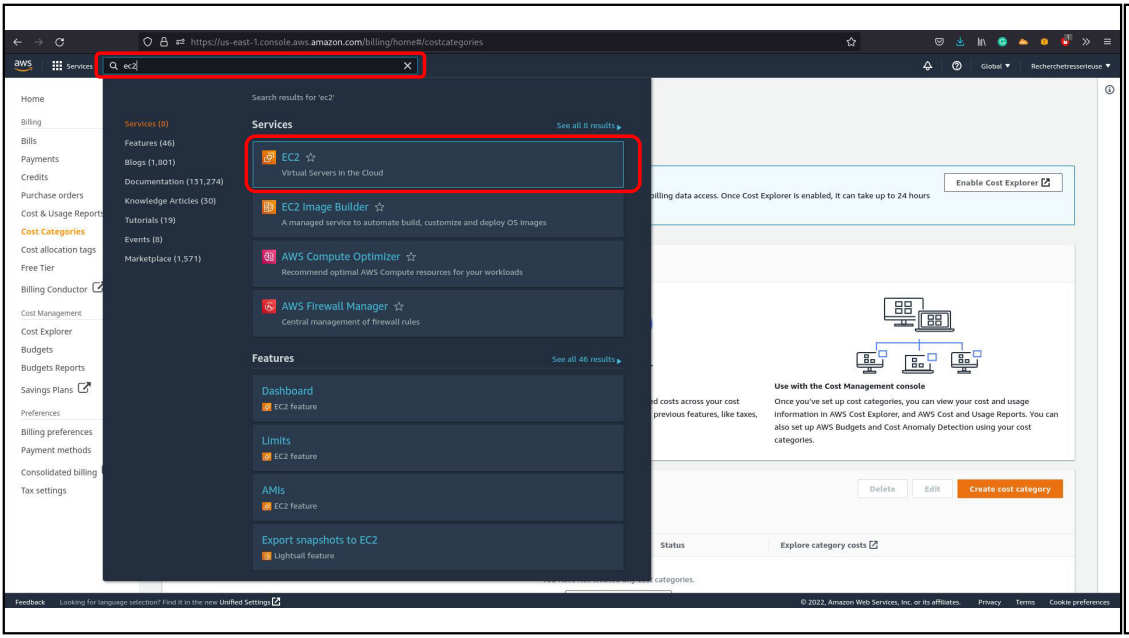
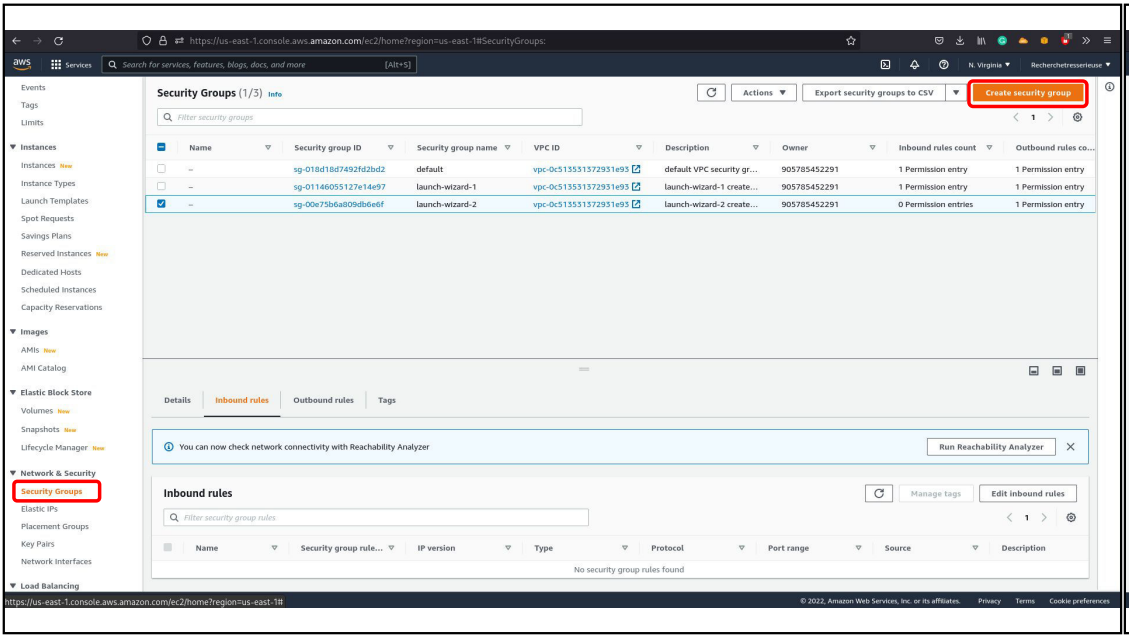
AWS Lab Work

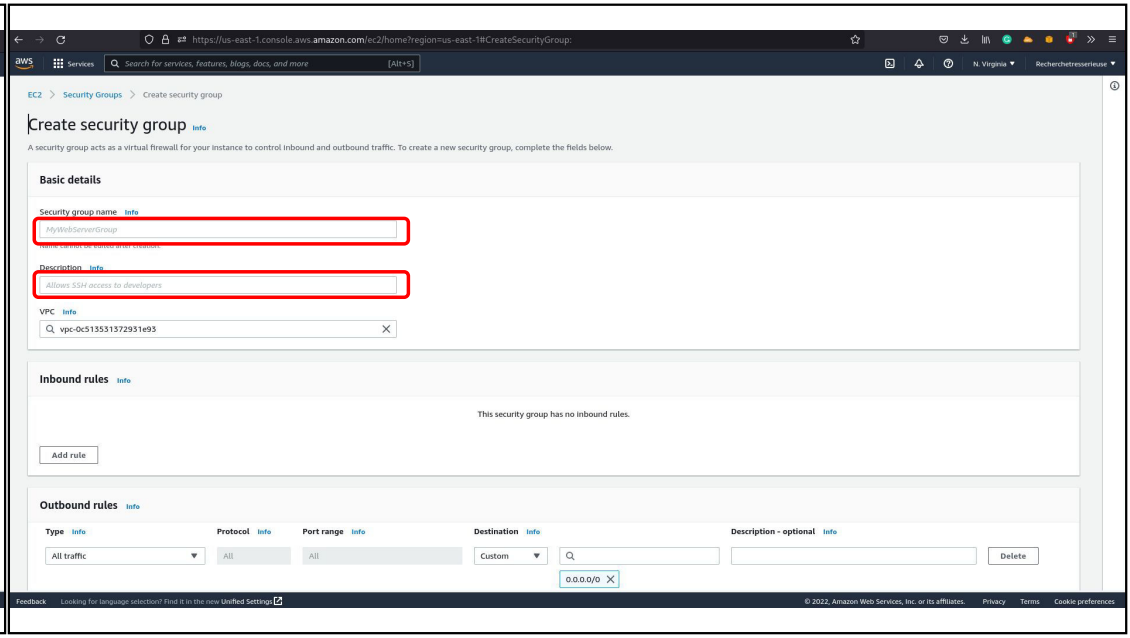
Boris TEABE
boris.teabe@inp-toulouse.fr

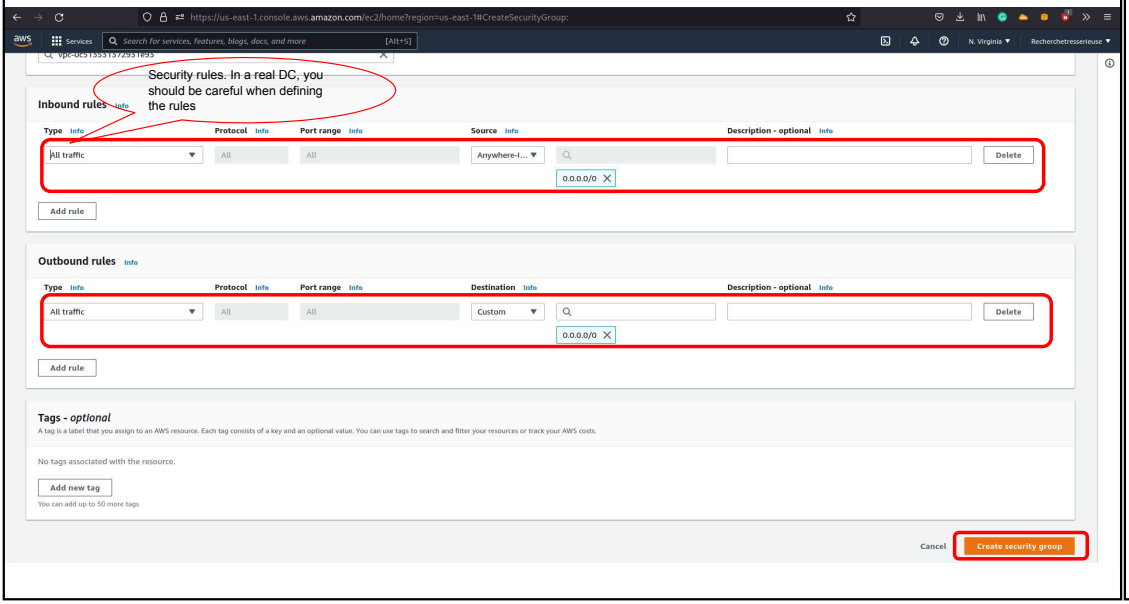
EC2 Service of AWS



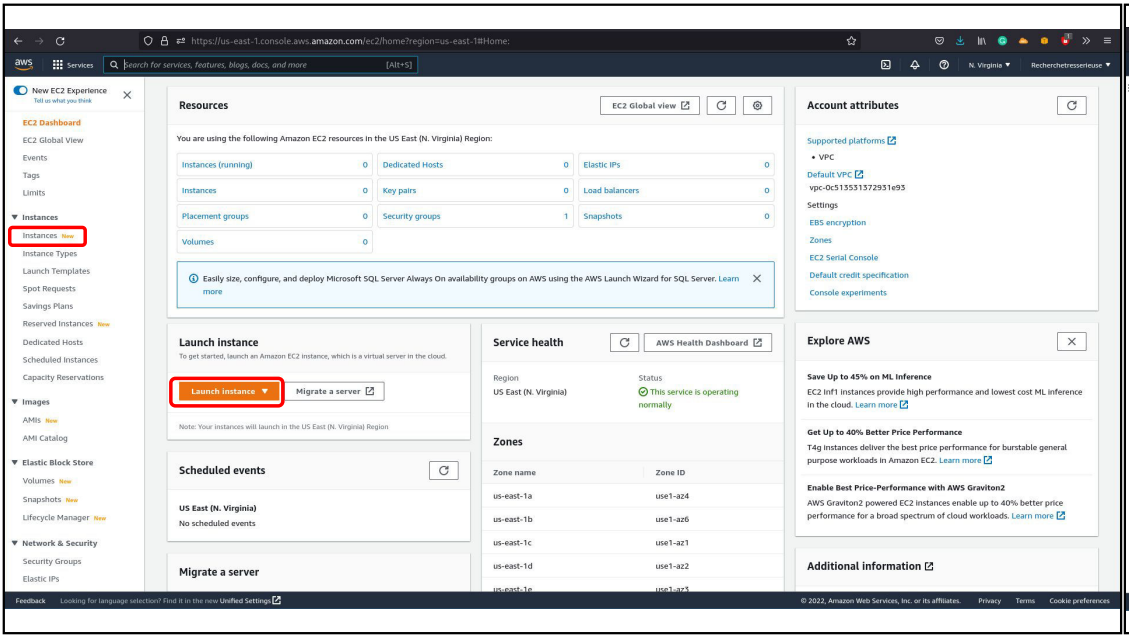
Security group creation (firewall)

