

**AWS CSAA Labs**

Let’s get our hands dirty, but before you begin, make sure that you have access to Celfocus AWS training account. You can signin on AWS console here - [**https://425239951089.signin.aws.amazon.com/console**](https://425239951089.signin.aws.amazon.com/console)or in our custom alias[**https://celfocus-training.signin.aws.amazon.com/console**](https://celfocus-training.signin.aws.amazon.com/console)

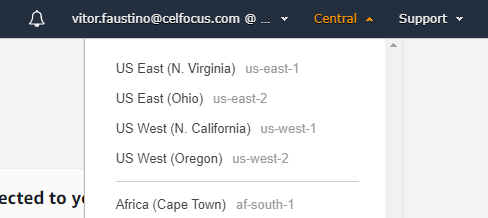
Your **username is always your Celfocus email** and you only be able to create VPC and EC2 resources on your **assigned region**. Each student **will have a number too**, don’t forget it, because you will need it throughout our labs.

Exam details: <https://d1.awsstatic.com/training-and-certification/docs-sa-assoc/AWS-Certified-Solutions-Architect-Associate_Exam-Guide.pdf>  
  
You can find additional course material here on GitHub:

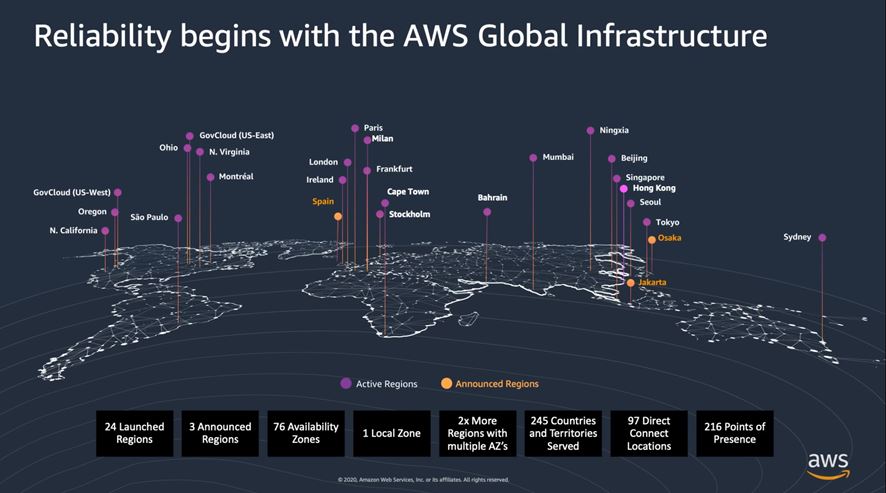
* [main.template](https://github.com/vitorfaustino/aws-csaa-training/blob/master/main.template) – Cloudformation script used to create IAM users, groups and permissions
* [AWS-CSAA-Study-Guide.docx](https://github.com/vitorfaustino/aws-csaa-training/blob/master/AWS-CSAA-Study-Guide.docx) – A Study Guide to help you pass the exam

TIP: Explore the cloudformation automation template on IAM, and try to understand Users, Groups and Roles (trustable entities) created and possible limitations you might have on this account.

Before you start our exercises with the account, be sure that you understand AWS Global Infrastructure, and the difference between Regions, Availability Zones and Edge Locations. AWS is always expanding their regions, you might find new ones on the top right dropdown menu.



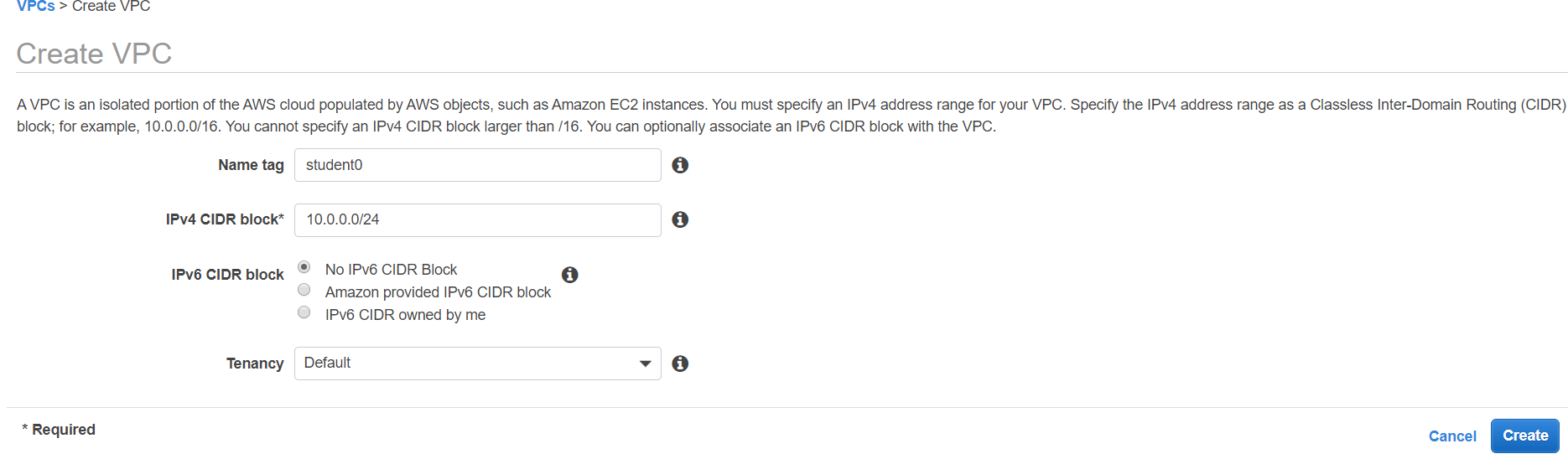
Discover all the infrastructure like Regions and Availability Zones here - <https://aws.amazon.com/about-aws/global-infrastructure/regions_az/?nc1=h_ls>



**Exercise 1 – Create a Web Server on a Public Subnet**

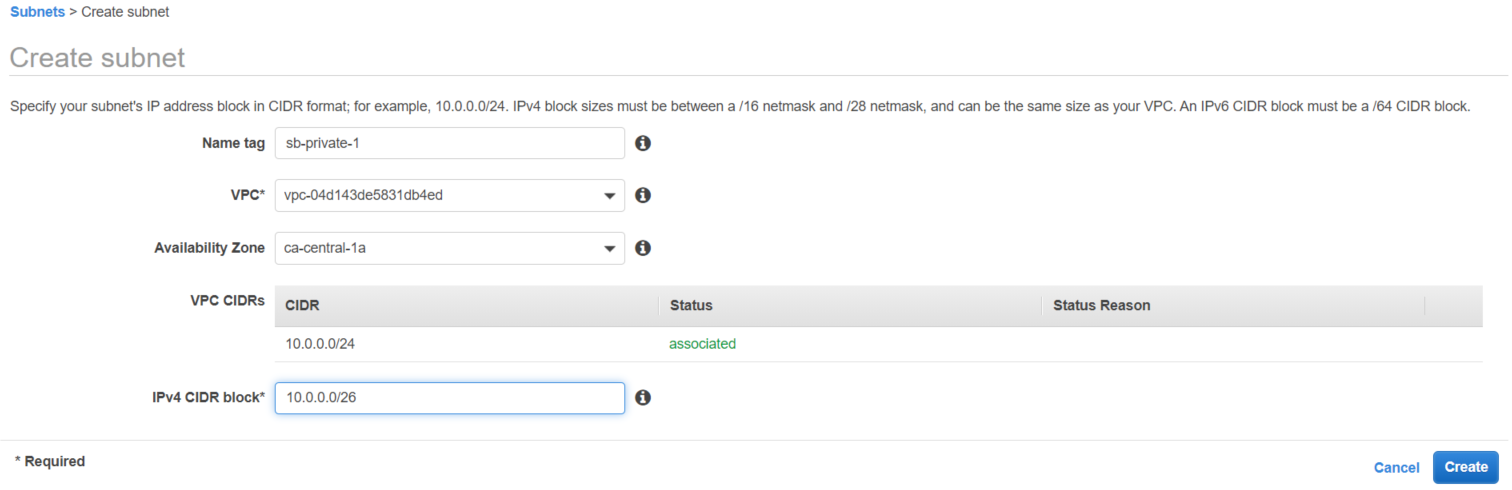
In this exercise you will create a webserver in a public subnet that should be availble from the internet. Before creating instances, VPC and Subnets are needed as the network layer for your infrastructure. You will need a route to an Internet Gateway and this EC2 must allow inbound connections.

