Bootcamp
Data Engineering

Day02
AWS

Bootcamp Data Engineering

Day02 - AWS - Flask - Terraform

Today, you will learn how to use Amazon Web Services. It has become the most popular cloud service provider in the world followed by Google Cloud Platform and Microsoft's Azure.

Amazon Web Services started in 2005 and it now delivers nearly 2000 services. Due to the large number of services and the maturity of AWS, it is a better option to start learning cloud computing.

If you never heard about the Cloud before, do not worry! You will learn step by step what the Cloud is and how to use it.

Notions of the day

Create an AWS account, set up a billing alarm.

The day will be divided into two parts. In the first one, you will learn to use a tool called Terraform which will allow you to deploy/destruct an AWS infrastructure. In the second part of the day, you will learn to use a software development kit (SDK) which will allow you to use Python in order to interact with AWS.

General rules

- The exercises are ordered from the easiest to the hardest.
- Your exercises are going to be evaluated by someone else, so make sure that your variable names and function names are appropriate and civil.
- Your manual is the internet.
- You can also ask any question in the dedicated channel in Slack: **42ai slack**.
- If you find any issue or mistakes in the subject please create an issue on our dedicated repository on Github: **Github issues**.

Foreword

Amazon Web Services started in 2005 and it now delivers nearly 2000 services. Due to the large number of services and the maturity of AWS, it is a better option to start learning cloud computing.

If you never heard about the Cloud before, do not worry! You will learn step by step what the Cloud is and how to use it.

Helper

- Your best friends for the day: **AWS documentation** and **Terraform documentation**.

Exercise 00 - AWS setup.

Exercise 01 - Simple Storage Service (S3).

Exercise 02 - Elastic cloud compute (EC2).

Exercise 03 - Flask API - List.

Exercise 04 - Flask API - Upload.

Exercise 05 - Flask API - Download.

Exercise 06 - Virtual Private Cloud.

Exercise 07 - Subnets.

Exercise 08 - IGW - Route table.

Exercise 09 - EC2 - Security groups.

Exercise 10 - Autoscaling group.

Exercise 11 - Load balancer.

Exercise 00 - AWS setup.

Turn-in directory:	ex00
Files to turn in:	
Forbidden function:	None
Remarks:	n/a

What is the Cloud?

Cloud computing is the on-demand delivery of IT resources and applications via the Internet with pay-as-you-go pricing. In fact, a cloud server is located in a data center that could be anywhere in the world.

Whether you run applications that share photos to millions of mobile users or deliver services that support the critical operations of your business, the cloud provides rapid access to flexible and low-cost IT resources. With cloud computing, you don't need to make large up-front investments in hardware and spend a lot of time managing that hardware. Instead, you can provision exactly the right type and size of computing resources you need to power your newest bright idea or operate your IT department. With cloud computing, you can access as many resources as you need, almost instantly, and only pay for what you use.

In its simplest form, cloud computing provides an easy way to access servers, storage, databases, and a broad set of application services over the Internet. Cloud computing providers such as AWS own and maintain the network-connected hardware required for these application services, while you provision and use what you need for your workloads.

As seen previously, Cloud computing provides some real benefits :

- **Variable expense**: You don't need to invest in huge data centers you may not use at full capacity. You pay for how much you consume!
- **Available in minutes**: New IT resources can be accessed within minutes.
- **Economies of scale**: A large number of users enables Cloud providers to achieve higher economies of scale translating at lower prices.
- **Global in minutes**: Cloud architectures can be deployed really easily all around the world.

Deployments using the cloud can be `all-in-cloud-based` (the entire infrastructure is in the cloud) or `hybrid` (using on-premise and cloud).

AWS global infrastructure

Amazon Web Services (AWS) is a cloud service provider, also known as infrastructure-as-a-service (`IaaS`). AWS is the clear market leader in this domain and offers much more services compared to its competitors.

AWS has some interesting properties such as:

- **High availability** : Any file can be accessed from anywhere
- **Fault tolerance**: In case an AWS server fails, you can still retrieve the files (the fault tolerance is due to redundancy).
- **Scalability**: Possibility to add more servers when needed.
- **Elasticity**: Possibility to grow or shrink infrastructure.

AWS provides a highly available technology infrastructure platform with multiple locations worldwide. These locations are composed of `regions` and `availability zones`.

Each region represents a unique geographic area. Each region contains multiple, isolated locations known as availability zones. An availability zone is a physical data center geographically separated from other availability zones (redundant power, networking, and connectivity).

You can achieve high availability by deploying your application across multiple availability zones.

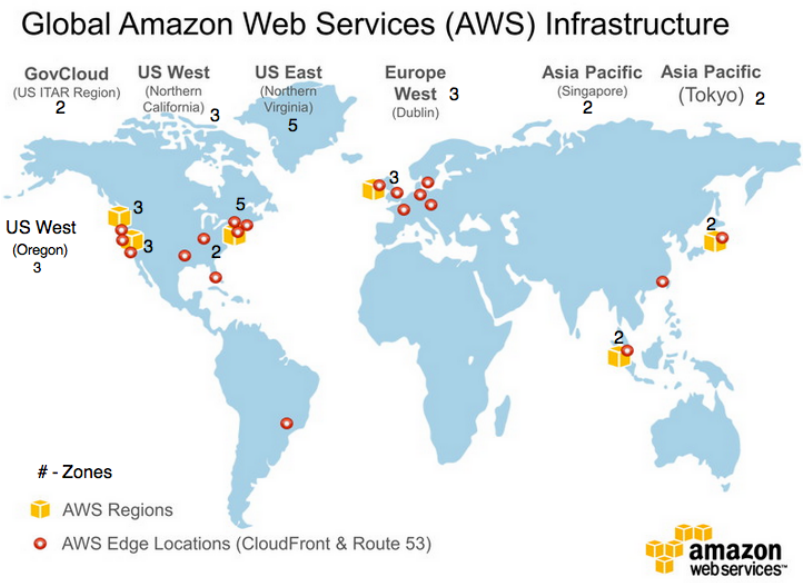

Figure 1: AWS regions

The `edge locations` you see on the picture are endpoints for AWS which are used for caching content (performance optimization mechanism in which data is delivered from the closest servers for optimal application performance). Typically consists of CloudFront (Amazon's content delivery network (CDN)).

AWS account registration

When you create a new account on AWS, you get an AWS Free Tier. The latter allows you to use some of the AWS resources for free each month for one year.

In order to create an AWS account, you must visit the following address.

Then, click on the "Create an account" button and fill the following form.