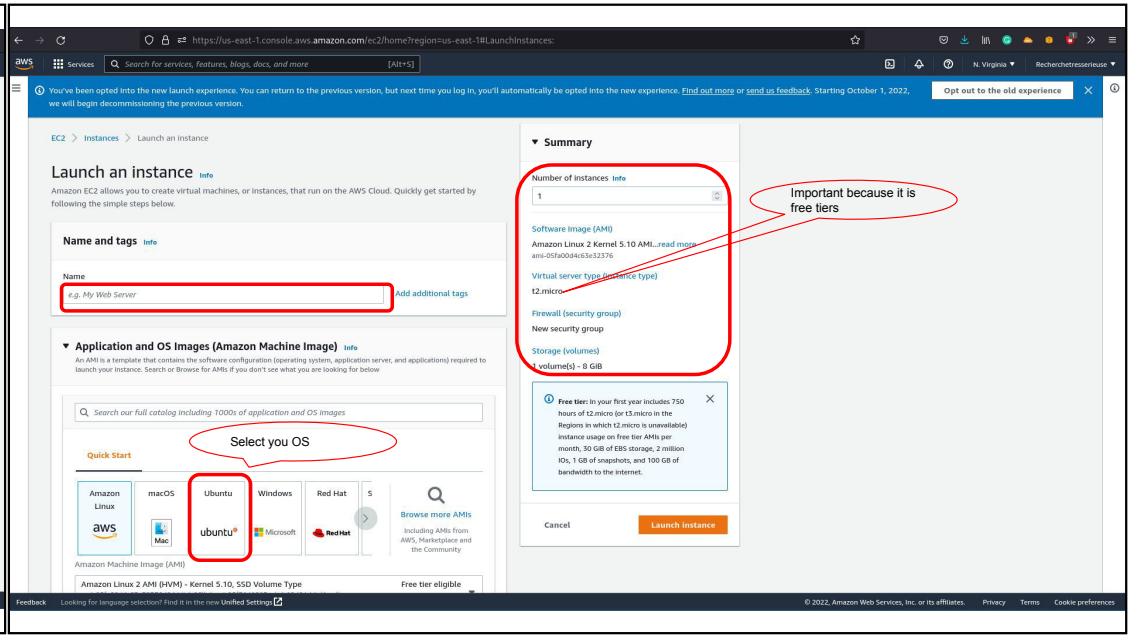


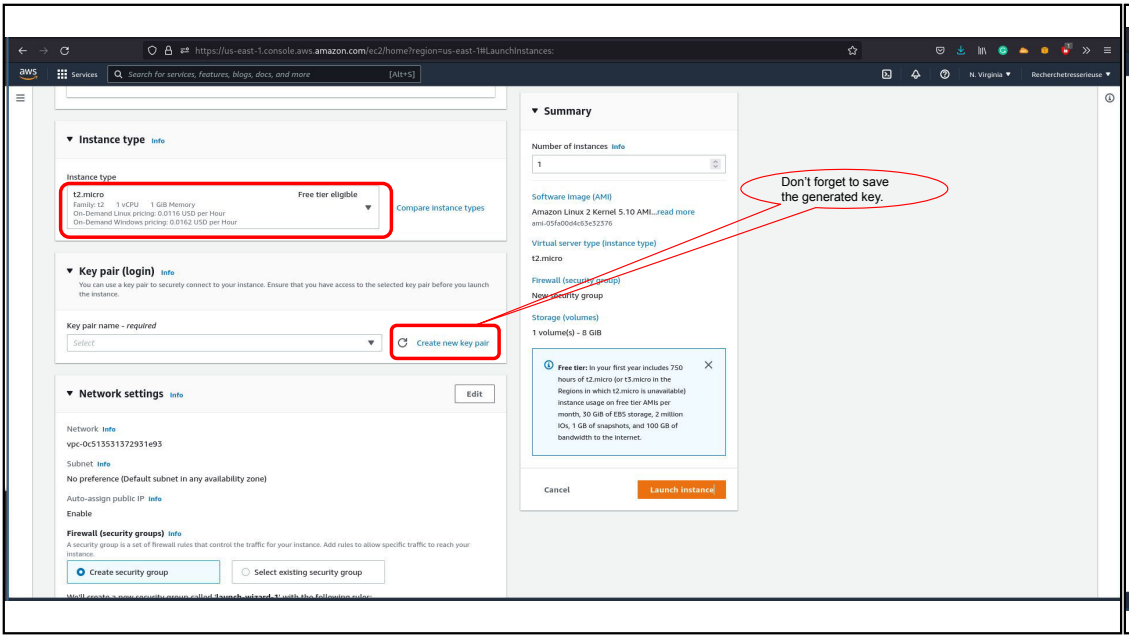


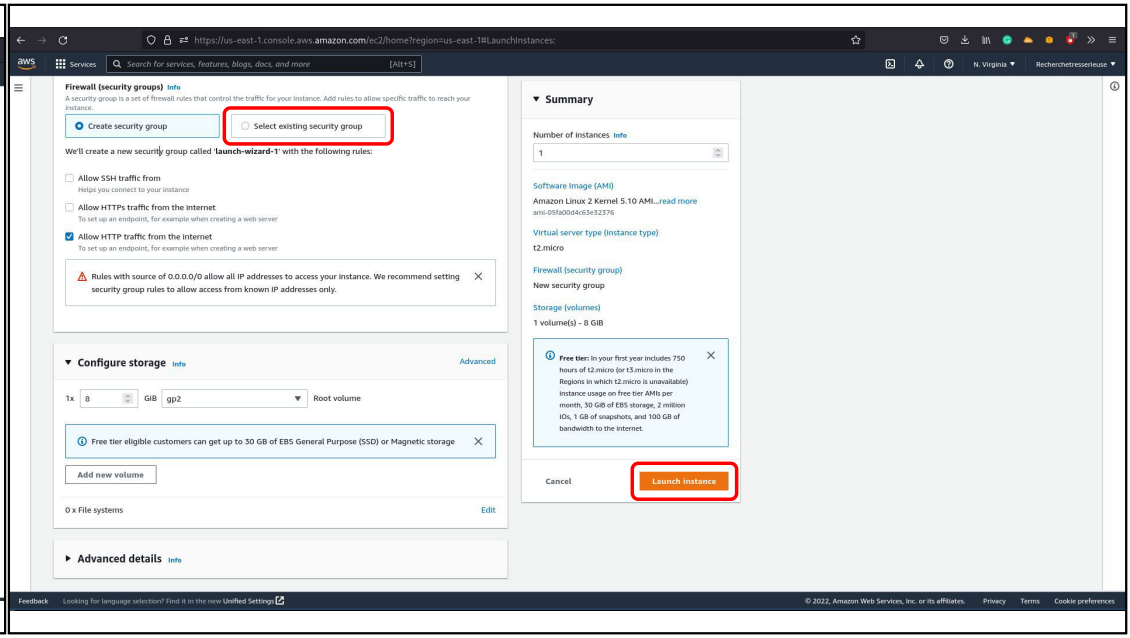


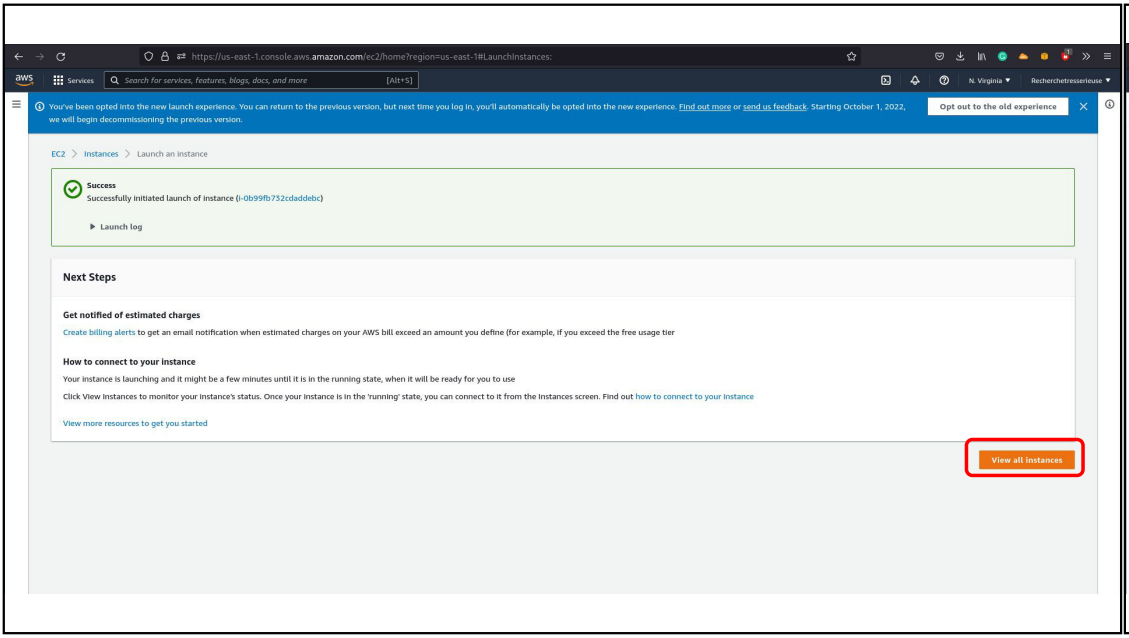
Create a new VM

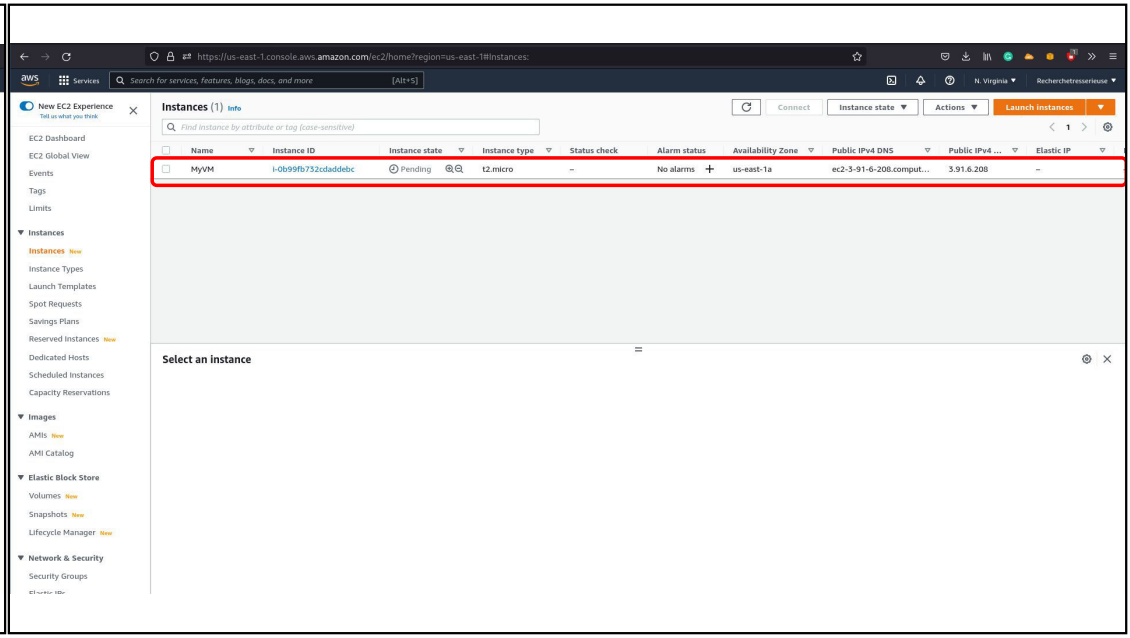

Launch instance
Migrate a server

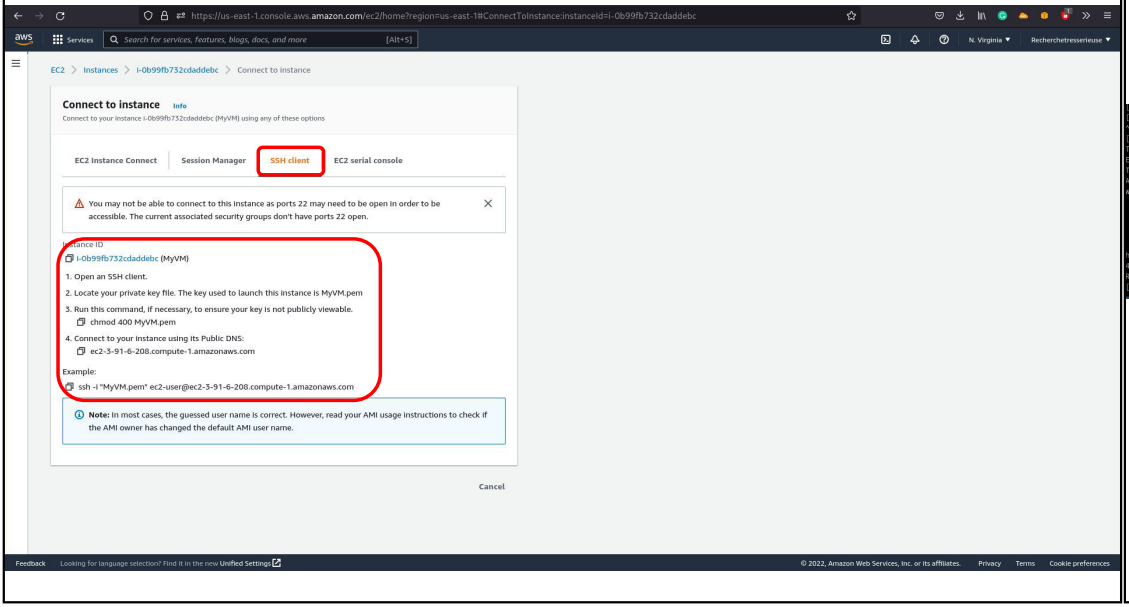

Important because it is free tiers
Select you OS


Don't forget to save the generated key.
Create new key pair

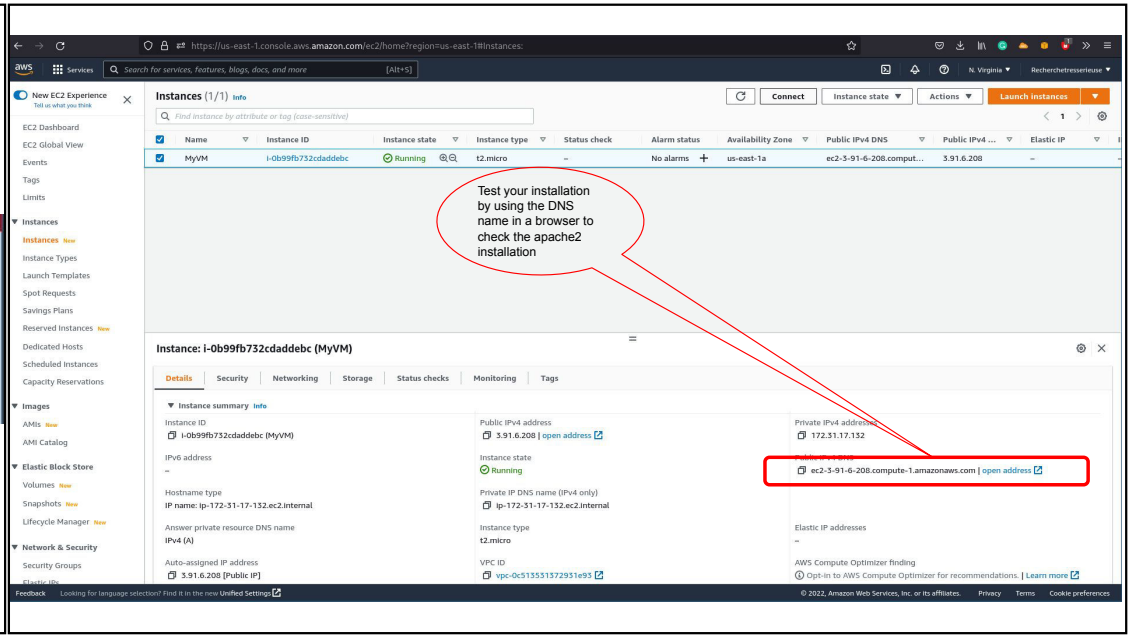

Launch instance


You've been opted into the new launch experience. You can return to the previous version, but next time you log in, you'll automatically be opted into the new experience. Find out more or send us feedback. Starting October 1, 2022, we will begin decommissioning the previous version.
Opt out to the old experience
EC2 > Instances > Launch an instance
Success
Successfully initiated launch of instance (i-0b99fb732cdaddebc)
Launch log
Next Steps
Get notified of estimated charges
Create billing alerts to get an email notification when estimated charges on your AWS bill exceed an amount you define (for example, if you exceed the free usage tier
How to connect to your instance
Your instance is launching and it might be a few minutes until it is in the running state, when it will be ready for you to use
Click View Instances to monitor your instance's status. Once your instance is in the 'running' state, you can connect to it from the Instances screen. Find out how to connect to your instance
View more resources to get you started
View all instances

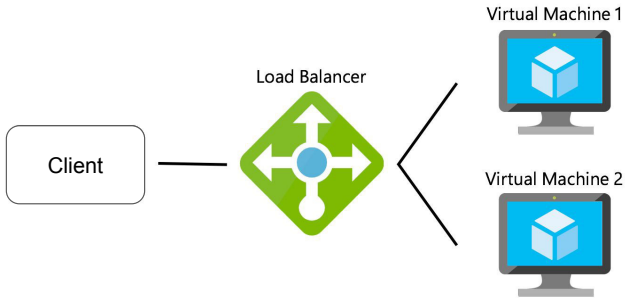

New EC2 Experience
EC2 Dashboard
EC2 Global View
Events
Tags
Limits
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Scheduled Instances
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Instances (1) Info
Connect
Instance state
Actions
Launch instances