1. Create a VPC 10.0.[**N**].0/24 with tenancy default. Replace **N** with your student number.



2. Create 2 public and 2 private subnets, each one with 64 hosts ( last octet .HHNNNNNN -> /26 to split 64 hosts). Replace **N** with your student number.

1. 10.0.[**N**].0/26 (Tag sb-private-1) – Choose AZ-A
2. 10.0.[**N**].64/26 (Tag sb-private-2) – Chose AZ-B
3. 10.0.[**N**].128/26 (Tag sb-public-1) – Choose AZ-A
4. 10.0.[**N**].192/26 (Tag sb- public-2) – Choose AZ-B



TIP: Try to understand CIDR and subneting concepts, important topic to setup VPCs and Subnets.

3. Enable option ***auto-assign public IPv4 address*** on both Public subnets.

4. Continue on VPC console and create a new Security Group with name **ssh-internet**, and add SSH Inbound rule (TCP Port 22). Keep Outbound rule open (All trafic).

TIP: Keep you EC2 instance more secure, you can add only your home/office network IP on SSH configuration. Be aware that management ports (e.g. SSH), should not be publicly accessible

5. Switch to **EC2 Console** and lauch a new instance with the following Setup:

1. **AMI**: Ubuntu Server 20.04 LTS (HVM)
2. **Instance Type**: t3.large
3. **Intance Details**: Choose your created VPC and Subnet sb-public-1. Confirm that **Auto-assign Public IP is enabled.** Keep all other default options
4. **Storage**: Keep the default 8 GB for Root Volume
5. **Tags**: Add Tags if you prefer
6. **Security Group**: Select existing SG **ssh-internet**
7. **Review:** Confirm all options and press **Launch** button
8. **Key Pair:** Create a new key pair(RSA type) and save the .pem file. You can reuse it for future instances.

TROUBLESHOOT: Your instance failed with a not authorized message? Go to IAM, find your groups, open JSON config and try to find out constraints regarding instance lauch. Repeat Step 5 with allowed configuration.

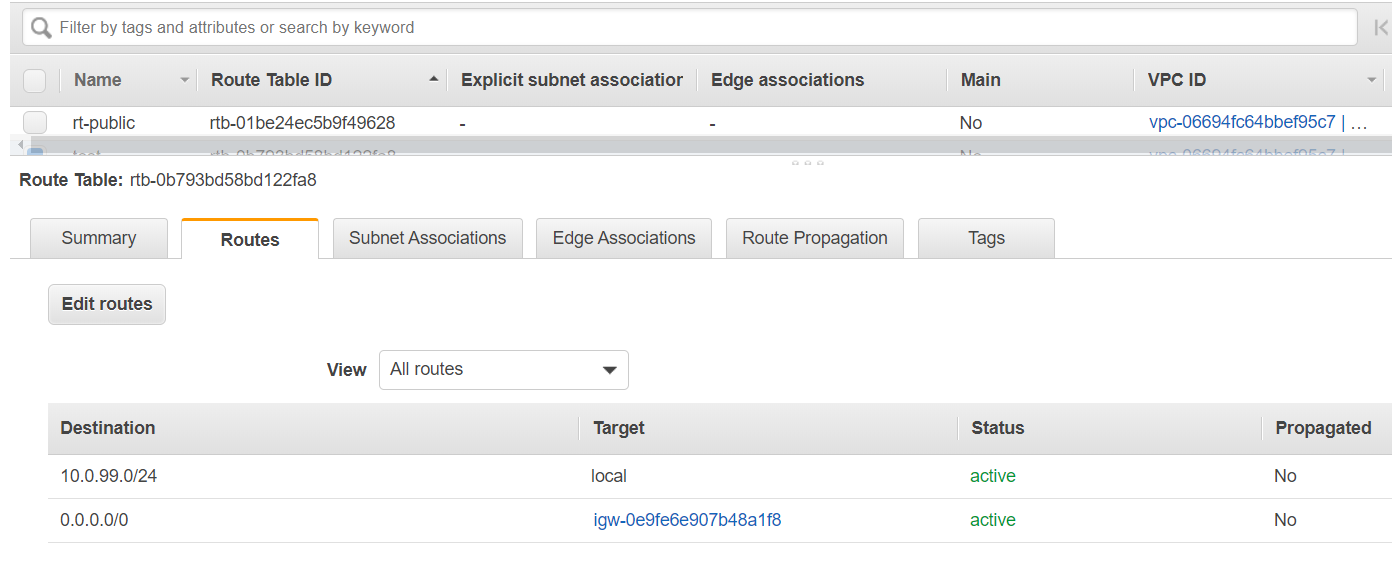
6. Download **MobaXterm** - <https://mobaxterm.mobatek.net/download.html>, and create a SSH connection to the Public IP provided to your EC2 instance, add .pem file and check username on instance connect details. Try to connect instance.

TROUBLESHOOT: You get a connection timed out error? Your subnet does not have Internet Gateway and a proper route to the internet. Follow next step.

7. Under VPC console, create a new **Internet Gateway**, and attach to it to your VPC.

8. Go to Route Tables and create a new Route Table. Select your route table, and add Route 0.0.0.0/0 to your newly created Internet Gateway. Associate this new Route table to only your Public Subnets.

Congrats! With this Setup you will get connectivity from Public Internet (Home/Office) to your instance.



**Exercise 2 – Setup Web Server**

In this exercise you will to install nginx service and access web server http (port 80) from the internet.

1. Login into EC2 created on previous exercise and run the following commands inside you Ubuntu instance:

*sudo apt-get update  
sudo apt-get install git nginx -y  
sudo service nginx restart  
sudo rm -R /var/www/html/\*  
sudo git clone https://github.com/vitorfaustino/eshop-fe-demo.git /var/www/html*