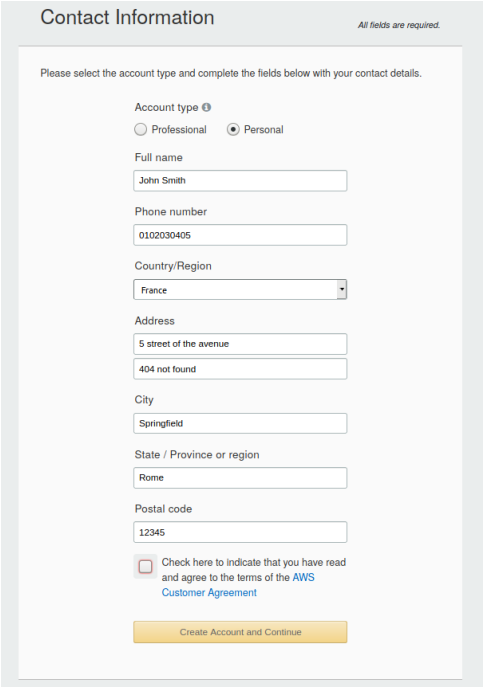

Figure 2: Account creation 1

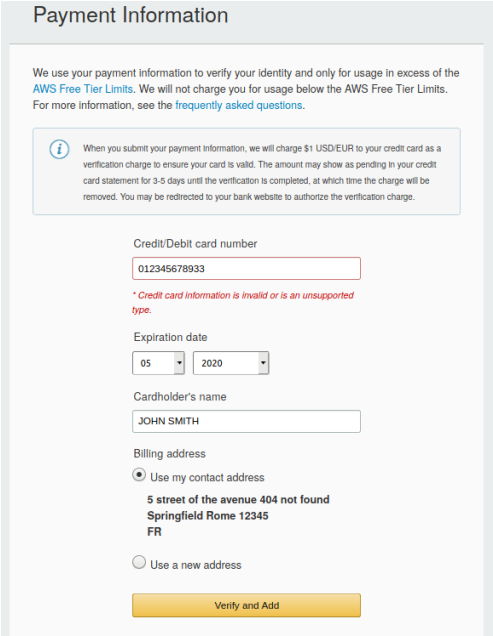

Figure 3: Account creation 2

Don't worry! Even if you enter your card number, this day is not going to cost you anything. First, indeed, AWS as a free tier usage that allows you to use a small amount of AWS resources for free. This will be sufficient enough for what you are going to do today. By the end of the day, you will have to entirely destroy your infrastructure (don't keep things running) !!!

In the next form, choose the `basic plan`, which is free.

To connect to AWS, use your email and password. You will then access to your root account. The root account **MUST NOT** be used directly when using AWS resources for security purposes. You will later create IAM users to use AWS resources in a safer way.

Discover AWS console

The console allows searching for specific services. By default, services are sorted by groups but they can also be sorted alphabetically.

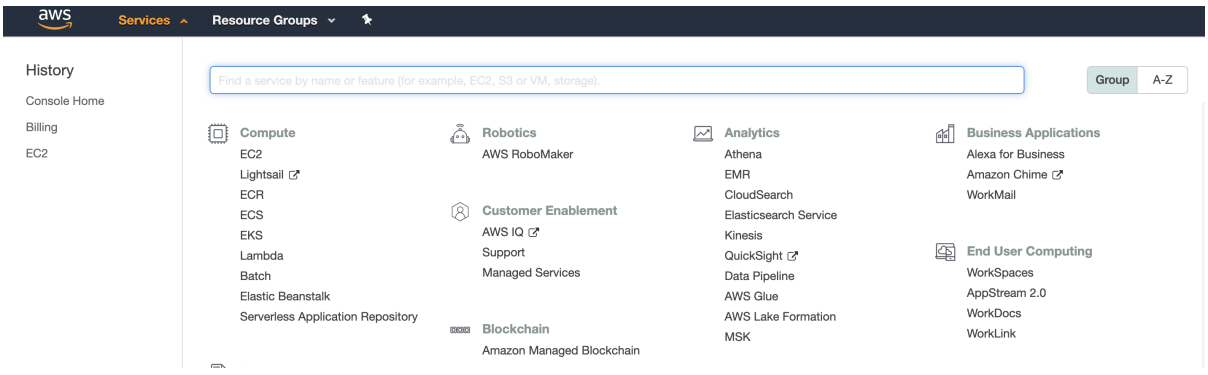

Figure 4: AWS console

You will notice there is a link for a resource group. Resource groups allow you to take a collection of AWS resources and assign them a tag (a label) so they can be managed as a group.

The pushpin is for one-click navigation. It allows you to create shortcuts for the most commonly used resources.

The alarm icon enables you to see all system alerts. Click it to see more details about the issues and their current status.

Then, you have account information and the selected AWS regions you are working on (note that all resources are not available for every region).

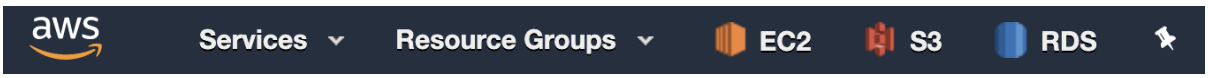

Figure 5: AWS pushpin

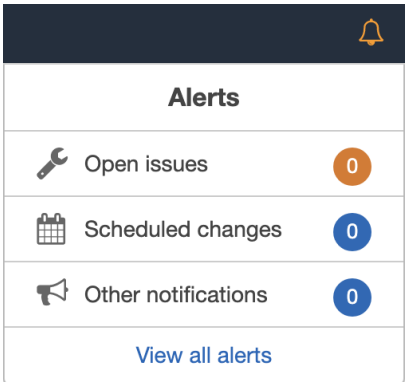

Figure 6: AWS alerts

The support center allows us to create cases when we encounter problems (we can see the support plan here, which will influence AWS response time).

Setting up a billing alarm

In order to avoid any big mistake (forget to destroy your infrastructure and it cost you some money), we are going to set an alarm to tell us if we have a cost superior to 1$.

We first type `billing` into the AWS search services bar.

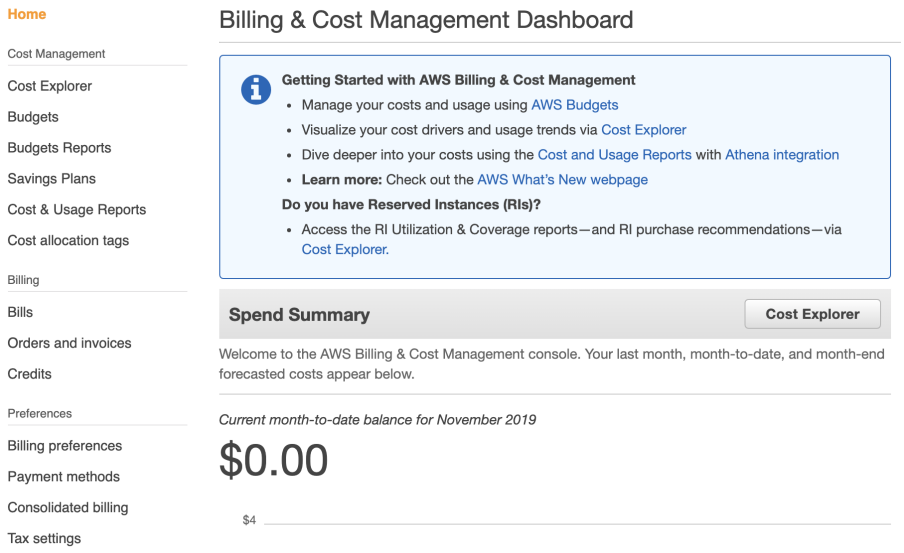

Figure 7: billing section

Then we can go in the section `Billing preferences` section and validate `Receive Free Tier Usage Alerts` (we have to enter an email). We can also set `Receive Billing Alerts` then we have to save preferences. We can now receive billing alerts !