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Public IPv4 DNS
Public IPv4 ...
Elastic IP
MyVM
i-0b99fb732cdaddebc
Pending
t2.micro
No alarms
us-east-1a
ec2-3-91-6-208.comput...
3.91.6.208
Select an instance

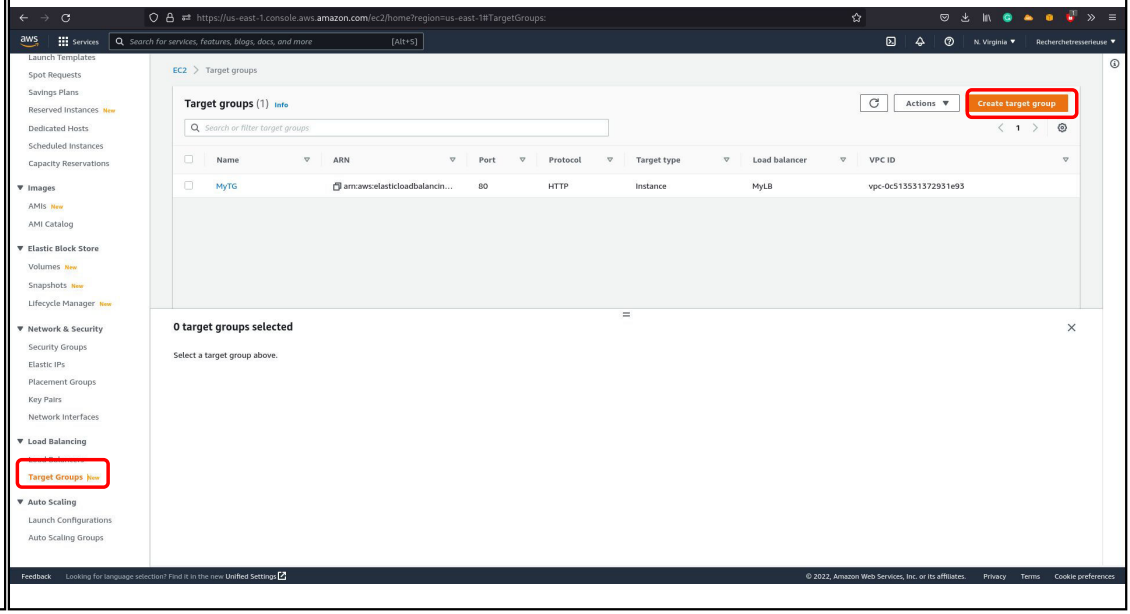

EC2 > Instances > i-0b99fb732cdaddebc > Connect to instance
Connect to instance Info
Connect to your instance i-0b99fb732cdaddebc (MyVM) using any of these options
EC2 Instance Connect
Session Manager
SSH client
EC2 serial console
You may not be able to connect to this instance as ports 22 may need to be open in order to be accessible. The current associated security groups don't have ports 22 open.
Instance ID
i-0b99fb732cdaddebc (MyVM)
1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is MyVM.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
chmod 400 MyVM.pem
4. Connect to your instance using its Public DNS:
ec2-3-91-6-208.compute-1.amazonaws.com
Example:
ssh -i "MyVM.pem" ec2-user@ec2-3-91-6-208.compute-1.amazonaws.com
Note: In most cases, the guessed user name is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI user name.
Cancel
Feedback
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

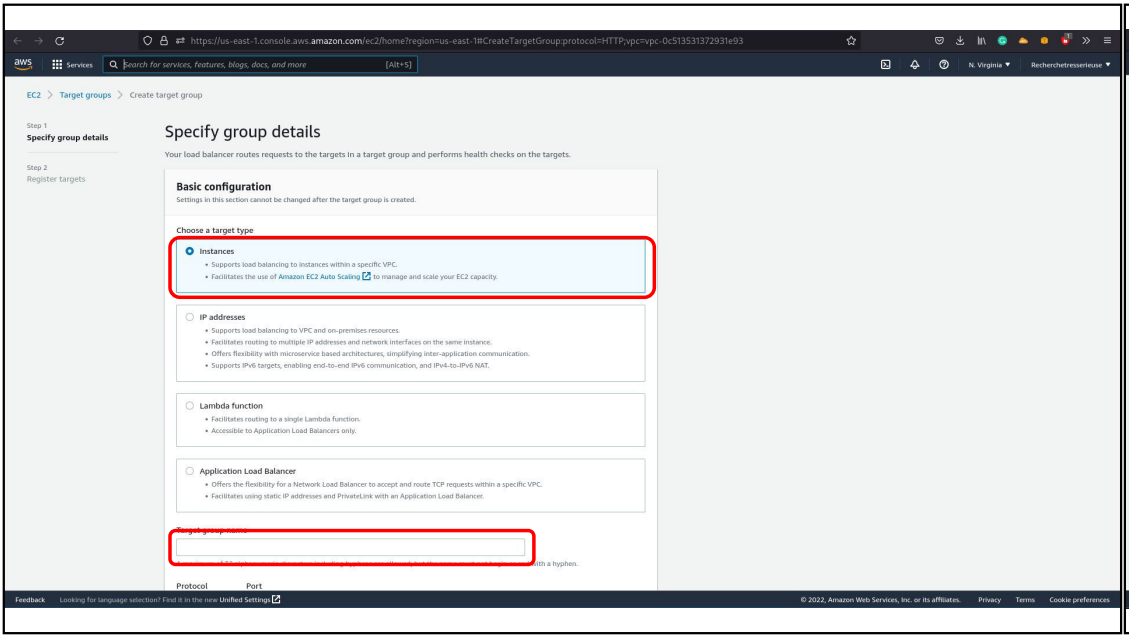

[tab@Tab-pc Downloads]$ chmod 400 MyVM.pem
[tab@Tab-pc Downloads]$ ssh -i "MyVM.pem" ec2-user@ec2-3-91-6-208.compute-1.amazonaws.com
^C
[tab@Tab-pc Downloads]$ ssh -i "MyVM.pem" ec2-user@ec2-3-91-6-208.compute-1.amazonaws.com
The authenticity of host [redacted] can't be established.
ED25519 key fingerprint is SHA256:9X96VZBpA6YnDIfC3Hey9P/dBrLsuw+cmJqiwvdbYAM.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-91-6-208.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
Amazon Linux 2 AMI
https://aws.amazon.com/amazon-linux-2/
4 package(s) needed for security, out of 11 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-17-132 ~]$

