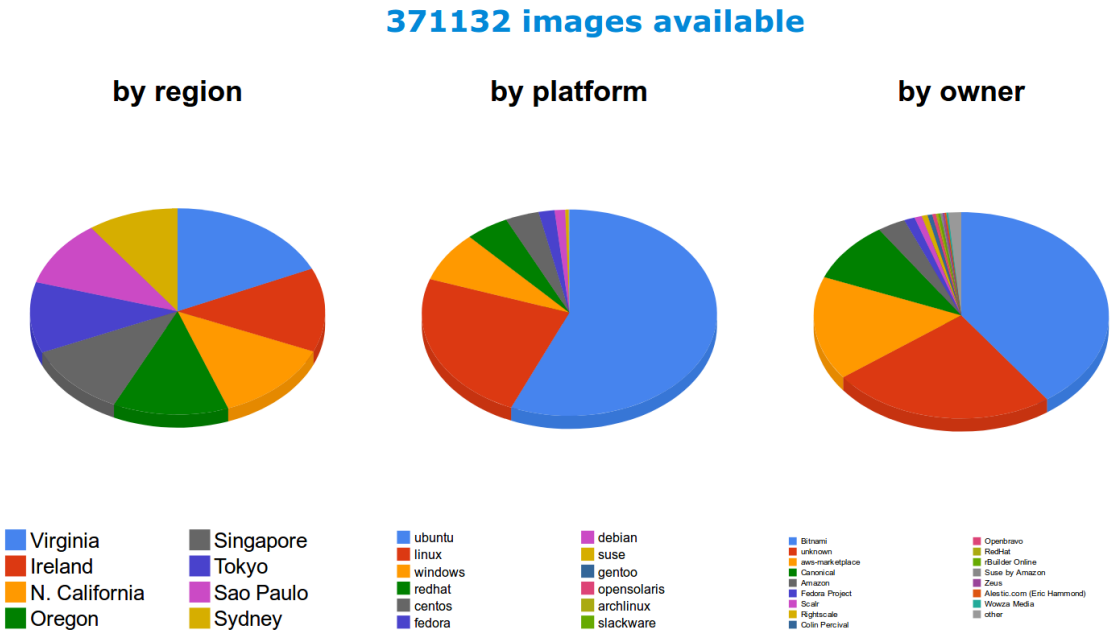
```
ssh -i practical-aws.pem ec2-user@ec2-198-51-100-1.compute-1.amazonaws.com
```

If you have a custom folder where you store your PEM files, like $HOME/my_keys/, then you should adapt the previous command to your case:

```
ssh -i $HOME/my_keys/practical-aws.pem ec2-user@ec2-198-51-100-1.compute-
1.amazonaws.com
```

Choosing EC2 OS

AWS provides you different possibilities when choosing the operating system you can use in an EC2 instance. In this guide, I usually make the choice of using Ubuntu. According to The Cloud Market, Ubuntu is the most used OS in AWS EC2 instances. In 2015, 70% of public cloud workloads and 54% of OpenStack clouds according to The Cloud Market and Zdnet.com.



You can use your preferred OS or your preferred GNU/Linux distribution, but you have to adapt some steps to your choice. So, for example, instead of using `sudo apt-get install nginx` to install Nginx in Ubuntu, you should use `sudo yum install nginx` if you are using CentOS. You can find instructions to install Nginx on Windows in the official documentation. If you don't have any problems using Ubuntu, just keep it since it will be easier for you to follow.

Useful Information

You can download Wordpress using this URL: https://wordpress.org/latest.zip

After unzipping WordPress, you need to install these PHP packages. It is possible that some of them are not required for a fresh Wordpress installation, but these are the most common libraries.

```
sudo apt install php7.0-mysql php7.0-curl php7.0-gd php7.0-intl php-pear php-
imagick php7.0-imap php7.0-mcrypt php-memcache  php7.0-pspell php7.0-recode
php7.0-sqlite3 php7.0-tidy php7.0-xmlrpc php7.0-xsl php7.0-mbstring php-gettext
php-fpm
```

You may also need the used Nginx configuration (configuration to copy/paste on `/etc/nginx/sites-available/default`) Note: Make sure to change wordpress.practicalaws.com by your domain or subdomain.

```
##
# You should look at the following URL's in order to grasp a solid understanding
# of Nginx configuration files in order to fully unleash the power of Nginx.
# http://wiki.nginx.org/Pitfalls
# http://wiki.nginx.org/QuickStart
# http://wiki.nginx.org/Configuration
#
# Generally, you will want to move this file somewhere, and start with a clean
# file but keep this around for reference. Or just disable in sites-enabled.
#
# Please see /usr/share/doc/nginx-doc/examples/ for more detailed examples.
##
# Default server configuration
```

```
#
server {
        listen 80 default_server;
        listen [::]:80 default_server;
        # SSL configuration
        #
        # listen 443 ssl default_server;
        # listen [::]:443 ssl default_server;
        #
        # Note: You should disable gzip for SSL traffic.
        # See: https://bugs.debian.org/773332
        #
        # Read up on ssl_ciphers to ensure a secure configuration.
        # See: https://bugs.debian.org/765782
        #
        # Self signed certs generated by the ssl-cert package
        # Don't use them in a production server!
        #
        # include snippets/snakeoil.conf;
        root /var/www/wordpress/wordpress;
        # Add index.php to the list if you are using PHP
        index index.php index.html index.htm index.nginx-debian.html;
        server_name wordpress.practicalaws.com;
        location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                #try_files $uri $uri/ =404;
                try_files $uri $uri/ /index.php?q=$uri&$args;
            # proxy_pass http://localhost:8080;
            # proxy_http_version 1.1;
            # proxy_set_header Upgrade $http_upgrade;
            # proxy_set_header Connection 'upgrade';
            # proxy_set_header Host $host;
            # proxy_cache_bypass $http_upgrade;
        }
        # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
        #
        location ~ \.php$ {
               include snippets/fastcgi-php.conf;
               # With php7.0-cgi alone:
               #fastcgi_pass 127.0.0.1:9000;
               # With php7.0-fpm:
               fastcgi_pass unix:/run/php/php7.0-fpm.sock;
        }
        # deny access to .htaccess files, if Apache's document root
        # concurs with nginx's one
        #
        #location ~ /\.ht {
        #       deny all;
        #}
}
# Virtual Host configuration for example.com
#
# You can move that to a different file under sites-available/ and symlink that
# to sites-enabled/ to enable it.
#
#server {
#       listen 80;
```

```
#       listen [::]:80;
#
#       server_name example.com;
#
#       root /var/www/example.com;
#       index index.html;
#
#       location / {
#               try_files $uri $uri/ =404;
#       }
#}
```

Choosing Your Database

Wordpress is made to use the MySQL database. Its codebase is very MySQL-centric. But you can also use some alternative databases like MariaDB (without adding any plugin or integration code) or other database engines like PostgreSQL or SQLite (not officially supported and need additional configurations).

To keep Wordpress portable and easy to use, choosing MySQL or MariaDB is a good choice.

AWS developed Aurora, a MySQL compatible database. According to AWS:

> Aurora is up to five times faster than standard MySQL databases and three times faster than standard PostgreSQL databases. It provides the security, availability, and reliability of commercial-grade databases at 1/10th the cost. Aurora is fully managed by Amazon Relational Database Service (RDS), which automates time-consuming administration tasks like hardware provisioning, database setup, patching, and backups. Aurora features a distributed, fault-tolerant, self-healing storage system that auto-scales up to 64TB per database instance. Aurora delivers high performance and availability with up to 15 low-latency read replicas, point-in-time recovery, continuous backup to Amazon S3, and replication across three Availability Zones.

You can use AWS Aurora with your Wordpress installation since it's MySQL compatible.

Route53 Record Types

Route53 supports different record types:

A Record Type AAAA Record Type CAA Record Type CNAME Record Type MX Record Type NAPTR Record Type NS Record Type PTR Record Type SOA Record Type SPF Record Type SRV Record Type TXT Record Type

- **A Record Type** The value for an A record is an IPv4 address
- **AAAA Record Type** The value for a AAAA record is an IPv6 address
- **CAA Record Type** CAA is the acronym of Certification Authority Authorization. This record is used to specify which certificate authorities are allowed to issue certificates for a domain.
- **CNAME Record Type** CNAME is the acronym of the Canonical Name record. It is a type of resource record in the DNS (Domain Name System), and it is used to specify that a domain name is an alias for another domain
- **MX Record Type** According to Wikipedia,

A mail exchanger record (MX record) is a type of resource record in the Domain Name System that specifies a mail server responsible for accepting email messages on behalf of a recipient's domain and a preference value used to prioritize mail delivery if multiple mail servers are available. For example, to use Google G Suite, you should configure your MX records with the following configuration:

```
ASPMX.L.GOOGLE.COM 1
ALT1.ASPMX.L.GOOGLE.COM 5
ALT2.ASPMX.L.GOOGLE.COM 5
ALT3.ASPMX.L.GOOGLE.COM 10
ALT4.ASPMX.L.GOOGLE.COM 10
```

1, 5, and 10 are the different priorities to give to each of the MX servers.

- **NAPTR Record Type** APTR (Name Authority Pointer) records are often used for applications in Internet telephony, NAPTR records are most commonly used with SIP protocol in conjunction with SRV records.
- **NS Record Type** The NS (Name Server) identifies the name servers for the hosted zone. NS indicates which DNS server is authoritative for a given domain.
- **PTR Record Type** PTR (Pointer records) associates an IP with a domain name. For instance, it could be used to to make 192.168.0.1 resolve to www.my-domain.com.
- **SOA Record Type** A start of authority (SOA) record provides information about a domain and the corresponding Amazon Route 53 hosted zone. Every domain must have a Start of Authority record.

SOA includes different information that you can find here.

- **SPF Record Type** SPF (Sender Policy Framework) is a simple email-validation system designed to detect email spoofing. This is deprecated.

RFC 7208, Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1, has been updated to say, "...[I]ts existence and mechanism defined in [RFC4408] have led to some interoperability issues. Accordingly, its use is no longer appropriate for SPF version 1; implementations are not to use it."

Now it is recommended to create a TXT record that contains the applicable value instead of using this type.

Example:

```
"v=spf1 include:amazonses.com ~all"
```

Read how to use TXT records with DKIM, SPF, and DMARC.

- **SRV Record Type** SRV (Service Record) defines the location, like the hostname and port number, of servers for specified services. The first three values are decimal numbers representing priority, weight, and port.

e.g :

```
1 2 80 my-hostname.m-domaine.com
```

- **TXT Record Type** TXT (Text Record) is a type of record used to let system administrators associate arbitrary text with a host. It is most commonly used with services and protocols like SPF, DKIM, DMARC, and DNS-SD ..etc.

Lesson 8 - Creating An Elastic File System - EFS

Introduction

EFS (Elastic File System) is described as simple, scalable file storage. It is usually used with Amazon EC2 instances in the AWS Cloud. EFS allows you to create and configure file systems in the cloud. One of the advantages of using EFS is its capacity elasticity: If you add files, it will grow automatically, and if you remove files, it will shrink automatically.

You can mount an EFS file system to more than one EC2 instance, which allows several instances to access the filesystem at the same time and share a common data source. An EFS filesystem can also be mounted to your on-premises servers, and in this case, you should use AWS Direct Connect in order to establish a dedicated network connection from your premises to AWS. EFS could be used in different scenarios like Big Data and analytics applications, web services, media processing, and backups.

EFS uses the NFS protocol, while NFSv4.0 is supported, AWS recommends that you use NFSv4.1.

Version 4 of NFS was released in December 2000, revised in RFC 3530, April 2003, and again in RFC 7530, March 2015. Versions 4 became the first version developed with the IETF (Internet Engineering Task Force). However, version 4.1 released in January 2010, has the ability to provide scalable parallel access to files distributed among multiple servers and version 4.2 was published in November 2016.

Practical Facts About EFS

To better understand how EFS works, let's enumerates the following 25 practical facts about Amazon Elastic File System:

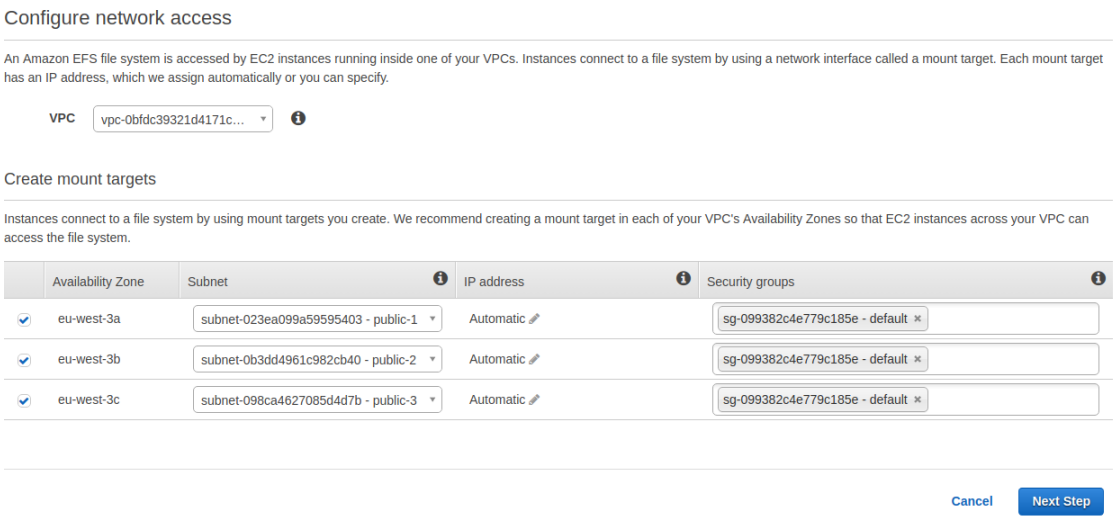
1. When you deploy an application that requires multiple virtual machines to access the same file system at the same time, EFS is the go-to tool if you use EC2 machines.
2. Think of EFS as a managed Network File System (NFS), that is easily integrated with other AWS services like EC2 or S3.
3. S3 could neither be an alternative to a Network File System nor a replacement for EFS. S3 is not a file system.
4. Sometimes when you think about using a service like EFS, you may also think about the "Cloud Lock" and its negative sides on your business, but to reduce the Time To Market, increase productivity and costs, EFS could be a good choice. Migrations are always possible later.
5. GlusterFS is an open-source alternative to EFS, but when thinking about the amount of work to manage it, the complexity to keep it stable and all of its maintenance efforts, some organizations would prefer a cloud lock over spending more money and time.
6. Choosing EFS or any other technology is a strategic choice, and every organization have its own choices.
7. EFS is extremely simple to use.
8. Creating an EFS file system can be done using the console or the CLI.
9. EFS uses NFSv4.x protocol that fixes issues of the v3 and comes with performance and security improvements while introducing a stateful protocol.

10. The replication of files between multiple availability zones in a region is covered automatically by EFS.
11. Making an EFS backup may decrease your production filesystem performance; the throughput used by backup counts towards your total file system throughput.
12. You can decide on your security (e.g., which EC2 instance could have access to an EFS file system) since EFS supports users and groups read, write and execute permissions, works with Security Groups, and could b integrated with AWS IAM.
13. EFS supports encryption.
14. EFS is SSD based storage, and its storage capacity and pricing will scale in or out as needed, so there is no need for the system administrator to do additional operations. It can grow to a petabyte-scale.
15. Throughput and IOPS (input/output operations per second) will scale according to how much your storage will grow/shrink.
16. It is possible to use your EFS from your on-premise data center and attach your storage directly to EFS using AWS Direct Connect.
17. EFS is expensive compared to EBS (almost 5 to 10x EBS pricing) - Make sure to check the prices for your region to get more detailed information.
18. EFS is not a magical solution for all your distributed FS problems; it can be slow in many cases. Test, benchmark, and measure to ensure your if EFS is a good solution for your use case.
19. EFS distributed architecture results in a latency overhead for each file read/write operation.
20. Most network file systems are not being used in the right way. This is a must-read: Amazon EFS Performance Tips.
21. If you have the possibility to use a CDN, don't use EFS, keep just files that can not be stored in a CDN.
22. It is evident to say, but don't use EFS as a caching system, sometimes you could be doing this unintentionally.
23. EFS now supports NFSv4 lock upgrading and downgrading, so yes, you can use an SQLite database that you store on an EFS file system... even if it was possible to do it before.
24. EFS is only compatible with Linux, if you're using another OS, find another solution.
25. Last but not least, even if EFS is a fully managed NFS, you could have performance problems in many cases, resolving this could take some time and needs some efforts, so brace yourself. A good way to measure and ameliorate performance is by integrating CloudWatch service and choosing the right metrics to focus on while benchmarking your application in a staging environment.

Creating Your First EFS Storage

In order yo do this, you can go to your AWS console, then click on create, choose your VPC.

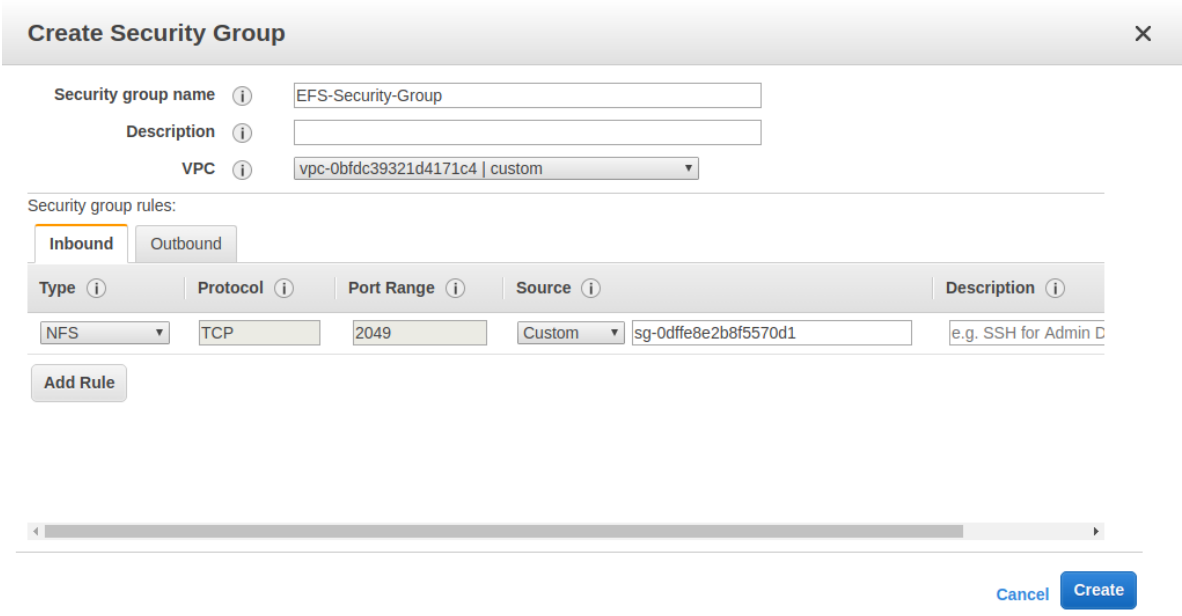
If your VPC supports N availability zones, you can create a filesystem with N availability zones. We are working with a VPC with 3 availability zones:



By default, AWS associates the default Security Group to your EFS filesystem, you can add yours. The Security Group acts like a firewall that controls the inbound and outbound traffic of your filesystem.

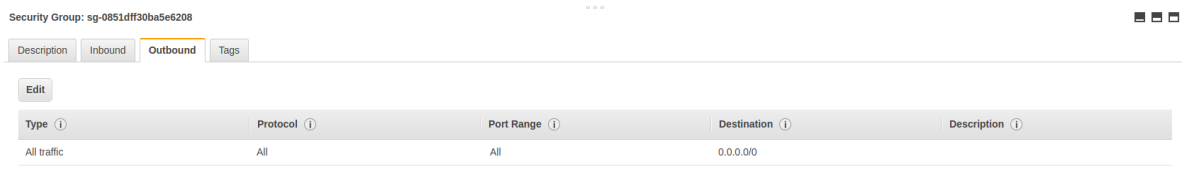
Note: You can specify up to 5 security groups.

In the following lesson, we are going to use an EC2 machine to connect to our filesystem. For this reason, we will create a new Security Group and add a rule to allow inbound access from the EC2 Security Group (the EC2 security group is identified as the source):



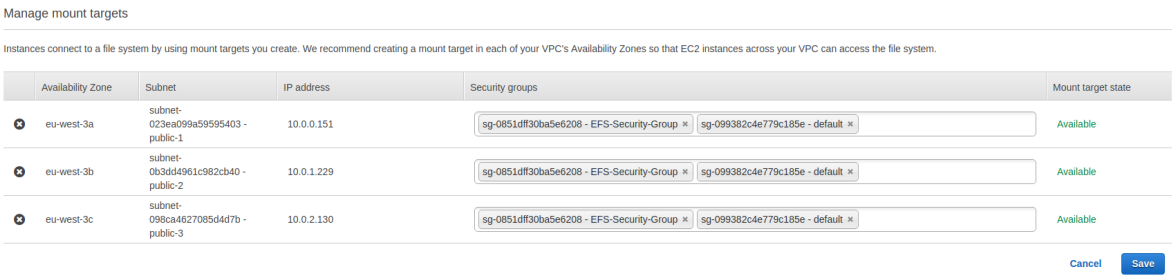
Notice the selected port: NFS.

For the outbound rules of our Security Group, we will keep the default values.



Now, we can add the new Security Group we created.

You can also create the EFS with the default Security Group then update it and add new Security Groups.



In the next step, you can choose if you want "Max I/O" performance mode, which is an optimized mode for applications where tens, hundreds, or thousands of EC2 instances are accessing the file system. You may also enable encryption.

Note: Choosing 3 availability zones will result in 3 mount targets.

Using AWS CLI To Create Your EFS Storage

We can do all of what's done before, using the CLI:

The first step is creating the filesystem using the following command with a creation token:

```
aws efs create-file-system --creation-token <a_random_token_you_choose> --region
<region>
```

This command will give you the filesystem id that we are going to use in the following commands:

```
{
    "PerformanceMode": "generalPurpose",
    "FileSystemId": "fs-1ca517d5",
    "CreationToken": "FileSystemForPracticalAWS",
    "LifeCycleState": "creating",
    "OwnerId": "998335703874",
    "SizeInBytes": {
        "Value": 0
    },
    "CreationTime": 1513728966.0,
    "NumberOfMountTargets": 0
}
```

If the filesystem is create we can create 3 mount targets in 3 different subnets where `sg-b88304c3` is the id of the security group we previously created:

First subnet:

```
aws efs create-mount-target --file-system-id fs-1ca517d5 --subnet-id subnet-
18e88743 --security-group sg-b88304c3 --region eu-west-1



---
{
    "IpAddress": "10.0.61.217",
    "MountTargetId": "fsmt-68d36fa1",
    "FileSystemId": "fs-1ca517d5",
    "OwnerId": "998335703874",
    "NetworkInterfaceId": "eni-f878cafb",
```

```
    "LifeCycleState": "creating",
    "SubnetId": "subnet-18e88743"
}
```

Second subnet:

```
aws efs create-mount-target --file-system-id fs-1ca517d5 --subnet-id subnet-
9493a7dd --security-group sg-b88304c3 --region eu-west-1



---
{
    "MountTargetId": "fsmt-6cd36fa5",
    "FileSystemId": "fs-1ca517d5",
    "SubnetId": "subnet-9493a7dd",
    "NetworkInterfaceId": "eni-8e547bbf",
    "LifeCycleState": "creating",
    "OwnerId": "998335703874",
    "IpAddress": "10.0.91.39"
}
```

Third subnet:

```
aws efs create-mount-target --file-system-id fs-1ca517d5 --subnet-id subnet-
55032132 --security-group sg-b88304c3 --region eu-west-1



---
{
    "MountTargetId": "fsmt-61d36fa8",
    "SubnetId": "subnet-55032132",
    "OwnerId": "998335703874",
    "LifeCycleState": "creating",
    "FileSystemId": "fs-1ca517d5",
    "NetworkInterfaceId": "eni-3cefde13",
    "IpAddress": "10.0.76.151"
}
```

We can describe the different mount targets using `aws efs describe-mount-targets --file-system-id fs-1ca517d5 --region eu-west-1` and you will get a similar output to the following one:

```
{
    "MountTargets": [
        {
            "IpAddress": "10.0.61.253",
            "LifeCycleState": "available",
            "SubnetId": "subnet-18e88743",
            "FileSystemId": "fs-62a517ab",
            "NetworkInterfaceId": "eni-008e3c03",
            "OwnerId": "998335703874",
            "MountTargetId": "fsmt-e6d36f2f"
        },
        {
```

```
            "IpAddress": "10.0.87.54",
            "LifeCycleState": "available",
            "SubnetId": "subnet-9493a7dd",
            "FileSystemId": "fs-62a517ab",
            "NetworkInterfaceId": "eni-985e71a9",
            "OwnerId": "998335703874",
            "MountTargetId": "fsmt-f8d36f31"
        },
        {
            "IpAddress": "10.0.79.11",
            "LifeCycleState": "available",
            "SubnetId": "subnet-55032132",
            "FileSystemId": "fs-62a517ab",
            "NetworkInterfaceId": "eni-1ee4d531",
            "OwnerId": "998335703874",
            "MountTargetId": "fsmt-f9d36f30"
        }
    ]
}
```


In each subnet, each mount target has an IP address associated with a network interface.

Lesson 9 - AWS Storage Type: EFS vs. EBS vs. S3

We already used EBS when we created our first EC2 machines. So why should we learn about another file system that works with EC2, since EBS is already here?

First of all, EBS and EFS are very different technologies.

It is true that EBS is faster than EFS. The distributed nature of EFS makes it slower. However, when it comes to throughput, EFS support a greater number of parallelized workloads and thread from a high number of EC2 instances.

Besides that, EFS has redundant storage across multiple AZs, while EBS stores to a single AZ.

If you need thousands of EC2 machines to access the same file system from multiple zones, this is not possible by EBS since the volume can be connected to a single machine, and this is the arena where EFS beats EBS.

We are not saying that EFS should be used in all cases and that the choice should not be EBS. But we should understand that given the technical differences, each technology has different use cases.

EBS can be used with boot volumes, databases, and other types of common data manipulations like ETL. EFS, on the other hand, should better be used with big data and analytics.

AWS claims that EFS provides the scale and performance required for big data applications that require high throughput to compute nodes coupled with read-after-write consistency and low-latency file operations. EFS is also used with image/video processing applications, contents like archives, or PDFs served by web servers and home directories.

The third type of storage that AWS provides is S3, which is a very popular object storage system and works with other services like CloudFront.

There are several differences between S3, EBS, and EFS; the main one is that S3 is not a file system but a key-based object store storage system built to store and retrieve any amount of data from anywhere on the Internet. The public access to S3 objects is usually done over HTTPS, and it supports 100 requests per second and can reach 300. It can be used to backup files (cold storage), web serving, content management, and media storage.

To understand all of the differences between these 3 technologies, this table may help:

	EBS	S3	EFS
Storage type	Block level storage	Object storage	Network file storage
Performance	From 250 MB/s to 1,000 MB/s of throughput	At least 3,500 requests/s to add data 5,500 requests/s to retrieve data	Can burst to 100 MiB/s of throughput.
Max Size	16TB	No limit	No limit
Max file size (indovidual files)	No limit	5Tb	16Tb
SLA	99.99%	99.99%	No public SLA (more details)
Access Control	IAM + security Groups	IAM + S3 Bucket/User Policies	IAM + security Groups
Access Throughput	Single EC2 instance in a single AZ	High (Millions of connections)	A thousand VMs
Backup	Snapshot	Versionning	EFS-to-EFS backup
VPC/Access	Accessible via 1 VPC	Public and private access	Accessible via 1 VPC
Scalability	No (manual scalability)	Highly scalable	Auto scalable (grow/shrink automatically)

Lesson 10 - Using EFS with an EC2 Instance (Video)

Note: Download the videos archive here.

We are going to re-create an EC2 machine, install Nginx and Wordpress, but we are going to use EFS this time.

Using the created EC2 machine, we are going to create an image that we can use later to start other similar EC2 machines (with the same OS and the same configurations: Wordpress + Nginx + EFS configuration ..etc.).

Note: In the video, I am using the new interface of RDS. If the option is still available, you will have the choice between using the new or the old interface.

Comments About Creating the EFS Mount Point

In order to automatically mount EFS at the system startup, you should add the mounting string to `/etc/fstab`. These are the recommended values for mount options:

```
nfsvers: 4.1
rsize: 1048576
wsize: 1048576
timeo: 600
retrans: 2
```

For this example, we created `/data` to mount the EFS filesystem. You can choose any other folder, e.g., `$HOME/data`.

In this case, you should not forget to change `/data` by `$HOME/data` when you edit `/etc/fstab` and when you configure Nginx website configuration using `/etc/nginx/sites-enabled/default`.

Final Notes

The best practice in production is using only the necessary resources, for example, not exposing the DB publicly and using tight security groups.

In the example provided in the video, our database can be publicly accessible since it uses an open security group, which increases the security risks.

But for practical reasons and in order to not add too much information at once, I made a choice to go slowly with details. Some people may find this kind of detail cumbersome while discovering AWS, even if for other people, things are obvious.

Also, the RDS instance should use a private subnet with the appropriate security groups, allowing the instance for inbound on 3306/TCP. If you want to administer your database, you can use the EC2 machine since we allowed SSH from everywhere; it can serve as a client to the database.

The best solution to consider in a production environment is creating a VPN or a bastion host, as we have seen.

The same security advice should be applied to EFS, which is publicly accessible in our example. Instead of using open security groups, you can set up tighter security rules by only choosing to allow the subnet of the EC2 machine using the EFS and only the NFS port.

Also, use strong passwords in production.

Lesson 11 - AWS Load Balancers

There are 3 types of load balancers. It may be confusing if this is the first time you use ELB (Elastic Load Balancer), and this is why we are going to study the differences between these 3 types.

We will make a choice to use the Classic Load Balancer (which is the old type of AWS load balancer) to show how to migrate from the old one to the Application Load Balancer. If you are using the VPC-EC2 instances, use the Application Load Balancer and don't follow the migration guide.

How Elastic Load Balancer Works

Load balancers are one of the most used infrastructure components, especially in medium to high traffic production environments. Basically, a load balancer acts as a reverse proxy and distributes network or application traffic across a pool of servers.

Load balancers are used for two main reasons. First of all, an application, especially if it is distributed, runs across multiple servers. Even in the traditional two-tier and three-tier architectures, your application always has frontend servers and backend servers.

In order to make these servers more resilient, there are two solutions mainly:

- Horizontal scaling
- Vertical scaling

While horizontal scaling means that you scale by adding more machines into your pool, vertical scaling is about adding more power (e.g., RAM) to an existing machine.

If you are running out of RAM or CPU, vertical scaling will solve your performance problem but not the availability of your application. If you decide that your API services, as an example, should have a server with 16GB of RAM, and even if you feed it with 32GB, there is no guarantee that a system problem occurs, and the machine goes down. This is when the horizontal scaling guaranteed both performance and uptime.

If you scale your API to 10 servers, it would be silly, too, if you allocate 10 public static IPs and use them in production. By adding a reverse proxy with load balancing capabilities, you will be able to have a single static public IP and use the capacity of all of your scaled pool.

Load balancers, in general, follow load balancing algorithms, like

- Round Robin (or "Next in Loop")
- Weighted Round Robin
- Source IP hash
- URL hash
- Least connections
- Weighted least connections.
- Random

AWS load balancers implement some of these algorithms; we are going to see this later.

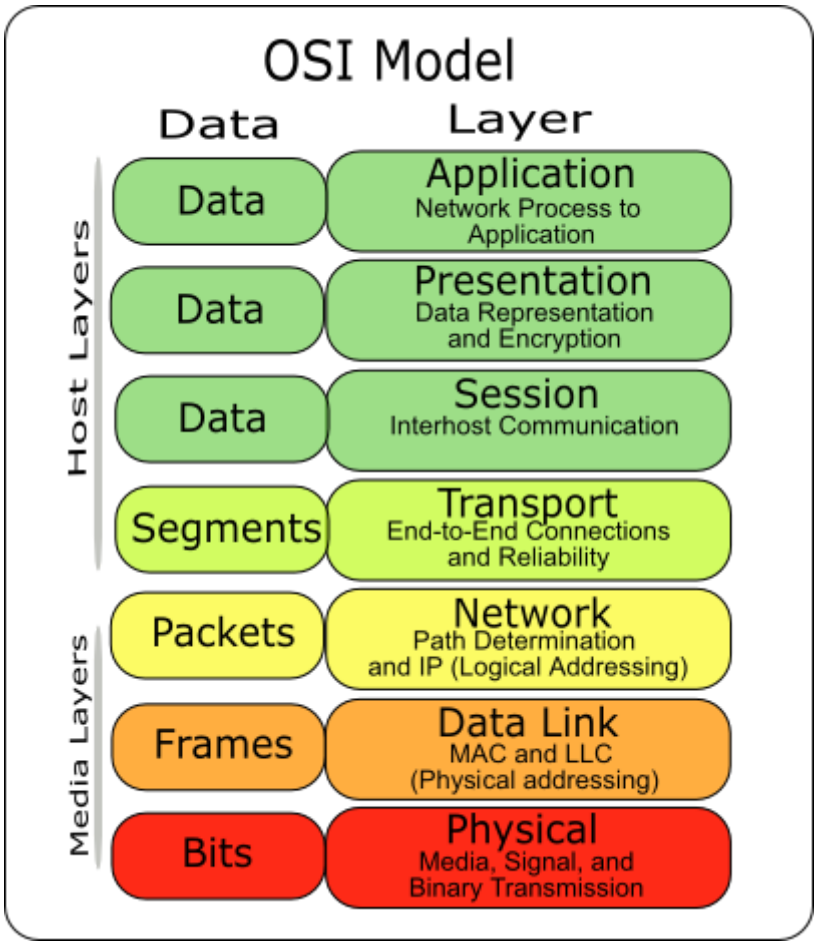
To summarize, load balancers are reverse proxy with load balancing capabilities. They are useful to increase the performance and uptime of a pool of servers running the same application. They follow load balancing algorithms. ELB is the managed load balancer of AWS.

Application Load Balancers

The first type of load balancer that AWS offers as a managed service is the Application Load Balancer.

This type of ELB acts at the HTTPS/HTTPS levels, which is the application layer.

Let's get back to some networking basics; HTTP(S) like FTP, IRC, SSH, and DNS are at the 7th layer of the OSI model. It makes routing based on the HTTP(S) path. In other words, if you have a website with two routes "website.com/path1" and "webstite.com/path2", the layer 7 load balancer can redirect the first request to a server or pool of servers, and the second path to another one. This is how the application load balancer or the 7th layer load balancer works.



source: wikimedia.org

So Application LB supports path-based routing and can route your requests to one or more ports on each instance in your cluster. You can't do this with a layer 3 or 4 load balancer.

We say that the Applications Load Balancer is "context-aware". Another common use case for ALB is redirecting HTTP to HTTPS-based upon rules, like hosts and paths. The Application Load uses the "X-forwarded-for" header containing the client IP address; this is not something we can do with other types of Load Balancers.

The Application Load Balancers are commonly used with microservices and containers.

Network Load Balancers

Network Load Balancer has an ultra-high performance. It supports TLS offloading at scale, centralized certificate deployment, support for UDP, and static IP addresses.

The Network Load Balancer operates at levels 3 and 4 of the OSI model and is not aware of higher-level protocols like HTTP/HTTPS. Since Network Load Balancers operates at lower levels, they are capable of handling millions of requests per second and securely maintaining low latency.

This type of load balancers is "context-less", and only care about the network packets and the information contained in these packets.

If you are not familiar with the OSI model, the types of data each layer uses.

Classic Load Balancers

In May 2009, Amazon introduced Elastic Load Balancing (ELB), Auto Scaling (which allows users to scale policies driven by metrics collected by Amazon CloudWatch), and Amazon CloudWatch (to track per-instance performance metrics like CPU load). The Application Load Balancer (ALB) was introduced later, and the first one was renamed Class Load Balancer, and it is still available to use.

Elastic Load Balancers are used with different AWS services, including EC2. To understand why Classic Load Balancers are "classic", we must understand that there two types of EC2 instances; EC2-Classic or EC2-VPC.

EC2-Classic is the first and original release of Amazon EC2. Instances run in a single, flat network that is shared with other customers. However, EC2-VPC instances run in a VPC that is logically isolated to only one AWS account.

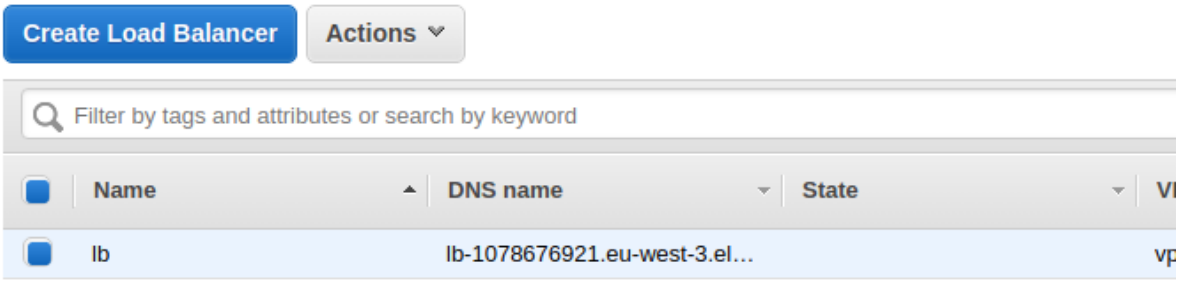
So if you have existing applications running in the EC2-Classic network, you should choose the Classic Load Balancer, otherwise choose the Network Load Balancer if you want network load balancing and Application Load Balancing if you need HTTP(S)-aware load balancing.