Now we need to go CloudWatch to configure a billing alert. Cloudwatch is the AWS monitoring system used to track performances of AWS resources.

Go to the `CloudWatch` section and then the `Alarm/Billing` subsection. Click `Create alarm`.

You have to `select metric`. Two types of metrics exist: `by service` or `Total estimated charges`. Select `Total estimated charges` and `USD` as currency. The threshold you will need is `static` and `greater or equal`

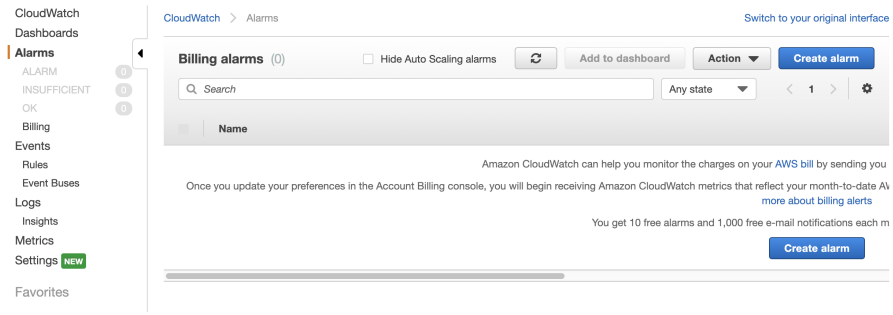

Figure 8: alarm

to 1 $.

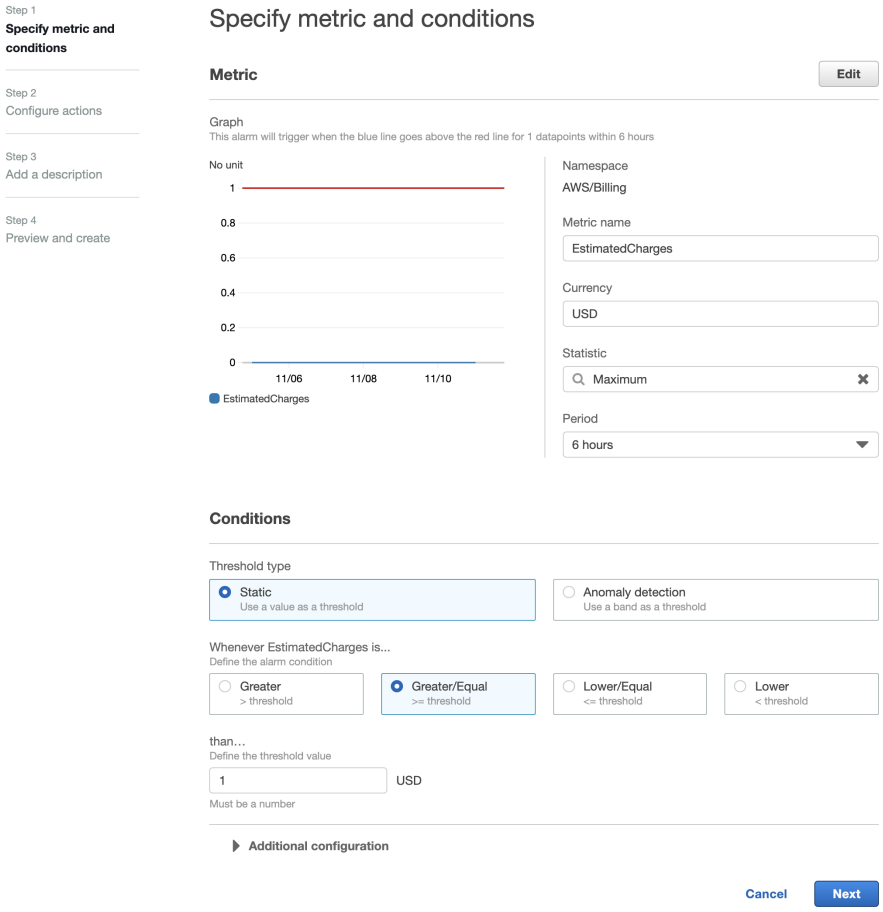

Figure 9: alarm condition

On the configure actions page, keep `in alarm`. For the notification, use SNS (Simple Notification Service). You are going to `create a new topic` with the name "BillingAlarm" and enter your email address. You can then create the SNS topic.

For the SNS topic to work, we need to confirm our email address (confirmation in the AWS email).

Choose `select an existing SNS topic` with "BillingAlarm".

Then, add the name and the description of the alarm. Your alarm is finally created!

Create an IAM user

AWS recommends not to use root accounts for day to day administrative tasks. Thus, you are going to create a new IAM user and attach the administrative access policy to it. As a best practice, AWS recommends granting users the minimum access to resources to accomplish their day to day tasks.

First, go to the `IAM` service (IAM stands for Identity and access management). To create a user, go to the dashboard into the `users` section, then `add user`.

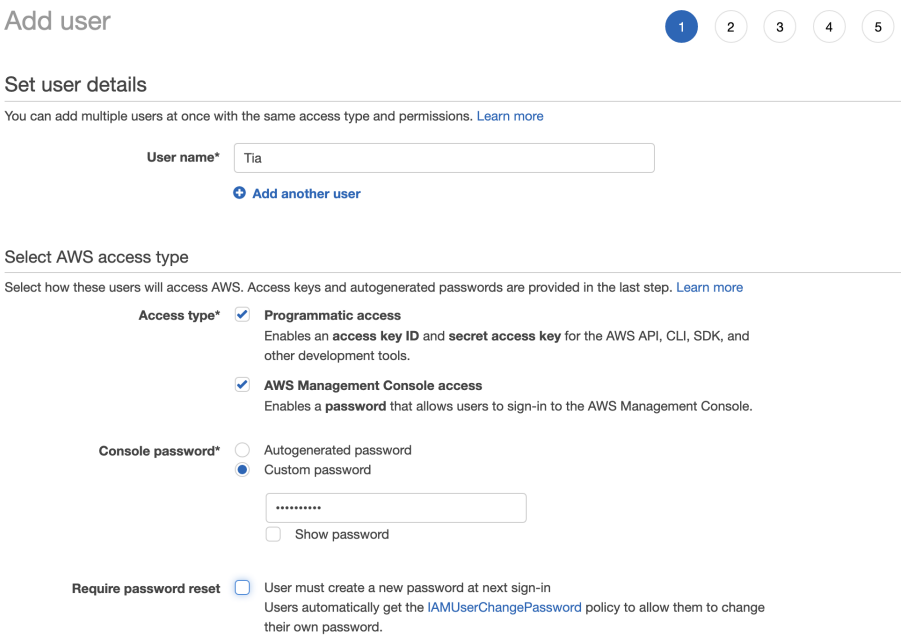

Figure 10: add user

Enter the name of the new user and select an access type. Select both `Programmatic access` and `AWS Management Console access`. Accesses will be seen later. Then, select an automatically generated password or custom password (here, a custom password is used). You are able to force the user to change of their password when they log in.