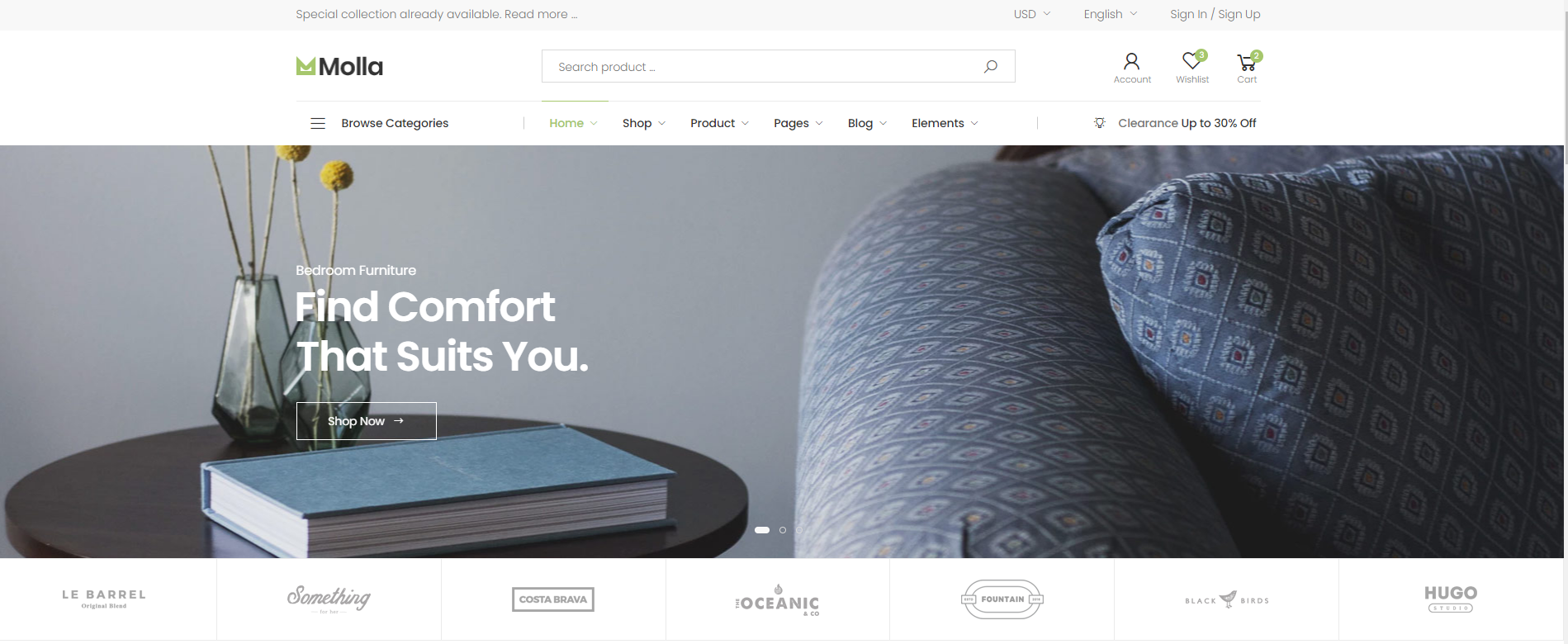
*rm -R /var/www/html/\**

*git clone https://github.com/vitorfaustino/feup-eshop.git /var/www/html*

2. Now that you have a NGINX server running, try to open webserver default page on your browser, [***http://myEC2PublicIP***](http://myEC2PublicIP)

TROUBLESHOOT: Your browser takes too much time connecting to NGINX website? Maybe some additional Port needs to be open.

3. Go to VPC and edit Security Group, adding **TCP Port 80** (HTTP) on Inbound Rules to Anywhere from Internet (**0.0.0.0/0**)

4. Try again to Open web page using instance public IP, you should have success now with our eShop web template!  
  
 

5. Terminate this EC2, launch a new instance using Ubuntu Server 20.04 LTS (HVM) as you did on previous exercise. In User data, automate software installation, adding below script:

TIP: Try to explore Network ACL inside VPC console, creating explicit Deny to SSH, first with Rule Number 99 and then 101, and understand the network behaviour.

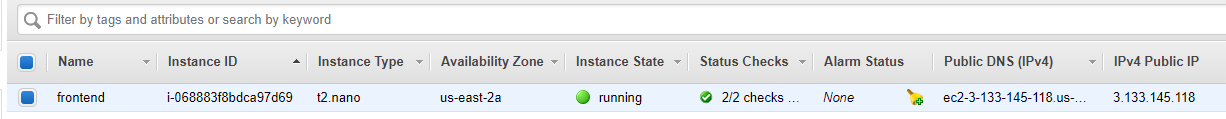
*#!/bin/bash  
apt-get update  
apt-get install git nginx -y  
service nginx restart  
rm -R /var/www/html/\*  
git clone https://github.com/vitorfaustino/eshop-fe-demo.git /var/www/html*

*rm -R /var/www/html/\**

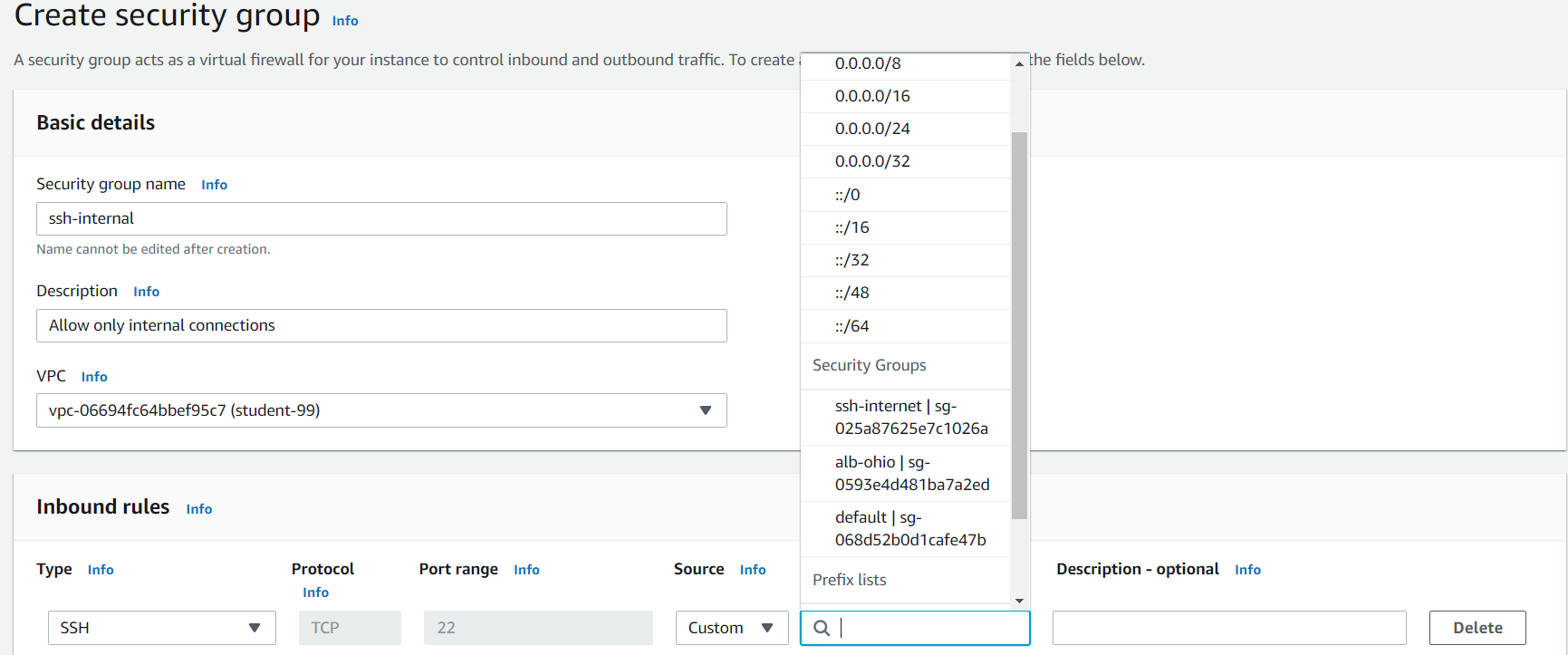
*git clone https://github.com/vitorfaustino/feup-eshop.git /var/www/html*

**Exercise 3 – Bastion Host**

You need to create a new EC2 instance in a private subnet, and need to access this machine throught SSH, without having any port exposed to the public internet. Use previously created frontend instance as a jump server or bastion host.  
  