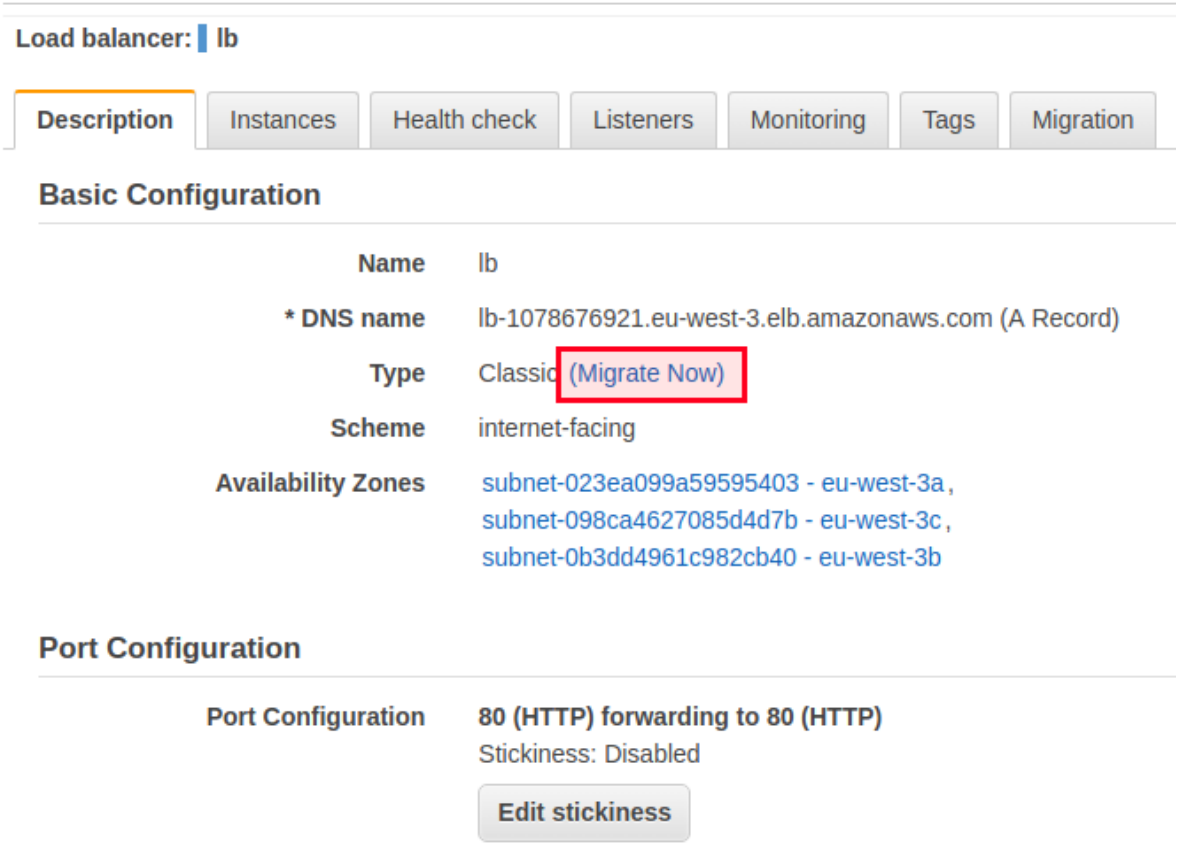
Note that you can use the Classic Load Balancer with EC2-VPC instances.

Both CLB and ALB supports features like sticky sessions, backend authentication and encryption, health checks, CloudWatch and autoscaling, access logs, connection draining, and cross-zone load balancing. However, only ALB supports WebSockets, HTTP/2, path-based routing, and routing to multiple ports of a single instance.

Migrating From Classic Load Balancer to Application Load Balancer

Migrating to ALB is easy; do not expect more than a few minutes to do it. There are 3 ways official ways to migrate a CLB to an ALB, the first one is creating a new EC2 instances pool and a new load balancer. The second way is using the migration wizard.





You will be asked to verify the configurations, once you click on "create", the migration will start.

Lesson 12 - Create a High Availability Wordpress Setup Using ELB, EC2, EFS (Video)

Note: Download the videos archive here.

In this lesson, we are going to use EC2 instances already configured to store data in EFS and autoscaling groups along with ELB to create an HA and scalable Wordpress platform.

Notes about the Video

ApacheBench

AB or ApacheBench is a single-threaded command line computer tool for measuring the performance of HTTP web servers. It was designed to test the Apache HTTP Server, but it is generic enough to test any other web server like Nginx, Tomcat, or IIS. This is an example of using AB:

```
ab -n 100 -c 10 "http://google.com"
```

This command will execute 100 HTTP GET requests, processing up to 10 requests concurrently to the specified URL "http://google.com". The -k option lets AB use the HTTP KeepAlive feature.

Other options are:

```
-n requests     Number of requests to perform
-c concurrency  Number of multiple requests to make at a time
-t timelimit    Seconds to max. to spend on benchmarking
                This implies -n 50000
-s timeout      Seconds to max. wait for each response
                Default is 30 seconds
-b windowsize   Size of TCP send/receive buffer, in bytes
-B address      Address to bind to when making outgoing connections
-p postfile     File containing data to POST. Remember also to set -T
-u putfile      File containing data to PUT. Remember also to set -T
-T content-type Content-type header to use for POST/PUT data, eg.
                'application/x-www-form-urlencoded'
                Default is 'text/plain'
-v verbosity    How much troubleshooting info to print
-w              Print out results in HTML tables
-i              Use HEAD instead of GET
-x attributes   String to insert as table attributes
-y attributes   String to insert as tr attributes
-z attributes   String to insert as td or th attributes
-C attribute    Add cookie, eg. 'Apache=1234'. (repeatable)
-H attribute    Add Arbitrary header line, eg. 'Accept-Encoding: gzip'
                Inserted after all normal header lines. (repeatable)
-A attribute    Add Basic WWW Authentication, the attributes
                are a colon separated username and password.
```

```
-P attribute    Add Basic Proxy Authentication, the attributes
                are a colon separated username and password.
-X proxy:port   Proxyserver and port number to use
-V              Print version number and exit
-k              Use HTTP KeepAlive feature
-d              Do not show percentiles served table.
-S              Do not show confidence estimators and warnings.
-q              Do not show progress when doing more than 150 requests
-l              Accept variable document length (use this for dynamic pages)
-g filename     Output collected data to gnuplot format file.
-e filename     Output CSV file with percentages served
-r              Don't exit on socket receive errors.
-m method       Method name
-h              Display usage information (this message)
-I              Disable TLS Server Name Indication (SNI) extension
-Z ciphersuite  Specify SSL/TLS cipher suite (See openssl ciphers)
-f protocol     Specify SSL/TLS protocol
                (SSL2, TLS1, TLS1.1, TLS1.2 or ALL)
```

Wordpress Mixed Content

In this part, we are going to see how to configure the load balancer to work with SSL. At the end of the video, you will see that Wordpress could be visited using the SSL certificate.

You can inspect your Wordpress website with HTTPS and see if there are any errors. If you have "mixed content" warnings or errors, you can fix this by installing the "SSL Insecure Content Fixer" Wordpress plugin.

Lesson 13 - Host a Static Website Using S3 (Video)

Note: Download the videos archive here.

Lesson 14 - Managing S3 Access & Policies

S3 offers many options to manage access, encryption, and logging. We are going to see how to use these features and how to manage your S3 data stores.

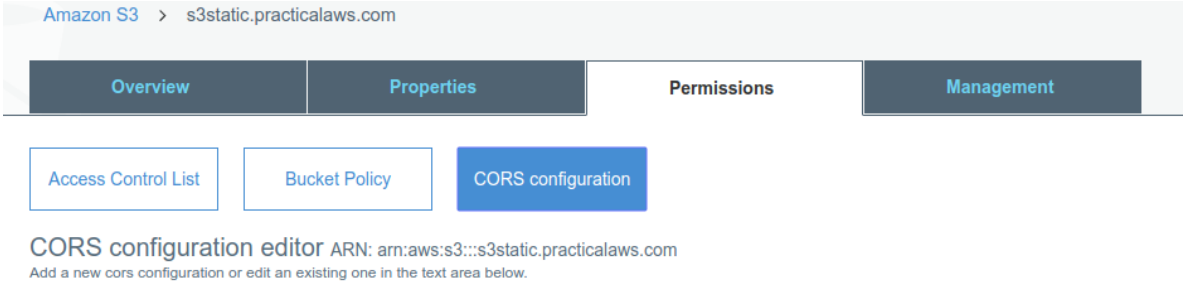
Cross-Origin Resource Sharing (CORS)

CORS defines how a web client loaded in one domain to interact with resources in a different domain. You can, for instance, allow one domain to interact with a bucket or a static website hosted on AWS S3.

Say, you have a website www.mywebsite.com that uses media content (e.g., photos, fonts ..) stored in an S3 bucket.

You have the choice of allowing everyone to use this content, but it's better to restrict the access to only your website. This could reduce your usage costs.

In order to configure CORS for S3, navigate to your S3 bucket then click on "Permissions" and "CORS management":



In the CORS configuration editor, you will find the default configuration:

```
<!-- Sample policy -->
<CORSConfiguration>
    <CORSRule>
        <AllowedOrigin>*</AllowedOrigin>
        <AllowedMethod>GET</AllowedMethod>
        <MaxAgeSeconds>3000</MaxAgeSeconds>
        <AllowedHeader>Authorization</AllowedHeader>
    </CORSRule>
</CORSConfiguration>
```

As you can notice, all origins are allowed by default, but if you want to allow only your domain name, change the configuration like the following:

```
<CORSConfiguration>
    <CORSRule>
        <AllowedOrigin>http://mywebsite.com</AllowedOrigin>
        <AllowedMethod>GET</AllowedMethod>
        <MaxAgeSeconds>3000</MaxAgeSeconds>
        <AllowedHeader>Authorization</AllowedHeader>
    </CORSRule>
</CORSConfiguration>
```

What if you are using a CloudFront distribution (with the domain mywebsite.com) that uses HTTPS?

In this case, you should choose the right protocol because you can configure your CloudFront to force redirect HTTP to HTTPS, or you have the choice to use both HTTP and HTTPS.

In the first case, make sure to add https://mywebsite.com instead of http://mywebsite.com; otherwise, the CloudFront distribution will not be able to access your bucket.

When you download your images and fonts from the S3 bucket, you will normally use the GET method.

What if you use POST or DELETE methods for other scenarios?

This can be resolved by changing your configuration and add the methods you will use.

```
<CORSConfiguration>
    <CORSRule>
        <AllowedOrigin>http://mywebsite.com</AllowedOrigin>
        <AllowedMethod>GET</AllowedMethod>
        <AllowedMethod>POST</AllowedMethod>
        <AllowedMethod>DELETE</AllowedMethod>
        <MaxAgeSeconds>3000</MaxAgeSeconds>
        <AllowedHeader>Authorization</AllowedHeader>
    </CORSRule>
</CORSConfiguration>
```

Another rule that you can configure is the **MaxAgeSeconds**. It's the time in seconds during which the browser will cache the response to a preflight OPTIONS request. When a browser caches this response, it will not make any new OPTIONS request during the configured time. The default value is 3000 seconds.

Note: According to the RFC2616, this method (OPTIONS) allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.

If the bucket allows HEAD, GET, PUT, DELETE, OPTIONS methods, the client should receive a response similar to this one:

```
200 OK
Allow: HEAD,GET,PUT,OPTIONS
```

Note: A preflight request is a CORS request that checks to see if this protocol (CORS) is understood.

Some other rules could be added like **ExposeHeader** that allows the client to access to a header. This example allows the client to access the "x-amz-request-id" header.

```
<ExposeHeader>x-amz-request-id</ExposeHeader>
```

Note: Some of AWS headers are only related to AWS. This type of header starts with "x-amz-". These are some of them:

- x-amz-content-sha256
- x-amz-date
- x-amz-security-token

AllowedHeader helps you to create a rule in order to specify which headers are allowed in a preflight request through the **Access-Control-Request-Headers** header

Note: The Access-Control-Request-Headers request header is used when sending a preflight request to let the distant server know which HTTP headers will be used. This is an example of an Access-Control-Request-Headers response:

```
Access-Control-Request-Headers: Content-Type
```

By default, the only authorized header in an S3 bucket is: **Authorization**

```
<!-- Sample policy -->
<CORSConfiguration>
    <CORSRule>
        ...
        <AllowedHeader>Authorization</AllowedHeader>
    </CORSRule>
</CORSConfiguration>
```

Note: We can use the "*" wildcard character in order to let many headers.

e.g:

```
<AllowedHeader>x-amz-*</AllowedHeader>
```

or

```
<AllowedHeader>*</AllowedHeader>
```

The wildcard could also be used with other elements like the **AllowedOrigin**:

e.g.:

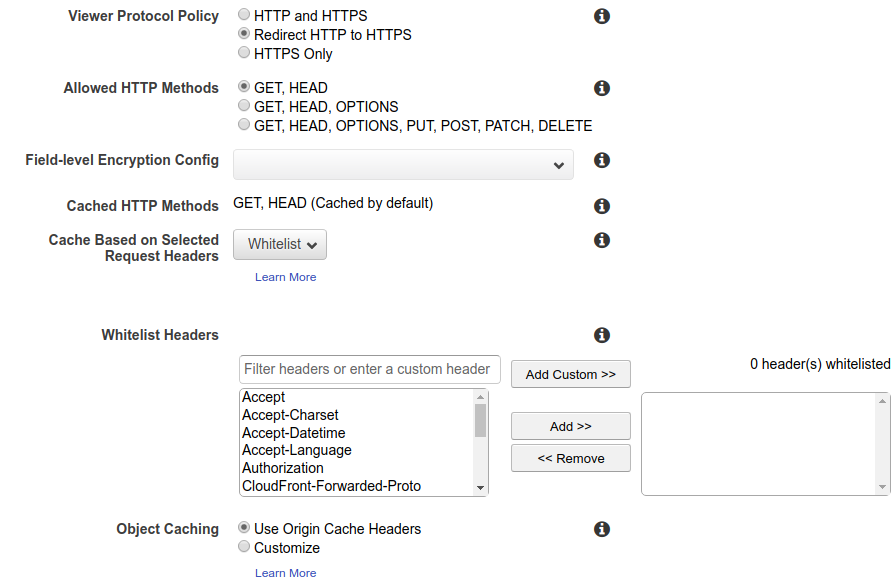
```
<AllowedOrigin>*</AllowedOrigin>
```

It is possible to create different configurations in the same file. In this example, we will use the POST method from one origin and disallow it on another one:

```
<CORSConfiguration>
 <CORSRule>
   <AllowedOrigin>http://www.mywebsite1.com</AllowedOrigin>
   <AllowedMethod>GET</AllowedMethod>
   <AllowedMethod>POST</AllowedMethod>
   <AllowedHeader>*</AllowedHeader>
 </CORSRule>
 <CORSRule>
   <AllowedOrigin>http://www.mywebsite2.com</AllowedOrigin>
   <AllowedMethod>GET</AllowedMethod>
   <AllowedHeader>*</AllowedHeader>
 </CORSRule>
</CORSConfiguration>
```

Remember that if you are hitting a CloudFront distribution instead of your S3 bucket, you should always make sure to enable the needed methods (GET, POST ..etc.) and whitelist the needed headers.

To do this, go to your CloudFront distribution Behaviour.

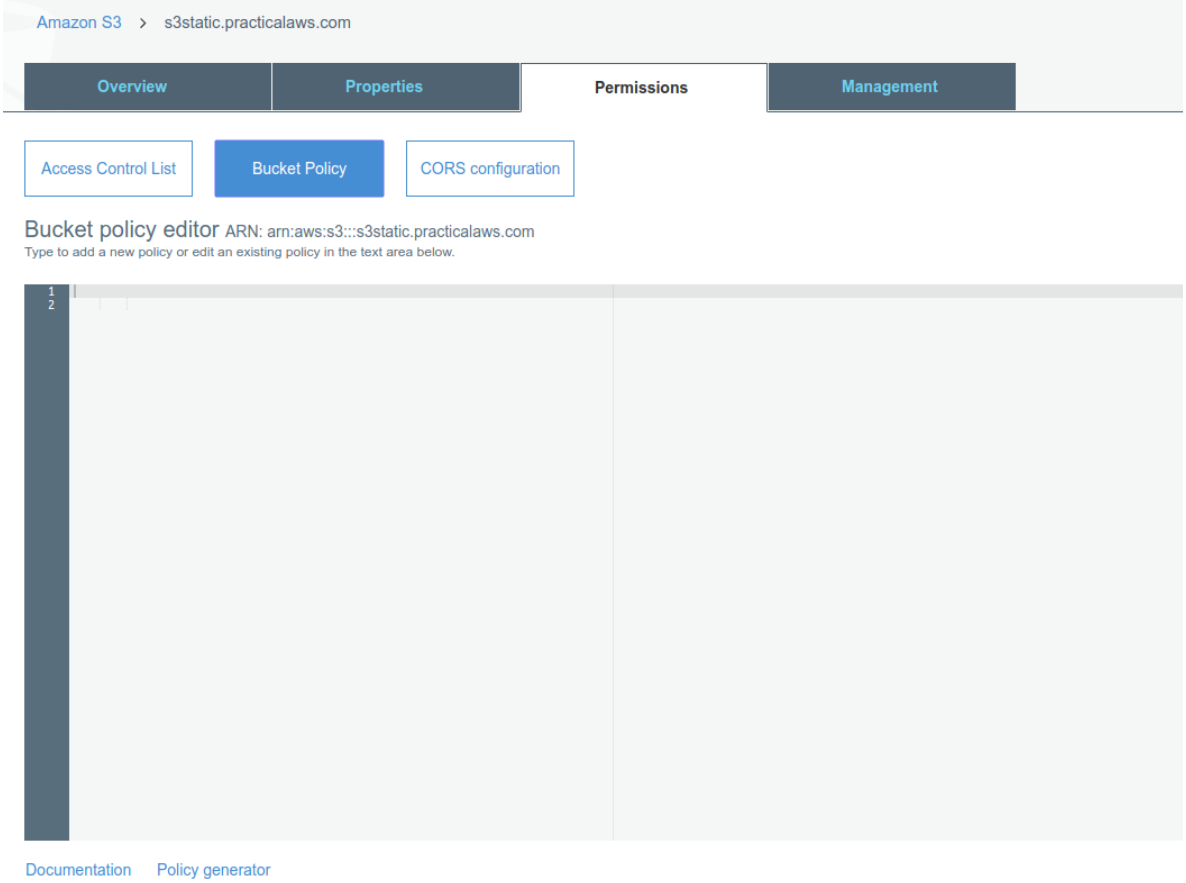


Another way of deciding how your S3 bucket will behave is by using the "Bucket Policy" feature. We are going to see this later.

```
<!-- Sample policy -->
<CORSConfiguration>
    <CORSRule>
        <AllowedOrigin>http://mywebsite.com</AllowedOrigin>
        <AllowedMethod>GET</AllowedMethod>
        <AllowedMethod>POST</AllowedMethod>
        <AllowedMethod>DELETE</AllowedMethod>
        <MaxAgeSeconds>3000</MaxAgeSeconds>
        <AllowedHeader>Authorization</AllowedHeader>
    </CORSRule>
</CORSConfiguration>
```

Managing Bucket Policies

We use the bucket policy to grant other AWS accounts, or IAM users access permissions for the bucket and the objects stored in it. Go to your AWS S3 console and click on "Bucket Policy:



If we are going to change the policy of the previously created bucket "s3static.practicalaws.com", then this is the ARN (Amazon Resource Name) of this bucket:

```
arn:aws:s3:::s3static.practicalaws.com
```

Use Case

Say your work in a startup that develops an Android application internally used by another company (let's call it Theta Corp). Your Android developers use S3 to store the new versions of the Android application, and your client downloads the new Android application (APK file).

At midnight, the Android application will list the S3 bucket and check if there is a newer version of the application, if it's the case, the mobile application has the right to download and install the latest version.

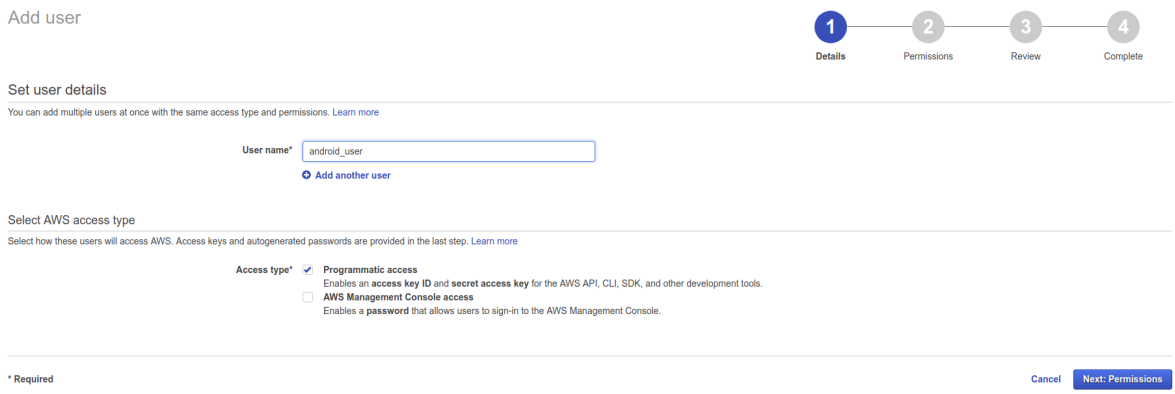
Theta Corp wanted to keep the application private, that's why the choice of using a public app store is not a good one, and that's why S3 is used as an app store for this application. Theta Corp used the application for its agents and wanted a maximum of privacy and security. Before your Android application could download the latest version, it should instantiate a new S3 client:

```
// Android code
AmazonS3Client s3Client = new AmazonS3Client(new
BasicAWSCredentials(ACCESS_KEY_ID, SECRET_KEY));
```

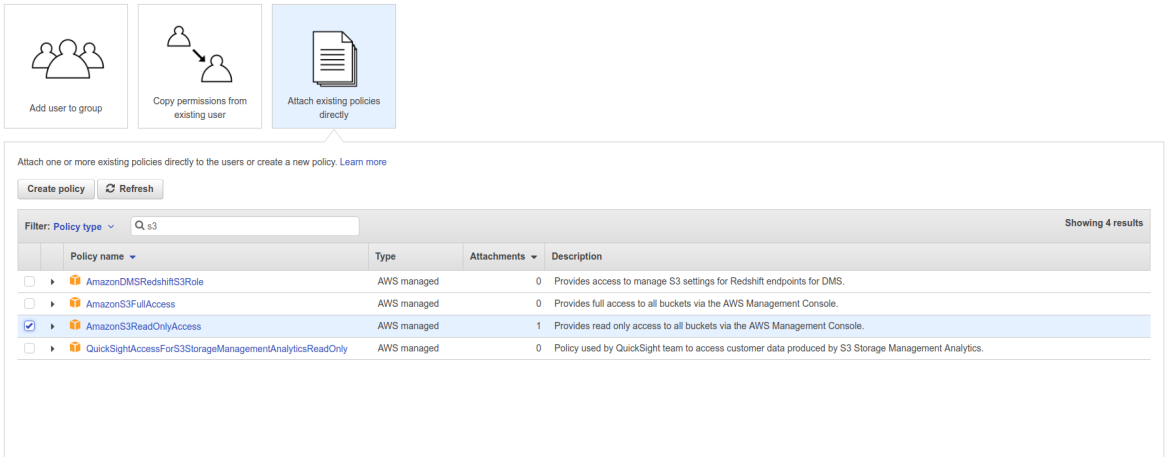
The first thing that may come to your mind is that the Android client should have the right to list and download the bucket and its objects, but never delete or upload files. This is why we are going to use a policy for the bucket.

The first step is creating a user, you can use the current user if you are working on a small project, but if you need more organization and user separation, it is better to create a new user.

Go to your IAM console and create a new user, then give it the right permissions:



Note: We do not need an AWS Management Console access. Only Programmatic access is needed. This user needs an S3 read-only access.

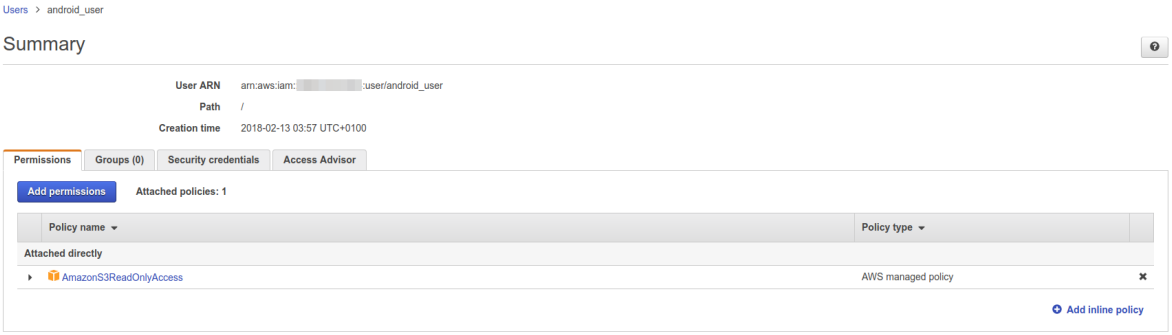


Click on next then “Create User”. After creating the user, AWS IAM will generate:

- Access key ID: "AKIAIBDN4L4S2CV4RBPA"
- Secret access key: "sL+FIT4rSaYdq+JfSpq+hz/VuqX0Dnp6sDYSmQEW"

Download the .csv file and keep it in a safe place. Your Android developers will use these generated keys.

Go back to your list of users and click on "android_user".



The blurred text is called AWS Principal. Let's consider this string as our Principal "998335703863". The user ARN becomes: `arn:aws:iam::998335703863:user/android_user`. Now we have these elements:

- The Principal: 998335703863
- The S3 bucket ARN: arn:aws:s3:::s3static.practicalaws.com

We can create a bucket policy:

```
{
    "Id": "Policy1518491345636",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1518491343637",
            "Action": [
                "s3:Get*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::s3static.practicalaws.com",
                "arn:aws:s3:::s3static.practicalaws.com/*"
            ],
            "Principal": {
                "AWS": [
                    "998335703863"
                ]
            }
        }
    ]
}
```

The ID element specifies an optional identifier for the policy we created.

The version element specifies the language syntax rules that are to be used to process our bucket policy. Only two values are possible:

- 2012-10-17: This is the current version of the policy language. Recommended.
- 2008-10-17: This was an earlier version of the policy language. Not recommended.

The Statement element is the main element in a bucket policy, and that's why it is required. The Sid (statement ID) is optional but must be unique within a JSON policy. You can assign a Sid value to each statement in a statement array. Action should contain the list of actions allowed.

We are going to grant our Android user all of the actions that starts with "Get":

- s3:GetBucketLocation

- s3:GetBucketCORS
- s3:GetIpConfiguration
- s3:GetAnalyticsConfiguration
- s3:GetObjectVersionForReplication
- s3:GetBucketAcl
- s3:GetBucketPolicy
- s3:GetObject
- s3:GetLifecycleConfiguration
- s3:GetObjectTorrent
- s3:GetBucketVersioning
- s3:GetReplicationConfiguration
- s3:GetBucketNotification
- s3:GetBucketLogging
- s3:GetInventoryConfiguration
- s3:GetBucketTagging
- s3:GetObjectVersionTorrent
- s3:GetObjectVersion
- s3:GetMetricsConfiguration
- s3:GetBucketRequestPayment
- s3:GetObjectAcl
- s3:GetBucketWebsite
- s3:GetObjectTagging
- s3:GetAccelerateConfiguration
- s3:GetObjectVersionTagging
- s3:GetObjectVersionAcl

The effect can allow or disallow the setup actions to a resource.

In our case, we are allowing the actions "s3:Get*" to:

- The bucket: `arn:aws:s3:::s3static.practicalaws.com`
- All of the objects inside this bucket: `arn:aws:s3:::s3static.practicalaws.com/*`

Principal is used to specifying the user (IAM user, federated user, or assumed-role user).

We are using an IAM user in our case.

In most cases, you don't need to write all of the JSON policy by yourself; you can generate it using the policy generator.

These are the steps to generate the policy automatically:

- Choose "S3 Bcuket Policy" as the "Type of Policy"
- Click on Allow
- Add your Principal
- Select "Amazon S3" in "AWS Service"
- Select all of the actions that starts with "Get"
- Add `arn:aws:s3:::s3static.practicalaws.com,arn:aws:s3:::s3static.practicalaws.com/*` to the list of ARNs.
- Click on "Add Statement"
- Click on "Generate"

You will have this generated JSON:

```
{
```

```
  "Id": "Policy1518492495045",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1518491343637",
      "Action": [
        "s3:GetBucketLocation",
        "s3:GetBucketCORS",
        "s3:GetIpConfiguration",
        "s3:GetAnalyticsConfiguration",
        "s3:GetObjectVersionForReplication",
        "s3:GetBucketAcl",
        "s3:GetBucketPolicy",
        "s3:GetObject",
        "s3:GetLifecycleConfiguration",
        "s3:GetObjectTorrent",
        "s3:GetBucketVersioning",
        "s3:GetReplicationConfiguration",
        "s3:GetBucketNotification",
        "s3:GetBucketLogging",
        "s3:GetInventoryConfiguration",
        "s3:GetBucketTagging",
        "s3:GetObjectVersionTorrent",
        "s3:GetObjectVersion",
        "s3:GetMetricsConfiguration",
        "s3:GetBucketRequestPayment",
        "s3:GetObjectAcl",
        "s3:GetBucketWebsite",
        "s3:GetObjectTagging",
        "s3:GetAccelerateConfiguration",
        "s3:GetObjectVersionTagging",
        "s3:GetObjectVersionAcl"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::s3static.practicalaws.com",
        "arn:aws:s3:::s3static.practicalaws.com/*"
      ],
      "Principal": {
        "AWS": [
          "998335703863"
        ]
      }
    }
  ]
}
```

The generated policy is the same as the one I used:

```
{
  "Id": "Policy1518492495045",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1518491343637",
      "Action": [
        "s3:Get*"
```

```
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::s3static.practicalaws.com",
        "arn:aws:s3:::s3static.practicalaws.com/*"
      ],
      "Principal": {
        "AWS": [
          "998335703863"
        ]
      }
    }
  ]
}
```

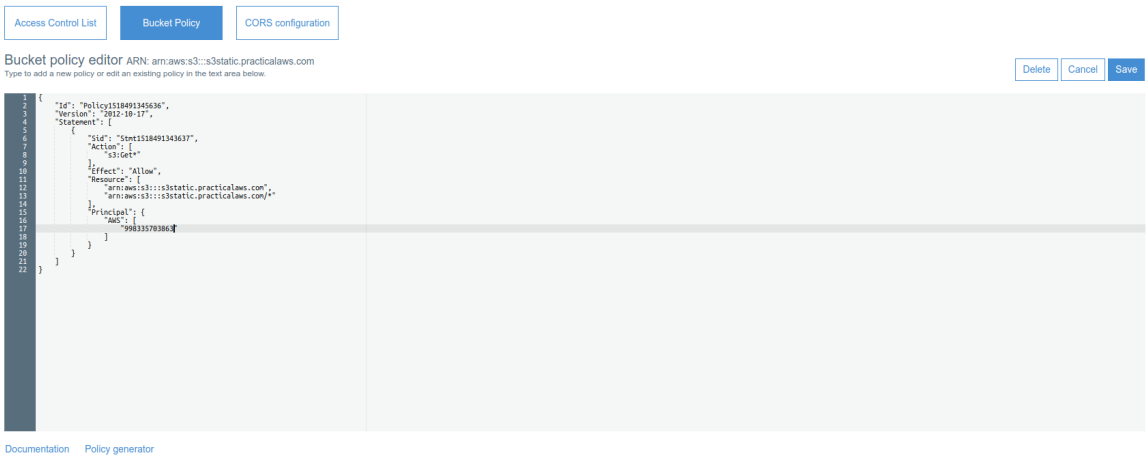
I replaced:

```
        "s3:GetBucketLocation",
        "s3:GetBucketCORS",
        "s3:GetIpConfiguration",
        "s3:GetAnalyticsConfiguration",
        "s3:GetObjectVersionForReplication",
        "s3:GetBucketAcl",
        "s3:GetBucketPolicy",
        "s3:GetObject",
        "s3:GetLifecycleConfiguration",
        "s3:GetObjectTorrent",
        "s3:GetBucketVersioning",
        "s3:GetReplicationConfiguration",
        "s3:GetBucketNotification",
        "s3:GetBucketLogging",
        "s3:GetInventoryConfiguration",
        "s3:GetBucketTagging",
        "s3:GetObjectVersionTorrent",
        "s3:GetObjectVersion",
        "s3:GetMetricsConfiguration",
        "s3:GetBucketRequestPayment",
        "s3:GetObjectAcl",
        "s3:GetBucketWebsite",
        "s3:GetObjectTagging",
        "s3:GetAccelerateConfiguration",
        "s3:GetObjectVersionTagging",
        "s3:GetObjectVersionAcl"
```

by:

```
        "s3:Get*"
```

Now copy the generated policy and paste it in the "Bucket policy editor".



Make sure to change the above values by your values (Principal, name of the bucket ..etc.). Now, when your Android developer wants to access the latest version of the APK file in the S3 bucket, she/he will be able to perform all of the actions that starts with "Get". Of course, the developer should use the good keys:

```
AmazonS3Client s3Client = new AmazonS3Client(new
BasicAWSCredentials("AKIAIBDN4L4S2CV4RBPA",
"sL+FIT4rSaYdq+JfSpq+hz/VuqX0Dnp6sDYSmQEW"));
```

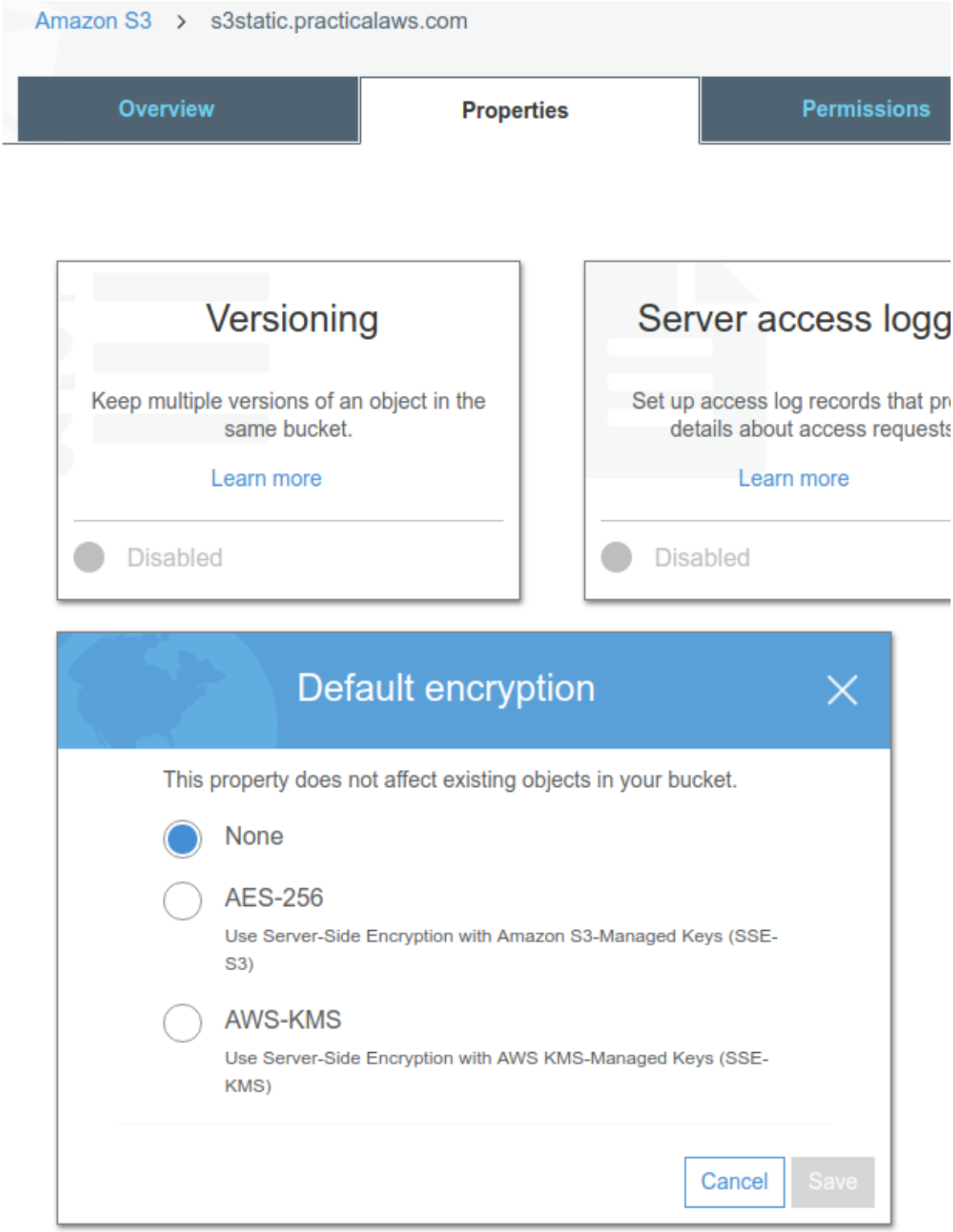
S3 Data Encryption

Amazon S3 uses TLS/SSL by default, but for some reason, Theta Corp also wanted to encrypt data at rest; that's why we are going to use data encryption.

AWS S3 supports 2 types of encryption:

- Server-side: Amazon S3 will encrypt a bucket object before saving it to the disk. An encrypted object is decrypted when downloaded.
- Client-side: You will manage the encryption process by yourself and use encryption keys to encrypt your objects before uploading them to S3.

The server-side encryption can be done using the AWS S3 dashboard.



There are two types of server-side encryption that S3 provides:

- Using AWS KMS-Managed Keys (**SSE-KMS**): Customer-provided encryption keys
- Using S3-Managed Encryption Keys (**SSE-S3**): Amazon provides and manage the keys

SSE-S3 is simpler to use than SSE-KMS since it is managed by AWS. SSE-S3 uses 256-bit Advanced Encryption Standard (AES-256).

Server-side encryption encrypts only the object data.

Let's create a new bucket:

```
aws s3 mb s3://enc-practicalaws
```

Upload a file:

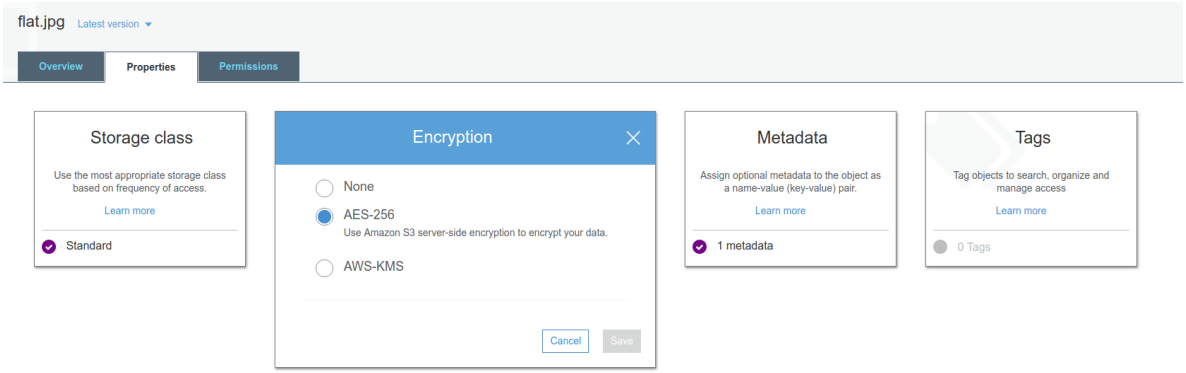
```
aws s3 cp my_app.apk s3://enc-practicalaws
```

Let's copy this file "app.apk" to the bucket:

```
aws s3 cp app.apk s3://enc-practicalaws
```

You can also upload your file using the AWS S3 dashboard.