The second step is to set permissions for the new user. For the first one, attach an `AdministratorAccess` policy (which grants the user full access to all AWS services).

Besides, a tag can be added to the user (optional). The last step is to review all the options set for the user. The user is finally created!

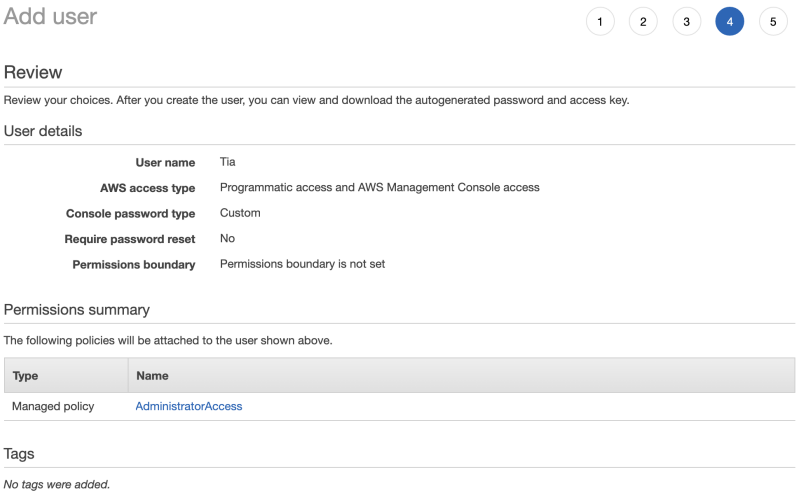

Figure 11: user review

At the end, you will see the following screen. The access key is really important as it will be used in the next exercise. Save the CSV file otherwise, you will never ever have access to the secret key again.

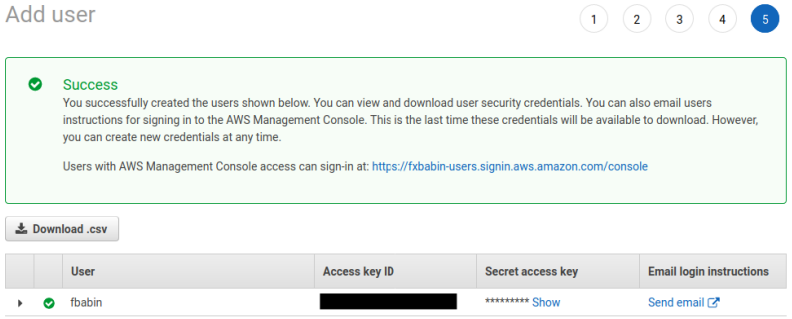

Figure 12: access key

By going in IAM, Users, and clicking your newly created user, the security credential panel will show you the sign-in link for the user. Use it to log to the console more safely.

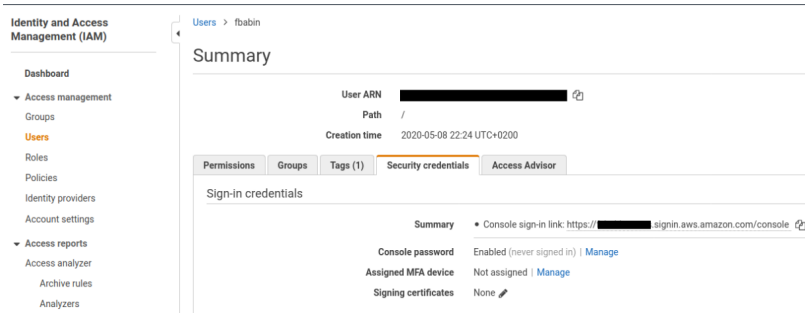

Figure 13: sign-in link

Exercise 01 - Simple Storage Service (S3).

Turn-in directory:	ex01
Files to turn in:	aws_presign_url.sh
Forbidden function:	None
Remarks:	n/a

AWS-CLI

We are going to use the AWS command-line interface. The first thing we need to do is install it.

```
curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o "AWSCLIV2.pkg"
sudo installer -pkg AWSCLIV2.pkg -target /
```

You should be able to run `aws --version` now.

We can setup our AWS account for the CLI with the command `aws configure`. You will need to enter :

- access key : in your `credentials.csv` file
- secret access key : in your `credentials.csv` file
- region : `eu-west-1` (Ireland)
- default output format : `None`

The AWS CLI is now ready!

S3 bucket creation

Amazon S3 provides developers and IT teams with secure, durable, and highly-scalable cloud storage. Amazon S3 is easy-to-use object storage with a simple web service interface that you can use to store and retrieve any amount of data from anywhere on the web. It is characterized by its :

- durability : it is designed to deliver 99.99% durability of objects over a given year.
- availability : S3 buckets are saved in at least 3 availability zones.
- scalability : bucket sizes will shrink/expand depending on the load.

A bucket is a container (web folder) for objects (files) stored in Amazon S3. Every Amazon S3 object is contained in a bucket. Buckets form the top-level namespace for Amazon S3, and bucket names are global. This means that your bucket names must be unique globally (across all AWS accounts).The reason for that is when we create a bucket, it is going to have a web address (ex : `https://s3-eu-west-1.amazonaws.com/example`).

Even though the namespace for Amazon S3 buckets is global, each Amazon S3 bucket is created in a specific region that you choose. This lets you control where your data is stored.

With your free usage you can store up to 5 Gb of data!

Exercise

In this exercise, you will learn to create an S3 bucket and use aws-cli.

- Connect to the console of your administrator user
- Create an S3 bucket starting with the prefix `day02-` and finished with whatever numbers you want.
- Using aws-cli copies `appstore_games.csv` file to the bucket. You can check the file was correctly copied using the AWS console.

- Using aws-cli create a presigned URL allowing you to download the file. Your presigned url must have an expiring time of 10 minutes.

Your AWS CLI command must be stored in the `aws_presign_url.sh` script.

Exercise 02 - Elastic cloud compute (EC2).

Turn-in directory:	ex02
Files to turn in:	os_name.txt
Forbidden function:	None
Remarks:	n/a

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing us to quickly scale capacity (up or down) depending on our needs.

Amazon EC2 allows you to acquire compute through the launching of virtual servers called instances. When you launch an instance, you can make use of the compute as you wish, just as you would with an on-premises server. Because you are paying for the computing power of the instance, you are charged per hour while the instance is running. When you stop the instance, you are no longer charged.

Two concepts are key to launching instances on AWS:

- **instance type** : the amount of virtual hardware dedicated to the instance.
- **AMI** : the software loaded on the instance.