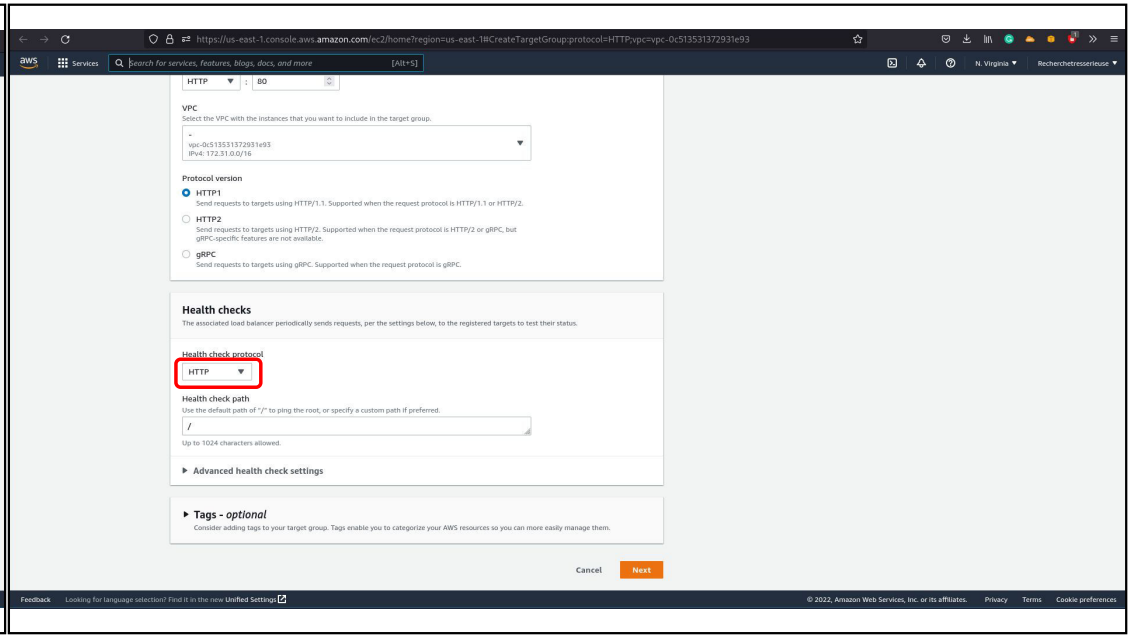


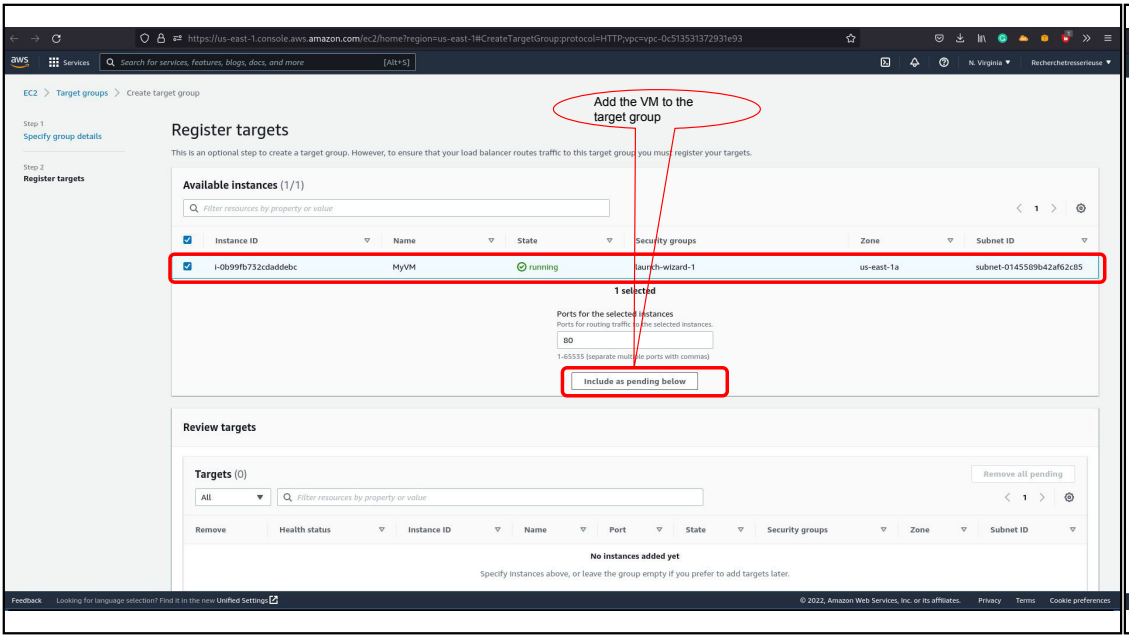
Create a Load balancer

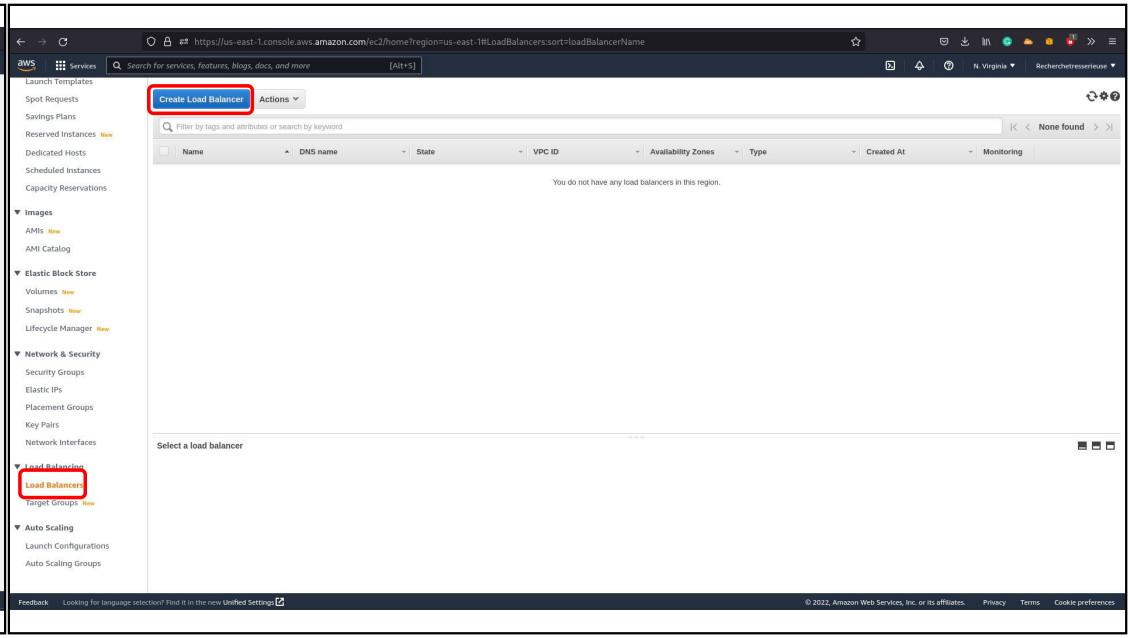


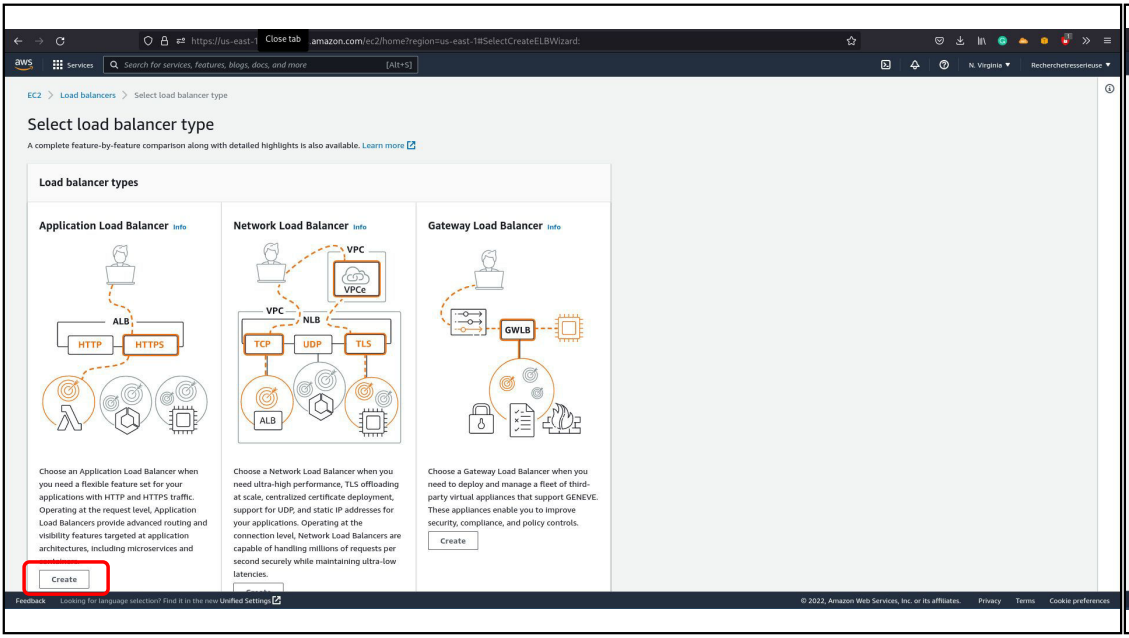


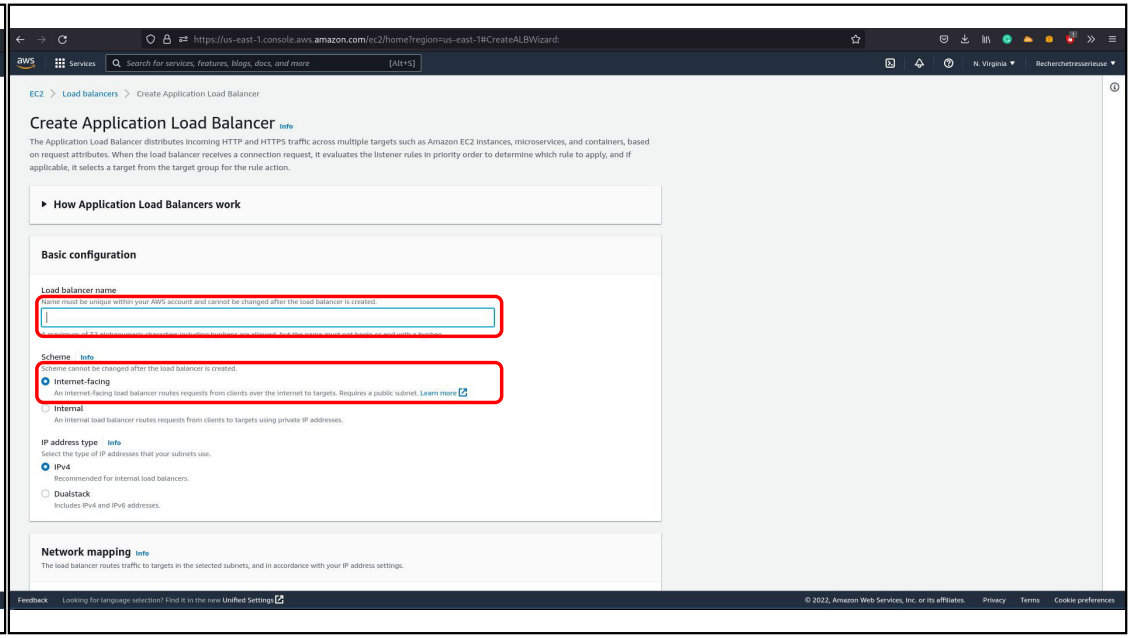

EC2 > Target groups > Create target group
Step 1
Specify group details
Step 2
Register targets
Specify group details
Your load balancer routes requests to the targets in a target group and performs health checks on the targets.
Basic configuration
Settings in this section cannot be changed after the target group is created.
Choose a target type
Instances
Supports load balancing to instances within a specific VPC.
Facilitates the use of Amazon EC2 Auto Scaling to manage and scale your EC2 capacity.
IP addresses
Supports load balancing to VPC and on-premises resources.
Facilitates routing to multiple IP addresses and network interfaces on the same instance.
Offers flexibility with microservice based architectures, simplifying inter-application communication.
Supports IPv6 targets, enabling end-to-end IPv6 communication, and IPv4-to-IPv6 NAT.
Lambda function
Facilitates routing to a single Lambda function.
Accessible to Application Load Balancers only.
Application Load Balancer
Offers the flexibility for a Network Load Balancer to accept and route TCP requests within a specific VPC.
Facilitates using static IP addresses and PrivateLink with an Application Load Balancer.
Protocol
Port

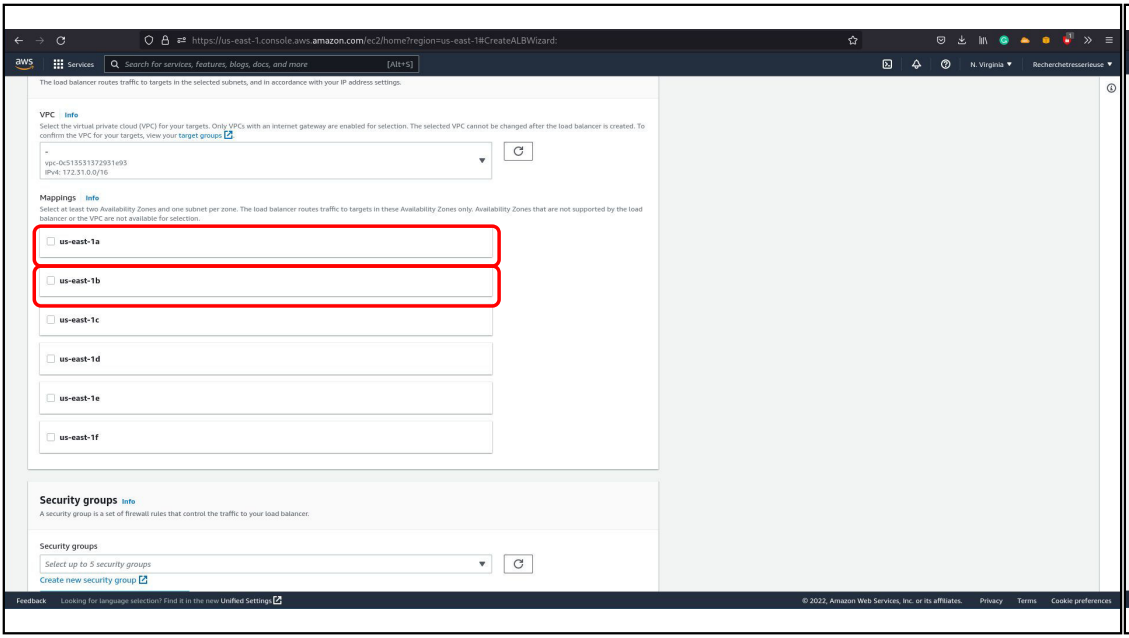

HTTP
80
VPC
Select the VPC with the instances that you want to include in the target group.
Protocol version
HTTP1
Send requests to targets using HTTP/1.1. Supported when the request protocol is HTTP/1.1 or HTTP/2.
HTTP2
Send requests to targets using HTTP/2. Supported when the request protocol is HTTP/2 or gRPC, but gRPC-specific features are not available.
gRPC
Send requests to targets using gRPC. Supported when the request protocol is gRPC.