We are now going to encrypt the uploaded file. On your AWS S3 dashboard, click on the file, click on "properties" then "Encryptio". Choose the AES-256 encryption (SSE-S3):



Click on "Save". Our APK file is now encrypted.

Your Android developer will upload the APK files to the bucket, but she/he may forget that any uploaded APK should be encrypted, she/he may upload a file without encryption ! In this case, we are going to use the bucket policy to deny unencrypted files upload.

```
    {
      "Sid": "DenyUnEncryptedObjectUploads",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::enc-practicalaws/*",
      "Condition": {
        "Null": {
          "s3:x-amz-server-side-encryption": "true"
        }
      }
    }
```

This will deny any request that does not include the "x-amz-server-side-encryption" header.

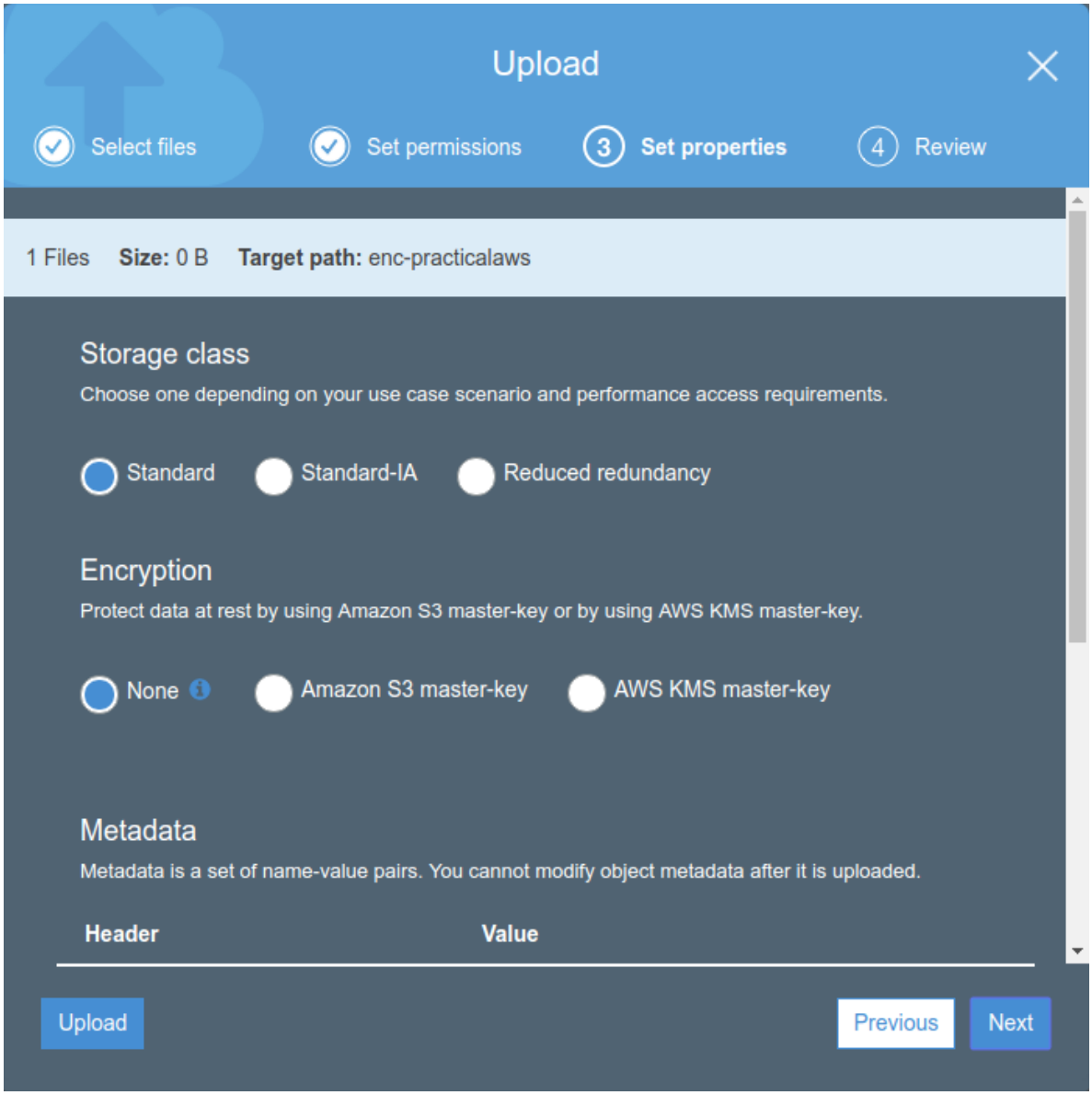
The Android developer may use another type of encryption like the AWS-KMS encryption, but we only want to use the AES-256 encryption:

```
    {
      "Sid": "DenyUnEncryptedObjectUploads",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::enc-practicalaws/*",
      "Condition": {
        "Null": {
          "s3:x-amz-server-side-encryption": "true"
        }
      }
    }
```

The bucket policy will include both statements, and it will look like this:

```
{
  "Version": "2012-10-17",
  "Id": "PutObjPolicy",
  "Statement": [
    {
      "Sid": "DenyIncorrectEncryptionHeader",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::enc-practicalaws/*",
      "Condition": {
        "StringNotEquals": {
          "s3:x-amz-server-side-encryption": "AES256"
        }
      }
    },
    {
      "Sid": "DenyUnEncryptedObjectUploads",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::enc-practicalaws/*",
      "Condition": {
        "Null": {
          "s3:x-amz-server-side-encryption": "true"
        }
      }
    }
  ]
}
```

Now, in order to test, upload a file with the wrong encryption or without encryption at all. You should have an error at this step.



Note: Amazon S3 encrypts each object with a unique key. It encrypts the key itself with a master key that it regularly rotates.

If you would like to get more details about uploading or downloading an encrypted file, you can get more logs by adding `--debug` to the AWS S3 CLI command:

```
aws s3 cp s3://enc-practicalaws/app.apk /tmp/ --debug
```

Example:

```
2018-02-13 15:44:48,
760 - MainThread - botocore.endpoint - DEBUG - Sending http request:
<PreparedRequest[
   HEAD
]>
2018-02-13 15:44:49,
522 - MainThread - botocore.parsers - DEBUG - Response headers:{
   'ETag':'"f24198ca0f9c4d75a9786b4308ea8470"',
   'Server':'AmazonS3',
   'x-amz-server-side-encryption':'AES256',
   'Date':'Tue, 13 Feb 2018 14:44:50 GMT',
   'Accept-Ranges':'bytes',
   'x-amz-request-id':'DECC475B893E025E',
```

```
   'x-amz-id-
2':'bySlyKxcghVN+rLG7yHMnNjOjZF/ypCOao7DJaZuc2o74le+8O2CT7d+CVq1BaygM1U3n94QwPM=
',
   'Content-Type':'application/octet-stream',
   'Content-Length':'207663',
   'Last-Modified':'Tue, 13 Feb 2018 14:43:05 GMT'
}
```

S3 Logging

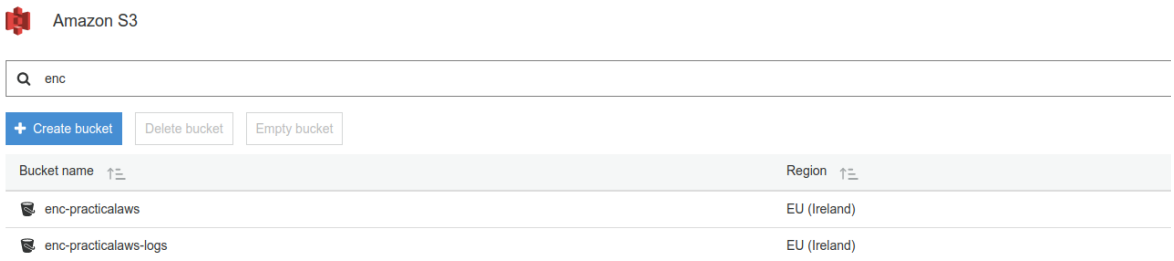
Theta Corp applications stored in S3 are secure, and everything seems to work fine, but in 3 months after starting the project, you realized that the costs associated with S3 are increasing. While the number of users is the same, and the frequency of deployment did not change, the bills of S3 services were increasing.

One of the things your team was thinking about was that some Android devices that are probably stuck in a loop of infinite downloads .. this scenario could happen, and it will consume your bandwidth and, of course, increase bills.

One of the first things to do in order to troubleshoot your system is activating logging.

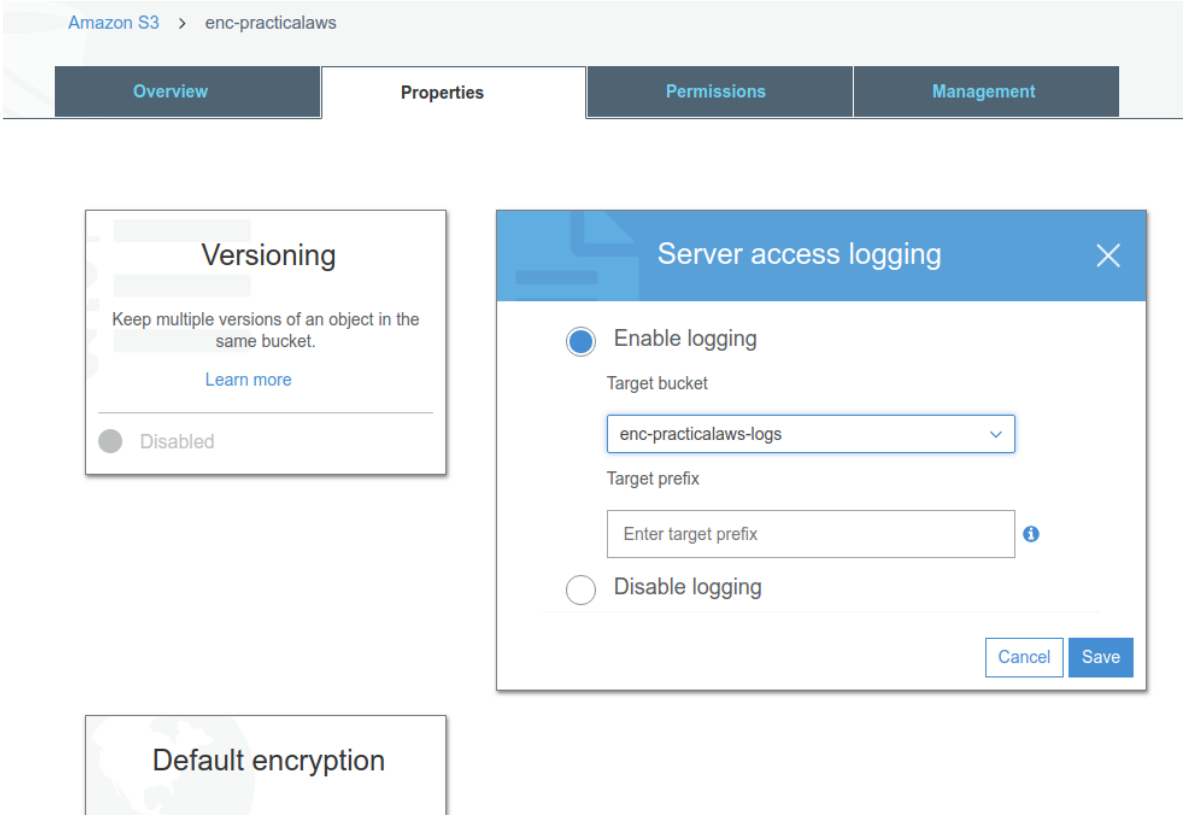
S3 logs are helpful in security and access audits. To activate S3 logs, you should create a new bucket where you will store the logs of the "enc-practicalaws" bucket.

Make sure that the source bucket (the bucket to log) and the target bucket (the bucket that stores logs) are in the same region. Let's name the target bucket "enc-practicalaws-logs":



Go to the source bucket "enc-practicalaws" and click on it, then click on properties and choose "Server access logging". Activate the option "Enable logging", then choose the target bucket "enc-practicalaws-logs".

If you would like to add a prefix to your logs to make them easily identified, you can do it by filling the prefix in the "Target prefix" text input. You can keep it empty if you don't need this.



Click on save.

Lesson 15 - Host a Static Website Using S3 and CloudFront (video)

Note: Download the videos archive here.

Notes about the Video

I want to draw your attention to the fact that Amazon Certificate Manager (ACM) that we use to generate an SSL certificate is not only available in N. Virignia region, but the service was made available by AWS in many other regions.

Lesson 16 - Optimizing CloudFront for Speed & Security

One of your clients, Larry, runs a part of his business website on a Wordpress instance. Larry uses a dedicated webserver to host his high traffic Wordpress website. He hosts thousands of videos watched by millions of visitors each day.

Larry thinks his web hosting company charges him a lot comparing to the bandwidth allocated to his dedicated server. Many of Larry's website visitors have the same problem: Videos buffer really slow.

While the dedicated server is hosted in the US, Larry's website visitors come from everywhere in the world. The goal of your client is to keep his dedicated server. He asked you if you have a solution to make video file streaming faster. If you followed this course, you would probably think about CloudFront.

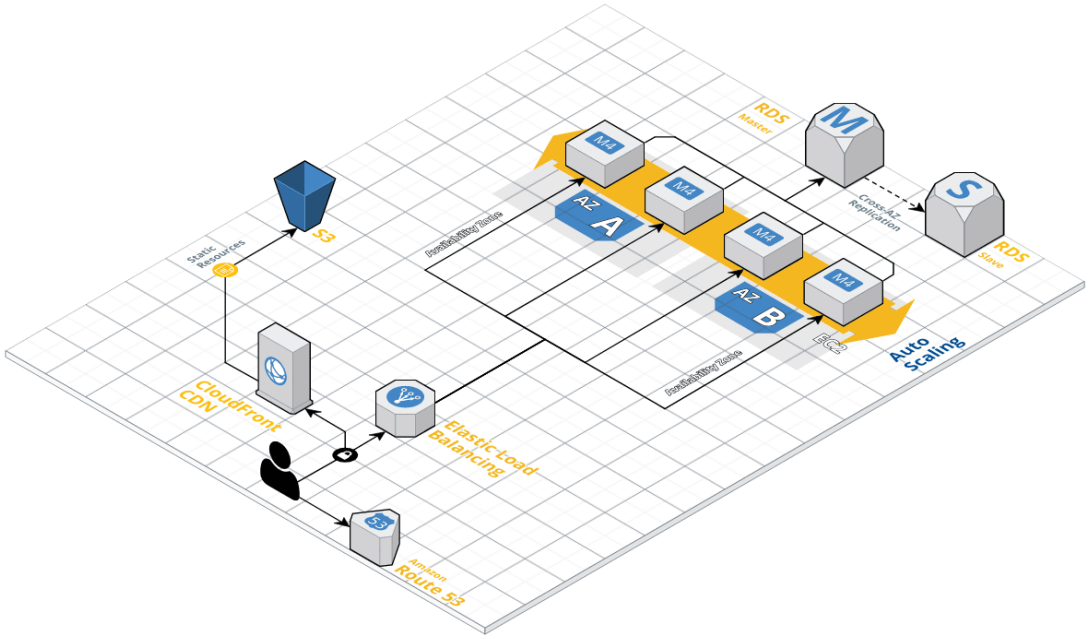
Using CloudFront With Wordpress

CloudFront is optimized for some use cases like:

- Hosting static contents like videos
- Hosting content consumed by people from different regions in the world
- Hosting content that could be cached in CloudFront "Regional Edge Caches"

Even if Larry's website uses Wordpress, videos are static elements that could be deployed to CloudFront. CloudFront will distribute videos in different regions in the world; this will decrease latency and increase the streaming speed and even costs.

This is a typical architecture of an HA Wordpress website using S3, CloudFront, Route53, ELB, EC2, and RDS:



The only difference is that Larry wants to keep his on-premises servers, but since we are going to host all the videos in CloudFront, we will keep Route53, CloudFront, and S3.

Videos will be deployed to S3; then, a CloudFront distribution should be created from the S3 bucket.

Larry's website uses the domain `larryswebsite.com`. We are going to use the following subdomain for the CloudFront distribution `videos.larryswebsite.com`. Now when Larry wants to post a video, he should use a similar URL to this one `videos.larryswebsite.com/video.mp4`.

To post the video to blog post, we should embed it using HTML:

```
<video width="320" height="240" controls>
<source src="http://videos.larryswebsite.com/video.mp4" type="video/mp4">
Your browser does not support the video tag.
</video>
```

Of course, there are some plugins you can use to integrate CloudFront in an easier way, but let's keep things simple.

Securing your CloudFront with AWS WAF

After some months, Larry sent you an email:

> Hello there, I noticed that my CloudFront bills are increasing every month. While I don't see any increase of visitors in my Google Analytics, AWS charges me for additional bandwidth.

After checking the CloudFront distribution "Top Referrers", you noticed that many other websites appear to be referrer in the list provided by CloudFront. This means that they are using the videos.

Anyone could just paste this code in a blog post and use Larry's website:

```
<video width="320" height="240" controls>
<source src="http://videos.larryswebsite.com/video.mp4" type="video/mp4">
</video>
```

Larry will pay for everyone using his content for free. This is when we can use AWS WAF.

Amazon defines AWS WAF as follows:

> AWS WAF is a web application firewall that helps protect your web applications from common web exploits that could affect application availability, compromise security, or consume excessive resources.

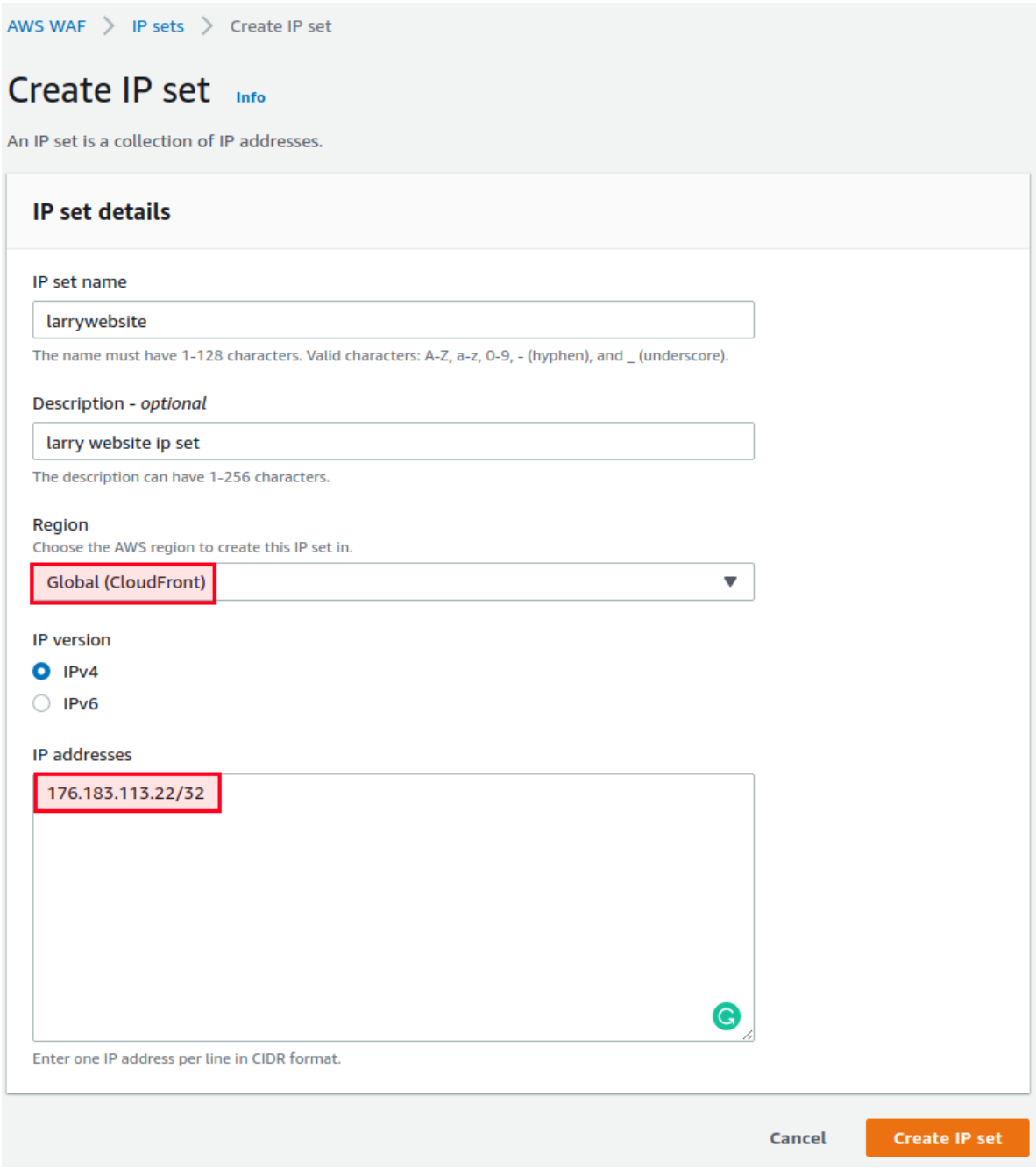
AWS WAF gives the user control over which traffic to allow or block to your web applications by defining customizable web security rules. We are going to use this feature in order to block all of the websites using Larry's videos, except - of course - larryswebsite.com.

You can also use AWS WAF to create custom rules that block common attack patterns, like :

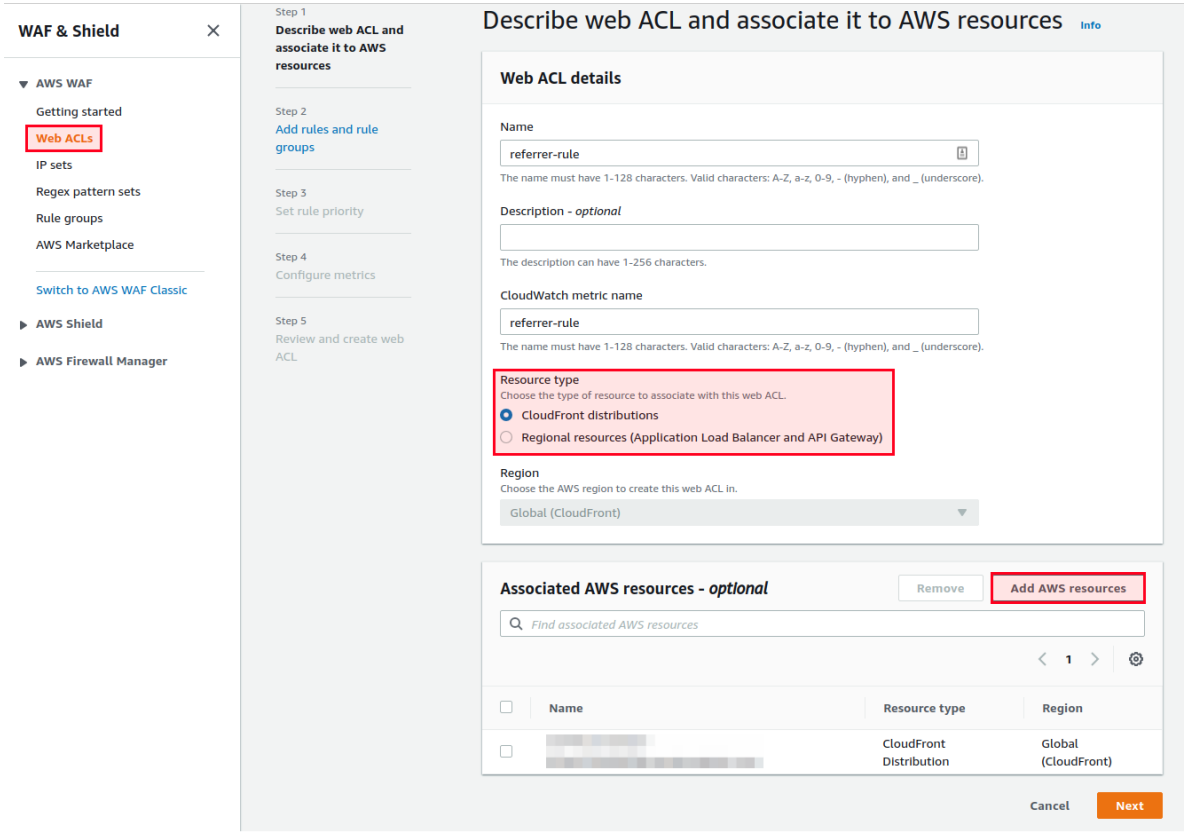
- SQL injection,
- Cross-site scripting

In order to forbid other websites from using Larry's videos, we are going to create a WAF Rule.

Before proceeding, let's create an "IP sets" that we are going to need later. Add the IP address or the IP address range of the dedicated website of Larry.



Now, choose "Web ACLs":



Make sure to choose the good resource type to apply WAF rules to. Add the CloudFront distribution, Click on Next.

Choose the "Rule builder", with the type "Regular rule". We also need to configure the rule to allow requests, only when the "Referrer" header filed, contains "larrywebsite.com/" after transforming the text to lowercase.

The request should also originate from the IP set we created previously. We should add it to the rule builder.

Rule type

Rule type

- IP set
  Use IP sets to identify a specific list of IP addresses.
- **Rule builder**
  Use a custom rule to inspect for patterns including query strings, headers, countries, and rate limit violations.
- Rule group
  Use a rule group to combine rules into a single logical set.

Rule builder

Rule visual editor | Rule JSON editor

You can use the JSON editor for complex statement nesting, for example to nest two OR statements inside an AND statement. The visual editor handles one level of nesting. For web ACLs and rule groups with complex nesting, the visual editor is disabled.

Rule

Validate

Name

LarryWebsiteRule

The name must have 1-128 characters. Valid characters: A-Z, a-z, 0-9, - (hyphen), and _ (underscore).

Type

Regular rule

If a request matches all the statements (AND)

Statement 1

Remove

Inspect

Header

Header field name

Referrer

Match type

Contains string

String to match

larrywebsite.com/

Text transformation

AWS WAF applies all transformations to the request before evaluating it. If multiple text transformations are added, then text transformations are applied in the order presented below with the top of the list being applied first.

Lowercase

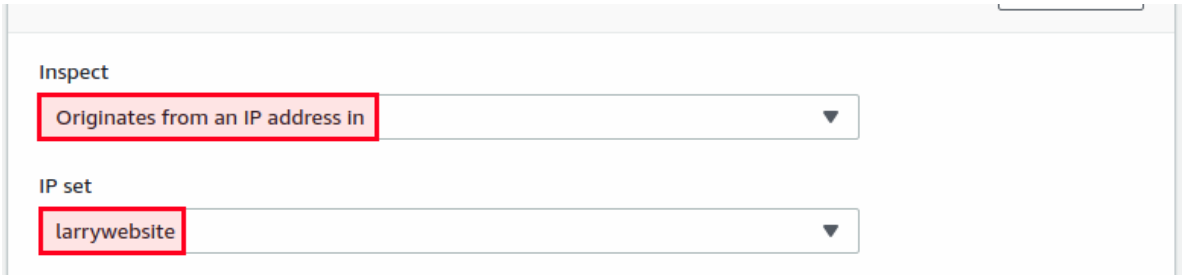
Add text transformation

At least one text transformation is required. You can add up to 3 text transformations.

AND

Statement 2

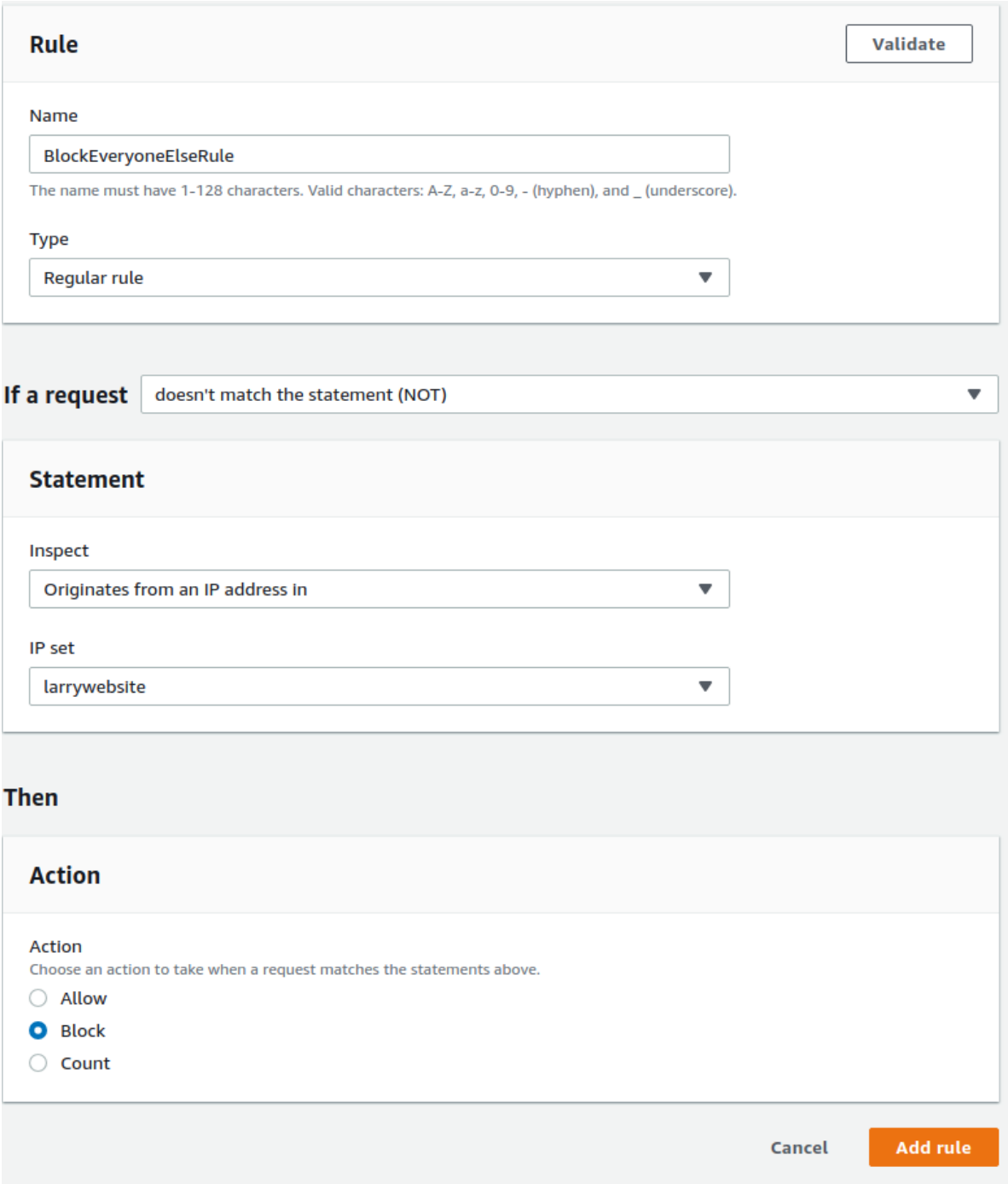
Remove



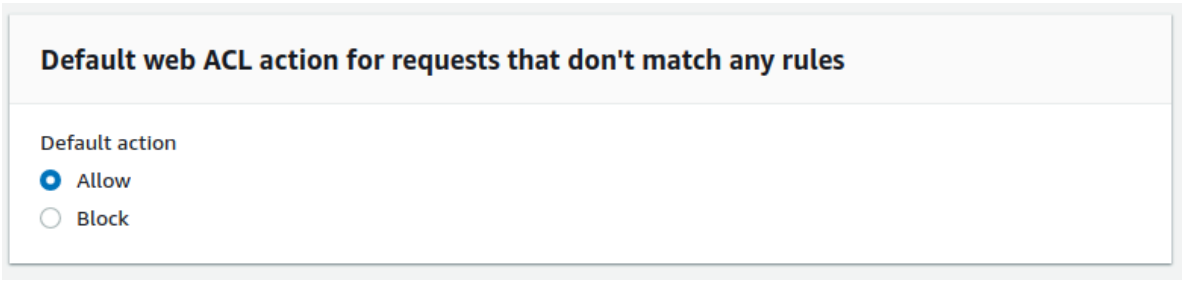
Note that you should configure the action to "Allows" and not block.

Finally, click on "Add Rule" and create the rule.

You can create a second rule to block every other server from using your CloudFront assets.



Or configure a default web ACL action for requests that don't match any rules.



Review and create your web ACL. At this step, you should wait for your CloudFront distribution to deploy the new configuration.

In order to test, let's use the curl command with a fake header and see the response:

```
» curl -H "Referer: https://notlarryswebsite.com" -I
https://static.larryswebsite.com/video.mp4
« HTTP/1.1 403 Forbidden
```

Testing with the right referer header should respond with "200 ok":

```
» curl -H "Referer: https://larryswebsite.com" -I
https://static.larryswebsite.com/video.mp4
« HTTP/1.1 200 OK
```

This ACL rule is based on the header referer; in other words, the website that calls a resource.

The CloudFront distribution we are using is `static.larryswebsite.com`, and it should be consumed and only consumed by the Wordpress website.

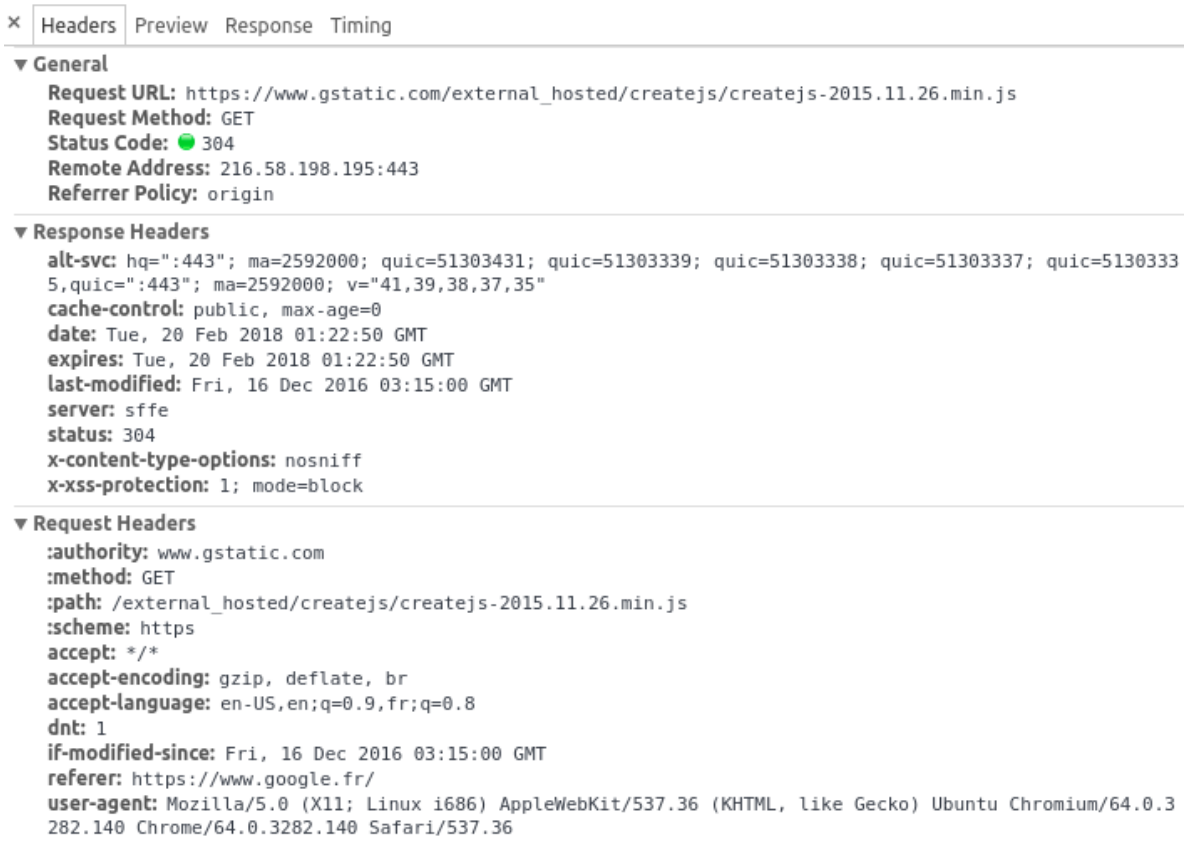
What we have seen until now is not only about security but also about reducing costs. There are many other techniques to reduce CloudFront costs. For example, we are going to discover how using Cache-Control could help us in having better control on CloudFront bills.

Reducing CloudFront Costs by Using Cache-Control

Larry is happy with your work, but he asked you after a few months if there is a way to reduce his CloudFront bills.

Using visitors' browsers, there is a way to reduce Larry's bills. Browser caching is an intelligent way of storing commonly-used data in quick-to-access locations in order to re-access this data as fast as possible.