The instance type defines the virtual hardware supporting an Amazon EC2 instance. There are dozens of instance types available, varying in the following dimensions:

- Virtual CPUs (vCPUs)
- Memory
- Storage (size and type)
- Network performance

Instance types are grouped into families based on the ratio of these values to each other. Today we are going to use t2.micro instances!

One of the impressive features of EC2 is autoscaling. If you have a website, with 100 users you can have your website running on a little instance. If the next day, you have 10000 users then your server can scale up by recruiting new ec2 instances to handle this new load! (it can also scale down of course)

Exercise

In this exercise, you will learn how to create and connect to an ec2-instance.

Follow these steps for the exercise:

- launch an ec2 instance with the AMI : `Amazon Linux 2 AMI`.
- choose `t2.micro` as instance type.
- create a key pair.
- connect in ssh to your instance using your key pair.
- get and save the os name of your instance in the `os_name.txt` file.
- terminate your instance.

Within minutes we have created a server and we can work on it!

Exercise 03 - Flask API - List.

Turn-in directory:	ex03
Files to turn in:	app.py, s3_funcs.py
Forbidden function:	None
Remarks:	n/a

Before getting into AWS infrastructure, we are going to discover how to interact with AWS resources using a Python SDK (Software Development Kit) called boto3.

To do that, we are going to work with a python microframework called Flask to create an API (a programmatic interface) to interact with your s3 bucket. Flask will run a service within our EC2 instance and with which we will be able to interact (through the instance IP and a specific port). To interact with an AWS resource, we have to grant certain permissions to the EC2 we previously created. The most secure way to do that is to use a role and to associate it with the EC2. Roles will grant the needed privileges when associated with a resource.

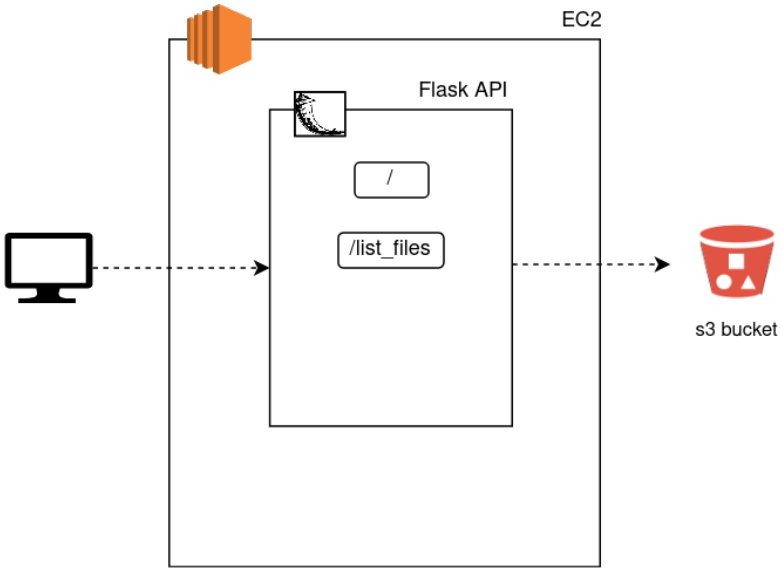

Figure 14: Flask API

nb: for a simplification of the following exercises we are going to use Flask directly like a development environment. If we wanted a more production-ready application we would add a webserver like Nginx and Gunicorn.

Exercise

In your AWS console:

- create an IAM EC2 role to allow S3FullAccess called `s3_access`.
- associate the created role to your EC2 instance.
- change security groups within your EC2 instance to allow traffic from port 5000 (this port will be used by Flask). You can either choose the whole internet or your IP as a source.

In your EC2 instance:

- install `python3` and the librairies `flask` and `boto3` (within python3)
- create a script app.py with 2 routes :
  - '/'
    - status : '200'
    - messsage : "Succesfully connected to day02 upload/download API"

- '/list_files' :
  - status : '200'
  - messsage : "Succesfully listed files on s3 bucket '<bucket_name>' "
  - content : list of files within the s3 bucket

The content you return with your Flask API has to be json formatted. You should use boto3 to interact with the s3 bucket you previously (`day02-...`).

Examples

For the route '/'

```
➜  ~ curl http://54.216.78.90:5000/
{
  "message": "Succesfully connected to day02 upload/download API",
  "status": "200"
}
➜  ~
```

Figure 15: Flask root

For the route '/list_files'

```
➜  ~ curl http://54.216.78.90:5000/list_files
{
  "content": [
    "airflow_scheduler.png",
    "toto.png"
  ],
  "message": "Succesfully listed files on s3 bucket 'day02-12345'",
  "status": "200"
}
➜  ~
```

Figure 16: Flask list files

Exercise 04 - Flask API - Upload.

Turn-in directory:	ex04
Files to turn in:	app.py, s3_funcs.py
Forbidden function:	None
Remarks:	n/a

We will continue to work on our Flask API to add new functionalities. This time we will work around file upload. Files upload can hardly be done without using proxy storage (data streams is an alternative to the proxy solution but are harder to implement so we will focus on an easier solution). Uploaded files will first be stored onto our EC2 before being sent to the s3 bucket.

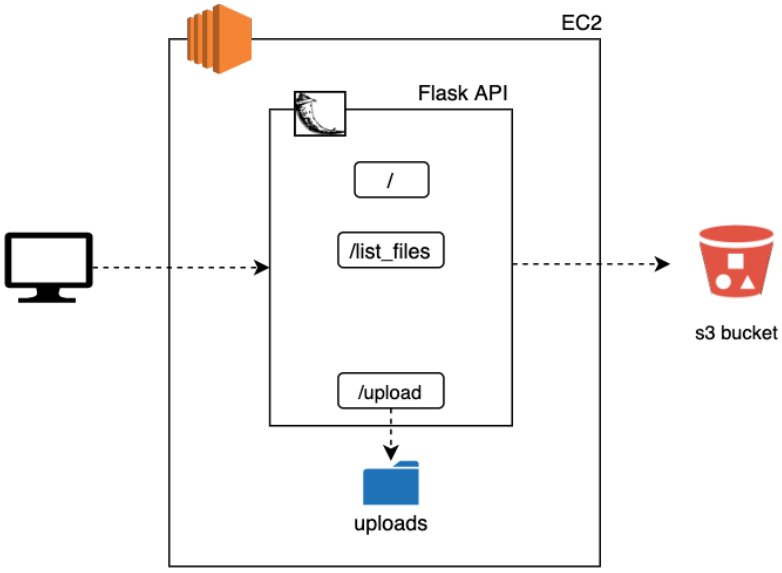

Figure 17: Flask API

Exercise

For this exercise you will have to create a directory called `uploads` which will be used to save files before sending them to the s3 bucket. Then you will have to add the '/upload' route to your Flask API. It has to follow those constraints:

```
- success

    - status : '200'

    - message : "Successfully uploaded '<file>' to the bucket '<bucket_name>'"

- file not found error

    - status : '404'

    - message : "The file '<file>' was not found"
```

The content you return with your Flask API has to be json formatted. You should use boto3 to interact with the s3 bucket you previously (`day02-...`).