1. Go to EC2 Console and change the name of the previously EC2 instance to **frontend**

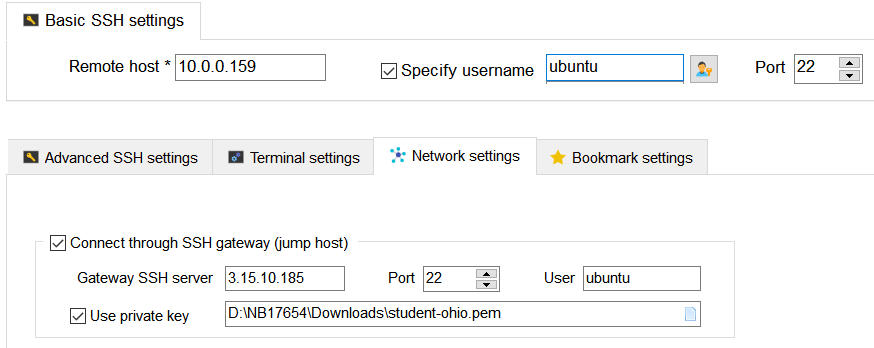


2. Create a new Security Group named **ssh-internal**, and add inboud SSH protocol. Only allow source connection from the **ssh-internet** Security Group.This way, only instances with this Security Group will access your private instance.



3. Launch a new EC2 with same specs you did on exercise 1. Only change the subnet, should be a private one and no public IP should be assigned to this instance. Security group should be **ssh-internal** that you created above.

4. Use MobaXterm to configure a connection to the internal IP (10.0.N.X). Use the public IP of our frontend on Gateway SSH Server. Use image below to help you on this setup.



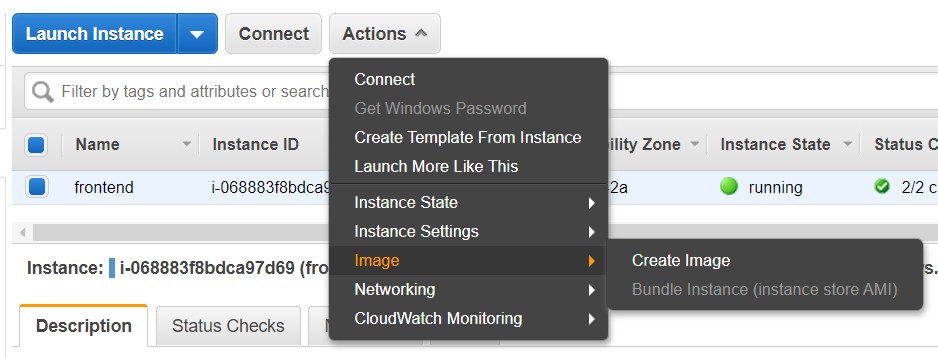
TIP: Change ssh-internal Security group and update inboud rule to your VPC CIDR block (e.g. 10.0.0.0). You should be able to access this internal EC2.

5. Test if you can connect to internal machine. Destroy this instance to save costs.

**Exercise 4 – Encrypted EBS Web Server**

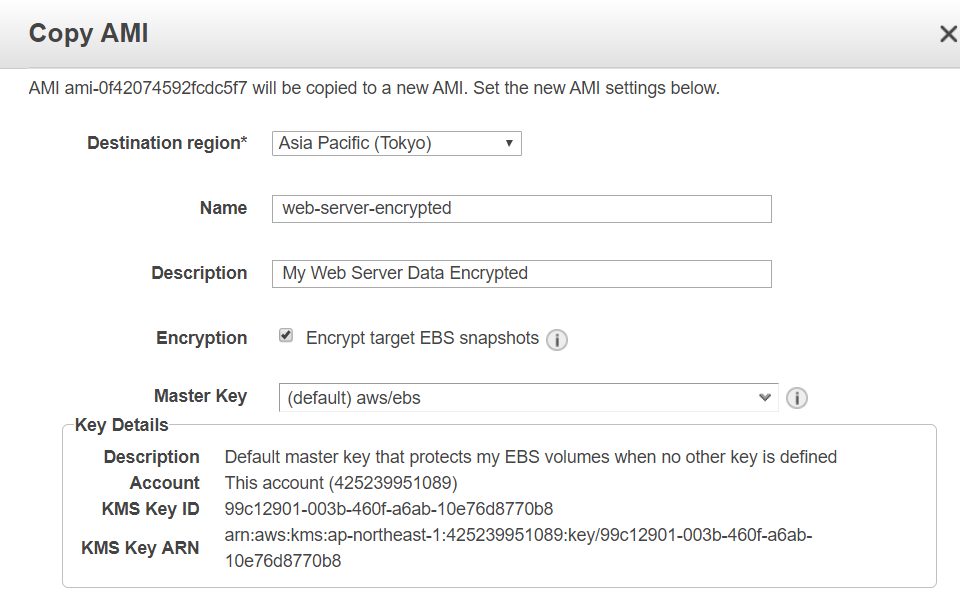
You need to encrypt all data from your Web Server at rest to be complaiant with security rules.

1. Create an image from your web server. Keep the original disk size and name it for example **web-server-unencrypted**, because data on this image is unencrypted



2. Go to Images/AMIs on EC2 console, and wait until the image is created.  
  


3. Once the image snapshot is created, choose option **copy AMI** to your own region and choose **Encryption** option. Encrypted data will be associated with a default KMS Master key, in this case aws/ebs. Wait until the copy is finished.



4. Select encrypted instance and choose **Lauch**. Follow same guidelines of exercise 1 to launch instance on public subnet.  
Only make a change on **Step3** (Configure instance), find **User data** textbox and place the following commands:  
  
5. Find the instance public IP and access it through your browser. You should see below message, with internal IP information from instance meta-data

TIP: Did you notice that we are not installing NGINX on User Data? Remember that we took a snaphot from an instance on which NGINX was already installed. So that image include OS and all Apps on top of it.

***#!/bin/bash  
echo "<h1>Welcome to my encrypted web server</h1> Internal IP:" > /var/www/html/index.nginx-debian.html  
curl http://169.254.169.254/latest/meta-data/local-ipv4 >> /var/www/html/index.nginx-debian.html  
service nginx restart***



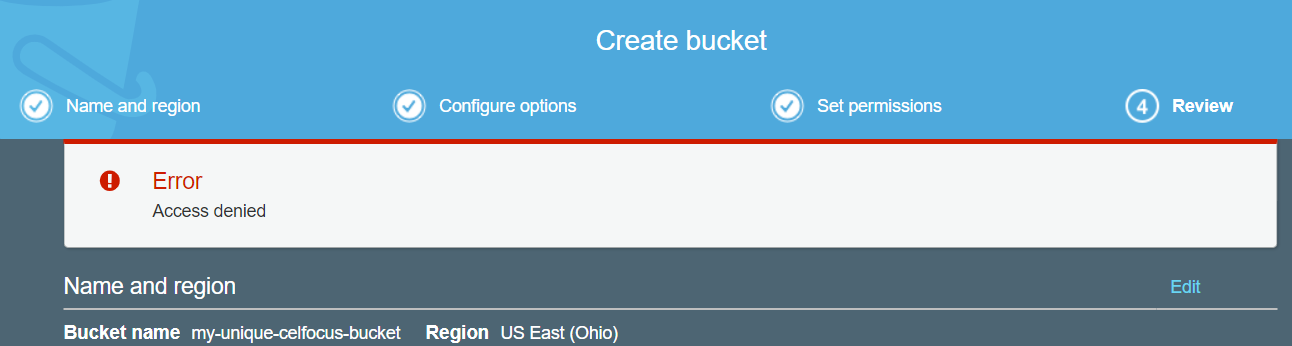
6. Teminate your unencrypted at rest instance web server to save costs.