"Cache-Control" header defines how a resource should be cached. By default, an object is cached in an edge for 24 hours. As an example, Google homepage uses "cache-control" to cache some Javascript files in your browser:



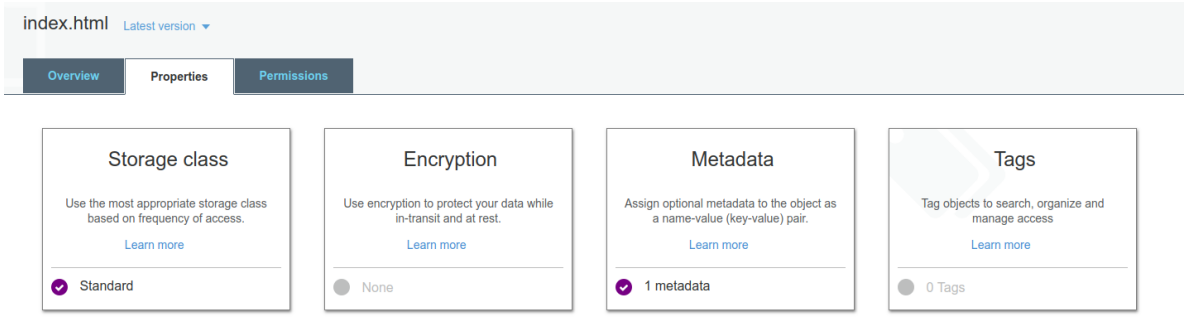
The cache-control max-age parameter in the previous screenshot is set to 0.

```
max-age=0
```

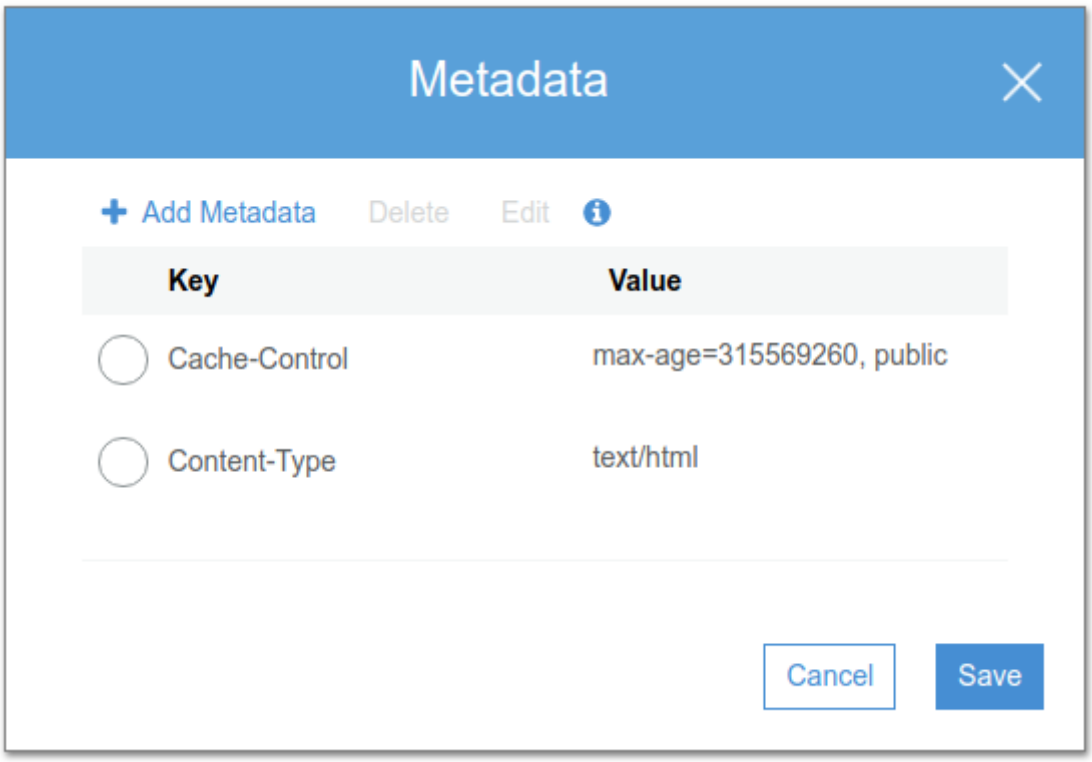
"max-age" determines the length of time in seconds that this resource should be cached. So, in reality, the Javascript file is not cached and will be loaded each time you hit F5 on Google's homepage.

If your videos' cache-control is also set to 0, a visitor who already watched the video will download it again from the CloudFront distribution if he wanted to watch it again and again.

In order to activate caching, go to your S3 bucket, click on the file you want to cache and choose the "Properties" tab, then click on "Metadata":



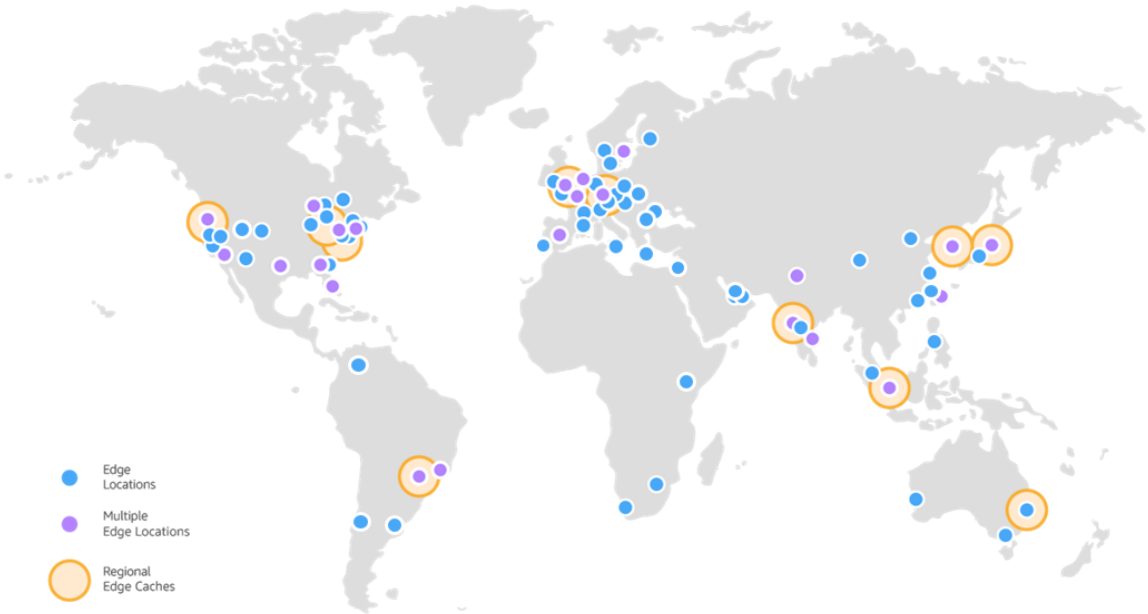
In this example, I set the cache-control max-age parameter to 10 years (in seconds). You should use the number of seconds suitable to your case.



Managing Edge Cache Expiration

When a user consumes a video hosted on CloudFront, it may be served directly from AWS, an edge cache.

The more requests your CloudFront distribution is able to serve from edge caches as a proportion of all requests (that is, the greater the cache hit ratio), the fewer viewer requests that CloudFront needs to forward to your origin to get the latest version or a unique version of an object.

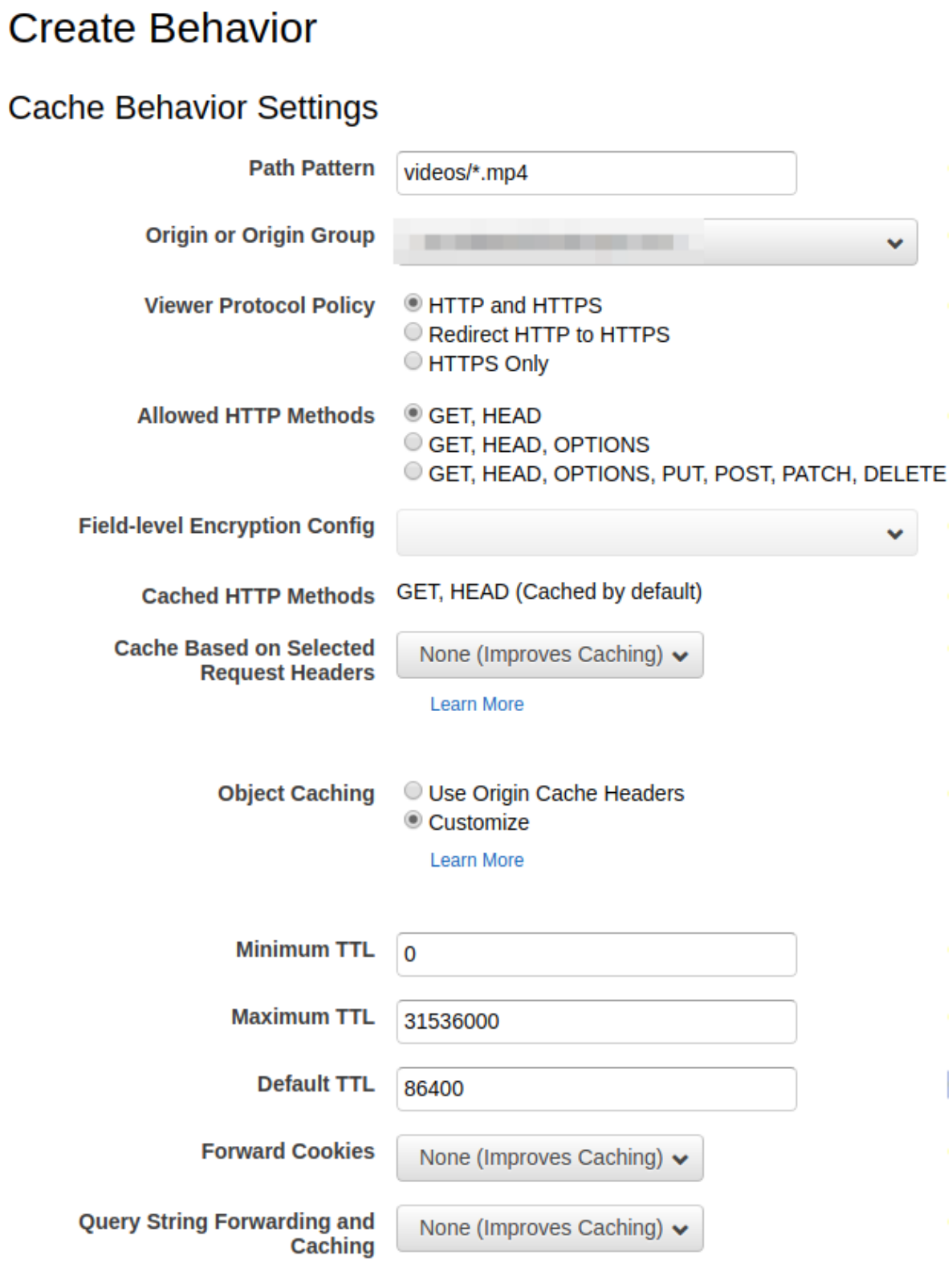


You can control how long your files stay in a CloudFront cache before CloudFront forwards another request to your origin. Serving your files from an edge cache increases the performance of your CDN.

By default, each file automatically expires after 24 hours, but you can change this.

To do that, you need to change the cache duration for all files that match the same path pattern; you can change the CloudFront settings for Minimum TTL, Maximum TTL, and Default TTL for a cache behavior.

Create a behavior, then in the "Path Pattern", specify which requests you want to route to the origin. For example, the path pattern "images/*.jpg" routes all requests for ".jpg" files in the images directory and in all subdirectories below the images directory to the Amazon S3 bucket or the web server that you specify in Origin.



Minimum TTL is the minimum amount of time, in seconds, that you want objects to stay in CloudFront caches before CloudFront forwards another request to your origin to determine whether the object has been updated. Minimum TTL interacts with HTTP headers such as Cache-Control max-age, Cache-Control s-maxage, and Expires and with Default TTL and Maximum TTL.

Maximum TTL is the amount of time, in seconds, that you want objects to stay in CloudFront caches before CloudFront forwards another request to your origin to determine whether the object has been updated. The value that you specify applies only when your origin adds HTTP headers such as Cache-Control max-age, Cache-Control s-maxage, and Expires to objects.

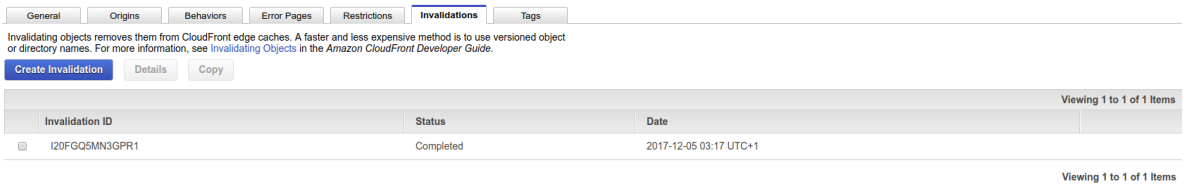
Default TTL is the default amount of time, in seconds, that you want objects to stay in CloudFront caches before CloudFront forwards another request to your origin to determine whether the object has been updated. The value that you specify applies only when your origin does not add HTTP headers such as Cache-Control max-age, Cache-Control s-maxage, and Expires to objects.

This is how CloudFront responds to cached versions of the same file:

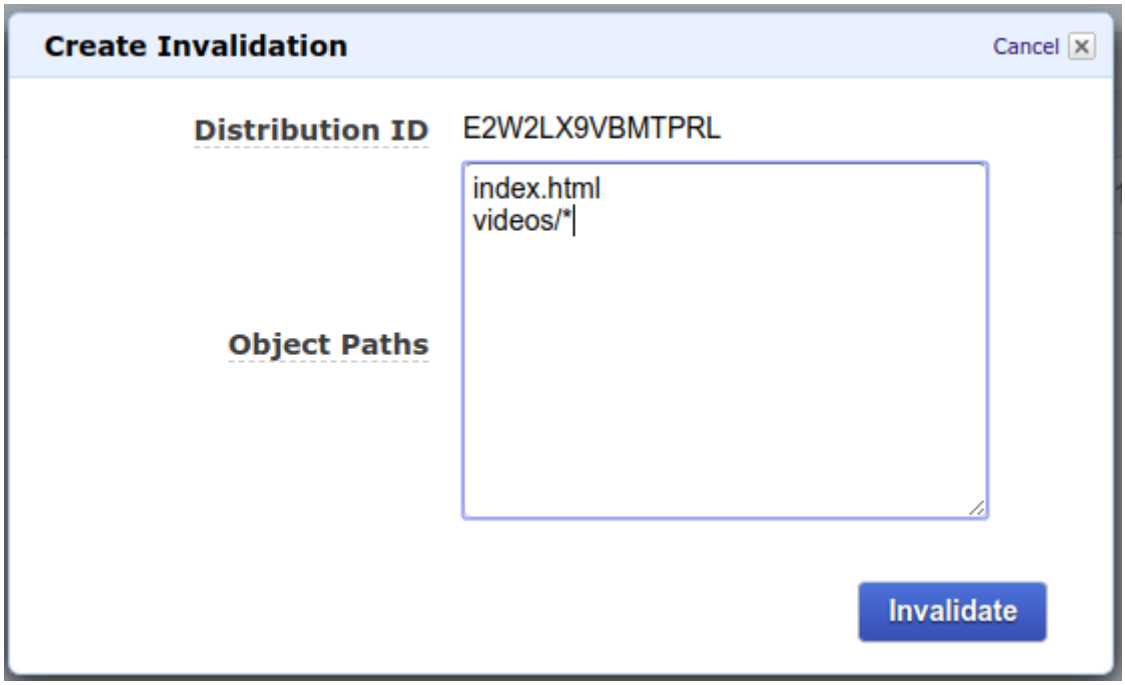
- The origin returns a 304 status code (Not Modified) if the CloudFront cache already has the latest version.
- The origin returns a 200 status code (OK) and the latest version of the file if the CloudFront cache does not have the latest version.

Invalidating a File in CloudFront

Invalidating objects means removing them from CloudFront edge caches. In order to invalidate a file, you should go to your CloudFront dashboard, click on the distribution to which your file is deployed.



Click on "Create Invalidation" and add the files you want to invalidate.



You can use a wildcard at the end of the path. If you use it in the middle of the string, CloudFront treats it as a character.

You can invalidate the cache using the CLI too:

```
aws cloudfront create-invalidation --distribution-id <distribution_id> --paths
"<paths>"
```

Example:

```
aws cloudfront  create-invalidation --distribution-id ENE8100A9Z6XW --paths
"/*"
```

Note: No additional charge for the first 1,000 paths requested for invalidation each month. Thereafter, $0.005 per path requested for invalidation.

Lesson 17 - Serverless Computing - AWS Lambda (Video)

Note: Download the videos archive here.

Additional Notes

Lambda Limits

AWS Lambda has limits, and in this section, we will list all of them (These limits may be subject to change in the future by AWS)

Concurrent executions:

1,000

Function and layer storage:

75 GB

Elastic network interfaces per VPC:

250

Function memory allocation:

128 MB to 3,008 MB, in 64 MB increments.

Function timeout:

900 seconds (15 minutes)

Function environment variables:

4 KB

Function resource-based policy:

20 KB

Function layers:

5 layers

Function burst concurrency:

500 - 3000 (varies per region)

Invocation frequency (requests per second):

- 10 x concurrent executions limit (synchronous – all sources)
- 10 x concurrent executions limit (asynchronous – non-AWS sources)

- Unlimited (asynchronous – AWS service sources)

Invocation payload (request and response)

- 6 MB (synchronous)
- 256 KB (asynchronous)

Deployment package size:

- 50 MB (zipped, for direct upload)
- 250 MB (unzipped, including layers)
- 3 MB (console editor)

Test events (console editor):

10

/tmp directory storage:

512 MB

File descriptors:

1,024

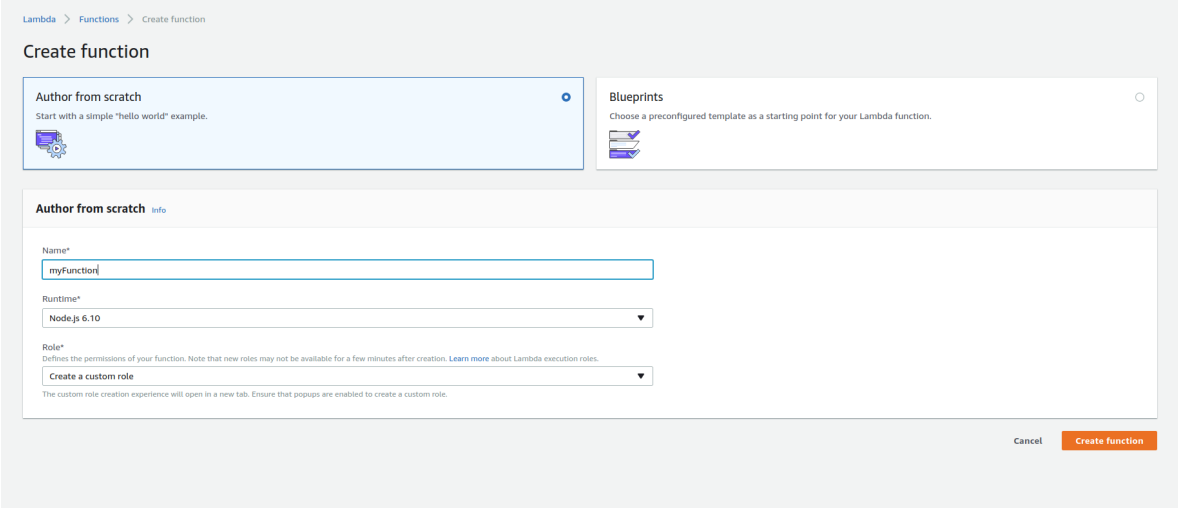
Execution processes/threads:

1,024

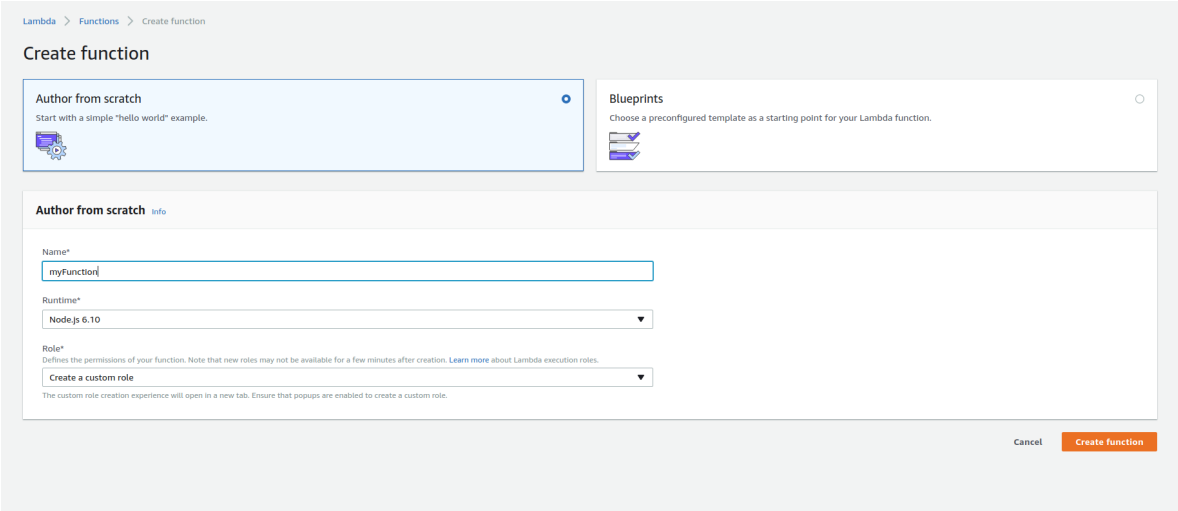
Lesson 18 - AWS Lambda - Hello World

Creating Our First AWS Lambda Function

First, log in to your AWS console and open the Lambda console. Click on "Create Function", give it a name and choose "Create a custom role".



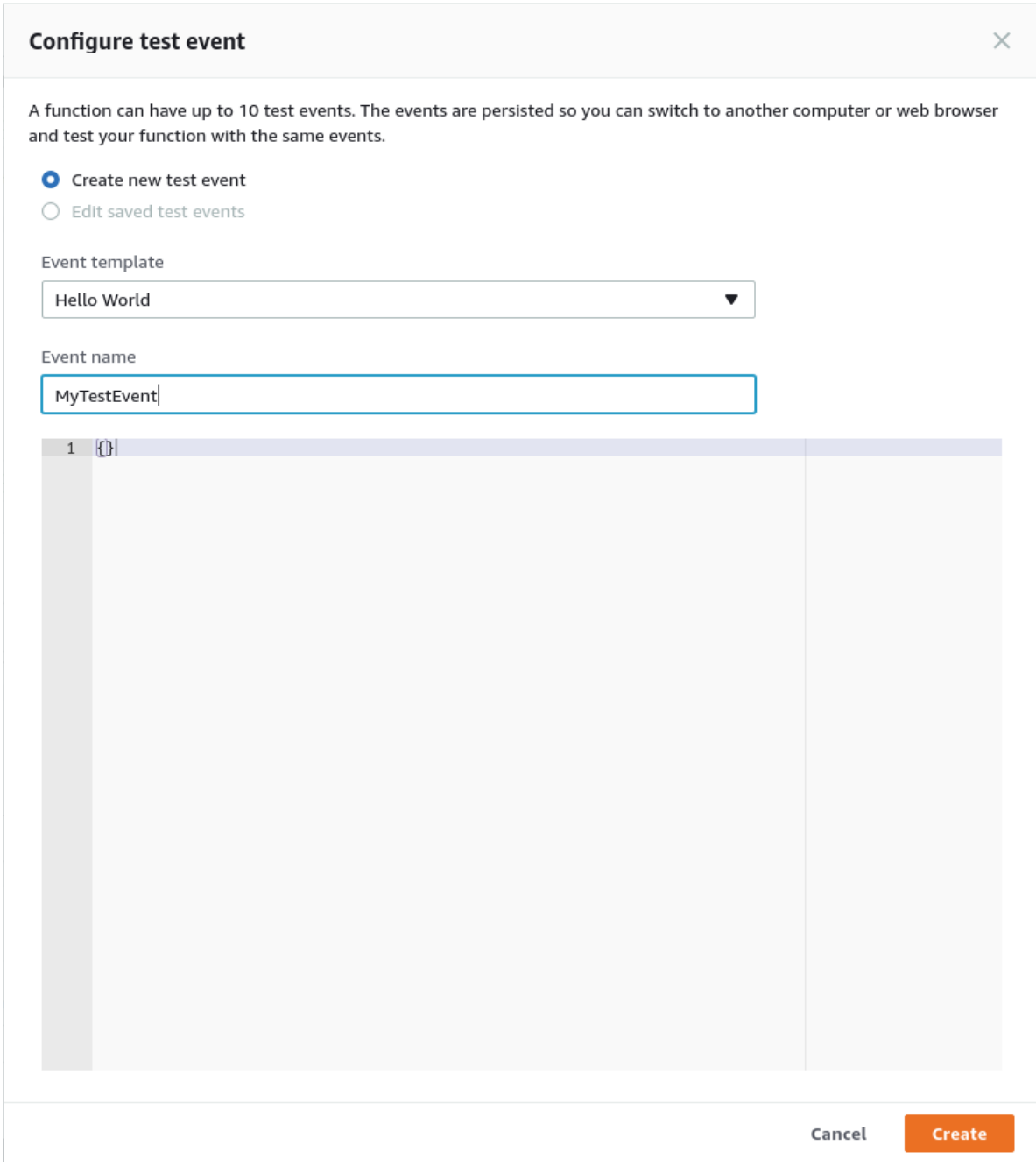
Keep the role with the default policy document since our "Hello World" function will not need any special permission.



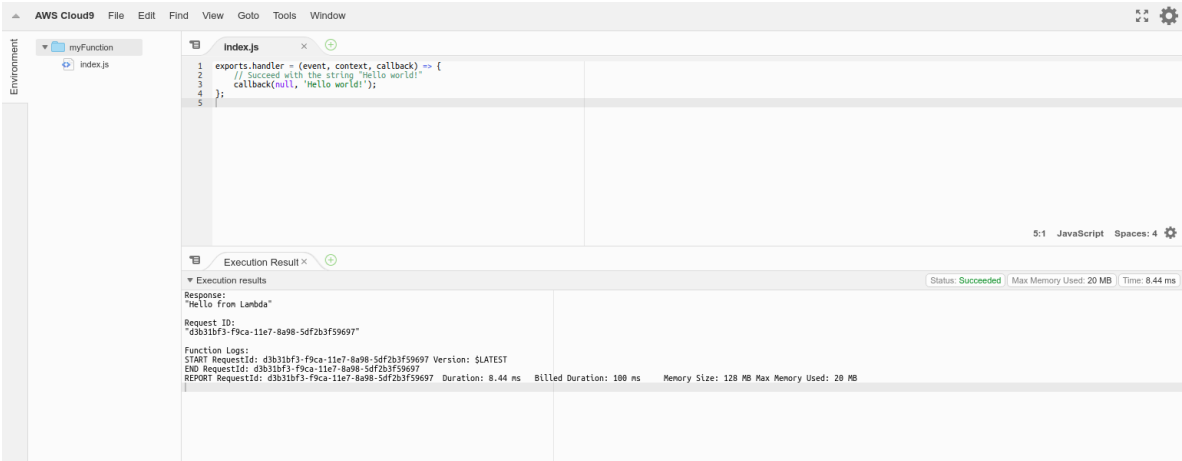
Click on "Allow" and make sure to choose "lambda_basic_execution" as a role. Now let's add this Node.js "Hello World" function:

```
exports.handler = (event, context, callback) => {
    // Succeed with the string "Hello world!"
    callback(null, 'Hello world!');
};
```

Click on "Test":

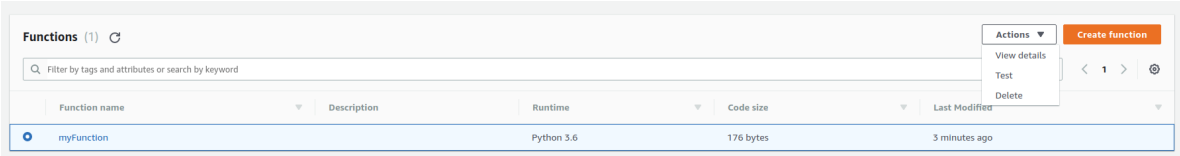


Our test event, once executed, will call the Lambda "Hello World" function. Click on "Create" followed by "Test", and you will notice the execution results:

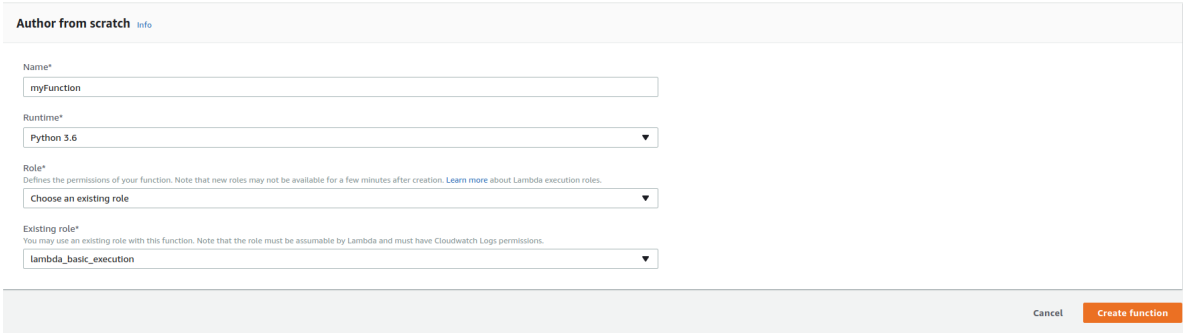


This is a basic function.

Using Lambda, you can use another language like Python. In order to do this, let's first remove the first function, so go back to your dashboard and click on "Action" then "Delete":



Now, create a new function and choose Python 3.6 or Python 3.8 as a runtime.



The Python Hello World function is the following:

```
def lambda_handler(event, context):
    return 'Hello from Lambda'
```

This function will return the string "Hello from Lambda" and exits.

Note: For Python, the function handler should be configured correctly in this form `file_name.function_name` . The file name should not also contain the extension of the file.

In my example, my file is called `lambda_function.py` and my function is defined as `lambda_handler`. The handler will be:

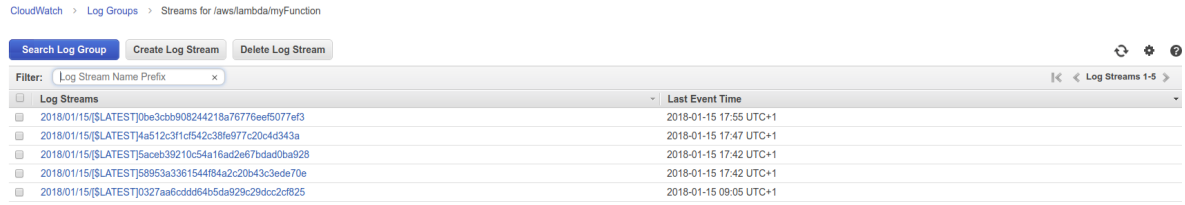
```
lambda_function.lambda_handler
```

AWS Lambda & CloudWatch

Amazon CloudWatch is the cloud monitoring service for AWS resources. It collects, tracks, and reacts to metrics as well as logs.

Our function logs could be viewed directly from the AWS Lambda dashboard.

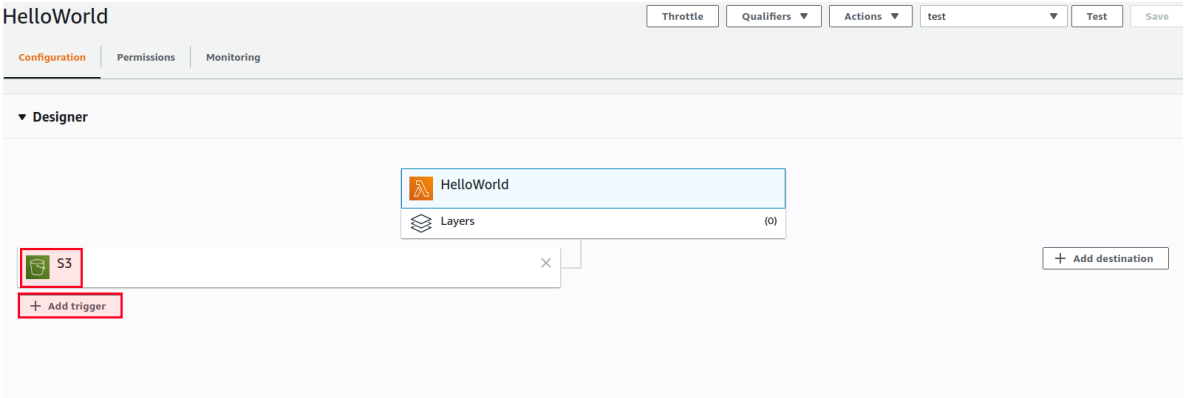
By default, this service collects a Lambda function logs automatically and stores it in a folder that you can find in the dashboard of AWS CloudWatch.



Triggering The Execution Of A Lambda Function

Let's now see how we can trigger our function using one of the AWS services. For this example, I'll use S3 as a trigger, and I'd need to execute my Hello World function when a file is uploaded to a bucket. In the function, we are going to read the content of the S3 file (we will upload text file).

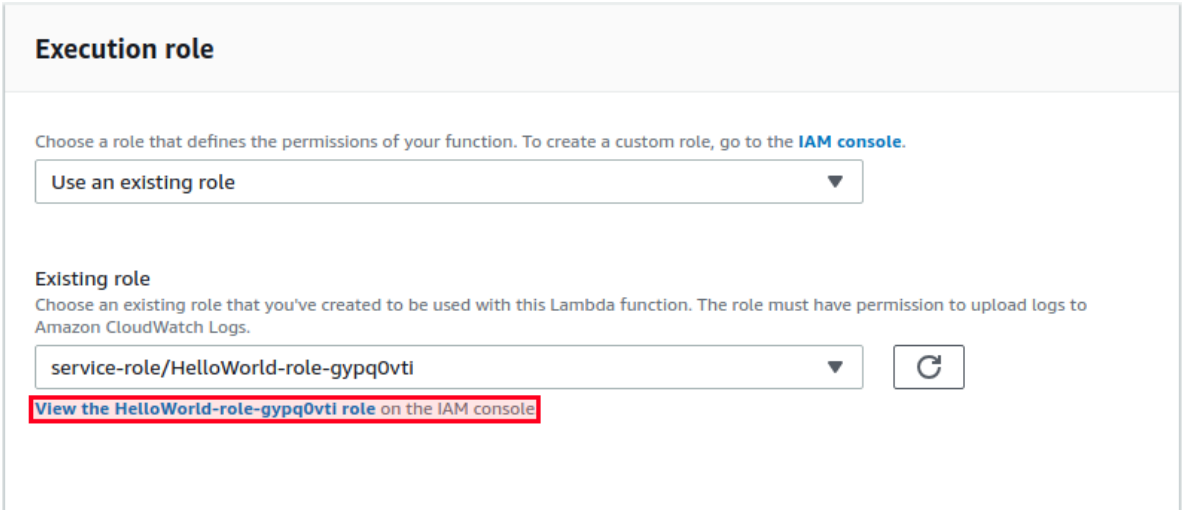
Click on S3 from "Designer" and add S3 as a trigger.



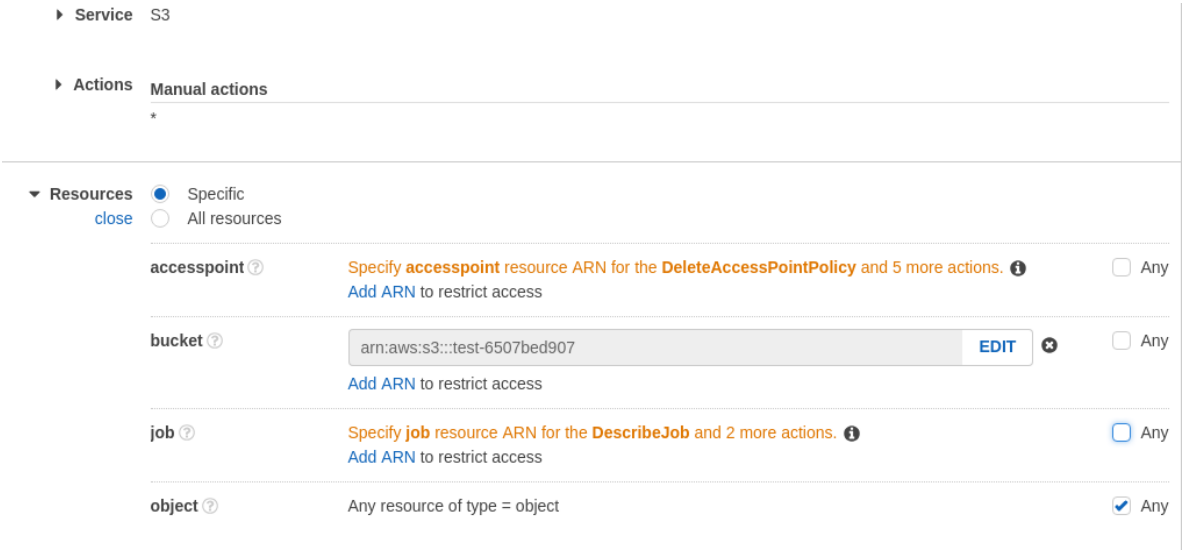
Let's now create a bucket:

```
aws s3 mb s3://test-6507bed907 --region eu-west-3


---
make_bucket: test-6507bed907
```

We also need to add some new rights to the Lambda function. Click on the execution role:



Then choose "Edit Policy" then click on "Add additional permissions". Add the permissions you need to **list**, **read**, and **write** from/in the bucket, then add the bucket ARN.



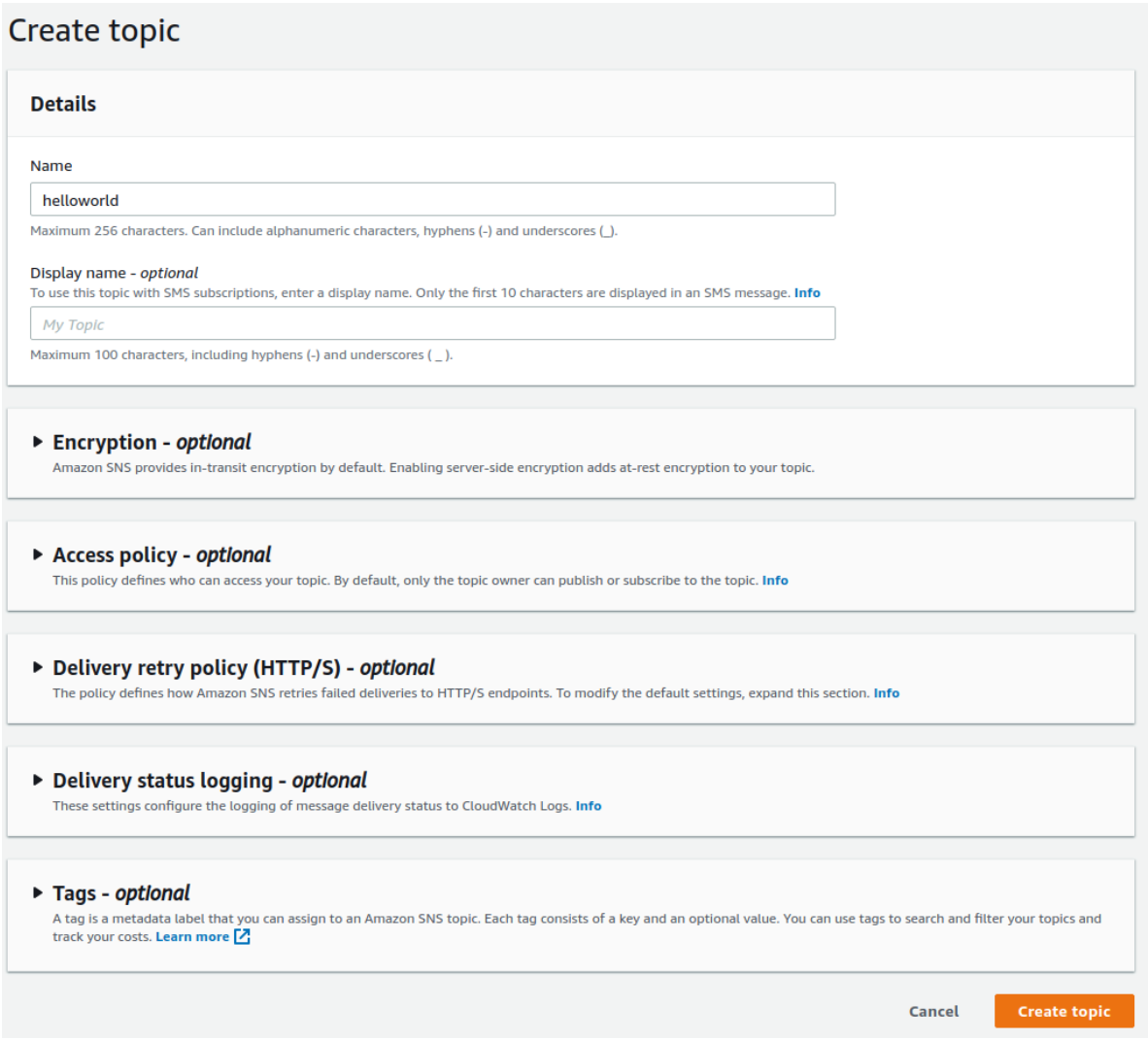
Click on "Any" on "object" to guarantee access to all objects in the bucket.