Examples

For the route '/upload'

```
➜  ~ curl -X POST -F files=@toto.png http://54.216.78.90:5000/upload
{
  "message": "Succesfully uploaded file 'toto.png' on s3 bucket 'day02-12345'",
  "status": "200"
}
➜  ~
```

Exercise 05 - Flask API - Download.

Turn-in directory:	ex05
Files to turn in:	app.py, s3_funcs.py
Forbidden function:	None
Remarks:	n/a

To finish our Flask API we will implement one last functionnality, the download. For this exercise, we are going to avoid the proxy solution. Indeed, we have an easier and faster solution to do that. Do you remember ? Yes, I am talking about presigned URLs !

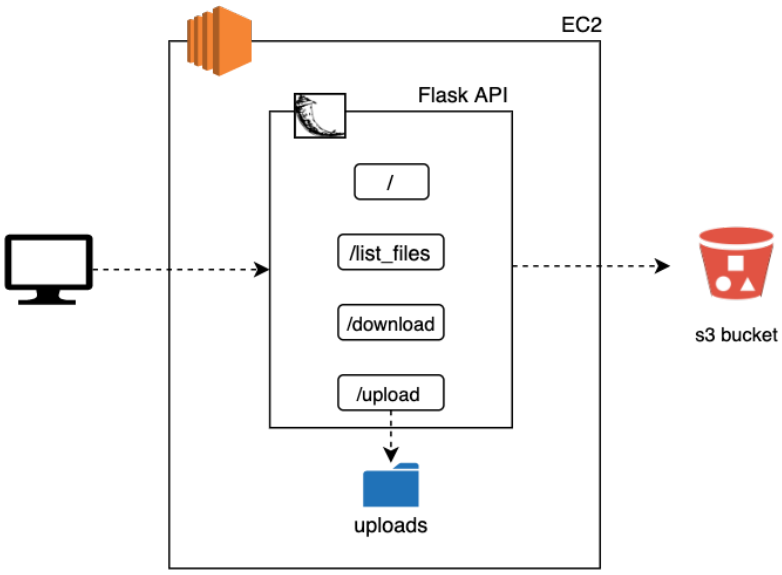

Figure 18: Flask API

Exercise

For this exercise you will have to create a directory a route '/download/`<file>`' which will create a presigned url with an expiring time of 2 minutes. Your route has to follow the following constraints:

```
- success

    - status : '200'

    - message : "Succesfully generated presigned url for '<file>'"

    - content : "<presigned_url>"

- file not found error

    - status : '404'

    - message : "The file '<file>' was not found on the bucket '<bucket_name>'"
```

Once you are done, create an AMI of your EC2 instance called `day02-flask-api-ami`. This will be used in further exercises.

The content you return with your Flask API has to be json formatted. You should use boto3 to interact with the s3 bucket you previously (`day02-...`).

Examples

For the route '/download'

Exercise 06 - Virtual Private Cloud (VPC).

Turn-in directory:	ex06
Files to turn in:	00_variables.tf, 01_networking.tf, 10_terraform.auto.tfvars
Forbidden function:	None
Remarks:	n/a

AWS and I lied to you! You thought deploying a server was that simple ? A huge part of the required stack for the deployment is hidden! This hidden layer uses a wizard configuration (default configuration suitable for most users). The default configuration includes :

- network (VPC, subnets, CIDR blocks)
- network components (routing table, Internet gateway, NAT gateway)
- security (NACLS, security groups)

The default configuration allows you to use many services however the default security is open to the whole internet which is quite insecure!

To deploy an AWS infrastructure at a production level, we need to handle those layers ourselves. As you already know, we are using our free tier. However, if you let your server run for weeks you will have to pay. We want to avoid this possibility. That's why we are going to use a tool to automatically deploy and destroy our infrastructure, Terraform.

Terraform is a tool to deploy infrastructure as code. It can be used for multiple cloud providers (AWS, Azure, GCP, ...). We are going to use it to deploy our own virtual network and servers.

All potentially critical data **MUST NOT** be deployed using infrastructure as code like terraform. If they are, they may be destroyed accidentally and you never want that to happen!

Terraform install

First, download the terraform software for macOS.

```
brew install terraform
```

You can now run the `terraform --version`. Terraform is ready!

Terraform is composed of three kinds of files:

- `.tfvars` : terraform variables.
- `.tf` : terraform infrastructure description.
- `.tfstate` : describe all the parameters of the stack you applied (is updated after an apply)

No further talking, let's deep dive into terraform !

VPC

For this first exercise, you have to create a VPC (Virtual Private Cloud) using terraform. A VPC emulates a network within AWS infrastructure.

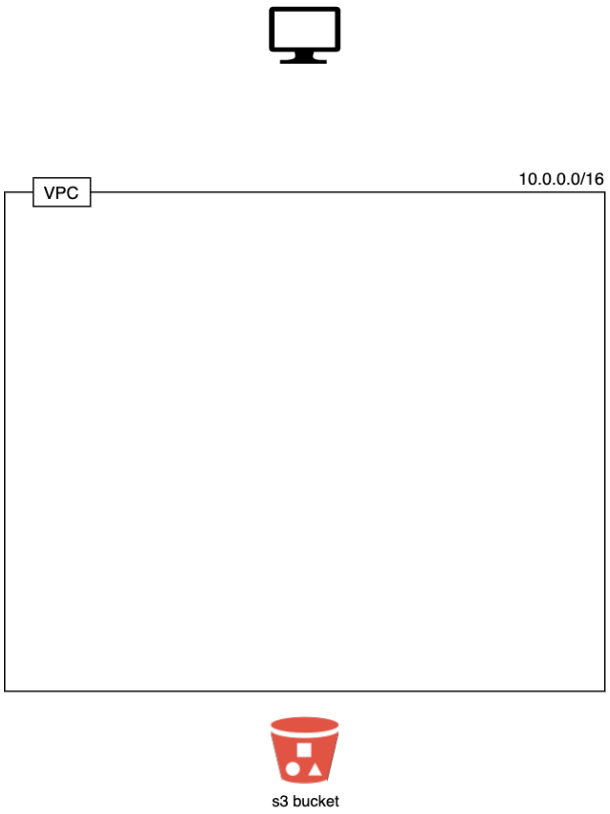

Figure 19: Flask API AWS infrastructure

Variables must be specified in variable files ! Your stack must only create a VPC.

For this exercise you must create 3 files:

- `00_variables.tf`: to declare variables
- `01_networking.tf`: to declare your stack
- `10_terraform.auto.tfvars`: to assign variables

You have to respect the following constraints:

- your vpc is deployed in Ireland (specified as the variable `region`).
- your vpc uses a `10.0.0.0/16` CIDR block.
- your vpc must enable DNS hostnames (this will be useful for the next exercises)
- you must add two tags:
  - `project_name` with the value `day02` (this allows us to easily track all resources associated to a project)
  - `Name` with the value `day02-vpc`

On your AWS console, you can go in the VPC section to check if your VPC was correctly created.