Health checks
The associated load balancer periodically sends requests, per the settings below, to the registered targets to test their status.
Health check protocol
HTTP
Health check path
Use the default path of "/" to ping the root, or specify a custom path if preferred.
/
Up to 1024 characters allowed.
Advanced health check settings
Tags - optional
Consider adding tags to your target group. Tags enable you to categorize your AWS resources so you can more easily manage them.
Cancel
Next

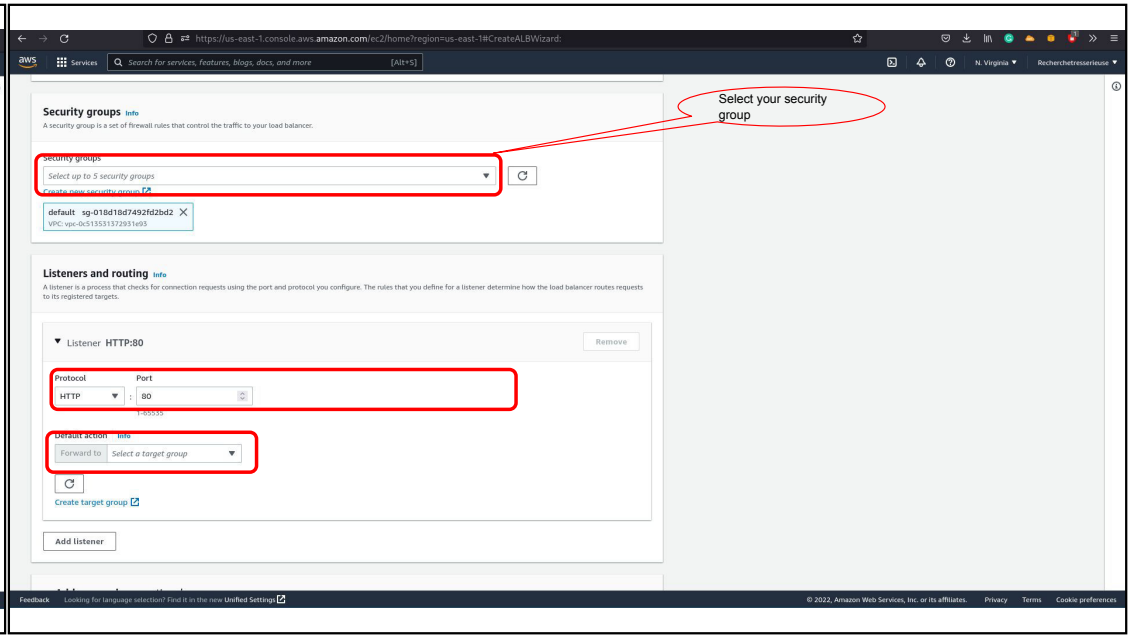

Add the VM to the target group
EC2 > Target groups > Create target group
Step 1
Specify group details
Step 2
Register targets
Register targets
This is an optional step to create a target group. However, to ensure that your load balancer routes traffic to this target group you must register your targets.
Available instances (1/1)
Instance ID
Name
State
Security groups
Zone
Subnet ID
i-0b99fb732cdaddebc
MyVM
running
launch-wizard-1
us-east-1a
subnet-0145589b42af62c85
1 selected
Ports for the selected instances
Ports for routing traffic to the selected instances.
80
1-65535 (separate multiple ports with commas)
Include as pending below
Review targets
Targets (0)
Remove all pending
Remove
Health status
Instance ID
Name
Port
State
Security groups
Zone
Subnet ID
No instances added yet
Specify instances above, or leave the group empty if you prefer to add targets later.

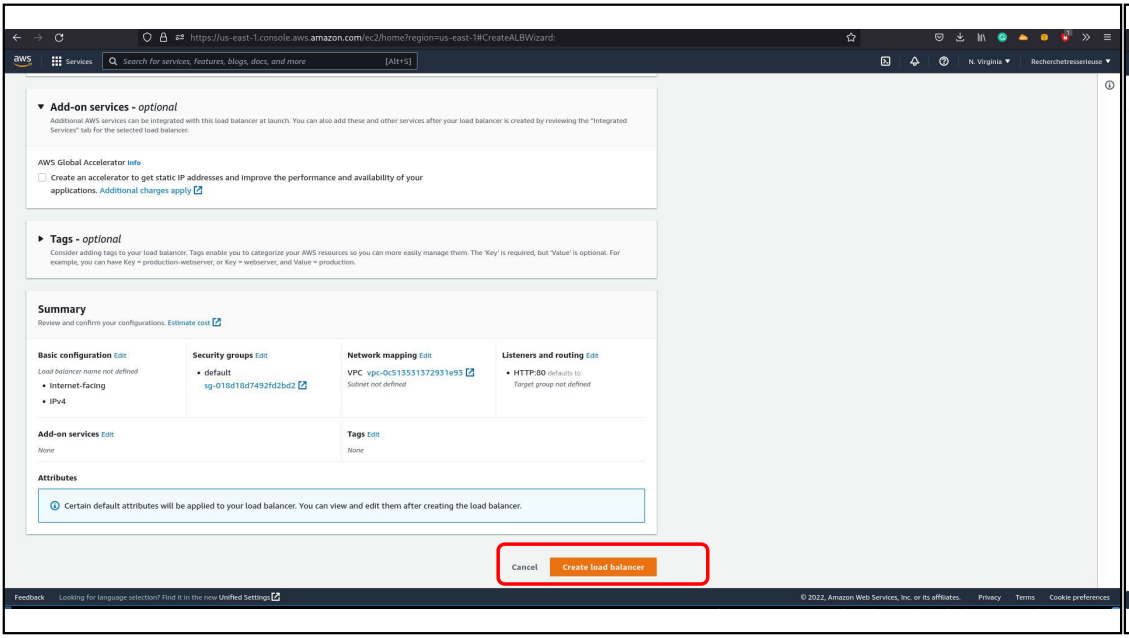

Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Scheduled Instances
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
Load Balancers
Target Groups
Auto Scaling
Launch Configurations
Auto Scaling Groups
Create Load Balancer
Actions
Filter by tags and attributes or search by keyword
None found
Name
DNS name
State
VPC ID
Availability Zones
Type
Created At
Monitoring
You do not have any load balancers in this region.
Select a load balancer