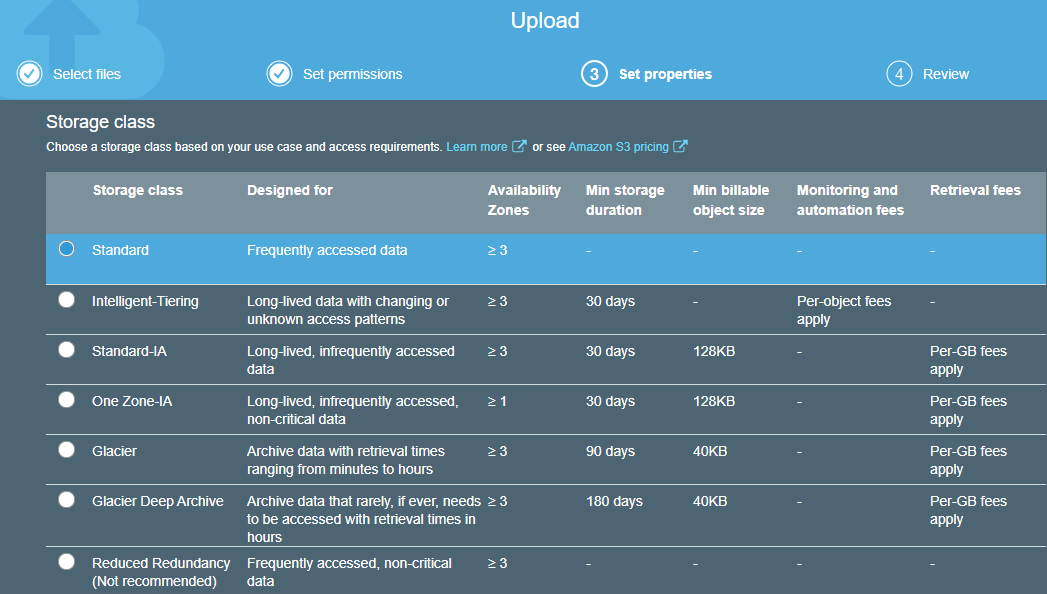
**Exercise 5 – S3 Bucket setup**

Create a S3 bucket on your region, and explore several ways to upload files.

1. Go to S3 console and create a bucket with any name you want on your select region, be aware that bucket names must be unique for **all AWS customers globally**.

TROUBLESHOOT: You get an **Access Denied** error? Explore IAM and try to understand bucket name restrictions applied to your user.



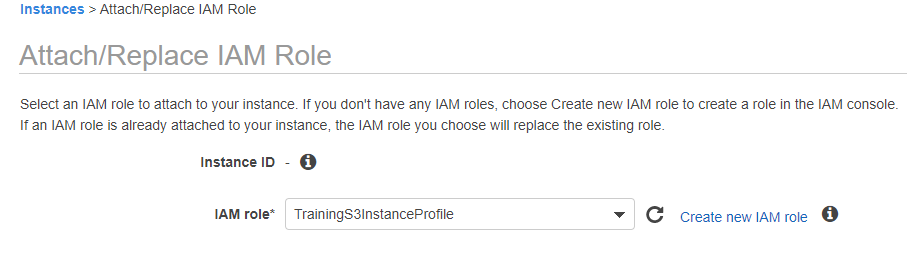
2. After finding the right way to create the bucket, enable version and encryption at rest.  
  
3. Upload a sample file, and explore the **storage class** you can configure to individual objects inside bucket.  
  


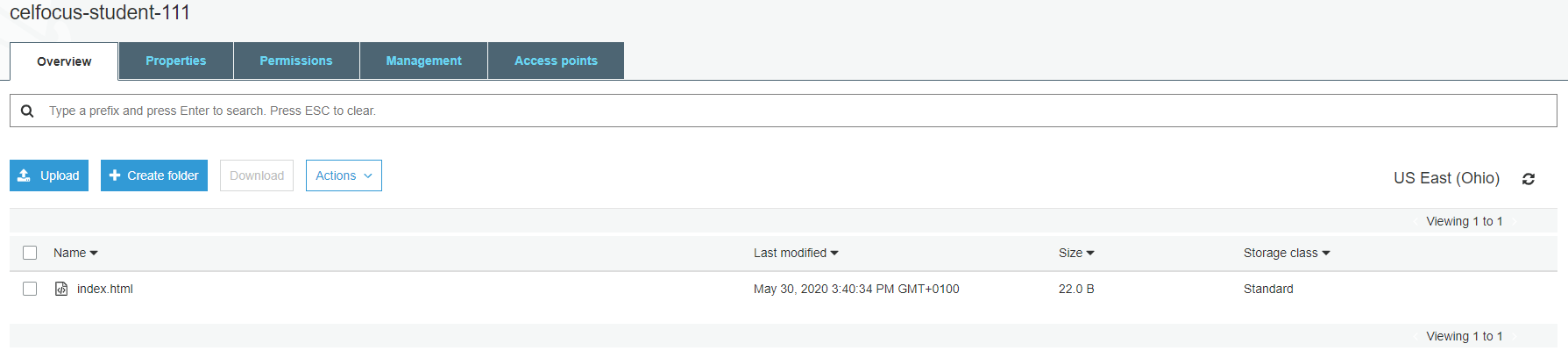
4. Now, access your web server EC2 instance created on previous exercise, and execute the following commands. The goal is to install **AWS CLI**, create and upload a file for the newly created S3 bucket. Replace ***celfocus-student-111*** by your bucket name  
  
5. Go to EC2 instance console and attach Role [**TrainingRoleS3InstanceProfile**, this will give permissions to your EC2 instance to upload files](https://us-east-2.console.aws.amazon.com/iam/home?region=us-east-2#roles/TrainingRoleS3ReadWrite) to your S3 bucket.

TROUBLESHOOT: Got an error uploading?   
**upload failed: ./index.html to s3://celfocus-student-111/index.html Unable to locate credentials**

Follow next steps.

***sudo apt-get install awscli -y  
echo "Generated from my EC2" > index.html  
aws s3 cp index.html s3://celfocus-student-111/index.html***

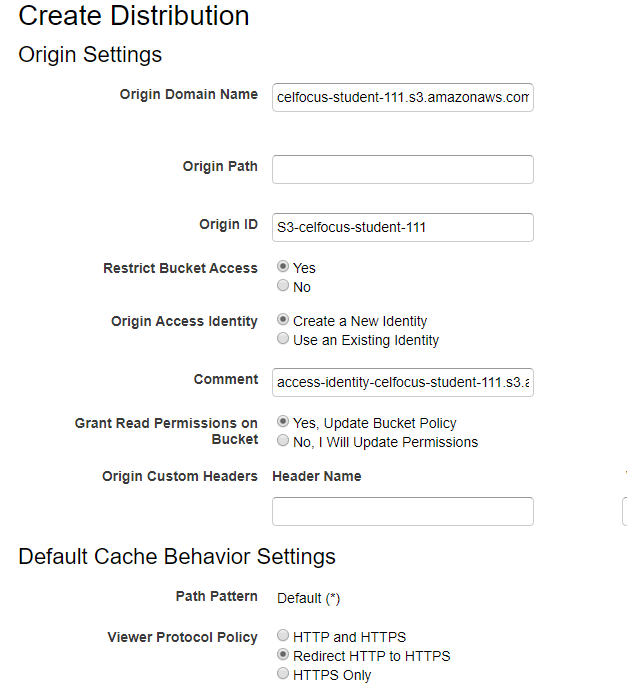


6. Retry above command to upload file on Step 4 to your S3 bucket, you will have success. Find your file on the S3 console to be sure you had success.   
**Notes**: This is a perfect example on how you can give permissions to AWS services, in this case an EC2, avoiding to use keys/access keys to communicate with other AWS services. Always choose roles, this is a tipical Exam question.

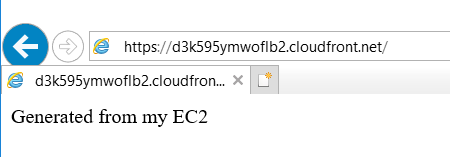
**Exercise 6 – Cloudfront distribution**

Imagine that you have users around the world accessing your **index.html** created on previous exercise, and you want to improve User Experience, delivering content closer from their location. Our customers are mainly based on Europe.