You can run `terraform destroy` to delete your stack.

Exercise 07 - Subnets.

Turn-in directory:	ex07
Files to turn in:	00_variables.tf, 01_networking.tf, 10_terraform.auto.tfvars
Forbidden function:	None
Remarks:	n/a

Within our newly created VPC we want to divide the network's IPs into subnets. This can be useful for many different purposes and helps isolate groups of hosts together and deal with them easily. In AWS, subnets are often associated with different availability zones which guarantees the high availability in case if one of the AWS data centers implied in our infrastructure is destroyed.

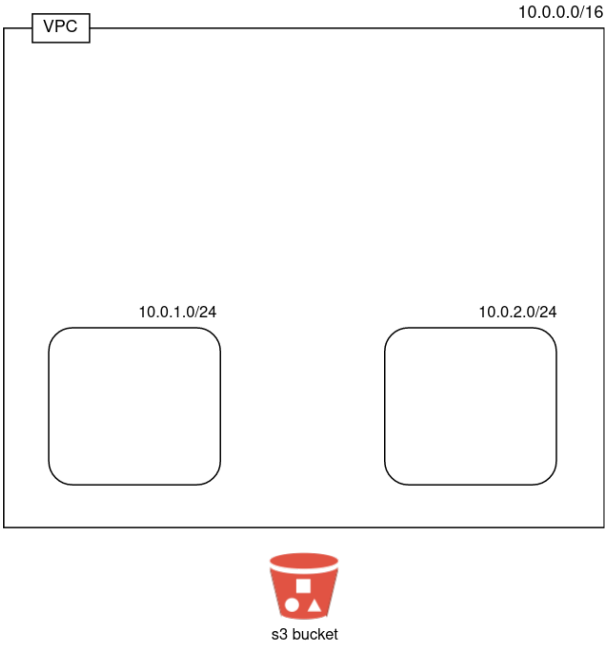

Figure 20: Flask API AWS infrastructure

Exercise

Within our previously created VPC, add 2 subnets with the following characteristics :

- their depends on the creation of the VPC (it has to be specified on terraform)
- your subnets will use `10.0.1.0/24` and `10.0.2.0/24` CIDR blocks.
- your subnets will use `eu-west-1a` and `eu-west-1b` availability zones.
- they must map public ip on launch.
- they must have two tags:
  - `project_name` with the value `day02`
  - `Name` with the value: `day02-sn-pub-1`/`day02-sn-pub-2` (depending on the subnet you create)

Exercise 08 - IGW - Route table.

Turn-in directory:	ex08
Files to turn in:	00_variables.tf, 01_networking.tf, 10_terraform.auto.tfvars
Forbidden function:	None
Remarks:	n/a

Let's continue our infrastructure ! We created a Network with VPC and divided our network into subnets into two different availability zones. However, our network is still not accessible from the internet (or your ip). First, we need to create an internet gateway and link it with our VPC. This first step will allow us to interact with the internet.

The subnets we created will be used to host our EC2 instances but our subnets are now disconnected from the internet and other IPs within the VPC. To fix this problem, we will create a route table (it acts like a combination of a switch (when you interact with IPs inside your VPC) and a router (when you want to interact with external IPs)).

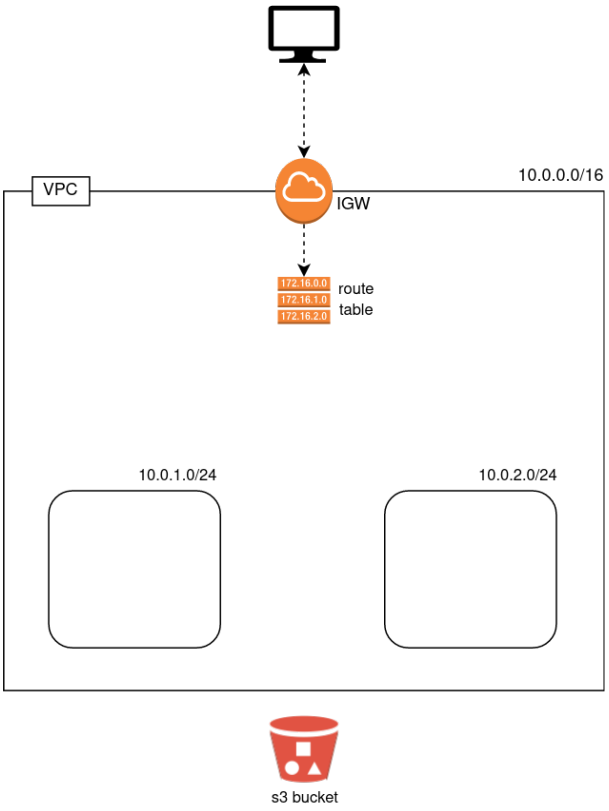

Figure 21: Flask API AWS infrastructure

Exercise

Create an Internet gateway (IGW) which depends on the VPC you created. Your IGW will need the tags:

- `project_name` with the value `day02`
- `Name` with the value `day02-igw`

Create a route table that depends on the VPC and the IGW. Your route table will have to implement a route linking the `0.0.0.0/0` CIDR Block with the IGW. The `0.0.0.0/0` is really important in the IP search process, it means if you cannot find the IP you are looking for within the VPC then search it on other networks (through the IGW). Your route table will need the following tags:

- `project_name` with the value `day02`
- `Name` with the value `day02-rt`

You thought you were finished ? We now need to associate our subnets to the route table ! Create route table associations for both of your subnets. They will depend on the route table and the concerned subnet.

Exercise 09 - EC2 - Security groups.

Turn-in directory:	ex09
Files to turn in:	02_ec2.tf, 08_security_groups.tf, *.tf, *.tfvars
Forbidden function:	None
Remarks:	n/a

We finally have an infrastructure we can work with ! Now let's work on our EC2 instance. We will use the ami we saved to provision our EC2 instance and automatically run our Flask application.