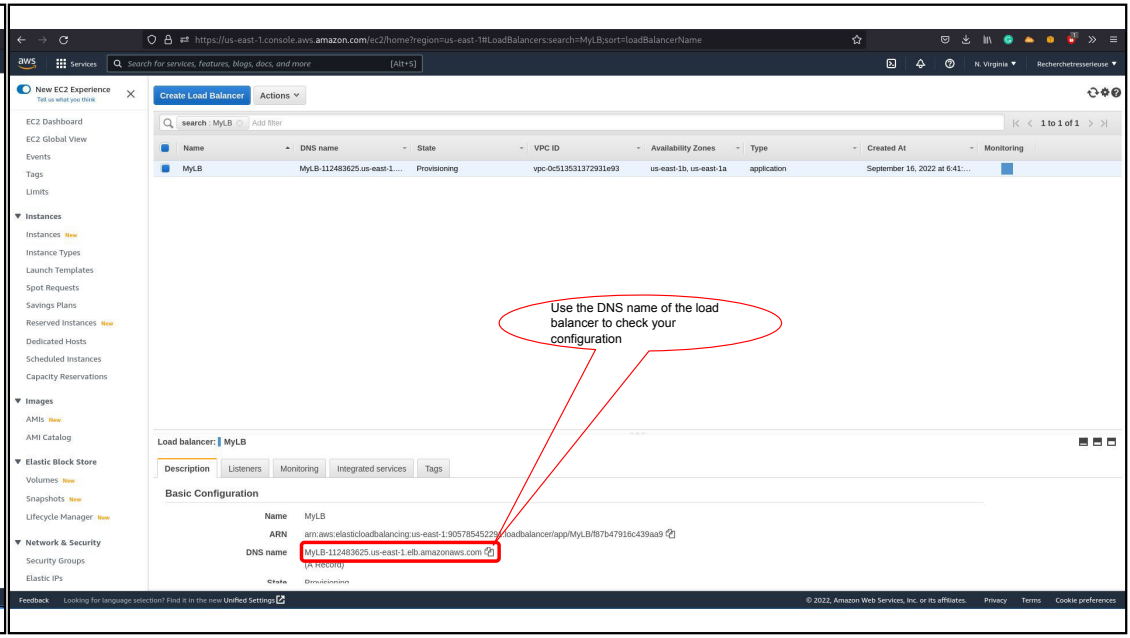

Close tab
EC2 > Load balancers > Select load balancer type
Select load balancer type
A complete feature-by-feature comparison along with detailed highlights is also available. Learn more
Load balancer types
Application Load Balancer Info
Network Load Balancer Info
Gateway Load Balancer Info
ALB
HTTP
HTTPS
VPC
VPCe
NLB
TCP
UDP
TLS
GWLB
Choose an Application Load Balancer when you need a flexible feature set for your applications with HTTP and HTTPS traffic. Operating at the request level, Application Load Balancers provide advanced routing and visibility features targeted at application architectures, including microservices and
Create
Choose a Network Load Balancer when you need ultra-high performance, TLS offloading at scale, centralized certificate deployment, support for UDP, and static IP addresses for your applications. Operating at the connection level, Network Load Balancers are capable of handling millions of requests per second securely while maintaining ultra-low latencies.
Choose a Gateway Load Balancer when you need to deploy and manage a fleet of third-party virtual appliances that support GENEVE. These appliances enable you to improve security, compliance, and policy controls.
Create


EC2 > Load balancers > Create Application Load Balancer
Create Application Load Balancer Info
The Application Load Balancer distributes incoming HTTP and HTTPS traffic across multiple targets such as Amazon EC2 instances, microservices, and containers, based on request attributes. When the load balancer receives a connection request, it evaluates the listener rules in priority order to determine which rule to apply, and if applicable, it selects a target from the target group for the rule action.
How Application Load Balancers work
Basic configuration
Load balancer name
Name must be unique within your AWS account and cannot be changed after the load balancer is created.
Scheme Info
Scheme cannot be changed after the load balancer is created.
Internet-facing
An internet-facing load balancer routes requests from clients over the internet to targets. Requires a public subnet. Learn more
Internal
An internal load balancer routes requests from clients to targets using private IP addresses.
IP address type Info
Select the type of IP addresses that your subnets use.
IPv4
Recommended for internal load balancers.
Dualstack
Includes IPv4 and IPv6 addresses.
Network mapping Info
The load balancer routes traffic to targets in the selected subnets, and in accordance with your IP address settings.


The load balancer routes traffic to targets in the selected subnets, and in accordance with your IP address settings.
VPC Info
Select the virtual private cloud (VPC) for your targets. Only VPCs with an internet gateway are enabled for selection. The selected VPC cannot be changed after the load balancer is created. To confirm the VPC for your targets, view your target groups
vpc-0c513531372931e93
IPv4: 172.31.0.0/16
Mappings Info
Select at least two Availability Zones and one subnet per zone. The load balancer routes traffic to targets in these Availability Zones only. Availability Zones that are not supported by the load balancer or the VPC are not available for selection.
us-east-1a
us-east-1b
us-east-1c
us-east-1d
us-east-1e
us-east-1f
Security groups Info
A security group is a set of firewall rules that control the traffic to your load balancer.
Security groups
Select up to 5 security groups
Create new security group

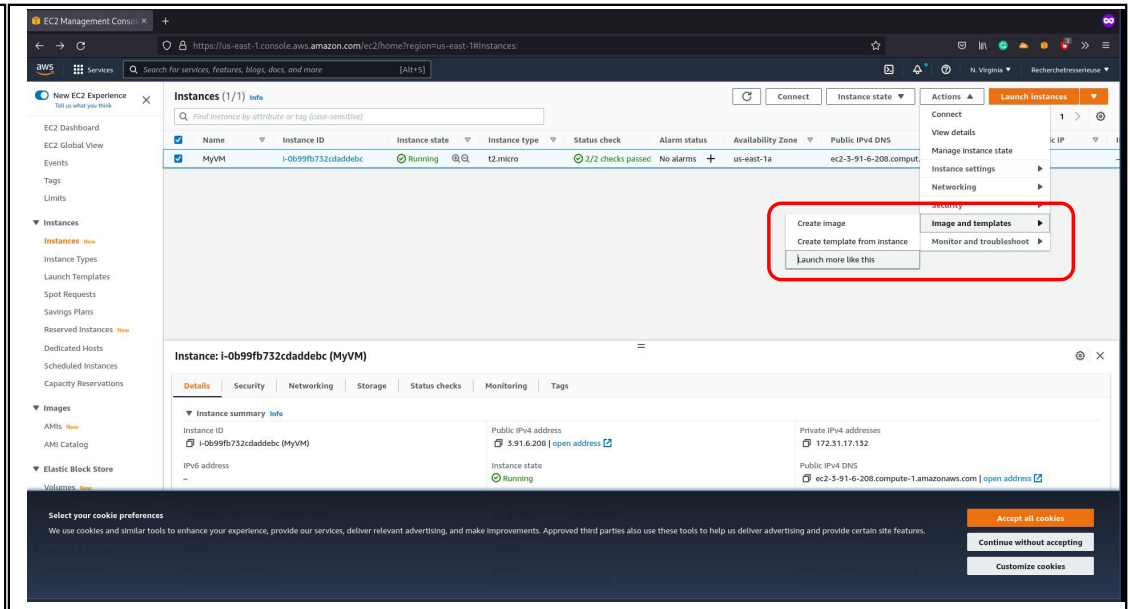

Select your security group
Security groups Info
A security group is a set of firewall rules that control the traffic to your load balancer.
Security groups
Select up to 5 security groups
Create new security group
default sg-018d18d7492fd2bd2
VPC: vpc-0c513531372931e93
Listeners and routing Info
A listener is a process that checks for connection requests using the port and protocol you configure. The rules that you define for a listener determine how the load balancer routes requests to its registered targets.
Listener HTTP:80
Remove
Protocol
Port
HTTP
80
1-65535
Default action Info
Forward to
Select a target group
Create target group
Add listener





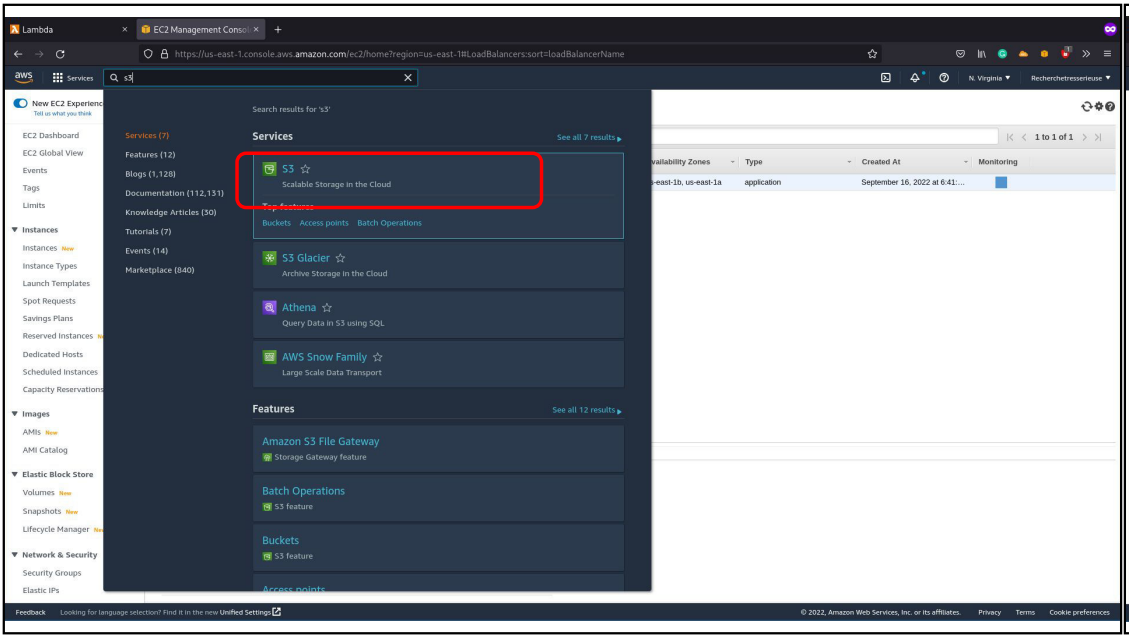
To Do: Add a new VM to you LB

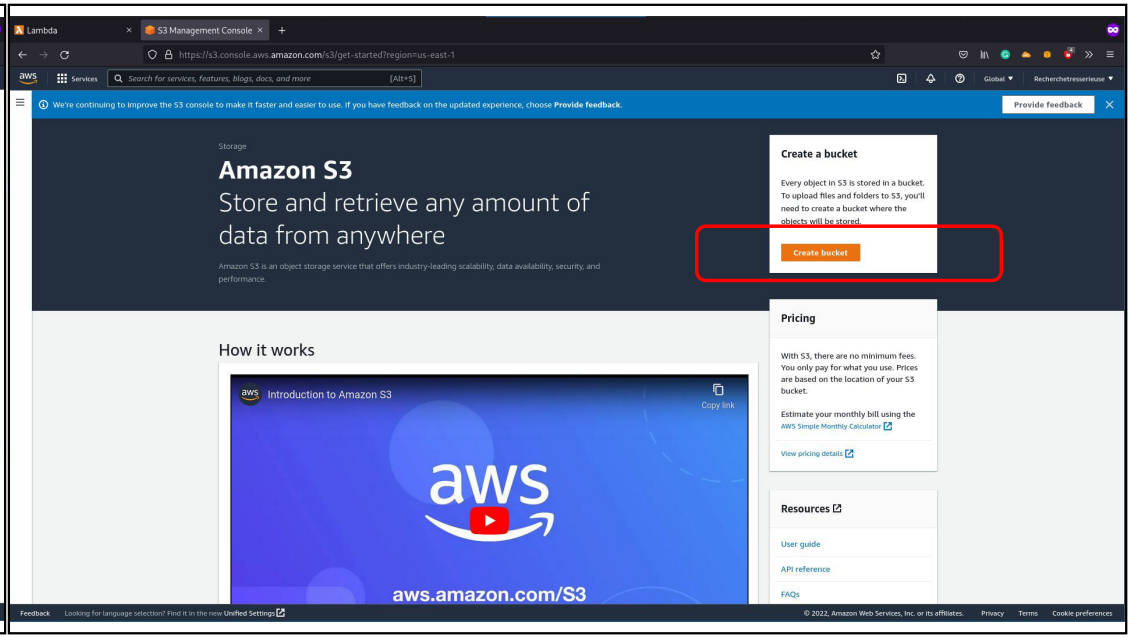
- Therefore, clone your running VM and add the the new VM to the target Group.
- Check if your load balancer balances the load between the two VMs (you can use the curl command from your terminal: ***curl -sv LoadBalancer DNSName***)
    - **Don't forget to connect to the new VM and check that apache2 is installed.**