1. Create a new Cloudfront web distribution. Find below image options, select your bucket from Origin Domain Name, and create a new **Origin Access Identity**, because we want to keep bucket private giving Cloudfront permissions to access content. On Distribution Settings and Price Class selecting **U.S., Canada and Europe** is enough. Be sure that defalut root object is **index.html** too. It can take sometime until distribution is Enabled.



2. After you cloudfront is deployed, find your domain name and try it in your browser with http, for example <http://d3k595ymwoflb2.cloudfront.net/>  
You will be redirected to https and get the content of your generated file from EC2.



**Exercise 7 – Using AWS CLI**

Ask your instructor for a personal key/access key associated with your IAM user and install AWS CLI V2 and configure it on your own laptop.

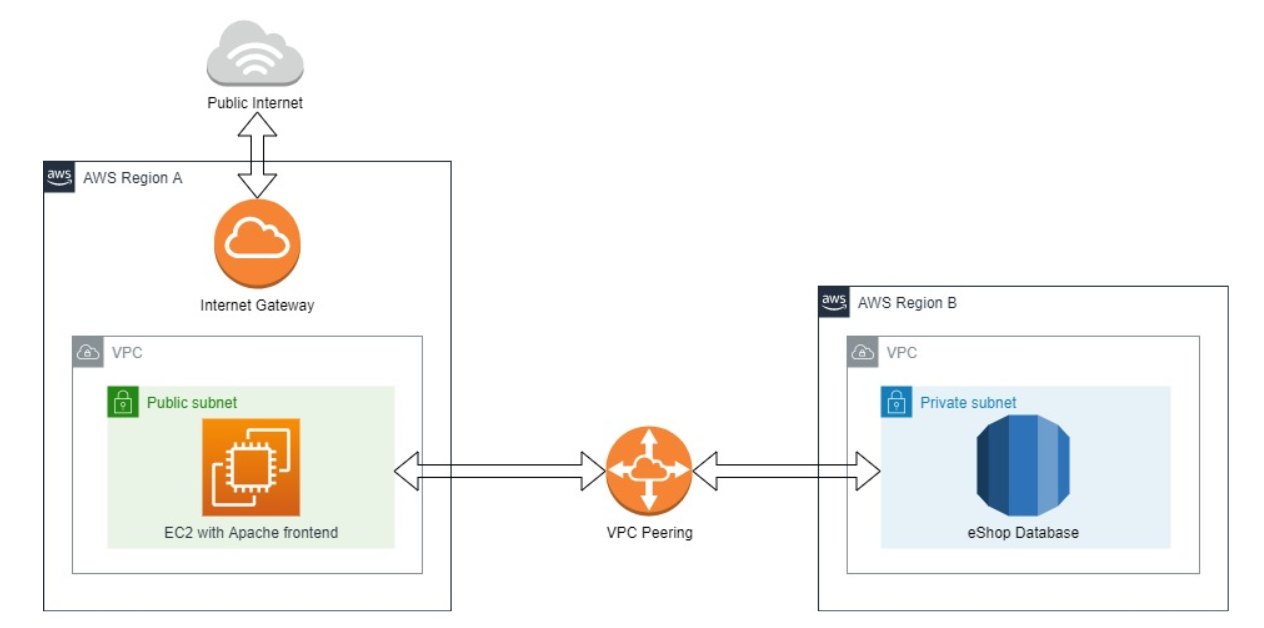
1. Repeat same task on exercise 5, and copy a new index file from your PC to the already created bucket S3, and test it out using Cloudfront.

2. Try to shutdown and start your EC2 using aws cli instead the AWS console.

TIP: Key/Access Key is associated with your user, hence you can do all tasks using this command line tool instead using the AWS console. Play around with it - <https://aws.amazon.com/cli/>

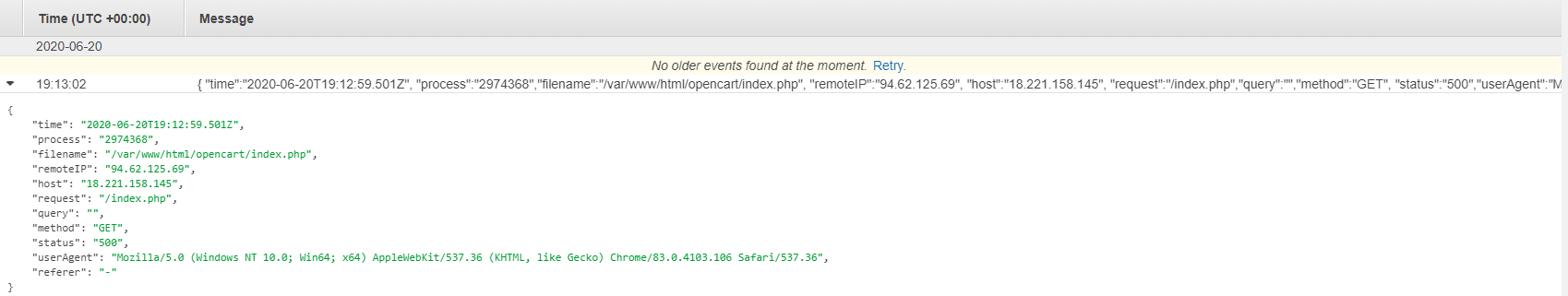
**Exercise 8 – eShop with EC2, RDS and Cloudwatch**

In this exercise you(Student A) need to find a buddy(Student B). You will need to setup an EC2 in a public subnet and connect to your buddy’s RDS on another region using global AWS backbone through a VPC peering. They will communicate using private IP address, without requiring gateways, VPN connections or separate network appliances. Traffic using Inter-Region VPC Peering always stays on the global AWS backbone and never traverses the public internet, thereby reducing threat vectors, such as common exploits and DDoS attacks.  
This is an important topic, you should always choose peering or VPC endpoints from a security and performance perspectives.   
Find below image with desired architecture for this eShop.



**You will get this** :

1. A pre-configured AMI with cloudwatch agent, apache and php installed with eShop application
2. Cloudwatch agent configuration can be found here inside EC2 ***/opt/aws/amazon-cloudwatch-agent/bin/config.json*** (file collector, memory and disk usage will be sent by agent to cloudwatch using attached Role)
3. Apache Logs format were changed to Json format ***/var/log/www/access/access.log.*** With this format we can explore data in Cloudwatch insights easily and create dashboards for our application layer



1. A pre-configured database snapshot with our eShop data

**You need to create this** :

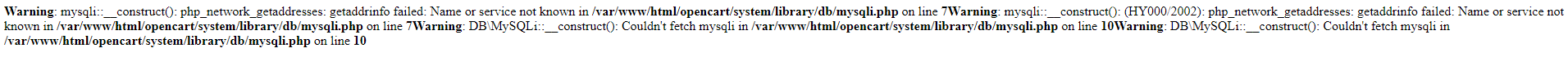
Student A (Region A):

1. Go to Canada region and copy AMI **eshop-celfocus-cloudwatch** to your region

TROUBLESHOOT: Problem via web console? Try to find an alternative way.

2. Go to **Cloudwatch** and delete all Log groups in your region

3. Lauch a **t2.nano** eShop AMI **eshop-celfocus-cloudwatch** in a **public subnet** with **Cloudwatch Role.** You need a Security Group to be able to access this instance with SSH (Home network) and HTTP (Public Internet).

4. Try to access instance with the public IP in your browser, you should get a PHP error.  


5. Explore Cloudwatch Log groups, you should find JSON records regarding apache access and error logs. You should notice same HTTP Status code 500.