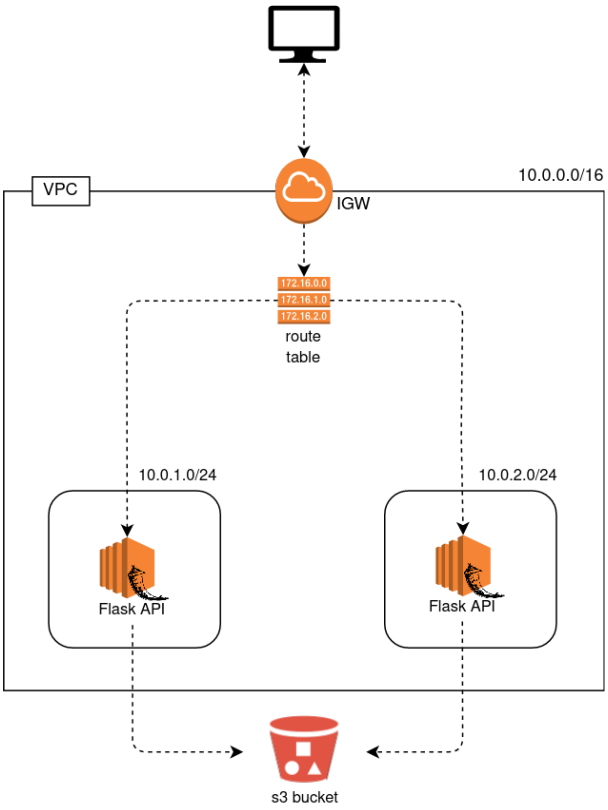

Figure 22: Flask API AWS infrastructure

Exercise

Create a security group called `day02-flask-sg` for your EC2 instance :

- this security group depends on the creation of the subnets
- it must allow ssh traffic from anywhere (or your IP only if you prefer)
- it must allow traffic through the port `5000` for your Flask application to work

Create an EC2 resource which :

- depends on the creation of the subnets
- must associate public IP address (it is useful if we want to ssh)
- must use :
  - a `t2-micro` instance type
  - the AMI you previously saved
  - the security group you just created

- the `s3_access` role you created

Even with all these setups one step is still missing, when starting our instance the Flask application is not running ! If we want our application to start when we instantiate our EC2 we will need to provide user data. User data is a script we will run when an instance starts.

Add user data to start your flask application. You will need to use `nohup` and `&` (background command) for your command to work.

The curl commands must work just after you applied your terraform infrastructure.

Exercise 10 - Autoscaling group.

Turn-in directory:	ex10
Files to turn in:	02_asg.tf, *.tf, *.tfvars
Forbidden function:	None
Remarks:	n/a

Any Cloud provider is based on a pay as you go system. This system allows us not to pay depending on the number of users we have. If we have 10 users our t2.micro EC2 may be sufficient for our Flask application but if tomorrow 1000000 users want to try our super API we have to find a way to scale our infrastructure !

This can be done trough autoscaling groups. The more traffic we have the more EC2 instances will spawn to handle the growing traffic. Of course those new instances will be terminated if the number of users goes down.

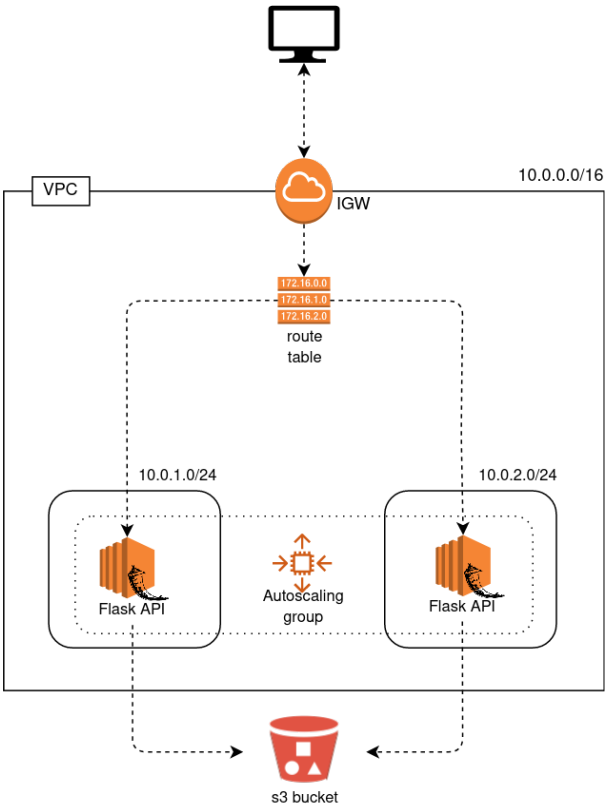

Figure 23: Flask API AWS infrastructure

Exercise

Transform your `02_ec2.t` terraform file into `02_asg.tf`. You will have to transform your code into an autoscaling group.

You need to implement a launch configuration with your EC2 parameters and add a create before destroy lifecycle.

Create an autoscaling group with :

- dependency on the launch configuration
- a link to the subnets of our vpc
- the launch configuration you previously created
- a minimal and maximal size of your autoscaling group will be 2 (this will allow us to always keep 2 instances up even if one terminates)

- a tag with:
  - `Autoscaling Flask` for a key
  - `flask-asg` for the value
  - the propagate at launch option

You should see 2 EC2 created within your console.

Exercise 11 - Load balancer.

Turn-in directory:	ex11
Files to turn in:	03_elb.tf, 02_asg.tf, *.tf, *.tfvars
Forbidden function:	None
Remarks:	n/a

Let's finish our infrastructure ! With our autoscaling group we now have 2 instances but we still need to go to our AWS console to search the IP of each EC2 instance. It's not convenient !

A solution is to create a load balancer. A load balancer as its name indicates will balance the traffic between EC2 instances (of our autoscaling group here).

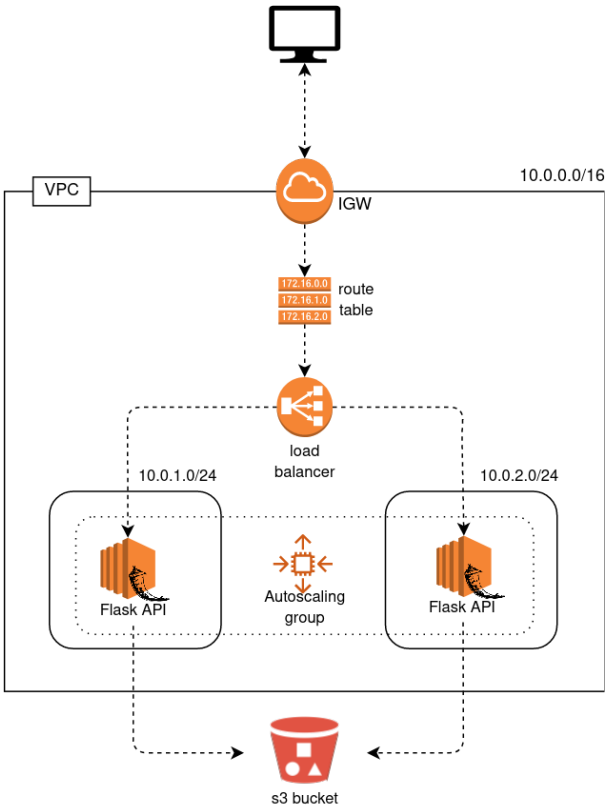

Figure 24: Flask API AWS infrastructure

Exercise

Create a security group for your load balancer. It must :

- depend on the vpc you created
- allow traffic from port 5000

Create a load balancer with :

- a health check on port 5000 every 30 sec and a healthy threshold at 2
- a listener on the port 5000
- the cross-zone load balancing option

In your autoscaling group add your load balancer and a health check type of type `ELB`.

Create a terraform output that will display the DNS name of your load balancer.

You should be able to use the DNS name of your load balancer to call the API now !