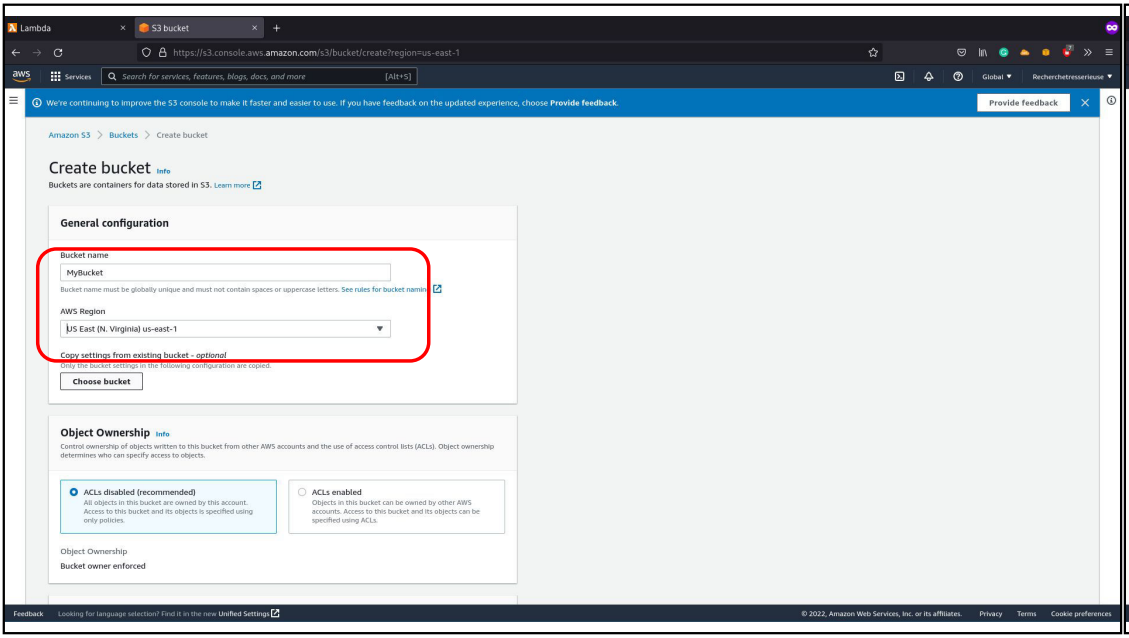


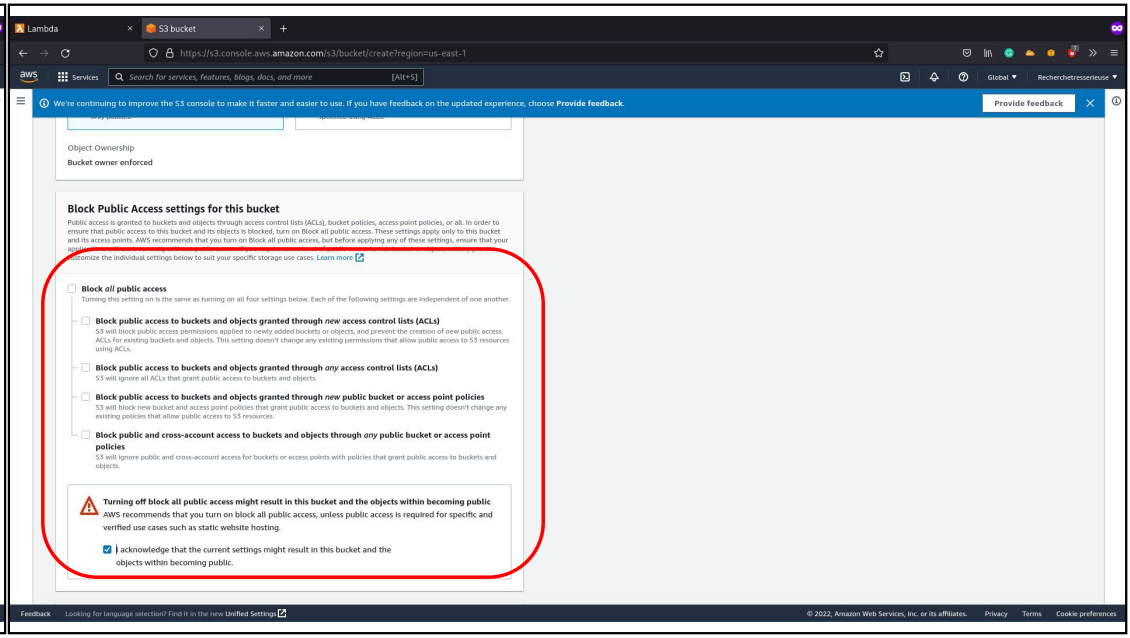
S3 and Lambda function

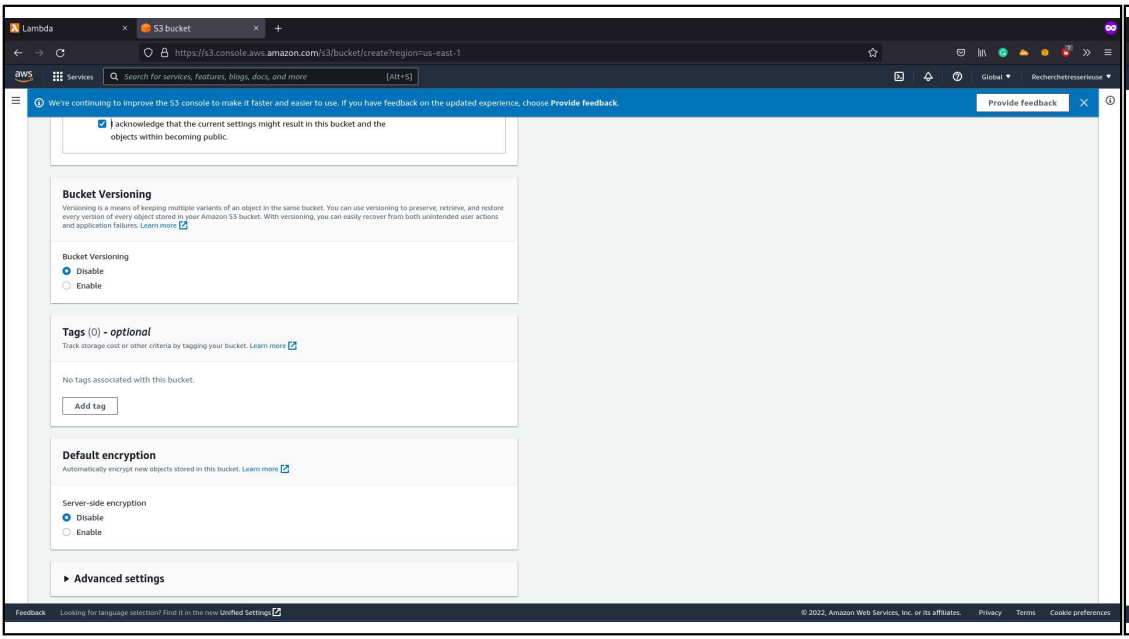
- Add a bucket to S3

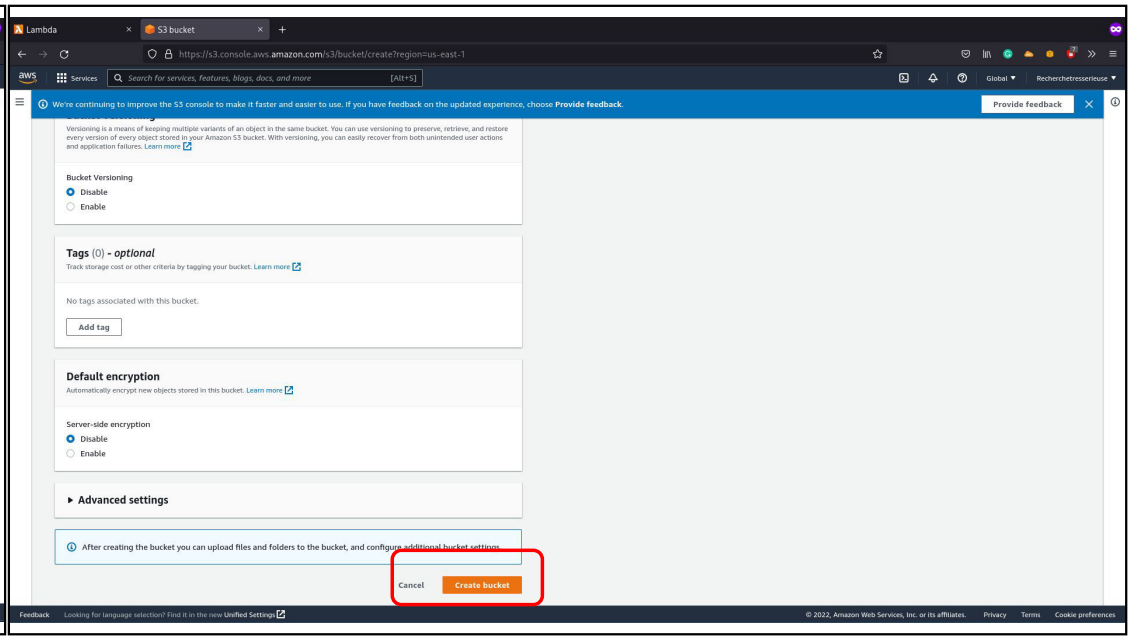


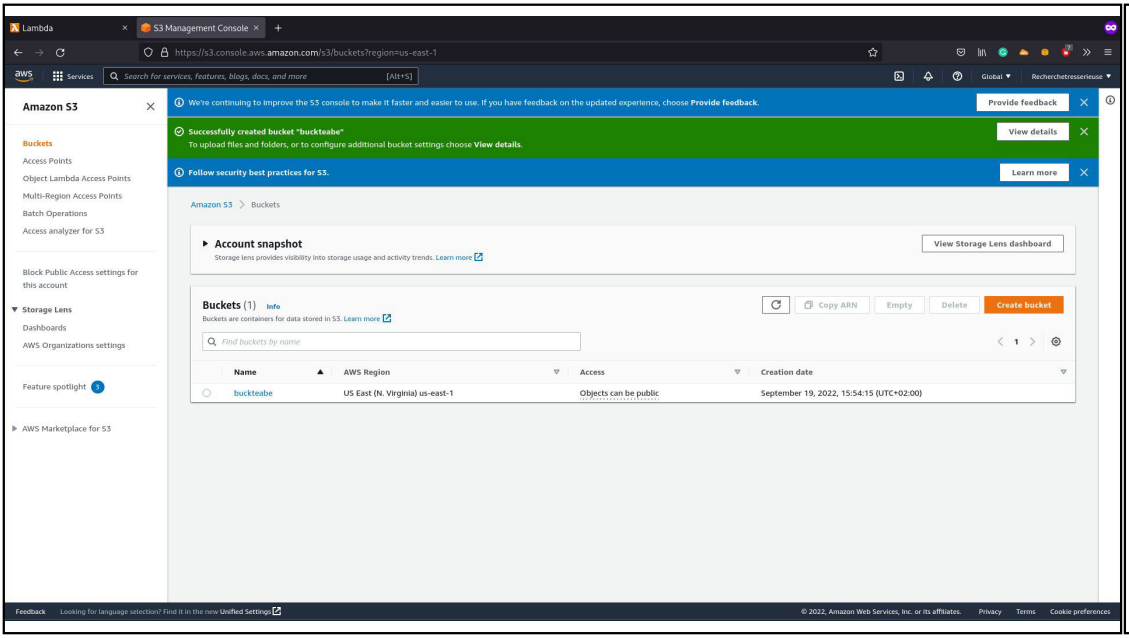



Lambda
S3 bucket
https://s3.console.aws.amazon.com/s3/bucket/create?region=us-east-1
We're continuing to improve the S3 console to make it faster and easier to use. If you have feedback on the updated experience, choose Provide feedback.
Provide feedback
Amazon S3 > Buckets > Create bucket
Create bucket Info
Buckets are containers for data stored in S3. Learn more
General configuration
Bucket name
MyBucket
Bucket name must be globally unique and must not contain spaces or uppercase letters. See rules for bucket naming
AWS Region
US East (N. Virginia) us-east-1
Copy settings from existing bucket - optional
Only the bucket settings in the following configuration are copied.
Choose bucket
Object Ownership Info
Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.
ACLs disabled (recommended)
All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.
ACLs enabled
Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.
Object Ownership
Bucket owner enforced
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

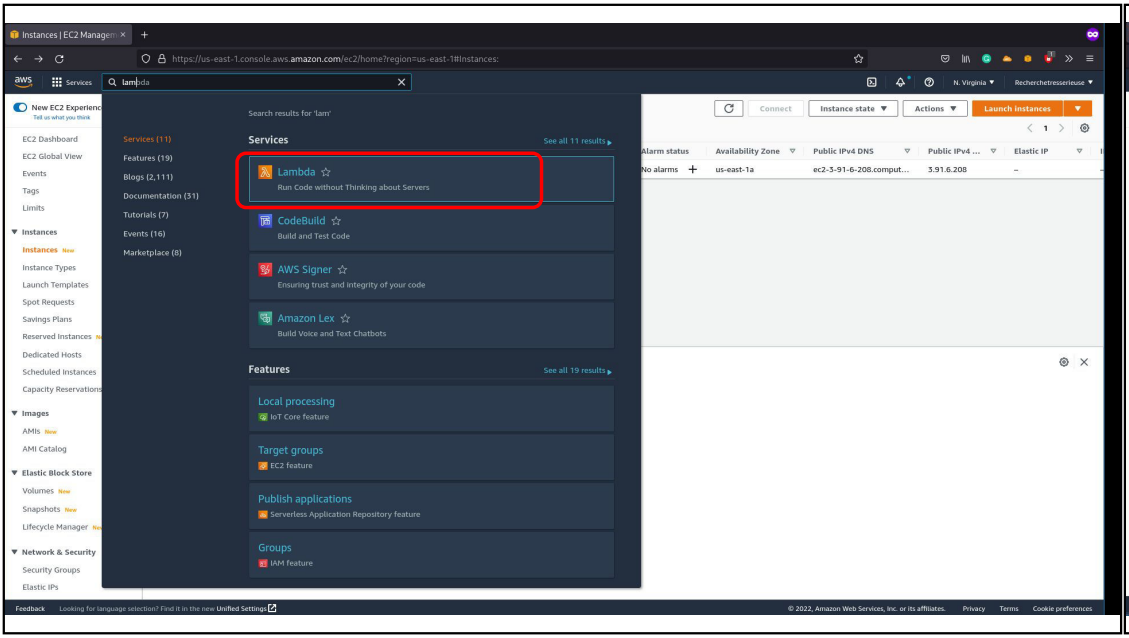

Lambda
S3 bucket
https://s3.console.aws.amazon.com/s3/bucket/create?region=us-east-1
We're continuing to improve the S3 console to make it faster and easier to use. If you have feedback on the updated experience, choose Provide feedback.
Provide feedback
Object Ownership
Bucket owner enforced
Block Public Access settings for this bucket
Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more
Block all public access
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.
Block public access to buckets and objects granted through new access control lists (ACLs)
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
Block public access to buckets and objects granted through any access control lists (ACLs)
S3 will ignore all ACLs that grant public access to buckets and objects.
Block public access to buckets and objects granted through new public bucket or access point policies
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