Now create a file and upload it to the S3 bucket:

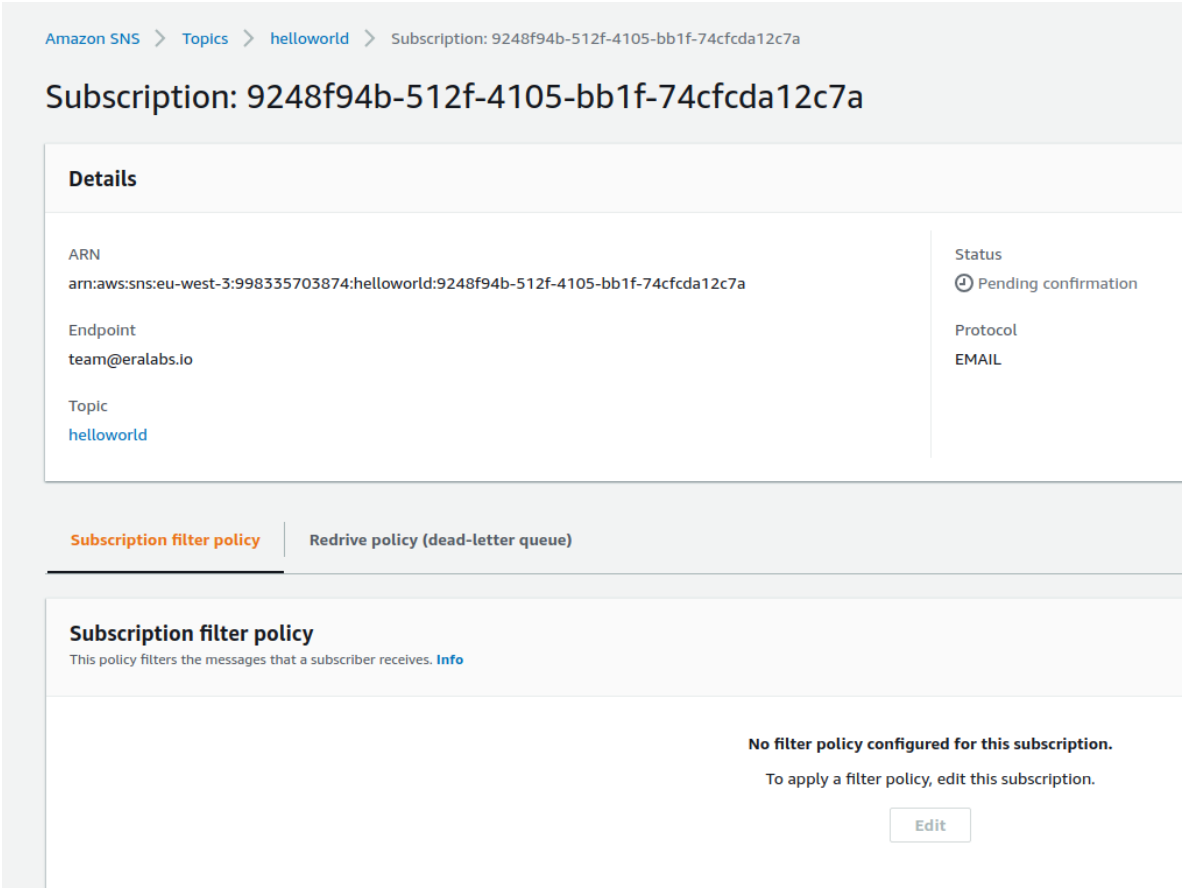
```
echo "This is a test content" > my_file.txt
aws s3 cp my_file.txt s3://test-6507bed907/
```

The log should now be visible in CloudWatch. Visit CloudWatch dashboard and search for the function name, then choose the right Log Group and you will be able to tail log and see that the function prints the content of the file to the log file:

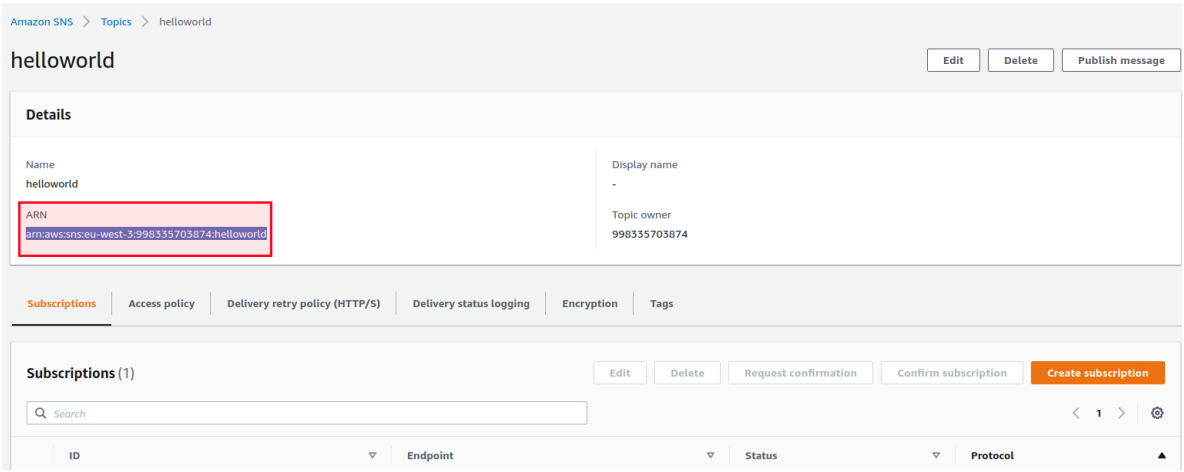
Create an SNS topic:



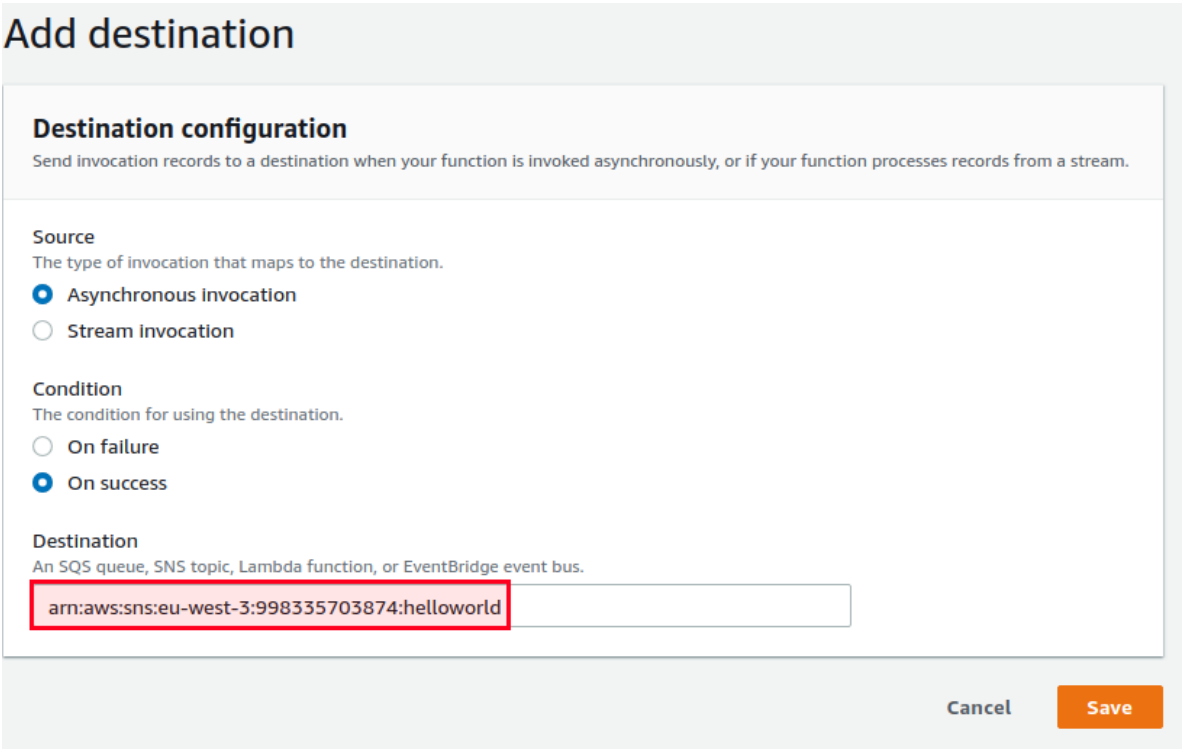
Configure this SNS with a new Topic and add "Email" as "Protocol", then your email in the email field.



Go back to the Topic list, copy the ARN of this SNS ressource.



Use the ARN in the Lambda dashboard to configure a target.



Now to test if you receive an email on success, you should re-upload a file and check your email. You should receive something very similar to:

```
{
   "version":"1.0",
   "requestContext":{
      "requestId":"xxxxx",
      "functionArn":"arn:aws:lambda:eu-west-
3:xxxxx:function:HelloWorld:$LATEST",
      "condition":"Success",
      "approximateInvokeCount":1
   },
   "requestPayload":{
      "Records":[
         {
             [...]
            "awsRegion":"eu-west-3",
            "eventName":"ObjectCreated:Put",
            "userIdentity":{
```

```
               "principalId":"AWS:AIDAJCLK5KQQMZKMCCQ7G"
            },
            "requestParameters":{
               "sourceIPAddress":"176.185.143.227"
            },
            "responseElements":{
               "x-amz-request-id":"xxxxx",
               "x-amz-id-2":"xxxxx"
            },
            "s3":{
               "s3SchemaVersion":"1.0",
               "configurationId":"xxxxx",
               "bucket":{
                  "name":"test-6507bed907",
                  "ownerIdentity":{
                     "principalId":"xxxxx"
                  },
                  "arn":"arn:aws:s3:::test-6507bed907"
               },
               "object":{
                  "key":"my_file.txt",
                  "size":23,
                  "eTag":"xxxxx",
                  "sequencer":"xxxxx"
               }
            }
         }
      ]
   },
   "responseContext":{
      "statusCode":200,
      "executedVersion":"$LATEST"
   },
   "responsePayload":null
}
```


It is also possible to use DynamoDB or AWS Kinesis to stream data to the SNS topic, or use an SQS queue as a target.

AWS Lambda & Environment Variables

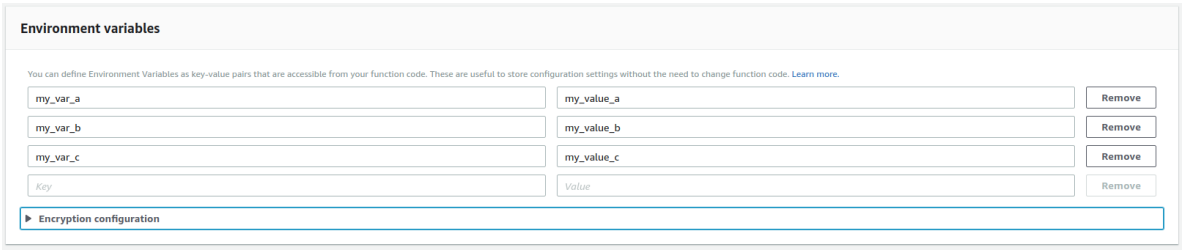
When developing an application or an AWS Lambda function, one of the best practices in keeping your environment variables outside of your code. An application configuration may vary between deploys:

- Staging
- Production
- Testing
- Developer's machine
- ..etc

e.g.: You database credentials may vary from one environment to another

To avoid the problems that these configurations may cause, You'll need to store your configurations in the environment.

In order to test this feature, add some environment variables:



Go back to the code editor and add this function:

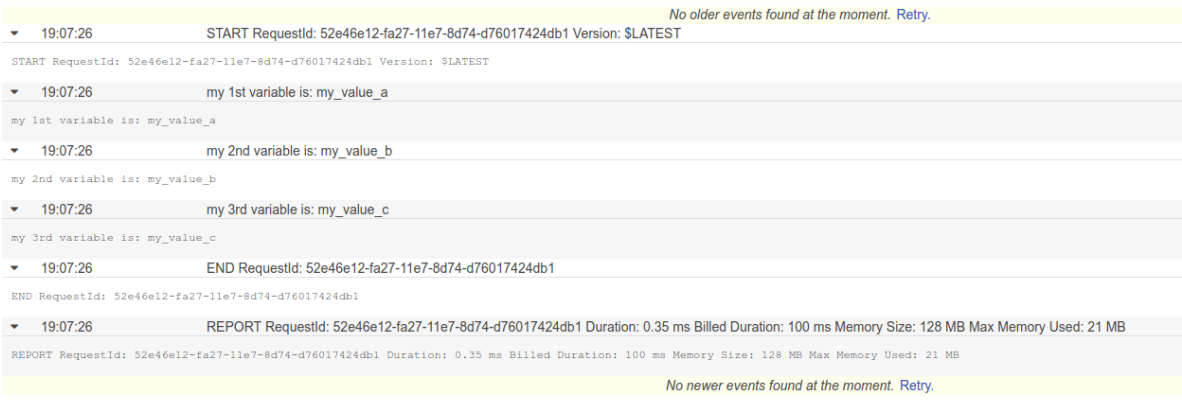
```
import os

def lambda_handler(event, context):
    print ( "my 1st variable is: " + os.environ["my_var_a"])
    print ( "my 2nd variable is: " + os.environ["my_var_b"])
    print ( "my 3rd variable is: " + os.environ["my_var_c"])
    return ""
```

As you can see in the code above, I used my code to get the environment variables:

```
    print ( "my 1st variable is: " + os.environ["my_var_a"])
    print ( "my 2nd variable is: " + os.environ["my_var_b"])
    print ( "my 3rd variable is: " + os.environ["my_var_c"])
```

The execution results will be similar to this one:

```
Response:
""

Request ID:
"52e46e12-fa27-11e7-8d74-d76017424db1"

Function Logs:
START RequestId: 52e46e12-fa27-11e7-8d74-d76017424db1 Version: $LATEST
my 1st variable is: my_value_a
my 2nd variable is: my_value_b
my 3rd variable is: my_value_c
END RequestId: 52e46e12-fa27-11e7-8d74-d76017424db1
REPORT RequestId: 52e46e12-fa27-11e7-8d74-d76017424db1    Duration: 0.35 ms
Billed Duration: 100 ms     Memory Size: 128 MB    Max Memory Used: 21 MB
```



AWS Lambda & VPC

By default, a Lambda function runs inside the default VPC. It's optional to configure the function to run inside a custom VPC, and the goal is accessing the same VPC resources, e.g., RDS database instances, Amazon ElastiCache clusters ..etc

Note: If your function requires external access to the Internet, you must ensure that the security group allows outbound connections and that your VPC has a NAT gateway.

Ensure your role has appropriate permissions to configure VPC.

To enable our Lambda function to access resources inside our private VPC, we must provide additional VPC-specific configuration information that includes VPC subnet IDs and security group IDs.

Lambda uses this information to set up ENIs (elastic network interfaces) that enable our function to connect securely to other resources within this VPC. The next paragraph explains how to create a role.

AWS Lambda and Credentials

Sometimes, you need to call other AWS services using your Lambda code, so you will use the AWS SDK; whether you are using Python, Node.js, or another programming language, you will need to instantiate a new AWS connection to a given service.

Example, with Boto3 (Python):

```
import boto3
client = boto3.client(
    's3',
    aws_access_key_id=ACCESS_KEY,
    aws_secret_access_key=SECRET_KEY,
    aws_session_token=SESSION_TOKEN,
)

# Or via the Session
session = boto3.Session(
    aws_access_key_id=ACCESS_KEY,
    aws_secret_access_key=SECRET_KEY,
    aws_session_token=SESSION_TOKEN,
)
```

If this code is used in a Lambda function (or any other AWS service), we do not need to add the credentials.

```
import boto3
client = boto3.client('s3')

# Or via the Session
session = boto3.Session()
```

AWS services do not need your credentials; the only thing you need to take care of is the different rights that your AWS Lambda function has (Execution Roles).

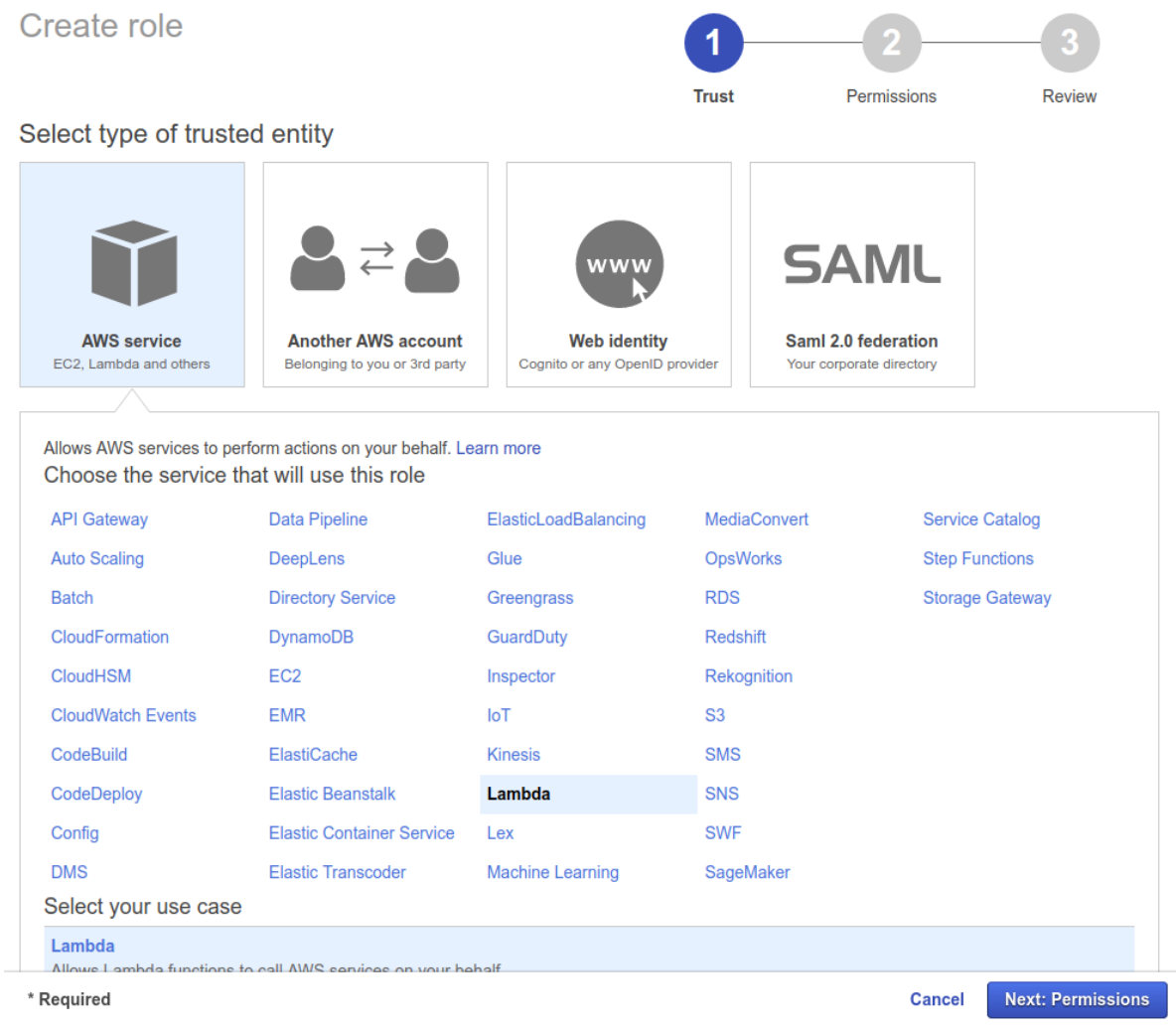
AWS Lambda & IAM Roles (Execution Roles)

To goal of creating a role is having a separate profile to which we can add 1 or more permissions. In this example, we are going to create a role and add the permission `AWSLambdaVPCAccessExecutionRole` that will allow a Lambda function to perform EC2 actions in order to create ENIs and access VPC resources as well as CloudWatch Logs actions to write logs.

It's described by AWS in the following Policy Document:

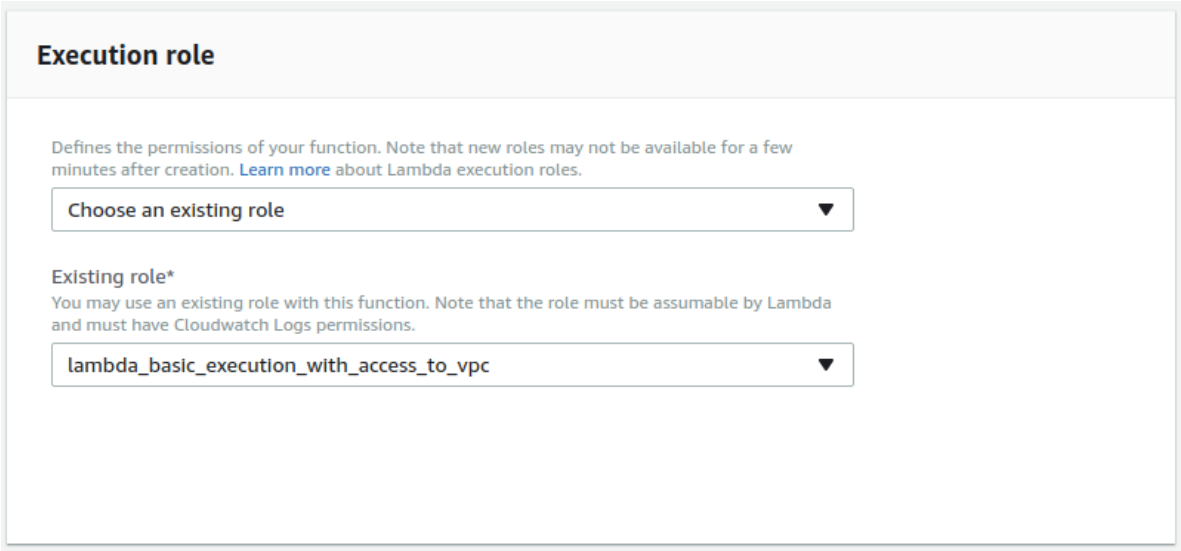
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface"
            ],
            "Resource": "*"
        }
    ]
}
```

To create an IAM role (execution role), sign in to the AWS Management Console and open the IAM console.



Click "Next", search for "AWSLambdaVPCAccessExecutionRole", activate it then click on "Next: Review". Give it a name like "lambda_basic_execution_with_access_to_vpc", a description and create the role.

Now you can go back to your function dashboard and select the new role.



AWS Lambda & The Code Entry Type

To add our function code to AWS Lambda, we have 3 possibilities:

- Using the cloud IDE Cloud9
- using a ZIP file
- Using S3

Using AWS Cloud IDE - Cloud9

This is what we have been doing in the first steps of this lesson.

Using a .ZIP Archive

We created our first AWS Lambda function online using AWS Cloud9, but it's possible to upload the code using a ZIP file.

When you code using your machine, creating a deployment package is useful. If you had included third party dependencies or any static or media file (photos, videos ..etc.), make sure to include them in your.ZIP archive.

e.g.: If you are using Node.js and you are going to need the node module "x11", you need to install first:

```
npm install x11
```

After installing this module, you will have these files in your machine:

```
node_modules/x11/lib/x11.js
node_modules/x11/package.json
```

These files should be included in the .ZIP package.

Let's take the example of a Python application that uses the library "requests".

Create a folder:

```
mkdir hello_world
```

Then a virtual environment:

```
virtualenv -p python3 hello_world
```

and activate it:

```
. bin/activate
```

We will create a function that returns the status of GET requests on our website `s3static.practicalaws.com`.

```
import requests

def lambda_handler(event, context):
    return requests.get('https://s3static.practicalaws.com').status_code
```

We should then install "requests":

```
pip install requests
```

The dependencies that we want to include in our archive file are located here:

```
$VIRTUAL_ENV/lib/python3.6/site-packages
```

Note that you should add the content of this directory and not the directory itself.

This is the structure of our application folder:

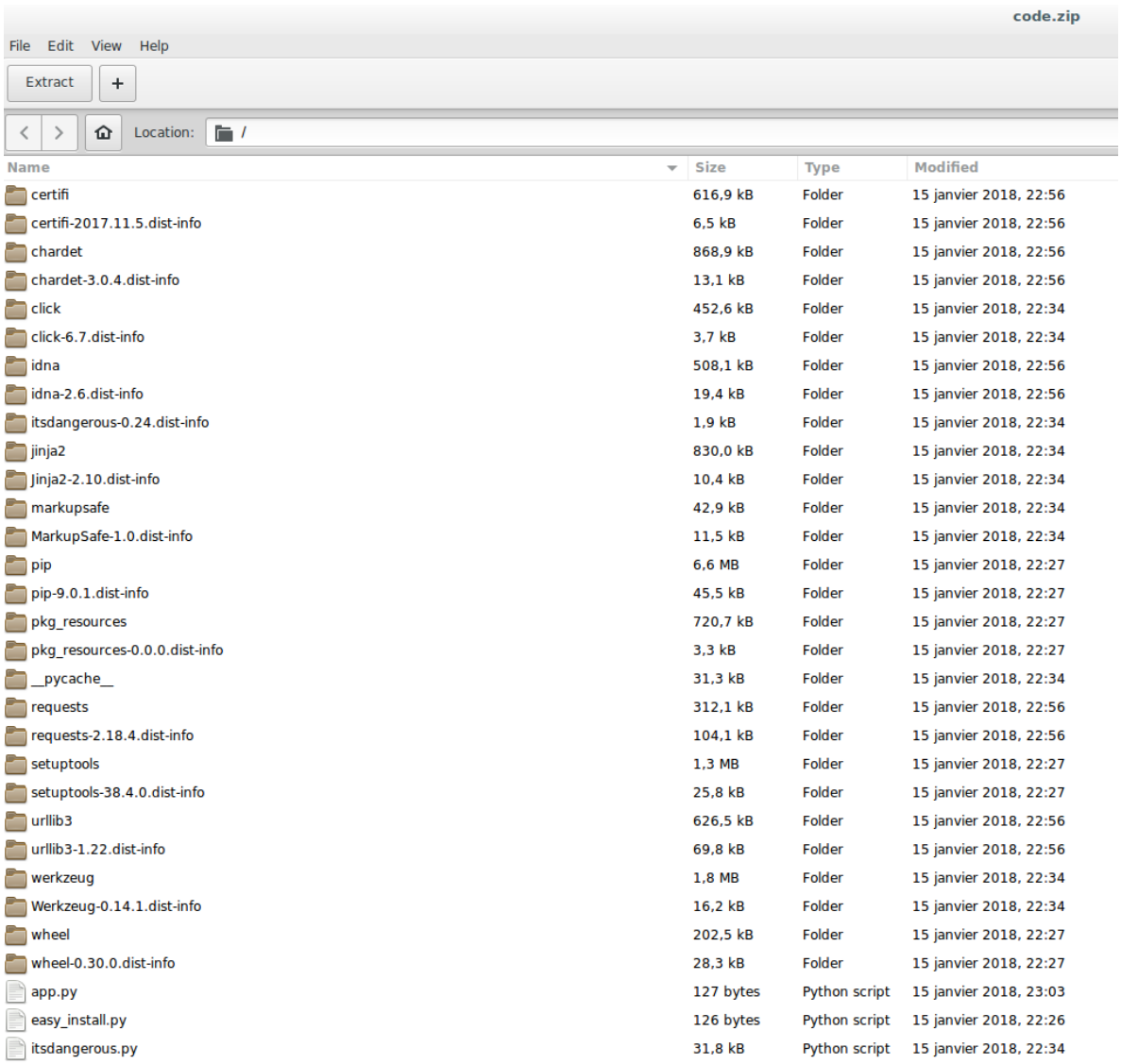
```
├── **app.py**
├── bin
│   ├── activate
│   ├── activate.csh
│   ├── activate.fish
│   ├── activate_this.py
│   ├── chardetect
│   ├── easy_install
│   ├── easy_install-3.5
│   ├── pip
│   ├── pip3
│   ├── pip3.5
│   ├── python -> python3
│   ├── python3
│   ├── python3.5 -> python3
│   ├── python-config
│   └── wheel
├── include
│   └── python3.5m -> /usr/include/python3.5m
├── **lib**
│   └── **python3.5**
```

```
│       ├── abc.py -> /usr/lib/python3.5/abc.py
│       ├── base64.py -> /usr/lib/python3.5/base64.py
│       ├── bisect.py -> /usr/lib/python3.5/bisect.py
│       ├── _bootlocale.py -> /usr/lib/python3.5/_bootlocale.py
│       ├── codecs.py -> /usr/lib/python3.5/codecs.py
│       ├── collections -> /usr/lib/python3.5/collections
│       ├── _collections_abc.py -> /usr/lib/python3.5/_collections_abc.py
│       ├── config-3.5m-i386-linux-gnu -> /usr/lib/python3.5/config-3.5m-i386-
linux-gnu
│       ├── copy.py -> /usr/lib/python3.5/copy.py
│       ├── copyreg.py -> /usr/lib/python3.5/copyreg.py
│       ├── distutils
│       ├── _dummy_thread.py -> /usr/lib/python3.5/_dummy_thread.py
│       ├── encodings -> /usr/lib/python3.5/encodings
│       ├── fnmatch.py -> /usr/lib/python3.5/fnmatch.py
│       ├── functools.py -> /usr/lib/python3.5/functools.py
│       ├── __future__.py -> /usr/lib/python3.5/__future__.py
│       ├── genericpath.py -> /usr/lib/python3.5/genericpath.py
│       ├── hashlib.py -> /usr/lib/python3.5/hashlib.py
│       ├── heapq.py -> /usr/lib/python3.5/heapq.py
│       ├── hmac.py -> /usr/lib/python3.5/hmac.py
│       ├── importlib -> /usr/lib/python3.5/importlib
│       ├── imp.py -> /usr/lib/python3.5/imp.py
│       ├── io.py -> /usr/lib/python3.5/io.py
│       ├── keyword.py -> /usr/lib/python3.5/keyword.py
│       ├── lib-dynload -> /usr/lib/python3.5/lib-dynload
│       ├── linecache.py -> /usr/lib/python3.5/linecache.py
│       ├── locale.py -> /usr/lib/python3.5/locale.py
│       ├── no-global-site-packages.txt
│       ├── ntpath.py -> /usr/lib/python3.5/ntpath.py
│       ├── operator.py -> /usr/lib/python3.5/operator.py
│       ├── orig-prefix.txt
│       ├── os.py -> /usr/lib/python3.5/os.py
│       ├── plat-i386-linux-gnu -> /usr/lib/python3.5/plat-i386-linux-gnu
│       ├── posixpath.py -> /usr/lib/python3.5/posixpath.py
│       ├── __pycache__
│       ├── random.py -> /usr/lib/python3.5/random.py
│       ├── reprlib.py -> /usr/lib/python3.5/reprlib.py
│       ├── re.py -> /usr/lib/python3.5/re.py
│       ├── rlcompleter.py -> /usr/lib/python3.5/rlcompleter.py
│       ├── shutil.py -> /usr/lib/python3.5/shutil.py
│       ├── **site-packages**
│       ├── site.py
│       ├── sre_compile.py -> /usr/lib/python3.5/sre_compile.py
│       ├── sre_constants.py -> /usr/lib/python3.5/sre_constants.py
│       ├── sre_parse.py -> /usr/lib/python3.5/sre_parse.py
│       ├── stat.py -> /usr/lib/python3.5/stat.py
│       ├── struct.py -> /usr/lib/python3.5/struct.py
│       ├── tarfile.py -> /usr/lib/python3.5/tarfile.py
│       ├── tempfile.py -> /usr/lib/python3.5/tempfile.py
│       ├── tokenize.py -> /usr/lib/python3.5/tokenize.py
│       ├── token.py -> /usr/lib/python3.5/token.py
│       ├── types.py -> /usr/lib/python3.5/types.py
│       ├── warnings.py -> /usr/lib/python3.5/warnings.py
│       ├── weakref.py -> /usr/lib/python3.5/weakref.py
│       └── _weakrefset.py -> /usr/lib/python3.5/_weakrefset.py
├── pip-selfcheck.json
└── share
```

```
    └── python-wheels
        ├── CacheControl-0.11.5-py2.py3-none-any.whl
        ├── chardet-2.3.0-py2.py3-none-any.whl
        ├── colorama-0.3.7-py2.py3-none-any.whl
        ├── distlib-0.2.2-py2.py3-none-any.whl
        ├── html5lib-0.999-py2.py3-none-any.whl
        ├── ipaddress-0.0.0-py2.py3-none-any.whl
        ├── lockfile-0.12.2-py2.py3-none-any.whl
        ├── packaging-16.6-py2.py3-none-any.whl
        ├── pip-8.1.1-py2.py3-none-any.whl
        ├── pkg_resources-0.0.0-py2.py3-none-any.whl
        ├── progress-1.2-py2.py3-none-any.whl
        ├── pyparsing-2.0.3-py2.py3-none-any.whl
        ├── requests-2.9.1-py2.py3-none-any.whl
        ├── retrying-1.3.3-py2.py3-none-any.whl
        ├── setuptools-20.7.0-py2.py3-none-any.whl
        ├── six-1.10.0-py2.py3-none-any.whl
        ├── urllib3-1.13.1-py2.py3-none-any.whl
        └── wheel-0.29.0-py2.py3-none-any.whl
```

In the .ZIP file, we only need the app.py file as well as the content of the folder `lib/python3.5/site-packages/`.

This is the content of my archive:



Note: The file `app.py` as well as the content of the folder `lib/python3.5/site-packages/` are included and have the same file-hierarchy level.

Now, on your Lambda dashboard, change "Code entry type" and set it to "Upload a .ZIP file".

Using S3

If you already have the .ZIP file, you can simply use S3 by uploading the same package to an S3 bucket and configure your AWS Lambda to use S3 as a deployment type.

```
aws s3 mb s3://practical-aws-lambda-code
aws s3 cp code.zip s3://practical-aws-lambda-code --region eu-west-1
```

Don't forget to change `eu-west-1` by your region.

Open your terminal and update the Lambda function using AWS CLI:

```
aws lambda update-function-code --function-name myFunction --region eu-west-1 --
s3-bucket practical-aws-lambda-code --s3-key code.zip
```

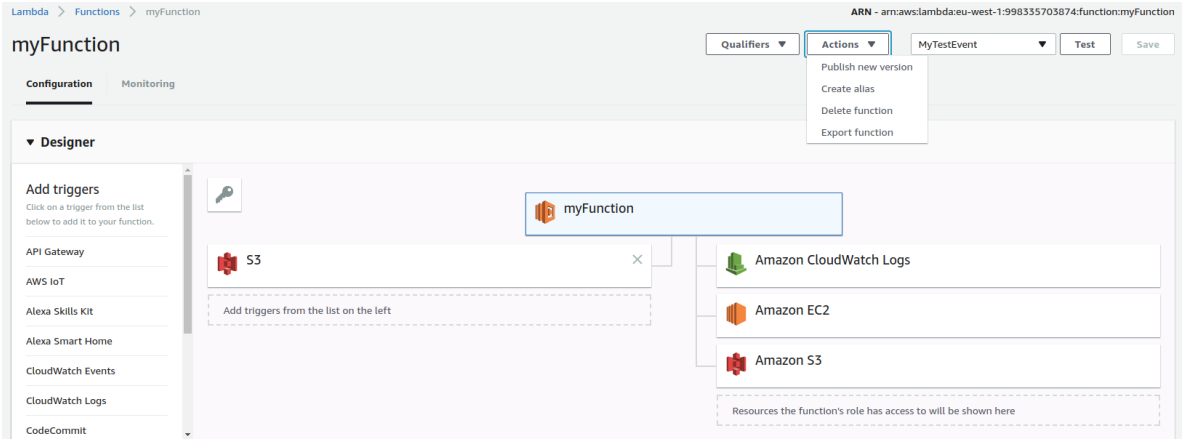
Change these values by your own values:

- --function-name
- --region
- --s3-bucket
- --s3-key

Lambda Versions

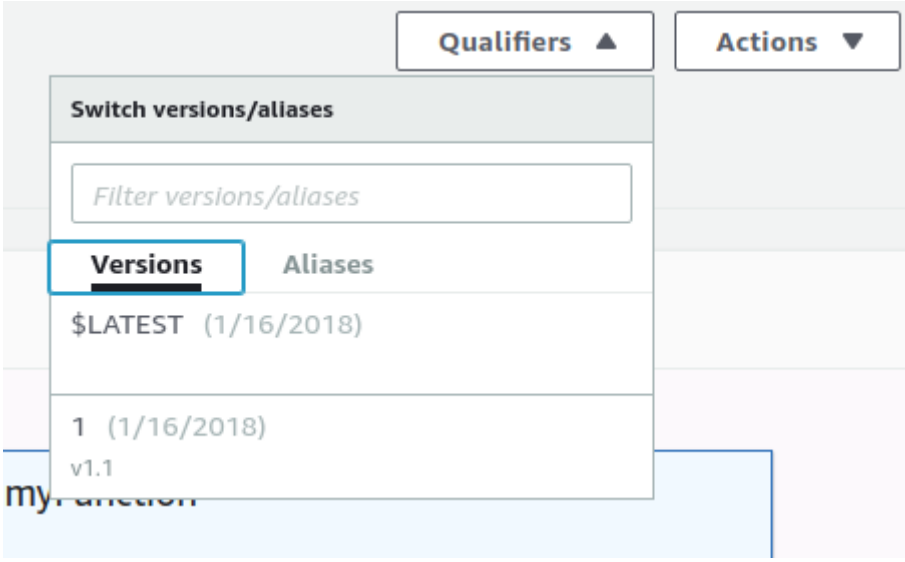
Lambda versions could be defined as the different snapshots that you can make of your project. A version has a number (id).

You can publish a version from the dashboard by clicking on "Publish new version":



> Publishing a new version will save a "snapshot" of the code and configuration of the $LATEST version. You will be unable to edit the new version's code after publishing it.

After publishing a version, click on "Qualifiers" and you'll notice that your new version was added.



You may also configure a test before publishing a version.

Say, you made some modifications to your function, and you want to publish the second version from the CLI.

First of all, if you keep your code without any modifications, Lambda will not increment the actual version. Let's make any modification on our function, like making the GET request use HTTP instead of https.