6. Create a VPC peering between your VPC and you buddy VPC, and ask your buddy RDS endpoint.

TROUBLESHOOT: You must work as a team, peering must be accepted and “green” from both sides. Be carefull with Routes and Security groups. Use telnet to test connectivity between EC2 and RDS endpoint.

7. Once you have connectivity, change endpoint in your application, **/var/www/html/opencart/config.php (DB\_HOSTNAME)**.

TIP: If the website is broken, is because opencart is not getting images, change this parameter as well on config file, and setup your public IP on **HTTP\_SERVER** section

Student B (Region B):

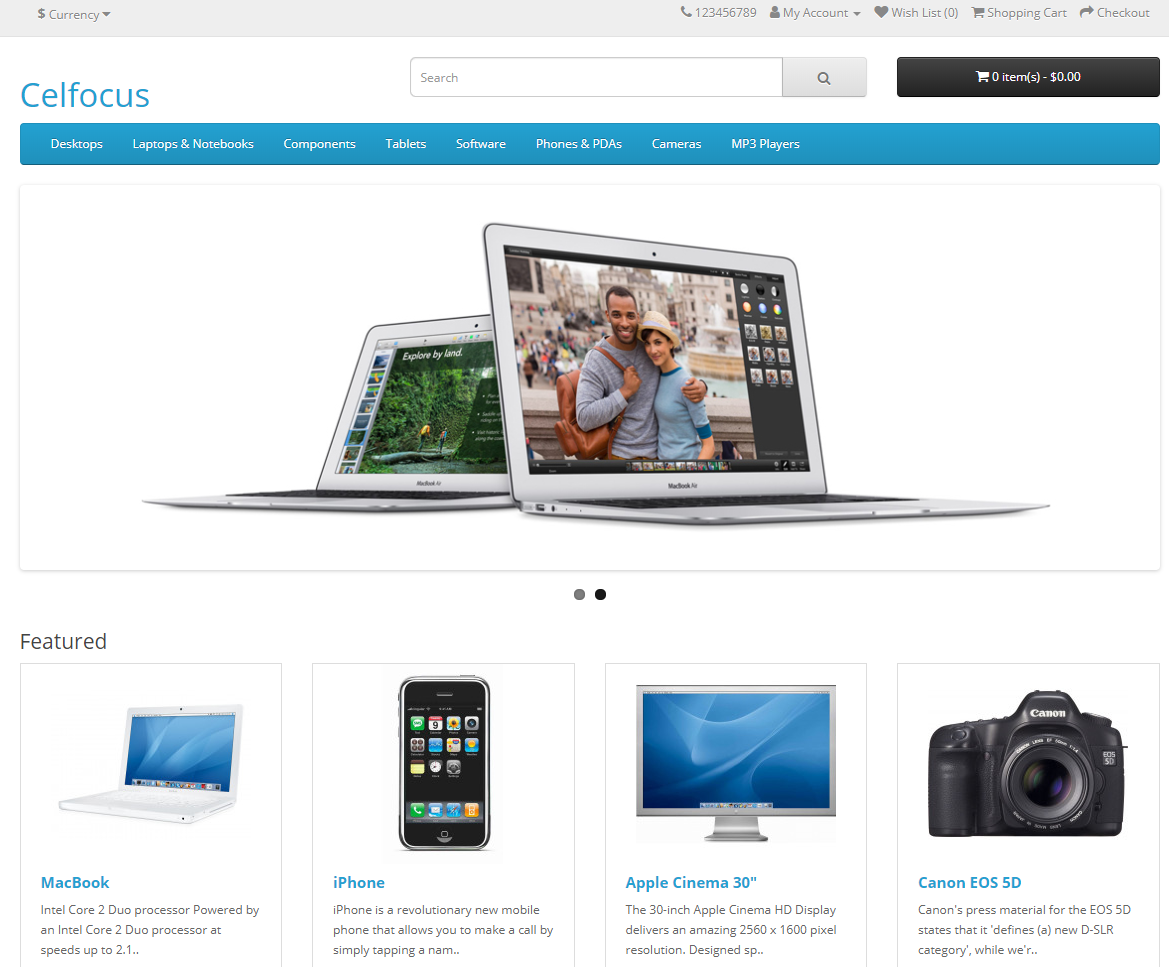
1. Go to Canada regions and copy snapshot eshop-base to your region

2. Go to RDS console and create a new Subnet Group, and add the **2 private subnets**

3. Once snapshot is created, restore from you region. Choose a **t2.micro** RDS instance and **Single AZ** and **eshop-db** as instance identifier. Don’t forget to add Subnet Group ceated before, instance should be private.

TIP: You should always choose Multi-AZ RDS instances for resilience purposes, not performance! In this case we choose Single AZ only to reduce costs.

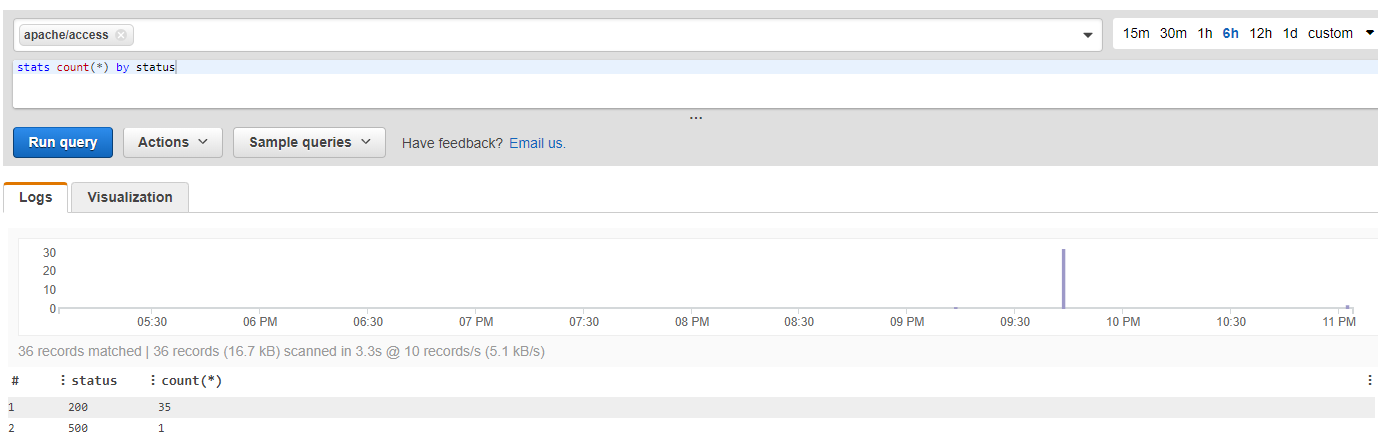
Congrats! If communication between frontend and backend was correctly done, you should see the eShop.



INFO: Despite the fact solution is working, it’s clearly an anti-pattern. Your frontend and backend is not in the same Region, RDS is single AZ, and EC2 instance is not on an Auto Scale group!

CHALLENGE!: Reverse roles, Student B to Setup the Frontend and Student A to Setup RDS

**Exercise 9 – Explore Cloudwatch Insights and Dashboards**

Explore eShop website, force some **404 pages** and go to Cloudwatch Logs Insights and perform this query to **Apache access log** [***stats count(\*) by status***]  
  
1. Create a new **WEB-<REGION>** dashboard and add this HTTP Status metric above

2. Add EC2 memory and CPU consumption to the same dashboard.

TIP: Memory is not given out-of-the-box, only with Cloudwatch agent is possible to have this metric from EC2.

**Exercise 10 – eShop Autoscale**

In this exercise, we need to change architecture to improve resilience to the Application Layer. We need to add an Application Load Balancer and setup an Autoscale group based on CPU consumption to eShop solution.  
You should be able to access eShop via Applicaton Load Balancer endpoint instead single EC2 Public IP address.

TIP: You need to preconfigure an image with opencart configuration adding ALB endpoint and RDS endpoint as well, to configure Auto-scaling Lauch Configuration

1. Create a new Security Group for ALB to allow ingress on Port 80

2. Create an Application Load Balancer

3. Change current EC2 with and setup ALB endpoint on **HTTP\_SERVER** section

4. Create an image based on this EC2 and setup the Launch Configuration/Auto-scaling (Scale Policy Min:2/Max:4)

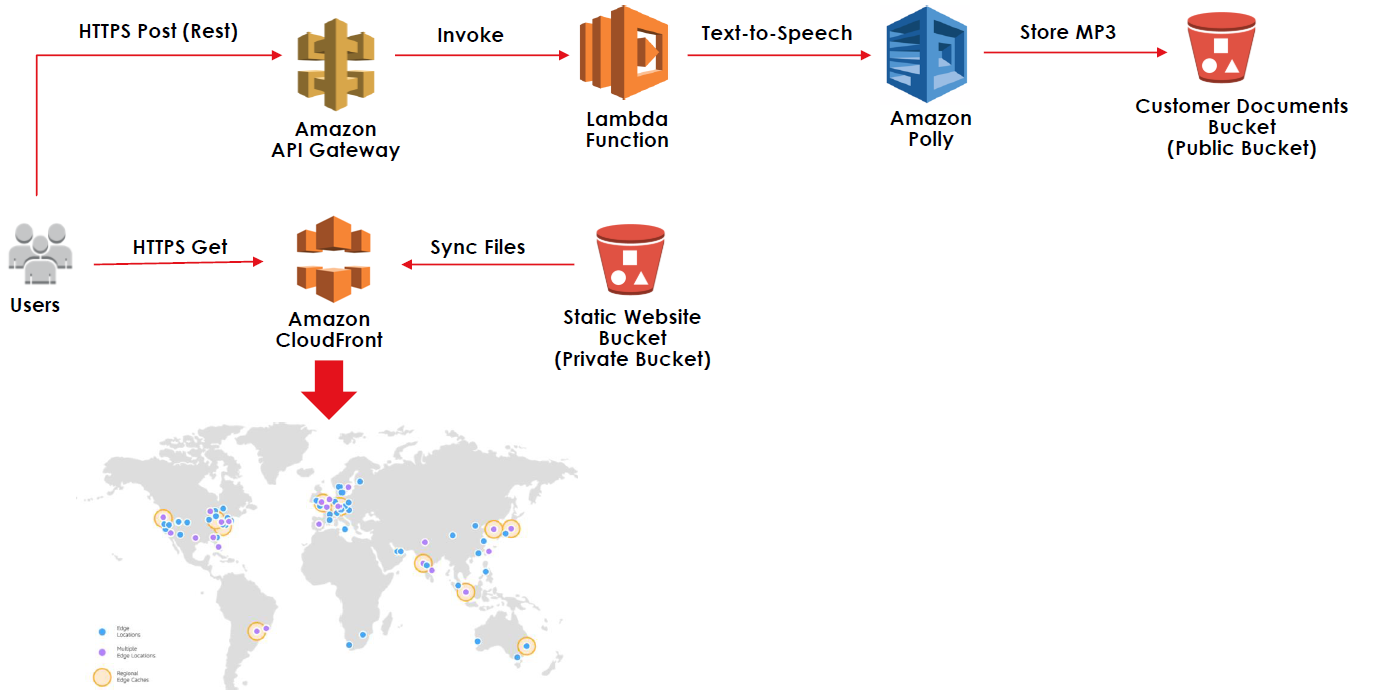
5. Change cloudwatch dashboard to show metrics of the Autoscale group instead of the EC2

CHALLENGE!: Use Jmeter or some other tool to test Autoscale. Terminate all EC2 to analyze behaviour and resilience of the frontend layer

**Exercise 11 – Serverless Solution**

Serverless allows you to build and run applications and services without thinking about servers. It eliminates infrastructure management tasks such as server or cluster provisioning, patching, operating system maintenance, and capacity provisioning.  
In this exercise you will build a solution with no EC2 instances, intead you will power your application with Lambda functions.

**Scenario:**   
Celfocus wins a new project for an Indian mobile operator – India Telecom  
This Telco company will start with a Pre-paid business, that will scale fast  
You need to build a high available frontend/backend application to support BSS - Process at peak load 10K topups/second  
All customers receive a voice message after Topup confirmation, English with Indian accent and Hindi (Native) is mandatory  
  
**Requirements:**  
**Customers**: Expected customers between 50K and 300M on first month  
**Demand**: Aggressive marketing campaigns can generate up to 10K top-up requests/second   
**Website**: Payment website using SSL and geographically distributed over 5 continents to improve UX  
**Availability:** 99.99% website availability and capability to handle 100K requests/second   
**Governance:** No Ops teams to manage servers and OS patching  
**Compliance**: Customer’s documents must be stored in a single region  
**Welcome Message**: India Telecom is a Customer-Focused Company, each top-up request must be acknowledged with a voice message of customer choice: English (Indian accent)/Hindi (Native)/Portuguese/French   
To help you on this, find below image with desired architecture for this setup. All this services will handle requirements by default.



1. Create a new Lambda with Runtime Python 3.9 and choose the role from IAM that will allow to interact with Polly(TrainingS3Polly). All lambda code can be found here - [**https://raw.githubusercontent.com/vitorfaustino/aws-csaa-training/master/serverless/lambda.py**](https://raw.githubusercontent.com/vitorfaustino/aws-csaa-training/master/serverless/lambda.py)  
  
2. Create a new Rest API Gateway in your region, and under resources a POST Method to point to your Lambda. You need to deploy your API, for example Stage:PROD. This will create a new endpoint you will need to reference from the frontend.

TIP: You might need to adjust Lambda timeout and add Lambda Parameters for the bucket to store the files.  
You can test your function with a simple JSON file, example:  
{   
"name": "Vitor",   
"msisdn": "918319400",  
 "topup\_qty": "23",  
 "language": "portuguese",  
 "message": "Queremos passar no exame A W S"  
}

3. Create a private S3 and a Cloudfront to host your frontend, please find all code here - [**https://github.com/vitorfaustino/aws-csaa-training/tree/master/serverless/cloudfront\_website**](https://github.com/vitorfaustino/aws-csaa-training/tree/master/serverless/cloudfront_website)Be aware that all communiction must be https and URL should point do index.html.   
  
4. You will need to edit index.html and add API Gateway endpoint at the bottom of the file.

TROUBLESHOOT: You are getting errors using frontend? Check you browser console and try to find out how to overcome this.

**Exercise 12 – Terraform (Bonus exercise)**

Terraform is an open-source infrastructure as code software tool created by HashiCorp. Users define and provide data center infrastructure using a declarative configuration language known as HashiCorp Configuration Language (HCL).  
Bear in mind that Terraform is not part of any AWS exam, you probably can get Cloudformation questions instead. Both Terraform and Cloudformation are good ways to automate infrastructure.

1. If you have several AWS profiles, set the right one. For example if your profile name is default, open commnad line and execute: set AWS\_PROFILE=default

2.Clone terraform project from GIT here - <https://github.com/vitorfaustino/terraform-training.git>

3.Download and extract latest terraform binary version available here - <https://releases.hashicorp.com/terraform/1.0.9/terraform_1.0.9_windows_amd64.zip>

4. Copy binary to git folder, you can perform a plan or apply it to create infrastructure using your own AWS access key already configured:

terraform plan

terraform apply

5. Adapt the code to create an EC2 with same configuration found in exercise 2, but using terraform automation this time. Virtual machine should be placed into public subnets already referenced in the code. Output elastic IP created and use it to test the web server.

6. Run destroy command to cleanup.