```
import requests

def lambda_handler(event, context):
    return requests.get('http://s3static.practicalaws.com').status_code
```

Archive your code and dependencies inside a ZIP package, and upload it to S3:

```
aws s3 cp code.zip s3://practical-aws-lambda-code --region eu-west-1
```

Update your code:

```
aws lambda update-function-code --function-name myFunction --region eu-west-1 --
s3-bucket practical-aws-lambda-code --s3-key code.zip
```

Publish a new version (2):

```
aws lambda publish-version --function-name myFunction --description "this is my
v2" --region eu-west-1 --output=text --query 'Version'
```

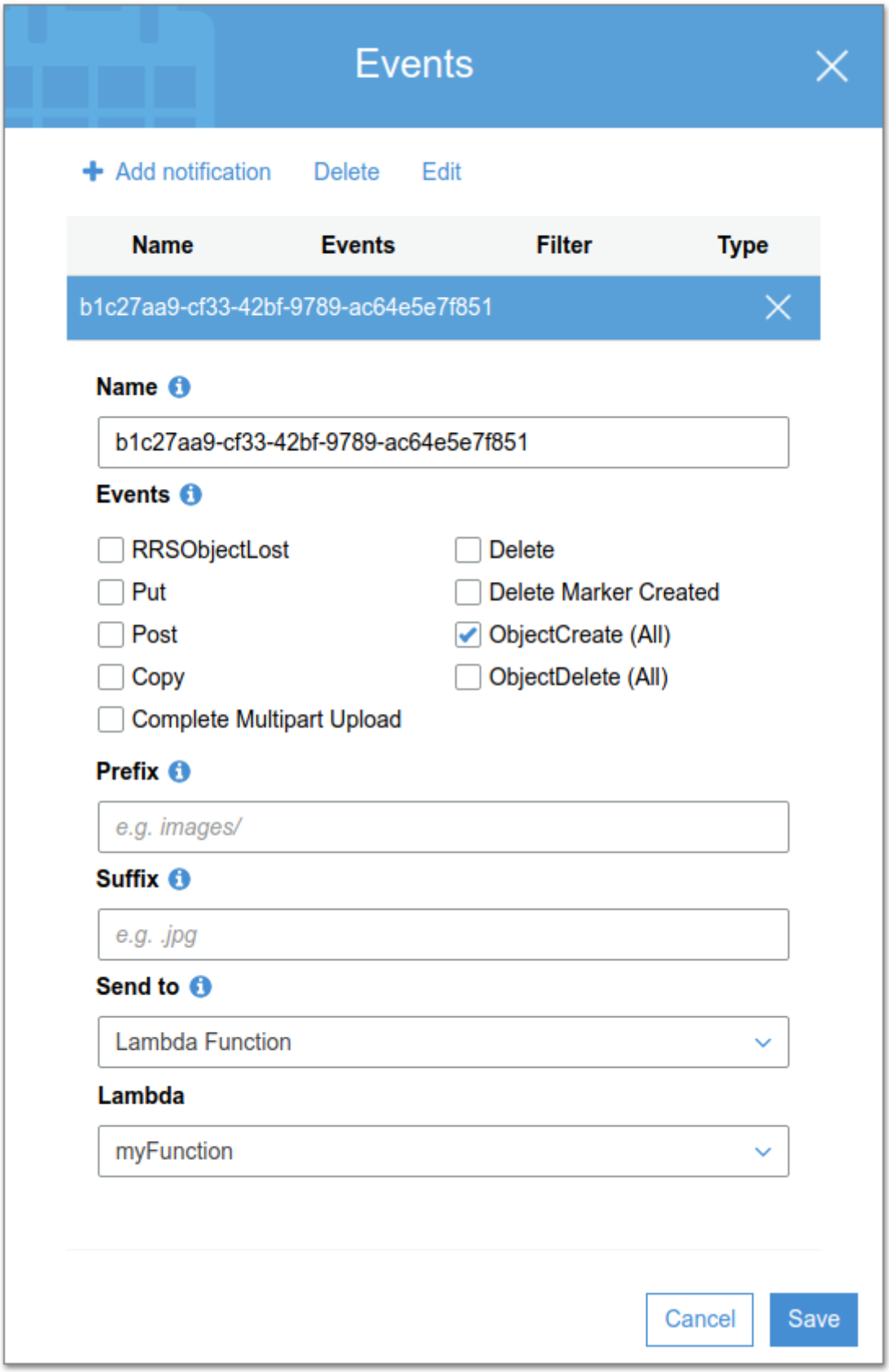
Lambda Aliases

An alias is a pointer to a specific version. Each alias has a unique ARN (Amazon Resource Name).

This could be useful for many scenarios, like pointing a PROD alias to the latest version of your code.

The alias called "Unqualified" is the default alias, and it always points to the default version $LATEST.

Aliases could also be used to attach triggers. In our case, any modification on S3 is a trigger that invokes the function. If you go to our bucket `s3static.practicalaws.com` and click on "Properties" then "Events", you will notice that the event source mapping information is stored in the bucket notification configuration. The Lambda function ARN is stored in this configuration:



In this case, when you publish a new version of your Lambda function, you need to update the notification configuration so that Amazon S3 invokes the correct version. The time between having the new version running in production and the update you will make to the S3 trigger could result in downtime, even if it could be for a small period of time, but it is still downtime.

This is when aliases are useful, you can attach the trigger to an alias instead of a version. You need to update the PROD alias to point to the latest stable version.

You can create an alias an attach it to a version using a similar command to this one:

```
aws lambda create-alias --region eu-west-1 --function-name myFunction --
description "My alias" --function-version "\$LATEST" --name PROD
```

You should, of course, adapt your region, function name, description, function version, and alias name.

Lesson 19 - Creating A Serverless Python API Using AWS Lambda & Chalice

Introduction

Chalice is a Python serverless microframework for AWS, created by Amazon Web Services. It allows developers to build and deploy applications that use Amazon API Gateway and AWS Lambda. When using the AWS cloud, there are some considerations a user should take, like IAM management. Chalice provides automatic IAM policy generation, which makes faster deploying an application.

Why Chalice?

First of all, it is created by AWS.

It allows the development as well as the deployment of Python serverless applications. Other frameworks will only help you develop applications. Finally, because I used it multiple times, I found it stable, easy, and intuitive.

Requirements

To follow this tutorial, make sure you are using python3.6 or a more recent version (3.8 is also ok).

If you are using Ubuntu, follow these instructions:

```
sudo add-apt-repository ppa:jonathonf/python-3.6
sudo apt-get update
sudo apt-get install python3.6
```

Other OSs installation instructions could be found here.

Before starting, you should have an AWS account, and you should have configured your credentials in:

```
~/.aws/credentials
```

If you haven't executed the AWS configuration command, you'll not find anything in your .aws folder. Use:

```
aws configure
```

Make sure you have "pip" and "virtualenv" installed:

```
sudo apt-get install python-pip
sudo pip install virtualenv
```

You can find the instructions for Mac and Windows for the version 15.1.0 here.

Now create a virtual environment if you don't like to touch your system libraries. Then activate it.

```
virtualenv -p python3.6 chalice_example
cd chalice_example
. bin/activate
```

Now install Chalice.

```
pip install chalice
```

Creating A Chalice Project

Let's create our project:

```
chalice new-project chalice_project
cd chalice_project
```

You'll find 2 files inside the project file:

```
.
├── app.py
└── requirements.txt
```

This is the default app.py program:

```
from chalice import Chalice

app = Chalice(app_name='chalice_project')


@app.route('/')
def index():
    return {'hello': 'world'}


# The view function above will return {"hello": "world"}
# whenever you make an HTTP GET request to '/'.
#
# Here are a few more examples:
#
# @app.route('/hello/{name}')
# def hello_name(name):
#    # '/hello/james' -> {"hello": "james"}
#    return {'hello': name}
#
# @app.route('/users', methods=['POST'])
# def create_user():
#     # This is the JSON body the user sent in their POST request.
#     user_as_json = app.current_request.json_body
#     # We'll echo the json body back to the user in a 'user' key.
#     return {'user': user_as_json}
#
# See the README documentation for more examples.
#
```

Creating Our Function

In our example, we will create a function that should return the md5sum of any string.

e.g:

```
import hashlib
m = hashlib.md5()
m.update("testing")
print m.hexdigest()
```

We need to install hashlib using PIP.

```
pip install hashlib
```

Open "app.py", add the libraries we want to import:

```
from chalice import Chalice
import hashlib
```

Create an application

```
app = Chalice(app_name='chalice_project')
```

Add the default / route function:

```
@app.route('/')
def index():
    return {'response': 'ok'}
```

Now create the main function we are going to use:

```
@app.route('/md5/{string_to_hash}')
def hash_it(string_to_hash):
    m = hashlib.md5()
    m.update(string_to_hash.encode("utf-8"))
    return {'response': m.hexdigest()}
```

Notice that the route we are going to use is /md5/{string_to_hash}. We are using "{}" because "string_to_hash" is a variable.

Deploying Our Function

Now when you execute the deployment command `chalice deploy`, these instructions will be executed:

```
Creating deployment package.
Creating IAM role: chalice_project-dev
Creating lambda function: chalice_project-dev
Creating Rest API
Resources deployed:
  - Lambda ARN: arn:aws:lambda:eu-west-1:998335703874:function:chalice_project-
dev
  - Rest API URL: https://05mtn3nsw7.execute-api.eu-west-1.amazonaws.com/api/
```

When you visit the provided url, you'll get the configured message for the / route:

Let's test this URL to create the md5sum of the string "PracticalAWS":

```
https://<Rest API URL>/md5/PracticalAWS
```

You should see something similar to the previous screenshot.

If you made an error in your code and don't want to deploy each time you'd like to test, activate the debug of Chalice. This is how our code looks like:

```
from chalice import Chalice
import hashlib
app = Chalice(app_name='chalice_project')
app.debug = True
@app.route('/')
def index():
    return {'response': 'ok'}
@app.route('/md5/{string_to_hash}')
def hash_it(string_to_hash):
    m = hashlib.md5()
    m.update(string_to_hash.encode("UTF-8"))
    return {'response': str(m.hexdigest())}
```

Have you noticed that the last deployment was for the development environment? To deploy to prod, use:

```
chalice deploy --stage prod



---
 --stage prod
Creating deployment package.
Creating IAM role: chalice_project-prod
Creating lambda function: chalice_project-prod
Creating Rest API
Resources deployed:
  - Lambda ARN: arn:aws:lambda:eu-west-1:998335703874:function:chalice_project-
prod
  - Rest API URL: https://9gp1u0v0g6.execute-api.eu-west-1.amazonaws.com/api/
```

We have two APIs, one for dev and the other for prod. Execute these commands to show both URLs:

```
chalice url --stage dev
chalice url --stage prod
```

Chalice created the new Lambda function; it also created a new API Gateway + IAM configurations for each function automatically. This would take more time if we were configuring each part manually or even using the CLI. Chalice helped us to create, deploy, and manage a function in a very intuitive and easy way.

Configurations

You can view our environments configurations by typing:

```
cat .chalice/deployed.json
```

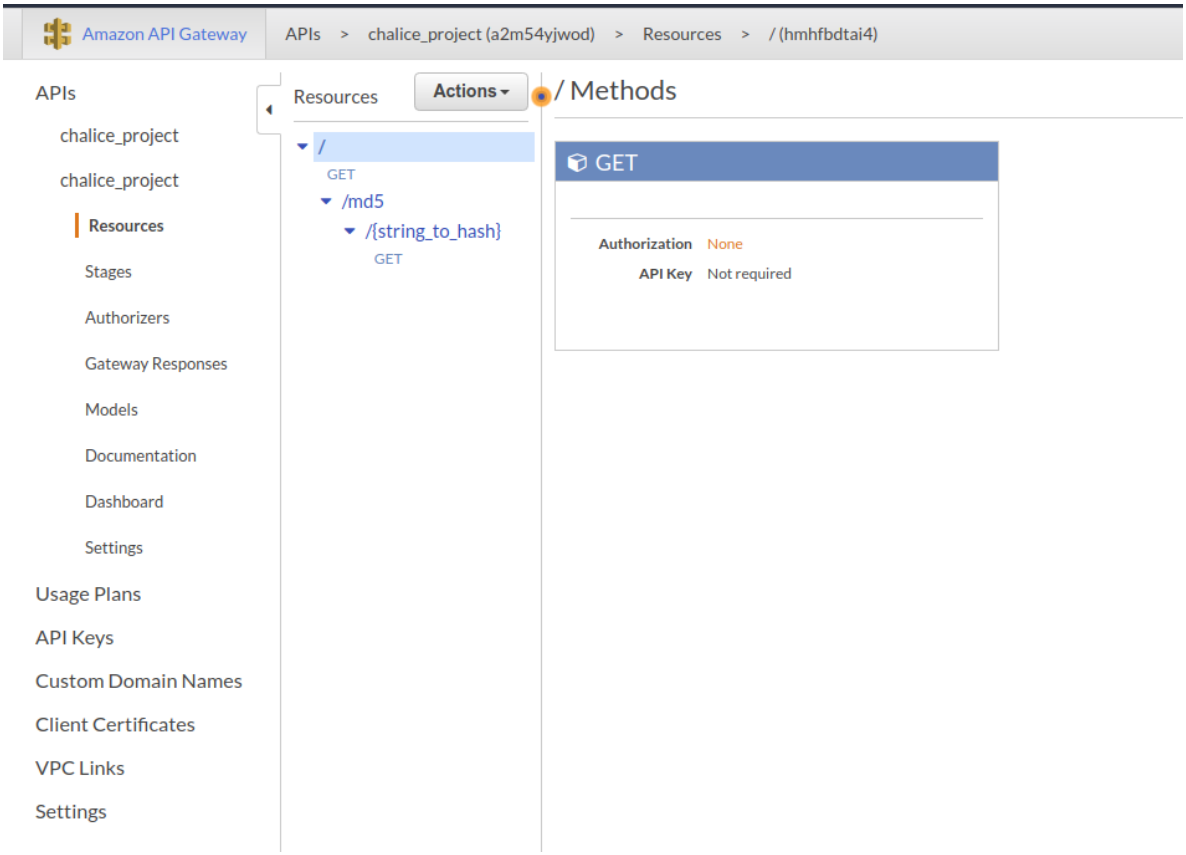
e.g:

```
{
  "dev": {
    "api_handler_name": "chalice_project-dev",
    "api_handler_arn": "arn:aws:lambda:eu-west-
1:998335703874:function:chalice_project-dev",
    "lambda_functions": {},
    "backend": "api",
    "chalice_version": "1.0.2",
    "rest_api_id": "rcnce4er7c",
    "api_gateway_stage": "api",
    "region": "eu-west-1"
  },
  "prod": {
    "api_handler_name": "chalice_project-prod",
    "api_handler_arn": "arn:aws:lambda:eu-west-
1:998335703874:function:chalice_project-prod",
    "lambda_functions": {},
    "backend": "api",
    "chalice_version": "1.0.2",
    "rest_api_id": "4wf2ohyot4",
```

```
    "api_gateway_stage": "api",
    "region": "eu-west-1"
  }
}
```

Using Your Own Domain

In order to use your own domain name, you need to go to the API Gateway dashboard and click on the newly created API. If you followed the same instructions and kept the same names, your API will be called "chalice_project".



Click on "Custom Domain Name" and add a new domain. In my case, I made the choice of using `md5sum.practicalaws.com`. Like you did for the certificate of `s3static.practicalaws.com`, make sure to generate a certificate for `md5sum.practicalaws.com`.

If you choose "Edge Optimised", which means that AWS is going to create a CloudFront distribution for us, remember to only use the region `us-east-1` for the certificate generation using ACM (Amazon Certificate Manager).

Clicking on save will give you the CloudFront distribution subdomain. The latter should be used in Route53, in order to make `md5sum.practicalaws.com` point to its alias (the alias of the CloudFront distribution).

Now our md5sum function could be called using:

```
http://md5sum.practicalaws.com/api/md5/<string_to_hash>
```

Deleting Resources

You can delete an environment resources, for instance, for development, type :

```
chalice delete --stage dev
```

Don't also forget to remove the production resources if you need that:

```
chalice delete --stage prod
```

Lesson 20 - Creating a Serverless Uptime Monitor & Getting Alerted by SMS — Lambda, Zappa & Python

Introduction

Zappa is, like Chalice, a python serverless microframework for AWS.

It allows developers to build and deploy serverless Python applications (including, but not limited to, WSGI web apps) on AWS Lambda + API Gateway.

This is an introduction (probably more) for those who would like to test and learn how to use this framework and deploy an API (or any other application) to AWS Lambda.

We are going to use a Python framework called Flask. Flask is a micro web framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine.

Requirements

We are going to use Python 3.6. For Ubuntu use:

```
sudo add-apt-repository ppa:jonathonf/python-3.6
sudo apt-get update
sudo apt-get install python3.6
```

Other OSs installation instructions could be found here.

Before starting, you should have an AWS account, and you should have configured your credentials in:

```
~/.aws/credentials
```

If you haven't executed AWS configuration command, you'll not find anything in your .aws folder. Use:

```
aws configure
```

Make sure you have "pip" and "virtualenv" installed:

```
sudo apt-get install python-pip
sudo pip install virtualenv
```

You can find the instructions for Mac and Windows for the version 15.1.0 here.

Now create a virtual environment if you don't like to touch your system libraries. Then activate it.

```
virtualenv -p python3.6 zappa_example
cd zappa_example
. bin/activate
mkdir zappa_project
cd zappa_project
```

Now install Zappa and Flask.

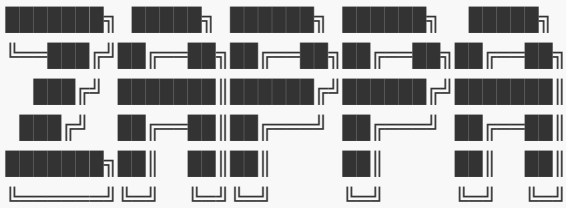
```
pip install zappa flask
```

Creating Our Zappa Project

Execute the "init" command, make sure you are inside "zappa_project" folder:

```
zappa init
```

When initializing a new Zappa project, the framework will ask you some questions that will be used to configure the project. I created a "dev" environment, with my "default" profile. I kept the default name of the bucket "zappa-67mw7olxz" and my app's function was set to "app.app". I didn't deploy my application for now.



```
Welcome to Zappa!

Zappa is a system for running server-less Python web applications on AWS Lambda
and AWS API Gateway.
This `init` command will help you create and configure your new Zappa
deployment.
Let's get started!

Your Zappa configuration can support multiple production stages, like 'dev',
'staging', and 'production'.
What do you want to call this environment (default 'dev'):

AWS Lambda and API Gateway are only available in certain regions. Let's check to
make sure you have a profile set up in one that will work.
Okay, using profile default!

Your Zappa deployments will need to be uploaded to a private S3 bucket.
If you don't have a bucket yet, we'll create one for you too.
What do you want to call your bucket? (default 'zappa-67mw7olxz'):

It looks like this is a Flask application.
What's the modular path to your app's function?
This will likely be something like 'your_module.app'.
Where is your app's function?: app.app
```

```
You can optionally deploy to all available regions in order to provide fast
global service.
If you are using Zappa for the first time, you probably don't want to do this!
Would you like to deploy this application globally? (default 'n')
[y/n/(p)rimary]:

Okay, here's your zappa_settings.json:

{
    "dev": {
        "app_function": "app.app",
        "aws_region": "eu-west-1",
        "profile_name": "default",
        "project_name": "zappa-project",
        "runtime": "python3.6",
        "s3_bucket": "zappa-67mw7olxz"
    }
}

Does this look okay? (default 'y') [y/n]:

Done! Now you can deploy your Zappa application by executing:

    $ zappa deploy dev

After that, you can update your application code with:

    $ zappa update dev

To learn more, check out our project page on GitHub here:
https://github.com/Miserlou/Zappa
and stop by our Slack channel here: https://slack.zappa.io

Enjoy!,
 ~ Team Zappa!
```

So, I used the "dev" environment and kept everything to the default values apart from the modular path to your app's function that was configured to `app.app` since my function will be written in app.py file and my Flask application will be called app.

Now you can see your configuration file under zappa_example:

```
cat zappa_settings.json



---
{
    "dev": {
        "app_function": "app.app",
        "aws_region": "eu-west-1",
        "profile_name": "default",
        "project_name": "zappa-project",
        "runtime": "python3.6",
        "s3_bucket": "zappa-67mw7olxz"
```

```
    }
}
```

Deploying Our Function

We can deploy our Zappa application by executing:

```
zappa deploy dev
```

This is the output of my deployment:

```
Calling deploy for stage dev..
Creating zappa-project-dev-ZappaLambdaExecutionRole IAM Role..
Creating zappa-permissions policy on zappa-project-dev-ZappaLambdaExecutionRole
IAM Role.
Downloading and installing dependencies..
 - sqlite==python36: Using precompiled lambda package
Packaging project as zip.
Uploading zappa-project-dev-1516130123.zip (5.1MiB)..
100%|
████████████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████████████████
████████████████████████████████| 5.30M/5.30M [00:03<00:00, 2.34MB/s]
Scheduling..
Scheduled zappa-project-dev-zappa-keep-warm-handler.keep_warm_callback with
expression rate(4 minutes)!
Uploading zappa-project-dev-template-1516130229.json (1.6KiB)..
100%|
████████████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████████████████
████████████████████████████████| 1.63K/1.63K [00:05<00:00, 309B/s]
Waiting for stack zappa-project-dev to create (this can take a bit)..
100%|
████████████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████████████████
████████████████████████████████| 4/4 [00:25<00:00,  6.59s/res]
Deploying API Gateway..
Deployment complete!: https://jj7zl1c9dg.execute-api.eu-west-1.amazonaws.com/dev
```

After deploying, your source code will be zipped and uploaded to Lambda.

Updating Our Function

This is the Flask application to start with:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello():
    return 'Hello\n'

if __name__ == "__main__":
    app.run()
```

The code should be added to "app.py" file; if you want to choose another filename, make sure that you configured Zappa to use this filename.

After adding the hello world code, we should update our deployment:

```
zappa update dev
```

This is the output of the latest update:

```
Calling update for stage dev..
Downloading and installing dependencies..
 - sqlite==python36: Using precompiled lambda package
Packaging project as zip.
Uploading zappa-project-dev-1516130558.zip (5.1MiB)..
100%|
████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████████
████████████████████████████████| 5.30M/5.30M [00:02<00:00, 1.96MB/s]
Updating Lambda function code..
Updating Lambda function configuration..
Uploading zappa-project-dev-template-1516130597.json (1.6KiB)..
100%|
████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████████
████████████████████████████████| 1.63K/1.63K [00:00<00:00, 6.25KB/s]
Deploying API Gateway..
Scheduling..
Unscheduled zappa-project-dev-zappa-keep-warm-handler.keep_warm_callback.
Scheduled zappa-project-dev-zappa-keep-warm-handler.keep_warm_callback with
expression rate(4 minutes)!
Your updated Zappa deployment is live!: https://08y69lrcek.execute-api.eu-west-
1.amazonaws.com/dev
```

Notice that the creation of the URL `https://08y69lrcek.execute-api.eu-west-1.amazonaws.com/dev` could tell us that an API Gateway was also created for our Lambda function.

Debugging Our Function

You may have some problems, so you will need to examine your logs. You can execute:

```
zappa tail
```

Creating an API Using Flask

Now let's add a method to handle a POST request with user data using Flask:

```
@app.route('/ping', methods=["POST"])
def ping():
    resp_dict = {}

    if request.method == "POST":
        data = request.form
        domain = data.get("domain", "")
```

```
        try:
            result = requests.get(domain,
timeout=timeout).elapsed.total_seconds()
            resp_dict = {"result": result, "response": "200"}
            response = Response(json.dumps(resp_dict), 200)

        except Exception as e:
            resp_dict = {"result": "error", "response": "408"}
            response = Response(json.dumps(resp_dict), 408)

    return response
```

This new method will simply return the response time of the website the user will send `requests.get(domain, timeout=timeout).elapsed.total_seconds()`.

The response is returned after being formatted using `Response(json.dumps(resp_dict), 200)` or `response = Response(json.dumps(resp_dict), 408)` in case the response is not "200 OK".

The example is quite simple, and our goal is not developing the best monitoring system in the industry but learning how we can code a Lambda function using Zappa.

Our Flask app will look like this:

```
from flask import Flask, Response, json, request
import requests
app = Flask(__name__)
@app.route('/')
def hello():
    return 'Hello World !'
@app.route('/ping', methods=["POST"])
def ping():
    resp_dict = {}

    if request.method == "POST":
        data = request.form
        domain = data.get("domain", "")

        try:
            result = requests.get(domain).elapsed.total_seconds()
            resp_dict = {"result": result, "response": "200"}
            response = Response(json.dumps(resp_dict), 200)

        except Exception as e:
            resp_dict = {"result": "error", "response": "408"}
            response = Response(json.dumps(resp_dict), 408)

    return response
if __name__ == "__main__":
    app.run()
```

You can run the Flask app locally:

```
export FLASK_APP=app.py && flask run
```

This will start a local server:

```
 * Serving Flask app "zappa_example.app.app"
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Testing Our Function

We can test it locally using CURL. Let's execute a CURL that will send a POST request with the site URL as POST "data". We will use "http://practicalaws.com" website.

```
curl --data "domain=http://practicalaws.com" http://127.0.0.1:5000/ping
```

This will give us the following output:

```
{"response": "200", "result": 0.05075}
```

Our website is up and running; we can see that the server response is "200". If you try a URL that doesn't exist, you will have a "408" response code:

```
curl --data "domain=http://UnknownSubDomain.practicalaws.com"
http://127.0.0.1:5000/ping
```

Output:

```
{"response": "408", "result": "error"}
```

Our application is running, and it's working. You should not forget to deploy it to production. Open the setting file `zappa_settings.json` and add prod settings, copy/paste the "dev" settings and change the "s3_bucket":

```
{
    "dev": {
        "app_function": "app.app",
        "aws_region": "eu-west-1",
        "profile_name": "default",
        "project_name": "zappa-project",
        "runtime": "python3.6",
        "s3_bucket": "zappa-67mw7olxz"
    },
    "prod": {
        "app_function": "app.app",
        "aws_region": "eu-west-1",
        "profile_name": "default",
        "project_name": "zappa-project",
        "runtime": "python3.6",
        "s3_bucket": "zappa-67mw7olxy"
    }
}
```

Then deploy to production:

```
zappa deploy prod
```

Deleting Our Function:

You can delete this serverless app and remove its API Gateway using:

```
zappa undeploy dev
```

Getting Alerted By SMS Using AWS SNS

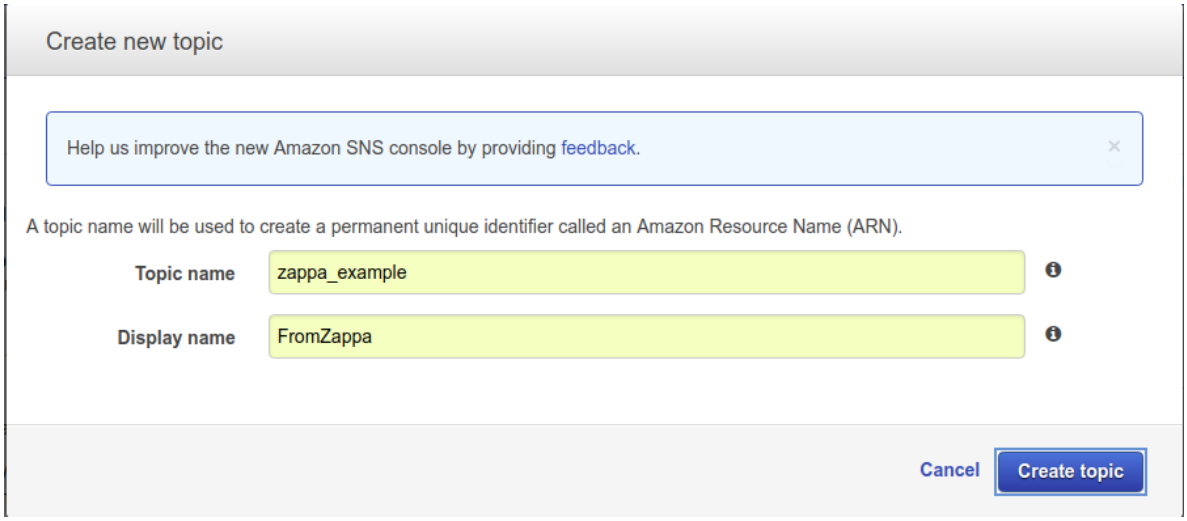
Suppose our website is down; we cant the user or the site engineer to be alerted using SMS.

For this example, I am going to use AWS SNS (Simple Notification Service), which is a Pub/Sub messaging and mobile notifications system.

Here is what we are going to do:

- We will create a "topic"
- We will create an SMS "subscription"
- The "subscription" will watch the topic for new messages
- When the website is down, Lambda should send a message to the "topic"

In order to do this, go to SNS dashboard and create a new topic:



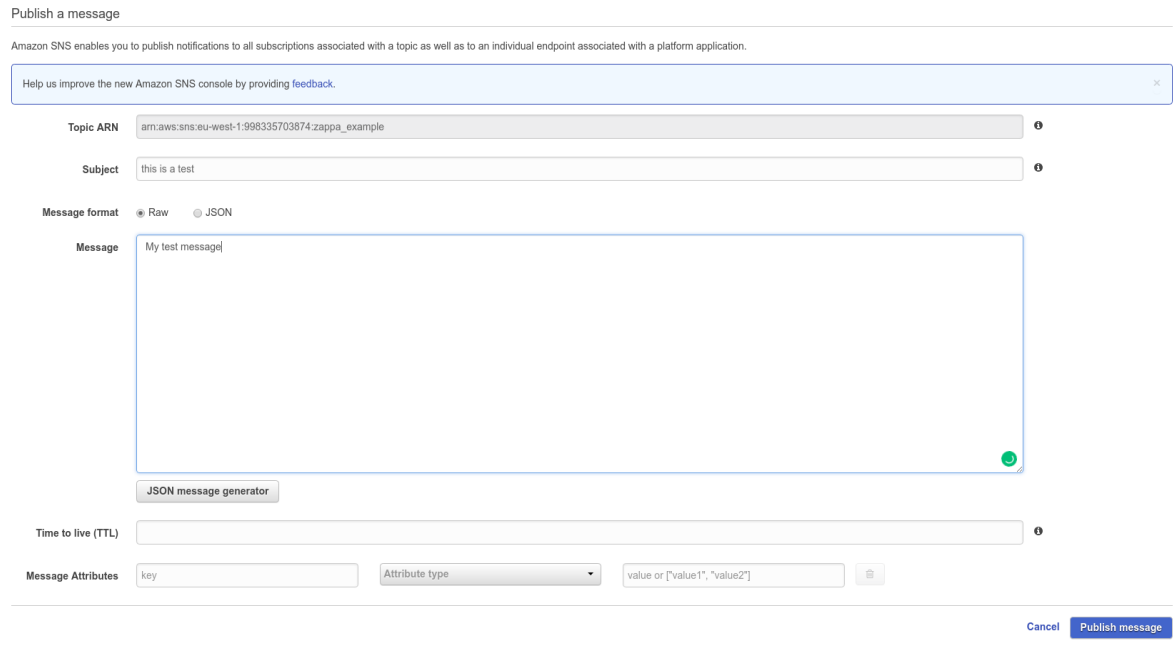
Like any resource in AWS, the new topic has an ARN (Amazon Resource Name), and you can view it on the topics dashboard. I will be using this : `arn:aws:sns:eu-west-1:xxxxxxxxxx:zappa_example`.

Now, go to the subscriptions dashboard and create a new subscription with these parameters:

- Topic ARN: Make sure to enter your topic ARN (e.g arn:aws:sns:eu-west-1:xxxxxxxxxx:zappa_example)
- The Protocol: SMS
- Endpoint: Enter your phone number

Now click on "Create Subscription".

In order to test this, go back to the topics list, choose the topic you created and click on "Publish to topic", choose



Write a test subject as well as message content and send it by clicking on "Publish message". You should instantly receive an SMS on the phone number you configured.

If everything is working, let's add this Python code that sends a message to a topic:

```
client = boto3.client('sns')

response = client.publish(
   TopicArn="arn:aws:sns:eu-west-1:xxxxxxxxxxxxxxx:zappa_example",
   Message="website %s is down" % str(domain),
   )
```

This code creates an SNS client using Boto3 and publishes it to that client. It takes at least two parameters: the Topic ARN and the Message to send. We need to make this code work for us if a website is down.

This is how our initial code will look like:

```
from flask import Flask, Response, json, request
import requests
import boto3
import json

app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello World !'

@app.route('/ping', methods=["POST"])
def ping():
    resp_dict = {}

    if request.method == "POST":
        data = request.form
        domain = data.get("domain", "")

        try:
```

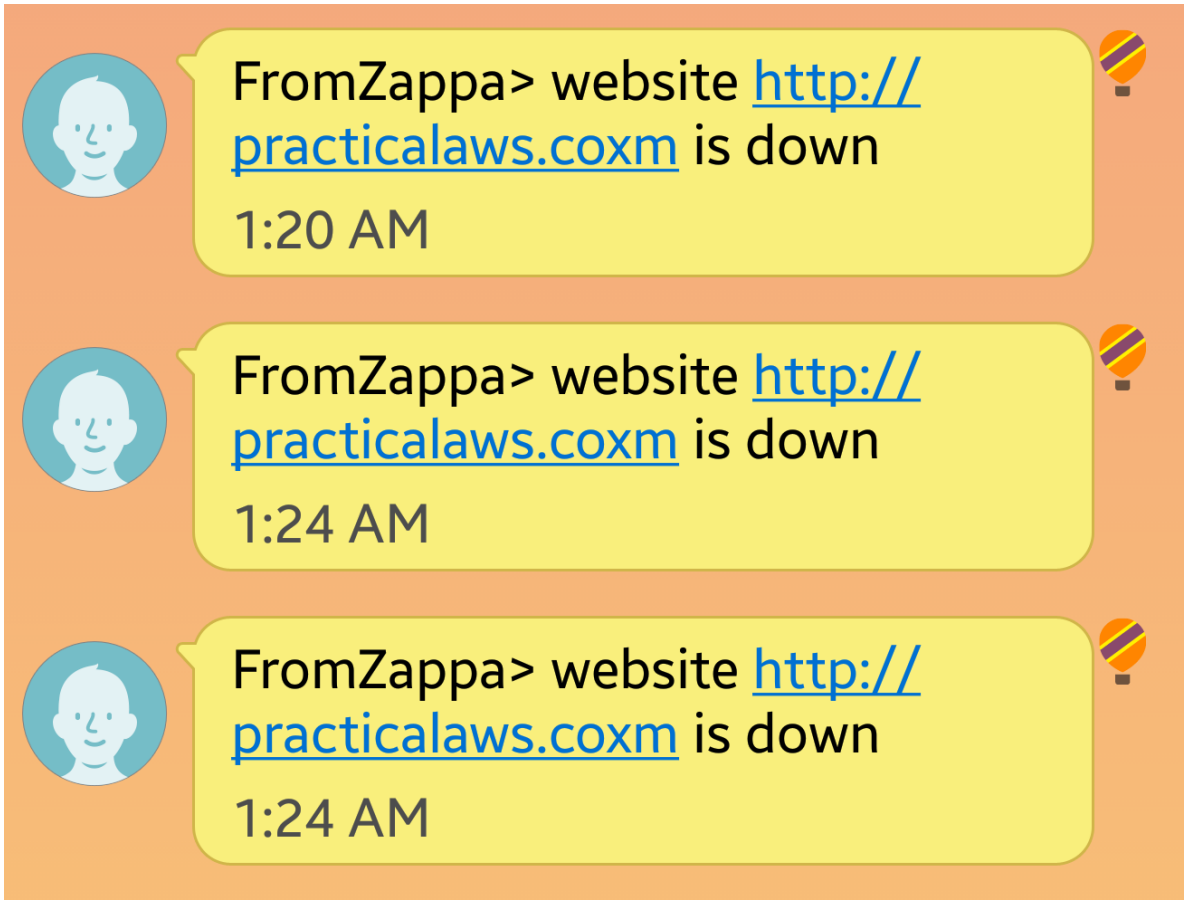
```
            result = requests.get(domain).elapsed.total_seconds()
            resp_dict = {"result": result, "response": "200"}
            response = Response(json.dumps(resp_dict), 200)

        except Exception as e:
            resp_dict = {"result": "error", "response": "408"}
            response = Response(json.dumps(resp_dict), 408)

            client = boto3.client('sns')
            client.publish(
                TopicArn="arn:aws:sns:eu-west-
1:xxxxxxxxxxxxxxxxx:zappa_example",
                Message="website %s is down" % str(domain),
            )




    return response
if __name__ == "__main__":
    app.run()
```

Let's update our code using `zappa update dev`. This will give you the URL of the API Gateway at the end of the deployment. We can test if we receive the SMS using a test URL that doesn't exist like `https://xxx.practicalaws.com`:

```
curl --data "domain=http://xxx.practicalaws.com" https://6ezc8qxb7f.execute-
api.eu-west-1.amazonaws.com/dev/ping
```

Make sure to change `http://xxx.practicalaws.com` and `https://6ezc8qxb7f.execute-api.eu-west-1.amazonaws.com` by your own values.

(In my test, I used wrong URLs like "http://practicalaws.comx" to force receiving the alert without setting my website down)

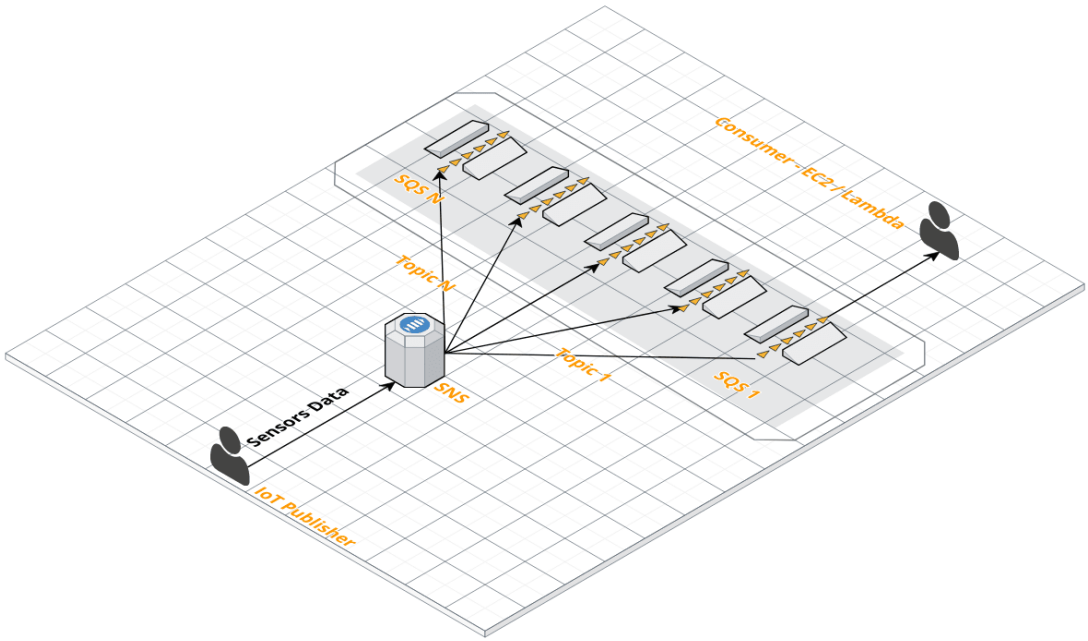

FromZappa> website http://practicalaws.coxm is down
1:20 AM
FromZappa> website http://practicalaws.coxm is down
1:24 AM
FromZappa> website http://practicalaws.coxm is down
1:24 AM

Lesson 21 - Prototyping A Pub/Sub System Using SNS & SQS For IoT, Inter-Process Communication, Microservices Architecture & Event-Based Systems

Introduction

Amazon SNS and SQS are AWS managed solutions. They are easy to configure, but you may find several open-source alternatives. For this use case, imagine that we have an IoT device (e.g., a RasberryPi) sending a sensor data like (e.g., temperature, pressure ..etc.) and sending it to either an EC2 machine or a Lambda function.

We can, of course, use Zappa, Chalice, or Serverless frameworks to deploy and manage our Lambda function.



The results in this benchmark are relative to the used internet connection. For your information, when writing this, I used a connection with a bandwidth of 9.73 Mbps download /11.09 Mbps upload.

I used the published code on a RasberryPi 3, but I adapted it to run on a computer or a server. You can use a Raspberry Pi or a computer/server as a publisher. The consumer is a backend AWS service.

Amazon Simple Notification Service

Amazon SNS is a fully managed push notification service that lets you send individual messages or fan-out messages to large numbers of recipients. SNS could be used to send push notifications to mobile device users, email recipients, or messages to other services (like SQS).

Amazon Simple Queue Service

Amazon SQS is a fully managed message queuing service. Amazon SQS is used to transmit any volume of data without losing messages. Amazon SQS can act as a FIFO queue (first-in, first-out).

Unix Philosophy & Microservice Based Software

I believe that microservices are changing the way we think about software and DevOps. I am not saying that it is the solution to every problem you have, but sharing the complexity of your stack between programs that work together (Unix Philosophy) could resolve many of them.

Adding a containerization layer to your microservices architecture using Docker or rkt and an orchestration tool (Swarm, K8S ..) are the first steps to build cloud-native applications. Containerization and orchestration will simplify the whole process from development to operations and help you manage the networking and increase your stack performance, scalability, and self-healing.

When using microservices, like many of you, I adopted the best practice and the philosophy of a single process per service. This is what I am also considering for this tutorial, so inter-process communication falls into the same thing as microservices containers, where each microservice runs a single isolated process that does one thing, does it well, and communicates with other processes (Unix Philosophy).

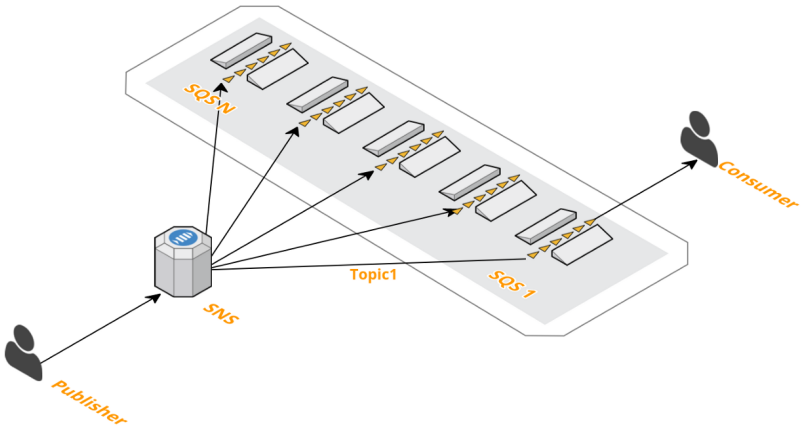
The UNIX philosophy is documented by Doug McIlroy in The Bell System Technical Journal from 1978:

- Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new “features”.
- Expect the output of every program to become the input to another, as yet unknown, program. Don’t clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don’t insist on interactive input.
- Design and build software, even operating systems, to be tried early, ideally within weeks. Don’t hesitate to throw away the clumsy parts and rebuild them.
- Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you’ve finished using them.

Developing software while keeping in mind the Unix philosophy principles will avoid you a lot of headaches.

A Common Architecture For Message-Based Microservices

A publisher sends a message to SNS with a pre-selected Topic, in accordance with it, SNS dispatch a message to a subscribed SQS queue.



POWERED BY CLOUDCRAFT.CO

In this tutorial, we will code a prototype using Python3 and Boto3.

Building The Publisher

Start by creating a topic on your SNS dashboard (I call it pub), then a virtual environment, and finally installing Boto3 (Python interface to Amazon Web Services).

```
virtualenv -p python3.6 sns_test
```

You can use your preferred Python3 version.

Activate the virtual environment:

```
cd sns_test
. bin/activate
```

This is a minimal publisher code:

```
import boto3
client = boto3.client('sns')

pub = client.publish(
                TopicArn=topic_arn,
                Message=my_message,
                )
```

Let's start by creating an SNS topic called "pub".

I need the following code to send a string "x" to the topic with the created topic which has the ARN `arn:aws:sns:eu-west-3:998335703874:pub`.

It should also sleep for 5 seconds before sending the next message, and the program should exit once 100 messages are sent.

I will also print to my terminal the date followed by the sent message and its ID:

```
import boto3, time, json, logging
from datetime import datetime

logging.basicConfig(filename="sns-publish.log", level=logging.DEBUG)

""" Connect to SNS """
client = boto3.client('sns')

""" Send 100 messages """
for x in range(100):

    message = str(x)
    pub = client.publish(
                TopicArn="arn:aws:sns:eu-west-3:998335703874:pub",
                Message=message,
            )


    m = json.loads(json.dumps(pub, ensure_ascii=False))


    message_id = m["MessageId"]

    print (str(datetime.now()) + " - Message Number " + message + " - With ID "
+ message_id)
    time.sleep(5)
```

The execution of `python pub.py` will give us something like this:

```
2018-01-17 07:31:28.341268 - Message Number 0 - With ID 82aa3e5e-1803-55f6-a9d9-
6c86e803abc9
2018-01-17 07:31:33.400489 - Message Number 1 - With ID d5710118-86f3-58ff-9684-
53c076f6a6c0
2018-01-17 07:31:38.465132 - Message Number 2 - With ID dcb430b2-6e0f-5306-aa1d-
626a9e05bd39
2018-01-17 07:31:43.520685 - Message Number 3 - With ID f0d5f8d4-173e-5906-90d9-
aa0ce7310739
2018-01-17 07:31:48.590203 - Message Number 4 - With ID 4be5bc51-bb0b-5e5b-b2f8-
ee894b1fd064
2018-01-17 07:31:53.649722 - Message Number 5 - With ID 7cbdc47e-a974-5073-820f-
96a6596d4e4c
```

Notice that each message sent to our AWS SNS topic has a unique ID. This is automatically generated by AWS.

AWS SQS Standard vs. FIFO Queue

There are two types of messages queues, standard, and FIFO.

According to AWS, the standard queue will ensure:

- **Unlimited Throughput:** Standard queues support a nearly unlimited number of transactions per second (TPS) per API action.
- **At-Least-Once Delivery:** A message is delivered at least once, but occasionally more than one copy of a message is delivered.

- **Best-Effort Ordering:** Occasionally, messages might be delivered in an order different from which they were sent.

AWS also gives some use cases for this type of queue.

- Decouple live user requests from intensive background work: let users upload media while resizing or encoding it.
- Allocate tasks to multiple worker nodes: process a high number of credit card validation requests.
- Batch messages for future processing: multiple schedule entries to be added to a database.

The FIFO queue, on the other hand, will ensure:

- **High Throughput**: FIFO queues support up to 300 messages per second (300 send, receive, or delete operations per second). When you batch 10 messages per operation (maximum), FIFO queues can support up to 3,000 messages per second. To request a limit increase, file a support request.
- **First-In-First-out Delivery**: The order in which messages are sent and received is strictly preserved.
- **Exactly-Once Processing**: A message is delivered once and remains available until a consumer process and deletes it. Duplicates are not introduced into the queue.

These are some use cases:

- Ensure that user-entered commands are executed in the right order.
- Display the correct product price by sending price modifications in the right order.
- Prevent a student from enrolling in a course before registering for an account.

In practice, if you are developing applications where the order of receiving and processing information is important and if you need the first input to be the first output, you should use the FIFO (first-in, first-out) queue.

If you don't care about the order or need higher throughput, then the standard queue is the best choice for you.

Building The Consumer

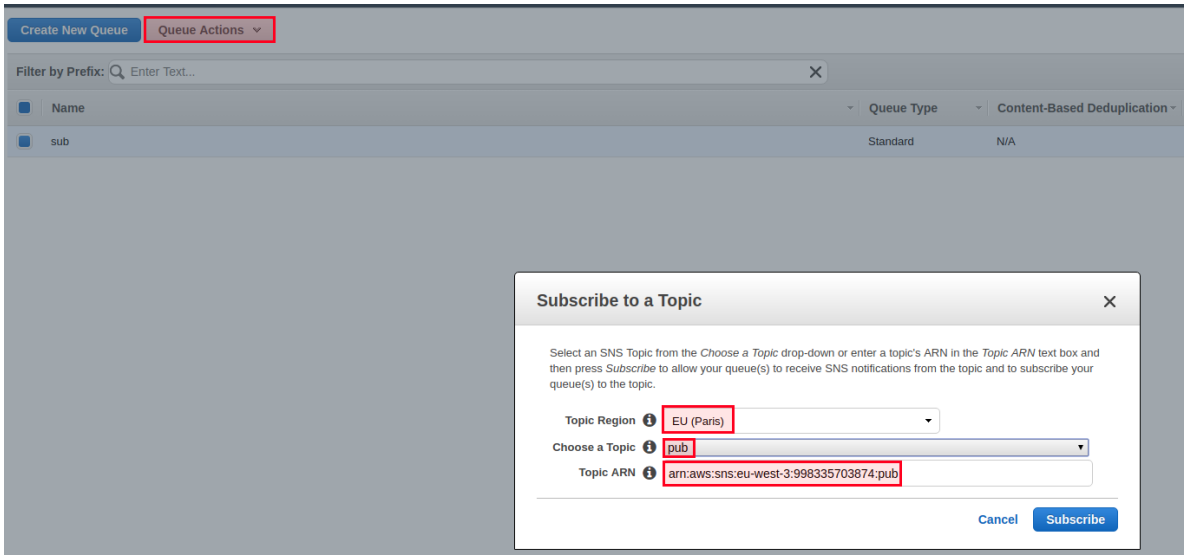
In our case, we don't need any specific order to process the inputs, so let's choose the regular SQS queue.

Open the SQS dashboard and create a standard queue. I called it "sub". You can customize how your SQS queue will work by adjusting values like :

- Default Visibility Timeout
- Message Retention Period
- Maximum Message Size
- Delivery Delay
- Receive Message Wait Time
- Dead Letter Queue Settings

You can also create a new queue and use it as a dead letter queue and use SSE for server-side encryption. Let's keep the default values since we don't have any specific requirements for our pub/sub application.

After creating the queue, click on "Subscribe Queue to SNS Topic" using the queue actions menu. Select your region and select the SNS topic you created for the publisher then hit "Subscribe".

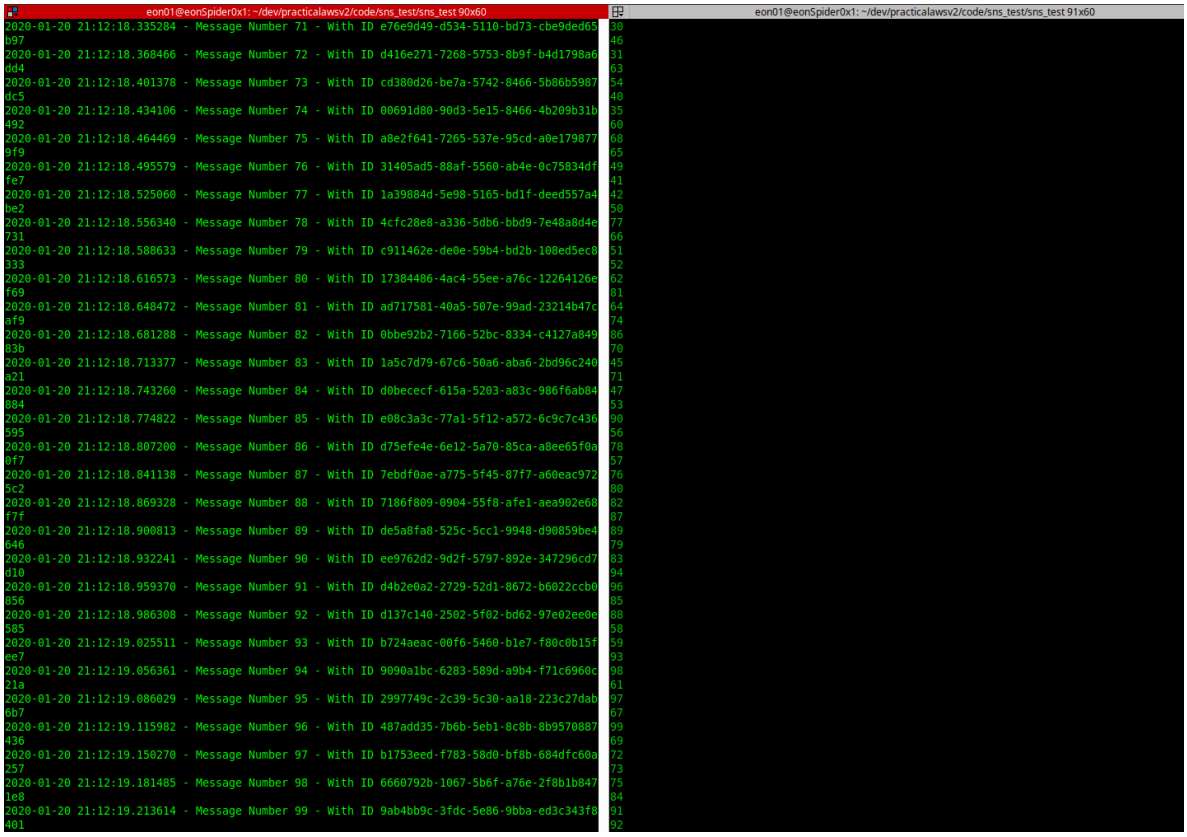


While our pub.py will send "1", "2", "3" ... "100" to the SNS topic, our SQS queue is subscribed to it and should receive the same messages without a particular order since we create a standard queue (not a FIFO one).

This is the code we are going to use for our SQS queue called "sub":

```
import boto3, time, json
import os
from datetime import datetime

sqs = boto3.resource('sqs')
queue = sqs.get_queue_by_name(QueueName='sub')


""" Process messages by printing out body """

while 1:
    for message in queue.receive_messages():
        print('%s ' % json.loads(message.body)["Message"]  )
        message.delete()
```

I split my terminal into two windows, started the subscriber first (left), then the publisher (right). This is a good way to test before deploying your code.



How Fast Is SNS+SQS?

In order to see how much time does the transportation of a text message takes, we are going to use the timestamp in both scripts. This is the simplified flow diagram: **Publisher (RasberryPi or a server/computer) -> SNS -> SQS -> Subscriber (AWS Infrastructure)** .

We are sending a message of almost 75B. In order to be more precise and make better measurements of the response time, I changed the two programs and added a line that prints the timestamp in each message creation or consumption.

In Python, you can use `time.time()` function.

The difference between the reception time and the sending time will be calculated (in accordance with the size of the data sent):

```
T = Time SQS received the message - Time when SNS Published the message
```

I used a spreadsheet with two columns where I pasted the different timestamps:

G	H	I	J	K
	1485806448	1485806447.9	0.1000001431	
	1485806448	1485806448	0.1000001431	
	1485806448	1485806448	0.05999994278	
	1485806448	1485806448	0.09999990463	
	1485806448	1485806448	0.06999993324	
	1485806448	1485806448	0.05999994278	
	1485806448	1485806448	0.04999995232	
	1485806448	1485806448	0.02999997139	
	1485806448	1485806448	0.05999994278	
	1485806448	1485806448	0.05000019073	
	1485806448	1485806448	0.04000020027	
	1485806448	1485806448	0.06999993324	
	1485806448	1485806448	0.05999994278	
	1485806448	1485806448	0.1099998951	
	1485806448	1485806448	0.08999991417	
	1485806448	1485806448	0.06999993324	
	1485806448	1485806448	0.04999995232	
	1485806448	1485806448	0.04999995232	
	1485806448	1485806448	0.04000020027	
	1485806448	1485806448	0.03000020981	
	1485806448	1485806448	0.02999997139	
	1485806448	1485806449	0.1099998951	
	1485806449	1485806449	0.120000124	
	1485806449	1485806449	0.07000017166	
	1485806449	1485806449	0.1099998951	
	1485806449	1485806449	0.1099998951	
	1485806449	1485806449	0.07999992371	
	1485806449	1485806449	0.06999993324	
	1485806449	1485806449	0.04999995232	

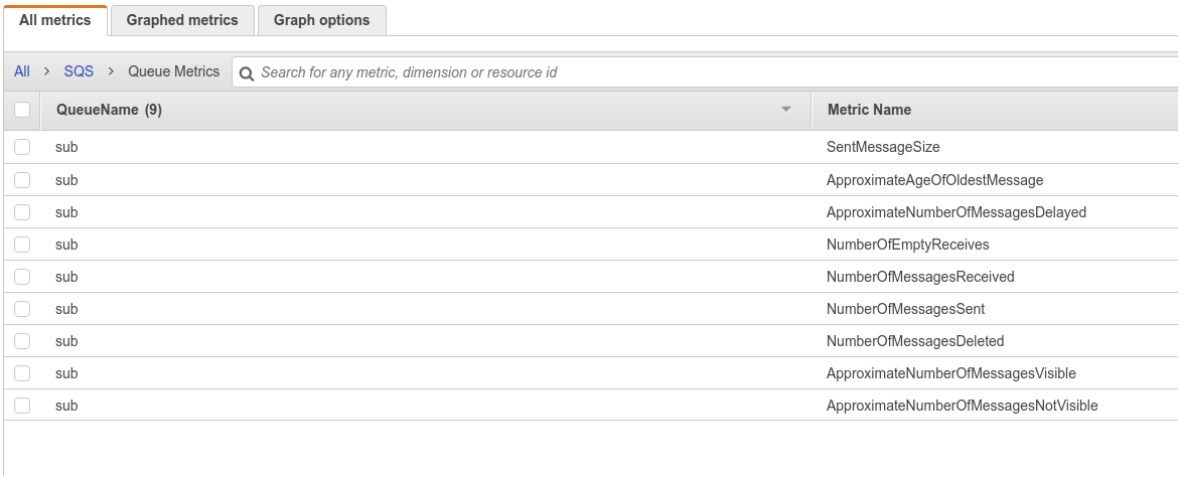
For every size, 20 serial requests are sent each time.

Here are the different data size sent from SNS to SQS:

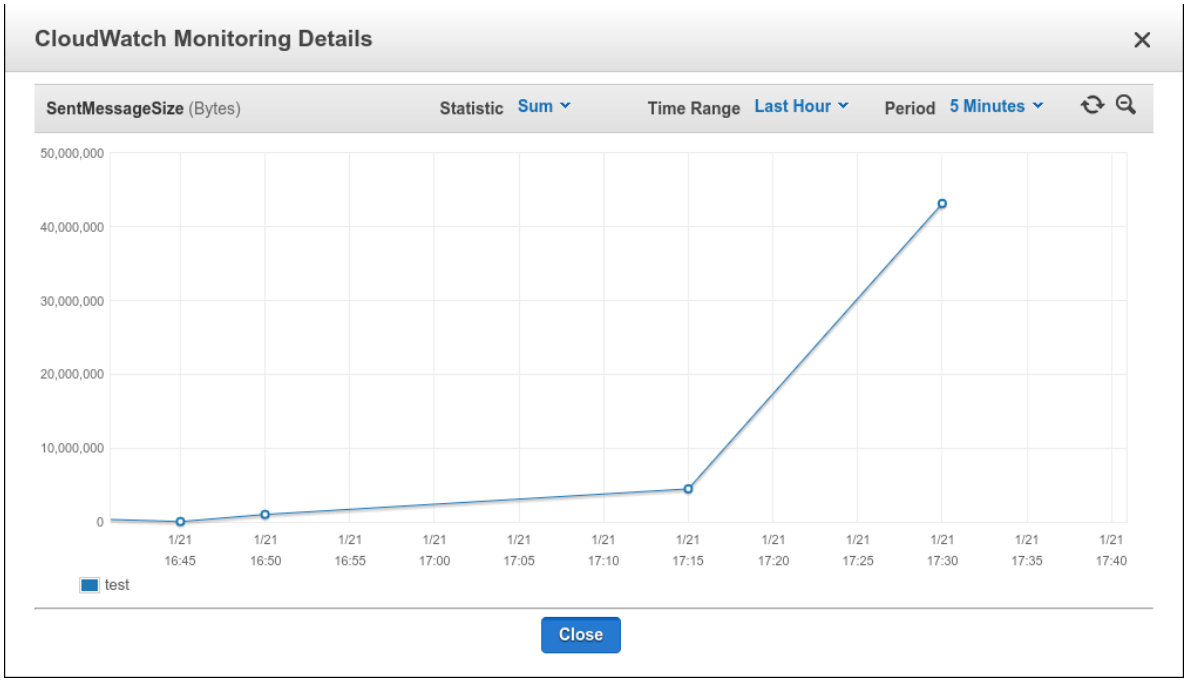
- 75B
- 700B
- 1.65KB
- 3.3KB
- 6.6KB
- 26.38KB

For the measurement part, we are going to use CloudWatch. Go to this service dashboard and click on "Metrics", you will find different services including SNS and SQS. If you choose SQS, for instance, you will notice that it has different metrics like:

- SentMessageSize
- ApproximateAgeOfOldestMessage
- ApproximateNumberOfMessagesDelayed
- NumberOfEmptyReceives
- NumberOfMessagesReceived
- ..etc

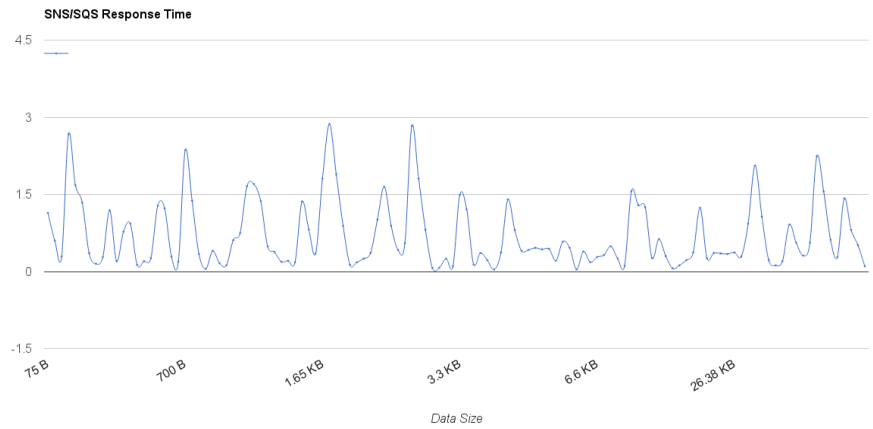


The message received by SQS, will not be the same as the sent data since other data and metadata are also sent with.

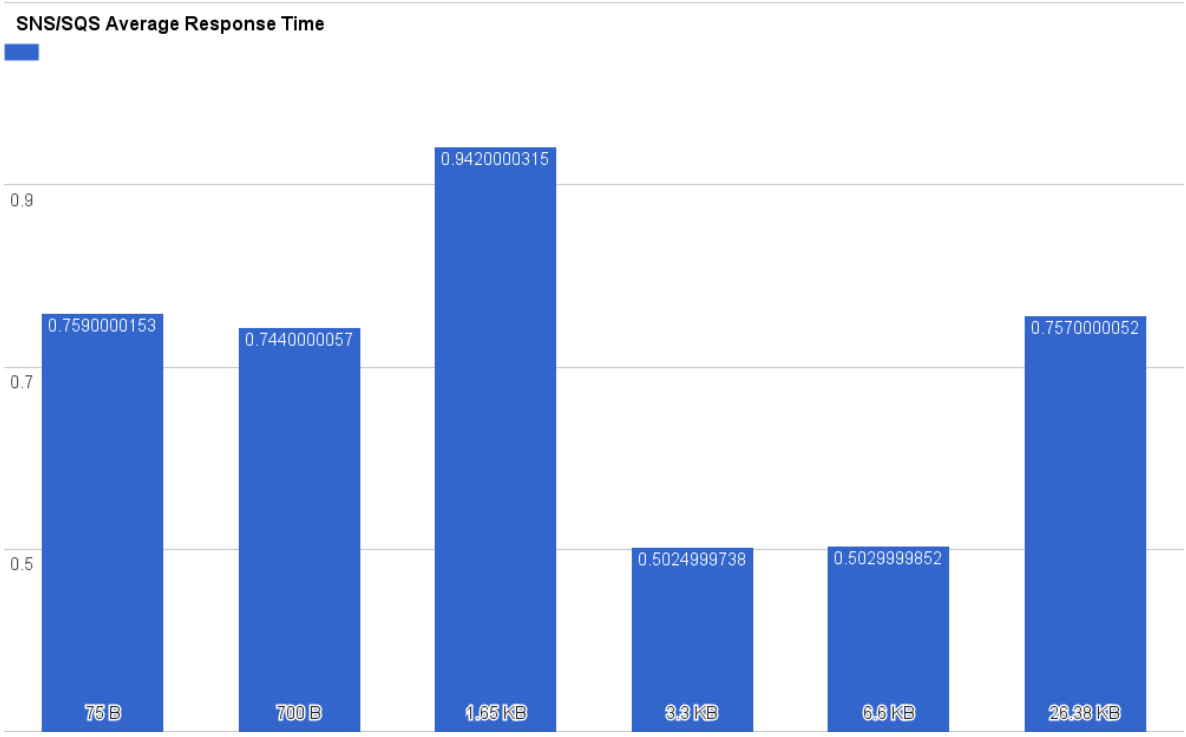


I stopped the benchmark at 26.38KB because there is a limit:

> With Amazon SNS and Amazon SQS, you now have the ability to send large payload messages that are up to 256KB (262,144 bytes) in size. To send large payloads (messages between 64KB and 256KB), you must use an AWS SDK that supports AWS Signature Version 4 (SigV4) signing.

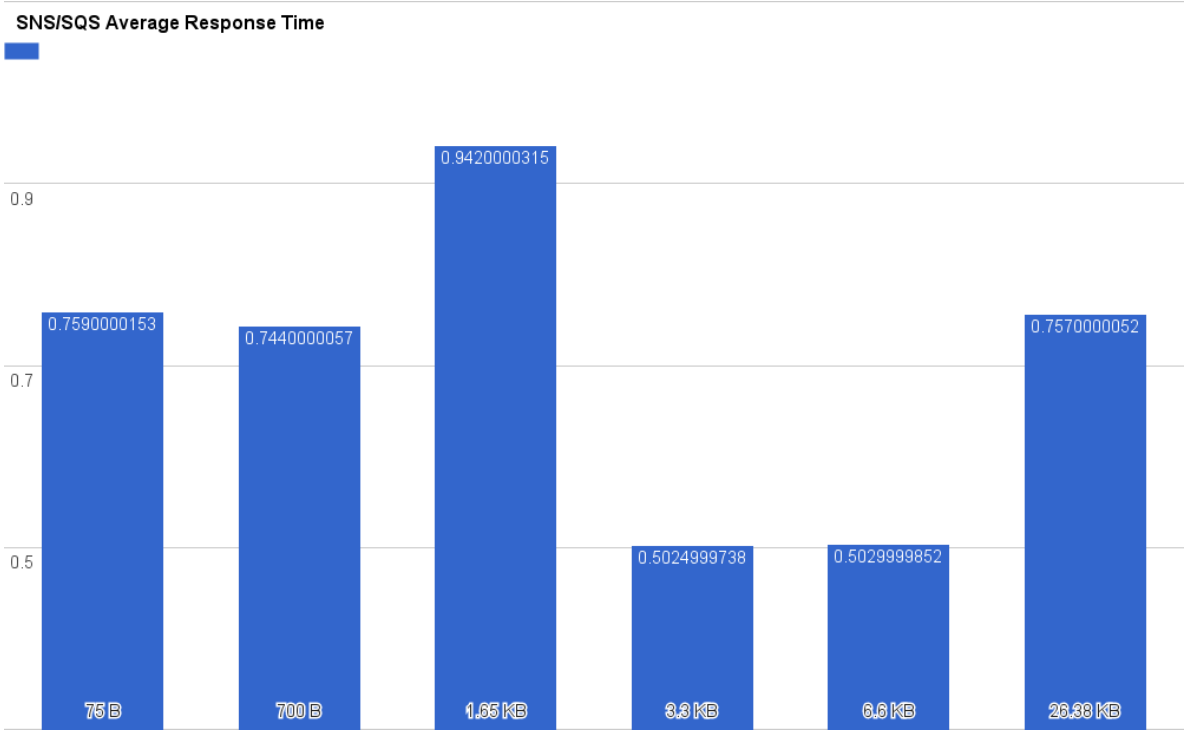


The response time is not changing if the data size increases, which is a good thing. Let's see the average response time in function of the data size:



For reasonable data sizes, the process of sending data to SNS that dispatch it to SQS + the time that my Python program takes to read the data is between 0.5 and 0.9 seconds.

During this benchmark, almost 1000 message was sent, I noticed that all the messages were delivered; there are no lost messages.



SNS/SQS In Multiple Regions

Apart from my Internet connection and laptop/Rasberry PI/server configuration, the speed depends on how you manage your SNS/SQS regions. (I've been using Dublin regions from Paris in this example.) To optimize the message transportation between publisher and subscriber, keep them as well as SNS and SQS in the same region.

If publishers are not in the same regions (say you have multiples publishers, one is in Asia, the other one is in Africa, and the third one is in Australia), the best thing to do here is to create multiple SNS/SQS each in a different region.

Conclusion

Microservices, distributed computing, IoT, and other new fields are changing how we make software, but one of the weaknesses in these fileds is networking. It could be complex sometimes, and messaging is impacted directly by the networking problems. Using SNS/SQS and a pub/sub model seems to be a good solution to create an inter-service messaging middleware.

The publisher/subscriber scripts that I used are not really optimized for load and speed, but our goal was implementing a practical use case that we can understand first and scale later.

For this example, I considered using a RasberryPi that sends messages to an EC2 instance. It is also possible to consider Amazon IoT service that lets you also connect and interact with devices with AWS cloud applications and other devices.

AWS IoT Device SDK enables your devices to connect, authenticate, and exchange messages with IoT Core using the MQTT, HTTP, or WebSockets protocols.

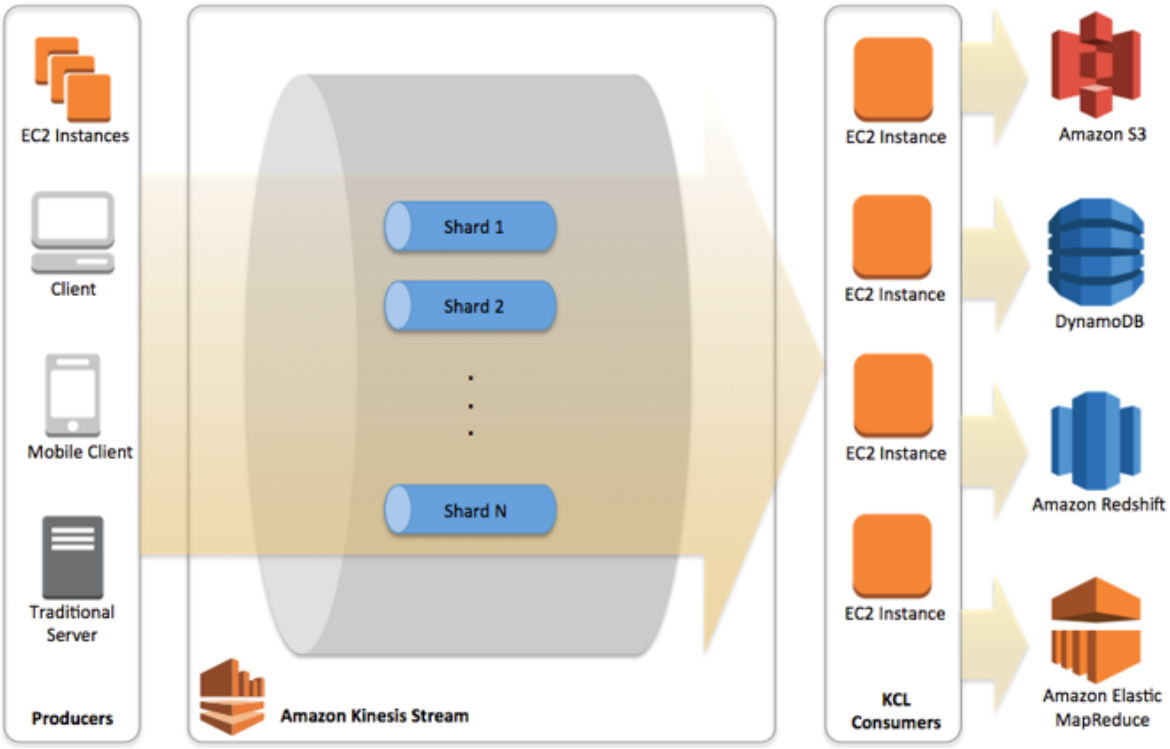
The same code and patterns could be implemented the communication between two Microservices or processes (e.g., a Docker containers communication, log streaming, event-based systems ..etc.)

Lesson 22 - Using Amazon Kinesis & Firehose to Stream & Consume Data in Real-Time

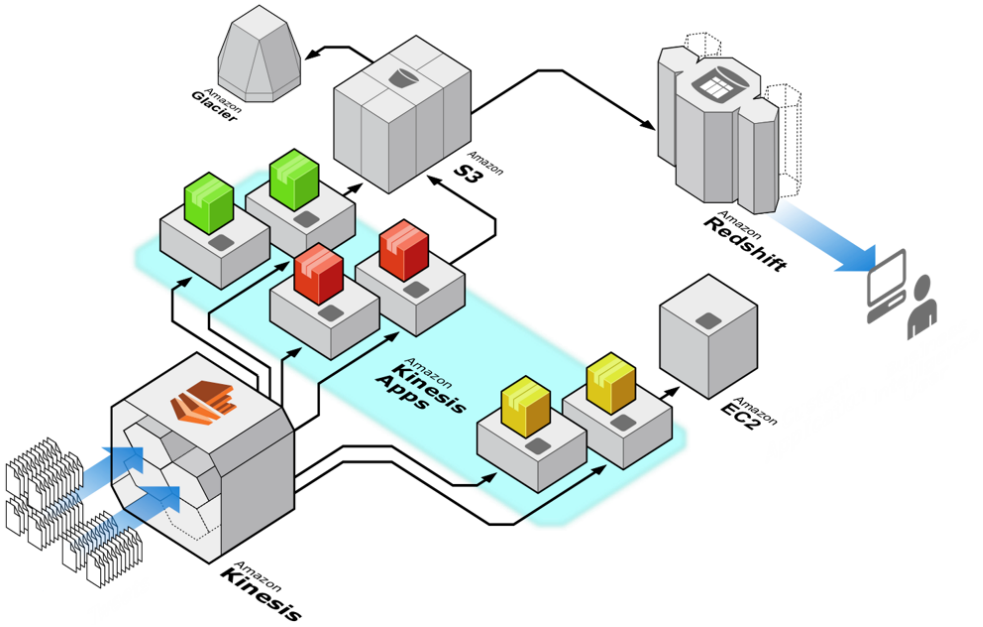
Introduction

In this part of the course, we are going to build a pseudo-real-time analytics and event processing system for large amounts of data. This data can be collected from multiple sources: IoT logs from servers, routers, distributed processing systems ..etc.

You are going to see the most important concepts about Kinesis and write a simple app in Python for a better understanding.



Kinesis offers good performances for data processing, and it is also easy to integrate with other Amazon services.



The benefit of Kinesis is also giving developers the ability to make calls, request data, and analyze it while it is transiting/streaming between the producer (the machine sending the stream) and the data stores (S3, Redshift, Elastic Search ..etc.).

To capture the streaming data, we are going to use Amazon Kinesis Firehose.

Amazon Kinesis Streams Concepts

To use Kinesis, you need to create a Kinesis Stream that will collect and stream data for ordered, replayable, real-time processing.

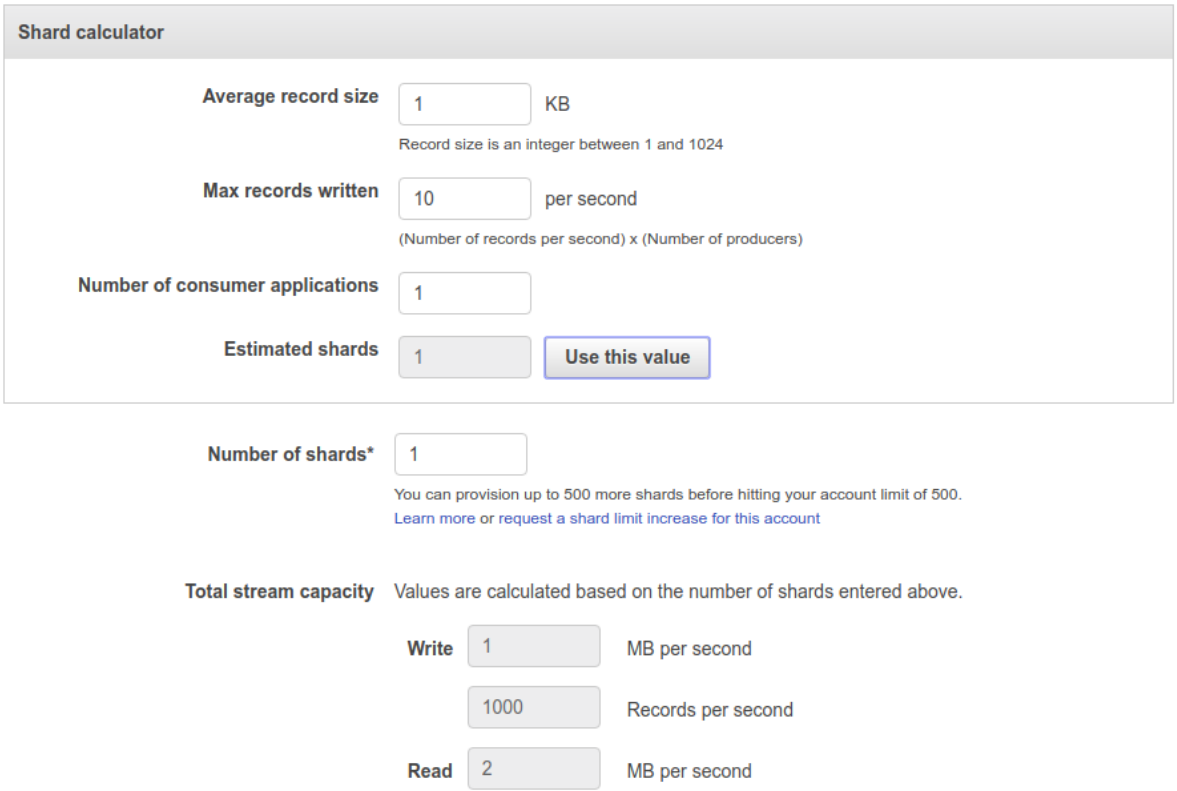
In our use case, we need to process data with our own applications, or using AWS managed services like Amazon Kinesis Data Firehose, Amazon Kinesis Data Analytics, or AWS Lambda.

Go to your console and just create a **data stream** with the name "KinesisStream".

In AWS Kinesis console, you will find different other use cases other than creating a simple data stream.

- **Delivery stream**: It allows you to continuously collect, transform, and load streaming data into destinations such as Amazon S3 and Amazon Redshift.
- **Analytics application**: It allows running continuous SQL queries on streaming data from Kinesis data streams and Kinesis Firehose delivery streams.
- **Video stream**: It helps in building applications to process or analyze streaming media.

Let's assume we are going to need 1KB as an average record size, 10 written records per second, and 1 consumer application. The shard calculator will let you know how many shards you will need for your Kinesis stream.



So one shard should enough. Create the stream with this configuration.

This is the output of `aws kinesis describe-stream --stream-name KinesisStream`:

```
{
    "StreamDescription": {
        "Shards": [
            {
                "ShardId": "shardId-000000000000",
                "HashKeyRange": {
                    "StartingHashKey": "0",
                    "EndingHashKey": "340282366920938463463374607431768211455"
                },
                "SequenceNumberRange": {
                    "StartingSequenceNumber":
"49603477465445893575205288340687793174278467900143566850"
                }
            }
        ],
        "StreamARN": "arn:aws:kinesis:eu-west-
3:998335703874:stream/KinesisStream",
        "StreamName": "KinesisStream",
        "StreamStatus": "ACTIVE",
        "RetentionPeriodHours": 24,
        "EnhancedMonitoring": [
            {
                "ShardLevelMetrics": []
            }
        ],
        "EncryptionType": "NONE",
        "KeyId": null,
        "StreamCreationTimestamp": 1579600126.0
    }
```

```
}
```

You can get information like:

- The stream name
- The stream ARN
- The shards
- The data retention period
- ..etc

Shards

A shard is the base throughput unit of a stream. We used 1, but you can configure your stream to use more shards. This number depends on how much data you are going to send per second because each shard provides a capacity of 1MB/sec data input and 2MB/sec data output. Also, it can support up to 1000 PUT records per second.

A shard is described by:

- ShardId (the shard unique identifier)
- EndingHashKey + StartingHashKey
- SequenceNumberRange
- ..etc

This is what makes a shard ensure the capacity/throughput and order of the messages.

Data Record

A record is the unit of data stored in a stream. In a Kinesis record you will find:

- A sequence number
- A partition key
- Data blob (1 megabyte (MB) maximum)

Partition Key

The partition key is used to segregate and route data records to different shards of a stream. Because you can create more than one shard, the Partition Key will help you in sending the data to a specific shard.

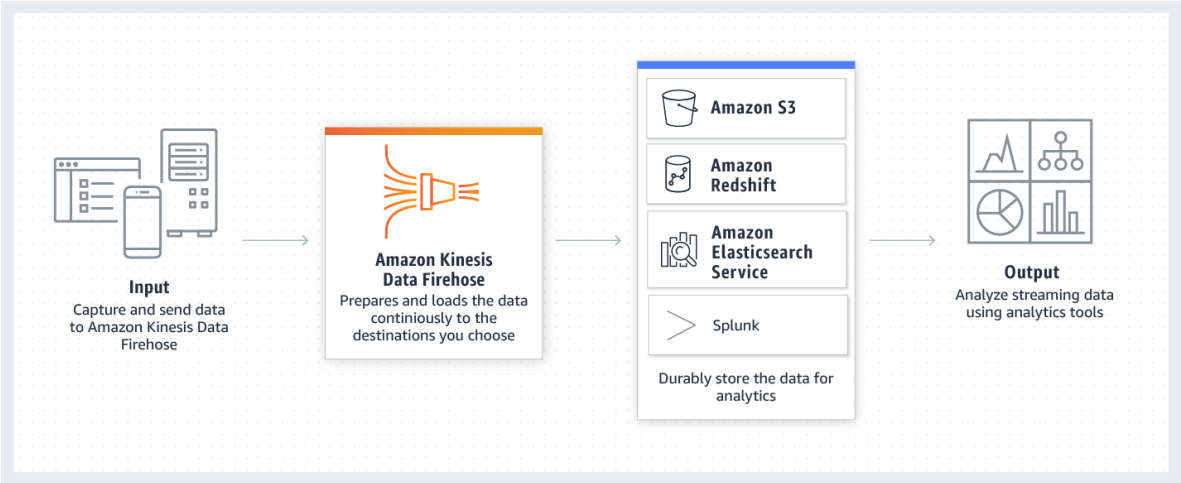
Sequence Number

A sequence number is a unique identifier for each data record. When a developer calls "PutRecord" or "PutRecords" from the data producer, a sequence number is assigned by Amazon Kinesis Streams to the data sequence. Sequence numbers increase with time, the longer the time period between "PutRecord" or "PutRecords" requests, the larger the sequence numbers become.

"PutRecord"/"PutRecords" writes a single/multiple data records into an Amazon Kinesis data stream.

Amazon Kinesis Firehose Concepts

Firehose is an Amazon managed service intended for delivering real-time streaming data to destinations: Amazon S3, Amazon Redshift, and Amazon Elasticsearch Service.



source: aws.amazon.com

Using Firehose, we can deliver the streaming data to a data store without writing and maintaining code. A simple configuration from that AWS console is sufficient.

In order to create a delivery stream, go to Firehose console, click on "Create a Delivery Stream" and give it a name (e.g., "FirehoseStream").

The next step is choosing the source of the records.

In a general case, Firehose is used to get data from a source in forms of records, then share it with a destination. The data could be transformed: Firehose can invoke a Lambda function and transform source records before delivery.

There are different delivery scenarios:

- Simply Delivering to S3 (encryption and compression are possible)
- Delivering to S3 and executing the COPY command to store the same data to Redshift
- Storing to Amazon Elastic Search
- Splunk

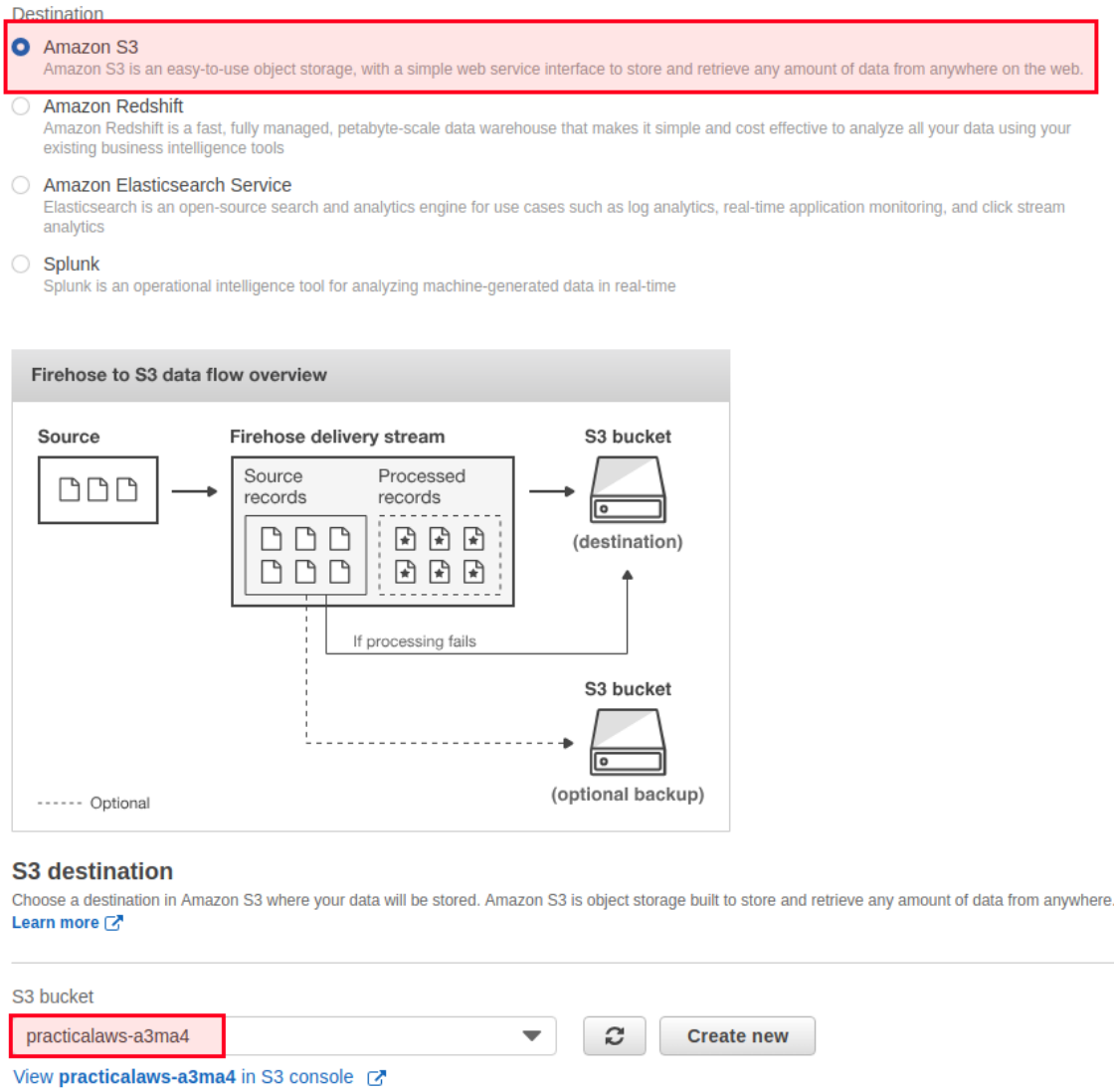
Choose the "KinesisStream", keep the other default configurations.

We do not need any transformation on the data, but you can create a Lambda function to transform source records before storing them in the destination.

Then choose S3 as a destination. You should, in this case, create a new bucket before:

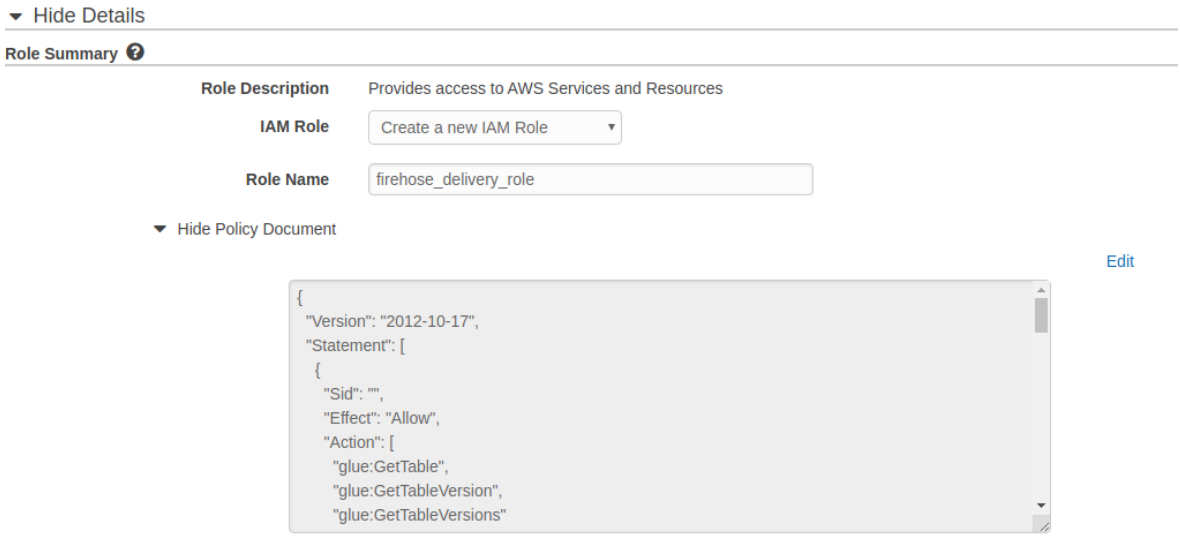
```
aws s3 mb s3://practicalaws-a3ma4
```

By default, Kinesis Data Firehose appends the prefix "YYYY/MM/DD/HH" (in UTC) to the data it delivers to Amazon S3. You can change these default settings and specify a custom prefix. We are going to keep the default settings.



Click on next, keep all the default values. We need to change neither the S3 buffer size nor the buffer interval. You can also activate the compression, encryption, and logging.

Before proceeding, make sure to create a custom role for the Firehose.



Click on next and review your delivery stream before proceeding, then create it.

Producing A Kinesis Stream Of Data

We are using a Python app that will request an API, get all the results in JSON and send them to the stream "KinesisStream" that we already created, you can create a virtual environment for this app:

```
virtualenv -p python3 kinesis_producer
```

Activate the virtual environment:

```
cd kinesis_producer/
. bin/activate
```

Don't forget to install the library boto3 that we are going to use:

```
pip install boto3
```

Now, let's create the producer `producer.py`:

You can choose whatever public API for your tests. In this example, we are going to consume a Pokemon API. The data we will get from this API will be streamed to the Kinesis cluster. To make this happen, we are going to use Boto3 Kinesis function:

```
kinesis.put_record(stream_name, data, partition_key)
```

You can test this API, so, for example, you can get information about Bulbasaur here https://pokeapi.co/api/v2/pokemon/1/ or Ivysaur here https://pokeapi.co/api/v2/pokemon/2/ ..etc.

This is the code that will read the first 10 Pokemon data and send it to Kinesis. The code is self-explanatory.

```
import boto3
import json
import requests
import time

url = "https://pokeapi.co/api/v2/pokemon/"
stream = "KinesisStream"
partition_key = "name"

kinesis_client = boto3.client('kinesis', region_name='eu-west-3')

for pokemon in range(1,10):
    route = url + str(pokemon)
```

```
    # resp type is byte
    resp = requests.get(route)

    kinesis_client.put_record(
                        StreamName=stream,
                        Data=json.dumps(resp.content.decode('utf8')),
                        PartitionKey=partition_key)
```

Make sure to change things like the `stream` and `region_name` by your own stream and region names.

We will use "name" as a partition key. Remember that "Partition key" is used to segregate and route data records to different shards of a stream. You can create more than one shard in order to send all the data with a specific partition key to a specific shard.

`resp.content.decode('utf8')` is simply the data we are sending and since it's type is "byte", we need to decode it in utf-8 when working with Python.

We are sending this data in JSON, that's why we added the `json.dumps()` function.

Before executing this code using `python producer.py`, make sure to install requests:

```
pip install requests
```

Now you can test your code, but let's first move to create the consumer.

Consuming A Kinesis Stream Of Data

The next step is creating a consumer application.

In a separate folder, create your virtual environment `virtualenv -p python3 kinesis_consumer`, install Boto3 ..etc.

To read data from Kinesis, we need the Kinesis id, the shard id, and the shard iterator.

We are going step by step, first by connecting a client to Kinesis, inspecting the stream, getting the shard ID and the shard iterator, and finally iterating through the next shard iterators.

```
import boto3
import time

stream = "KinesisStream"
partition_key = "my_key"

# Connect to Kinesis
kinesis_client = boto3.client('kinesis', region_name='eu-west-3')

# Inspecting the stream
response = kinesis_client.describe_stream(StreamName=stream)

# Get the shard ID
my_shard_id = response['StreamDescription']['Shards'][0]['ShardId']

# Create a shard iterator with the type LATEST
shard_iterator = kinesis_client.get_shard_iterator(
                                    StreamName=stream,
```

```
                                    ShardId=my_shard_id,
                                    ShardIteratorType='LATEST')


my_shard_iterator = shard_iterator['ShardIterator']

# Get the record using the shard iterators
record_response = kinesis_client.get_records(ShardIterator=my_shard_iterator)

# Iterating through the next shard iterators
# Each data record can be up to 1 MiB in size, and each shard can read up to 2
MiB per second.
# You can ensure that your calls don't exceed the maximum supported size or
throughput
# by using the Limit parameter to specify the maximum number of records that
GetRecords can return.
while 'NextShardIterator' in record_response:
    record_response = kinesis_client.get_records(

 ShardIterator=record_response['NextShardIterator'],
                                    Limit=2)

    # printing the result to the screen
    print(record_response)

    # sleeping for 3 seconds
    # time.sleep(3)
```

You can also use one of the following shard iterator types:

- AT_SEQUENCE_NUMBER
- AFTER_SEQUENCE_NUMBER
- AT_TIMESTAMP
- TRIM_HORIZON

AWS documentation describes each type as the following:

The following are the valid Amazon Kinesis shard iterator types:

- **AT_SEQUENCE_NUMBER** - Start reading from the position denoted by a specific sequence number, provided in the value `StartingSequenceNumber`.
- **AFTER_SEQUENCE_NUMBER** - Start reading right after the position denoted by a specific sequence number, provided in the value `StartingSequenceNumber`.
- **AT_TIMESTAMP** - Start reading from the position denoted by a specific time stamp, provided in the value `Timestamp`.
- **TRIM_HORIZON** - Start reading at the last untrimmed record in the shard in the system, which is the oldest data record in the shard.
- **LATEST** - Start reading just after the most recent record in the shard, so that you always read the most recent data in the shard.

Now, if you start the consumer app without producing data at the same time, you will see empty records scrolling through the screen continuously.

```
{
   'Records':[
   ],
   'NextShardIterator':'[...]',
```

```
   'MillisBehindLatest':0,
   'ResponseMetadata':{
      'RequestId':'[...]',
      'HTTPStatusCode':200,
      'HTTPHeaders':{
           [...]
         'content-type':'application/x-amz-json-1.1',
         'content-length':'284'
      },
      'RetryAttempts':0
   }
}
```

Streaming the Data and Consuming it

Now that we wrote the producer and the consumer code, we should simply execute both of them at the same time.

For simplicity reasons I am going to produce the data using Unix watch command which will produce new data every 2 seconds:

```
watch "python producer"
```

You can also change the producer code and create an infinite loop:

```
import boto3
import json
import requests
import time

url = "https://pokeapi.co/api/v2/pokemon/"
stream = "KinesisStream"
partition_key = "name"

kinesis_client = boto3.client('kinesis', region_name='eu-west-3')


while 1:
    for pokemon in range(1,10):
        route = url + str(pokemon)

        # resp type is byte
        resp = requests.get(route)

        kinesis_client.put_record(
                            StreamName=stream,
                            Data=json.dumps(resp.content.decode('utf8')),
                            PartitionKey=partition_key)
```

In another terminal, start the consumer app:

```
python consumer.py
```

You will be able to see the data caught by the consumer as soon as it is sent by the producer.

Amazon Kinesis Streams Limits

Streams have following limits (These limits may be subject to change by Amazon in the future.):

- Shards are limited to 50 for the following regions only: US East, US West, EU Ireland, while all other regions have a default shard limit of 25.
- Data records are accessible for a default of 24 hours (the retention period). It could be configured up to 168 hours = 7 days.
- A data blob maximum size is 1 MB.
- Functions like CreateStream, DeleteStream, ListStreams, MergeShards, SplitShard, GetShardIterator can provide up to 5 transactions per second. While DescribeStream can provide up to 10 transactions per second.
- A call to GetRecords can retrieve 10 MB of data: 5 transactions per second for reads, up to a maximum total data read rate of 2 MB per second.
- PutRecord and PutRecords have a limit of 1,000 records per second / 1 MB per second.
- GetShardIterator times out after 5 minutes of inactivity

Retrieving Firehose Data

So, where is Firehose in all of this?

We already configured Firehose to put the generated data of the Kinesis stream in an S3 bucket.

Now, after testing how Kinesis works using the Pokemon API data, if you go to the bucket you created when configuring the AWS Firehose, you will find the same data stored there. This is what Firehose stored on the S3 bucket:

```
aws s3 ls s3://practicalaws-a3ma4/<year>/<month>/<day>/<hour>/
```

By default, Firehose store files using the format YYYY/MM/DD. This is an example:

```
aws s3 ls s3://practicalaws-a3ma4/2020/01/21/10/
```

Output:

```
FirehoseStream-1-2020-01-21-10-33-51-d97f319d-dfd0-4758-b2fc-8f8646b9c409
```

You can download and check this file using:

```
aws s3 cp s3://practicalaws-a3ma4/2020/01/21/10/FirehoseStream-1-2020-01-21-10-
33-51-d97f319d-dfd0-4758-b2fc-8f8646b9c409 <destination>
```

Kinesis vs. SQS

We used SQS, and at first sight, it seems that both Kinesis and SQS do the same things. So, what are the differences?

The main difference is the services that you can use with Kinesis and not SQS. For example, if you want to transform data before storing it using a Lambda function, you should use Kinesis.

Firehose does not integrate with SQS, so if you need to set up a data delivery infrastructure to S3 or other services, you should go for Kinesis.

SQS can not integrate with Redshift, which is a scalable data warehouse used to store and analyze data and can extend to data lakes (e.g., S3). Unlike SQS, a user can use Kinesis to replay data, while it is not possible/very limited with SQS. Kinesis can persist data for up to 7 days on the stream.

SQS, for instance, is better at handling jobs that can fail or jobs that you can re-queue. SQS offers a dead letter queue that handles this type of data, while Kinesis does not offer such a built-in feature.

On the other hand, Kinesis can also allow up to 5 consumers on a stream simultaneously while SQS does not offer this feature.

The number of integration to store and analyze data that we can use with Kinesis make it optimize for different use cases.

In other words, even if we functionally can do almost the same things using Kinesis and SQS, but at a scale and in real-world use cases, things may differ. Kinesis is designed to ingest and digest large volumes of streaming data, while SQS is designed as a message broker.

Amazon defines Kinesis and SQS as follows:

Amazon Kinesis Data Streams enables real-time processing of streaming big data. It provides an ordering of records, as well as the ability to read and/or replay records in the same order to multiple Amazon Kinesis Applications. The Amazon Kinesis Client Library (KCL) delivers all records for a given partition key to the same record processor, making it easier to build multiple applications reading from the same Amazon Kinesis data stream (for example, to perform counting, aggregation, and filtering).

Amazon Simple Queue Service (Amazon SQS) offers a reliable, highly scalable, hosted queue for storing messages as they travel between computers. Amazon SQS lets you easily move data between distributed application components and helps you build applications in which messages are processed independently (with message-level ack/fail semantics), such as automated workflows.

It is also wise to look at the limits of each service beforehand. The costs should be something to consider when choosing one of these services. Using Kinesis as a message borker will certainly make you spend more money than necessary.

This is what Amazon recommends:

We recommend Amazon Kinesis Data Streams for use cases with requirements that are similar to the following:

- Routing related records to the same record processor (as in streaming MapReduce). For example, counting and aggregation are simpler when all records for a given key are routed to the same record processor.
- Ordering of records. For example, you want to transfer log data from the application host to the processing/archival host while maintaining the order of log statements.
- The ability for multiple applications to consume the same stream concurrently. For example, you have one application that updates a real-time dashboard and another that archives data to Amazon Redshift. You want both applications to consume data from the same stream concurrently and independently.
- Ability to consume records in the same order a few hours later. For example, you have a billing application and an audit application that runs a few hours behind the billing

application. Because Amazon Kinesis Data Streams stores data for up to 7 days, you can run the audit application up to 7 days behind the billing application.

We recommend Amazon SQS for use cases with requirements that are similar to the following:

- Messaging semantics (such as message-level ack/fail) and visibility timeout. For example, you have a queue of work items and want to track the successful completion of each item independently. Amazon SQS tracks the ack/fail, so the application does not have to maintain a persistent checkpoint/cursor. Amazon SQS will delete "acked" messages and redeliver failed messages after a configured visibility timeout.
- Individual message delay. For example, you have a job queue and need to schedule individual jobs with a delay. With Amazon SQS, you can configure individual messages to have a delay of up to 15 minutes.
- Dynamically increasing concurrency/throughput at read time. For example, you have a work queue and want to add more readers until the backlog is cleared. With Amazon Kinesis Data Streams, you can scale up to a sufficient number of shards (note, however, that you'll need to provision enough shards ahead of time).
- Leveraging Amazon SQS's ability to scale transparently. For example, you buffer requests and the load changes as a result of occasional load spikes or the natural growth of your business. Because each buffered request can be processed independently, Amazon SQS can scale transparently to handle the load without any provisioning instructions from you.

Lesson 23 - AWS Cheat Sheet

AWS CLI

Installation

```
pip install awscli
```

Help

```
aws help
```

Configuration

```
aws configure
```

CLI config files

```
~/.aws/credentials
~/.aws/config
```

Output Formats

The AWS CLI supports four output formats:

- **json** – The output is formatted as a JSON string.
- **yaml** – The output is formatted as a YAML string. (Available in the AWS CLI version 2 only.)
- **text** – The output is formatted as multiple lines of tab-separated string values. This can be useful to pass the output to a text processor, like grep, sed, or awk.
- **table** – The output is formatted as a table using the characters +|- to form the cell borders. It typically presents the information in a "human-friendly" format that is much easier to read than the others but not as programmatically useful.

In the following commands, you can use one of the above values instead of `<output>`.

Volumes

Describing volumes

```
aws ec2 describe-volumes
```

Describing filtered volumes:

```
aws ec2 describe-volumes --filters  Name=status,Values=creating | available |
in-use | deleting | deleted | error
```

e.g, describing all deleted volumes:

```
aws ec2 describe-volumes --filters  Name=status,Values=deleted
```

Filters can be applied to the attachment status:

```
aws ec2 describe-volumes --filters  Name=attachment.status,Values=attaching |
attached | detaching | detached
```

e.g: describing all volumes with the status “attaching”:

```
aws ec2 describe-volumes --filters  Name=attachment.status,Values=attaching
```

This is the generic form. Use –profile `<your_profile_name>`, if you have multiple AWS profiles or accounts.

```
aws ec2 describe-volumes --filters Name:'tag:Name',Values: ['some_values'] --
profile <your_profile_name>
```

Describing volumes using a different aws user profile

```
aws ec2 describe-volumes --filters  Name=status,Values=in-use  --profile
<your_profile_name>
```

Listing Available Volumes IDs

```
aws ec2 describe-volumes --filters  Name=status,Values=available |grep
VolumeId|awk '{print $2}' | tr '\n|,|"' ' '
```

With “profile”:

```
aws ec2 describe-volumes --filters  Name=status,Values=available  --profile
<your_profile_name>|grep VolumeId|awk '{print $2}' | tr '\n|,|"' ' '
```

Deleting a Volume

```
aws ec2 delete-volume --region <region> --volume-id <volume_id>
```

Deleting Unused Volumes.. Think Before You Type :-)

```
for x in $(aws ec2 describe-volumes --filters  Name=status,Values=available  --
profile <your_profile_name>|grep VolumeId|awk '{print $2}' | tr ',|"' ' '); do
aws ec2 delete-volume --region <region> --volume-id $x; done
```

With “profile”:

```
for x in $(aws ec2 describe-volumes --filters  Name=status,Values=available  --
profile <your_profile_name>|grep VolumeId|awk '{print $2}' | tr ',|"' ' '); do
aws ec2 delete-volume --region <region> --volume-id $x --profile
<your_profile_name>; done
```

Creating a Snapshot

```
aws ec2 create-snapshot --volume-id <vol-id>
aws ec2 create-snapshot --volume-id <vol-id> --description "snapshot-$(date
+'%Y-%m-%d_%H-%M-%S')"
```

Creating an Image (AMI)

```
aws ec2 create-image --instance-id <instance_id> --name "image-$(date +'%Y-%m-
%d_%H-%M-%S')" --description "image-$(date +'%Y-%m-%d_%H-%M-%S')"
```

Creating AMI Without Rebooting the Machine

```
aws ec2 create-image --instance-id <instance_id> --name "image-$(date +'%Y-%m-
%d_%H-%M-%S')" --description "image-$(date +'%Y-%m-%d_%H-%M-%S')" --no-reboot
```

You are free to change the AMI name `image-$(date +'%Y-%m-%d_%H-%M-%S')` to the name of your choice.

AMIs

Listing AMI(s)

```
aws ec2 describe-images
```

Describing AMI(s)

```
aws ec2 describe-images --image-ids <image_id> --profile <profile> --region
<region>
```

e.g:

```
aws ec2 describe-images --image-ids ami-e24dfa9f --profile terraform --region
eu-west-3
```

Listing Amazon AMIs

```
aws ec2 describe-images --owners amazon
```

Using Filters

e.g.: Describing Windows AMIs that are backed by Amazon EBS.

```
aws ec2 describe-images --filters "Name=platform,Values=windows" "Name=root-device-type,Values=ebs"
```

e.g.: Describing Ubuntu AMIs

```
aws ec2 describe-images --filters "Name=name,Values=ubuntu*"
```

Lambda

Using AWS Lambda with Scheduled Events

```
sid=Sid$(date +%Y%m%d%H%M%S); aws lambda add-permission --statement-id $sid --action 'lambda:InvokeFunction' --principal events.amazonaws.com --source-arn arn:aws:events:<region>:<arn>:rule/AWSLambdaBasicExecutionRole --function-name function:<awsents> --region <region>
```

IAM / User and Groups Management

Get Current Identity

```
aws sts get-caller-identity
```

or

```
aws iam get-user
```

Get Another User Information

```
aws iam get-user --user-name <username>
```

List Users

```
aws iam list-users
```

List Policies

```
aws iam list-policies
```

List Groups

```
aws iam list-groups
```

Get Users in a Group

```
aws iam get-group --group-name <group_name>
```

Describing a Policy

```
aws iam get-policy --policy-arn arn:aws:iam::aws:policy/<policy_name>
```

Create a User

```
aws iam create-user --user-name <username>
```

Password Management

List Global Password Policies

```
aws iam get-account-password-policy
```

Delete Password Policy

```
aws iam delete-account-password-policy
```

Key Management

List Access Keys

```
aws iam list-access-keys
```

List Keys

```
aws iam list-access-keys
```

List the Access Key IDs for an IAM User

```
aws iam list-access-keys --user-name <user_name>
```

List the SSH Public Keys for a User

```
aws iam list-ssh-public-keys --user-name <user_name>
```

List Access Keys

```
aws iam list-access-keys
```

List Access Keys of a User

```
aws iam list-access-keys --user-name <username>
```

Create an Access Key

```
aws iam create-access-key --user-name <username> --output <output>
```

When an Access Key was Last Used

```
aws iam get-access-key-last-used --access-key-id <access_key_id>
```

Deactivate an Access Key

```
aws iam update-access-key --access-key-id <access_key_id> --status Inactive --
user-name <username>
```

Delete an Access Key

```
aws iam delete-access-key --access-key-id <access_key_id> --user-name <username>
```

List Keypairs

```
aws ec2 describe-key-pairs
```

Create a Keypair

```
aws ec2 create-key-pair --key-name <key_name> --output <output>
```

Import an Existing Keypair

```
aws ec2 import-key-pair --key-name <key_name> --public-key-material
file://</path/to/id_rsa.pub>
```

Delete a Keypair

```
aws ec2 delete-key-pair --key-name <key_name>
```

S3 API

Listing Buckets

```
aws s3api list-buckets
```

Or

```
aws s3 ls
```

e.g

```
aws s3 ls --profile eon01
```

Listing Only Bucket Names

```
aws s3api list-buckets --query 'Buckets[].Name'
```

Getting a Bucket Region

```
aws s3api get-bucket-location --bucket <bucket_name>
```

e.g

```
aws s3api get-bucket-location --bucket practicalaws.com
```

Listing the Content of a Bucket

```
aws s3 ls s3://<bucket_name> --region <region>
```

e.g

```
aws s3 ls s3://practicalaws.com

aws s3 ls s3://practicalaws.com --region eu-west-1

aws s3 ls s3://practicalaws.com --region eu-west-1 --profile eon01
```

Syncing a Local Folder with a Bucket

```
aws s3 sync <local_path> s3://<bucket_name>
```

e.g

```
aws s3 sync . s3://practicalaws.com --region eu-west-1
```

Copying Files

```
aws s3 cp <file_name> s3://<bucket_name>
```

Or:

```
aws s3 cp <file_name> s3://<bucket_name>/<folder_name>/
```

To copy all files from a filder, look at “Copying Folders”. Or use the following example, where I copy the content of the folder “images (contains images) in the remote folder “images”.

```
cd images
aws s3 cp . s3://saltstackfordevops.com/images --recursive --region us-east-2
```

Copying Folders

```
aws s3 cp <folder_name>/ s3://<bucket_name>/ --recursive
```

To exclude files:

```
aws s3 cp <folder_name>/ s3://<bucket_name>/ --recursive --exclude "
<file_name_or_a_wildcard>"
```

e.g., To only include a certain type of files (PNG) and exclude others (JPG)

```
aws s3 cp practicalaws.com/ s3://practicalaws-backup/  --recursive --exclude
"*.jpg"  --include "*.png"
```

e.g., To exclude a folder

```
aws s3 cp practicalaws.com/ s3://practicalaws-backup/ --recursive --exclude
".git/*"
```

Removing a File from a Bucket

```
aws s3 rm s3://<bucket_name>/<object_name>
```

e.g

```
aws s3 rm s3://practicalaws.com/temp.txt
```

Deleting a Bucket

```
aws s3 rb s3://<bucket_name> --force
```

If the bucket is not empty, use –force.

e.g

```
aws s3 rb s3://practicalaws.com --force
```

Emptying a Bucket

```
aws s3 rm s3://<bucket_name>/<key_name> --recursive
```

e.g

In order to remove tempfiles/file1.txt and tempfiles/file2.txt from practicalaws.com bucket, use:

```
aws s3 rm s3://practicalaws.com/tempfiles --recursive
```

Remove all objects using:

```
aws s3 rm s3://practicalaws.com/tempfiles
```

VPC

Creating A VPC

```
aws ec2 create-vpc --cidr-block <cidr_block> --regiosn <region>
```

e.g

```
aws ec2 create-vpc --cidr-block 10.0.0.0/16 --region eu-west-1
```

Allowing DNS hostnames

```
aws ec2 modify-vpc-attribute --vpc-id <vpc_id> --enable-dns-hostnames "
{\"Value\":true}" --region <region>
```

Subnets

Creating A Subnet

```
aws ec2 create-subnet --vpc-id <vpc_id> --cidr-block <cidr_block> --
availability-zone <availability_zone> --region <region>
```

Auto Assigning Public IPs To Instances In A Public Subnet

```
aws ec2 modify-subnet-attribute --subnet-id <subnet_id> --map-public-ip-on-
launch --region <region>
```

Internet Gateway

Creating An IGW

```
aws ec2 create-internet-gateway --region <region>
```

Attaching An IGW to A VPC

```
aws ec2 attach-internet-gateway --internet-gateway-id <igw_id> --vpc-id <vpc_id>
--region <region>
```

NAT

Setting Up A NAT Gateway

Allocate Elastic IP

```
aws ec2 allocate-address --domain vpc --region <region>
```

then use the AllocationId to create the NAT Gateway for the public zone in

```
aws ec2 create-nat-gateway --subnet-id <subnet_id> --allocation-id
<allocation_id> --region <region>
```

Route Tables

Creating A Public Route Table

Create the Route Table:

```
aws ec2 create-route-table --vpc-id <vpc_id> --region <region>
```

then create a route for an Internet Gateway.

Now, use the outputted Route Table ID:

```
aws ec2 create-route --route-table-id <route_table_id> --destination-cidr-block
0.0.0.0/0 --gateway-id <igw_id> --region <region>
```

Finally, associate the public subnet with the Route Table

```
aws ec2 associate-route-table --route-table-id <route_table_id> --subnet-id
<subnet_id> --region <region>
```

Creating A Private Route Tables

Create the Route Table

```
aws ec2 create-route-table --vpc-id <vpc_id> --region <region>
```

then create a route that points to a NAT Gateway

```
aws ec2 create-route --route-table-id <route_table_id> --destination-cidr-block
0.0.0.0/0 --nat-gateway-id <net_gateway_id> --region <region>
```

Finally, associate the subnet

```
aws ec2 associate-route-table --route-table-id <route_table_id> --subnet-id
<subnet_id> --region <region>
```

Security Groups

List Security Groups

```
aws ec2 describe-security-groups
```

Create a Security Group

```
aws ec2 create-security-group --vpc-id <vpc_id> --group-name
<security_group_name> --description "<description>"
```

List a Securty Group

```
aws ec2 describe-security-groups --group-id <security_group_id>
```

Open Port/Protocol For Your IP

```
aws ec2 authorize-security-group-ingress \
    --group-id <security_group_id> \
    --protocol <protocol_name> \
    --port <port_number> \
    --cidr $my_ip/24
```

Open Port/Protocol to CIDR

```
aws ec2 authorize-security-group-ingress \
    --group-id <security_group_id> \
    --protocol <protocol_name> \
    --port <port_number> \
    --cidr <CIDR>
```

Revoke a Rule

```
aws ec2 revoke-security-group-ingress \
    --group-id <security_group_id> \
    --protocol <protocol_name> \
    --port <port_number> \
    --cidr <CIDR>
```

Delete a Security Group

```
aws ec2 delete-security-group --group-id <security_group_id>
```

CloudFront

Listing Distributions

In some cases, you need to set up this first:

```
aws configure set preview.cloudfront true
```

Then:

```
aws cloudfront list-distributions
```

Invalidating Files From a Distribution

To invalidate index and error HTML files from the distribution with the ID Z2W2LX9VBMAPRX:

```
aws cloudfront create-invalidation --distribution-id Z2W2LX9VBMAPRX  --paths
/index.html /error.html
```

To invalidate everything in the distribution:

```
aws cloudfront create-invalidation --distribution-id Z2W2LX9VBMAPRX  --paths
'/*'
```