Block public and cross-account access to buckets and objects through any public bucket or access point policies
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.
Turning off block all public access might result in this bucket and the objects within becoming public
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.
I acknowledge that the current settings might result in this bucket and the objects within becoming public.
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

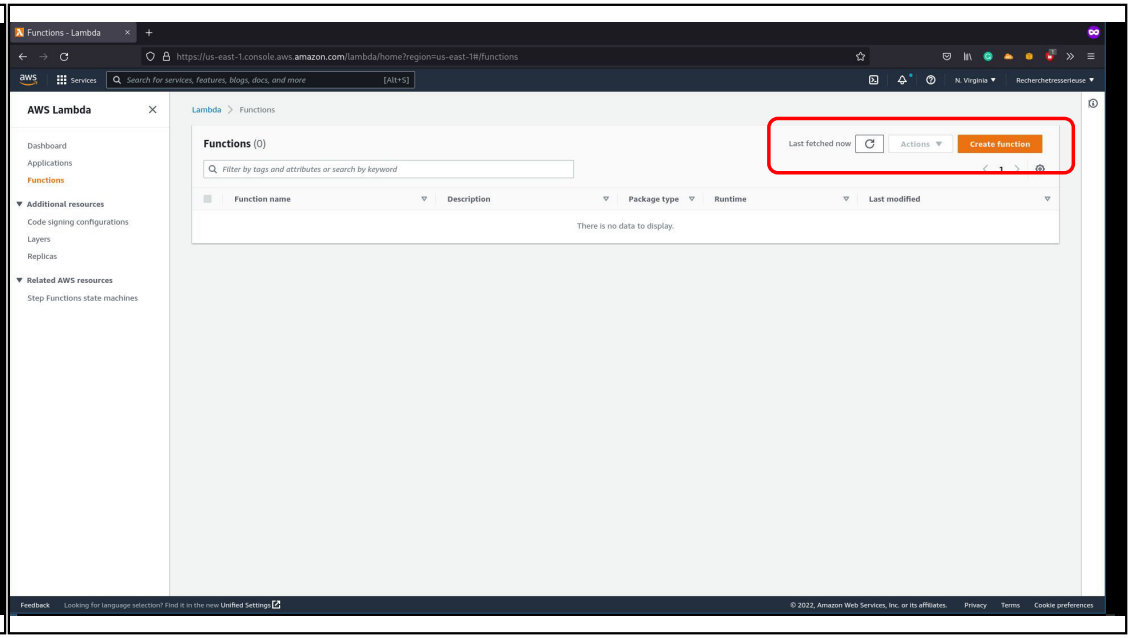

Lambda
S3 bucket
https://s3.console.aws.amazon.com/s3/bucket/create?region=us-east-1
We're continuing to improve the S3 console to make it faster and easier to use. If you have feedback on the updated experience, choose Provide feedback.
Provide feedback
I acknowledge that the current settings might result in this bucket and the objects within becoming public.
Bucket Versioning
Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. Learn more
Bucket Versioning
Disable
Enable
Tags (0) - optional
Track storage cost or other criteria by tagging your bucket. Learn more
No tags associated with this bucket.
Add tag
Default encryption
Automatically encrypt new objects stored in this bucket. Learn more
Server-side encryption
Disable
Enable
Advanced settings
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

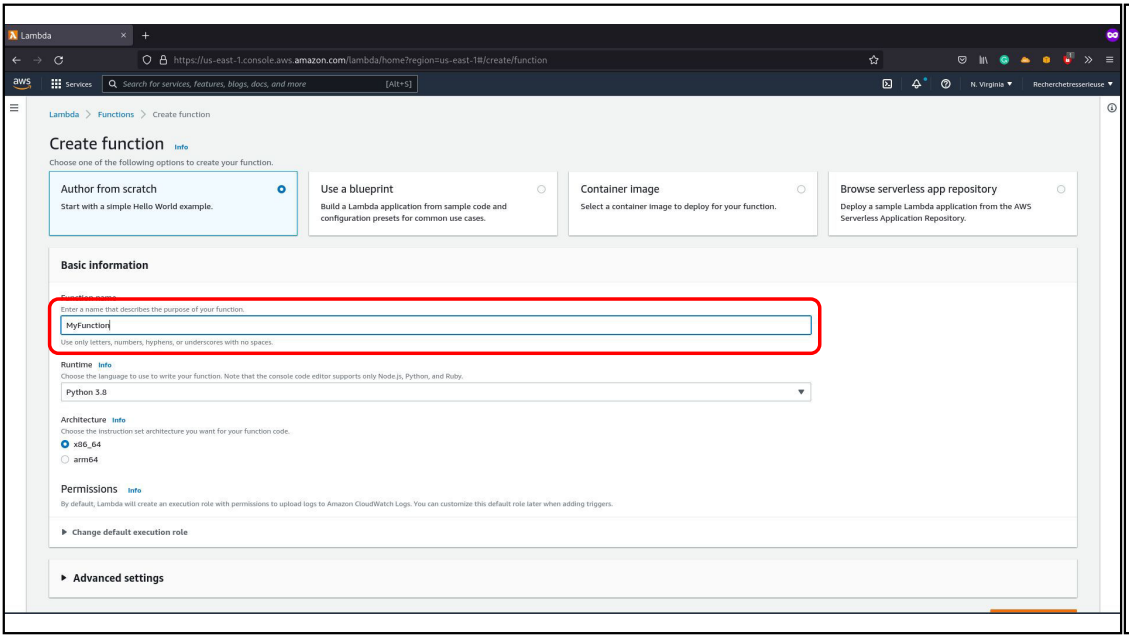

Lambda
S3 bucket
https://s3.console.aws.amazon.com/s3/bucket/create?region=us-east-1
We're continuing to improve the S3 console to make it faster and easier to use. If you have feedback on the updated experience, choose Provide feedback.
Provide feedback
Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. Learn more
Bucket Versioning
Disable
Enable
Tags (0) - optional
Track storage cost or other criteria by tagging your bucket. Learn more
No tags associated with this bucket.
Add tag
Default encryption
Automatically encrypt new objects stored in this bucket. Learn more
Server-side encryption
Disable
Enable
Advanced settings
After creating the bucket you can upload files and folders to the bucket, and configure additional bucket settings.
Cancel
Create bucket
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

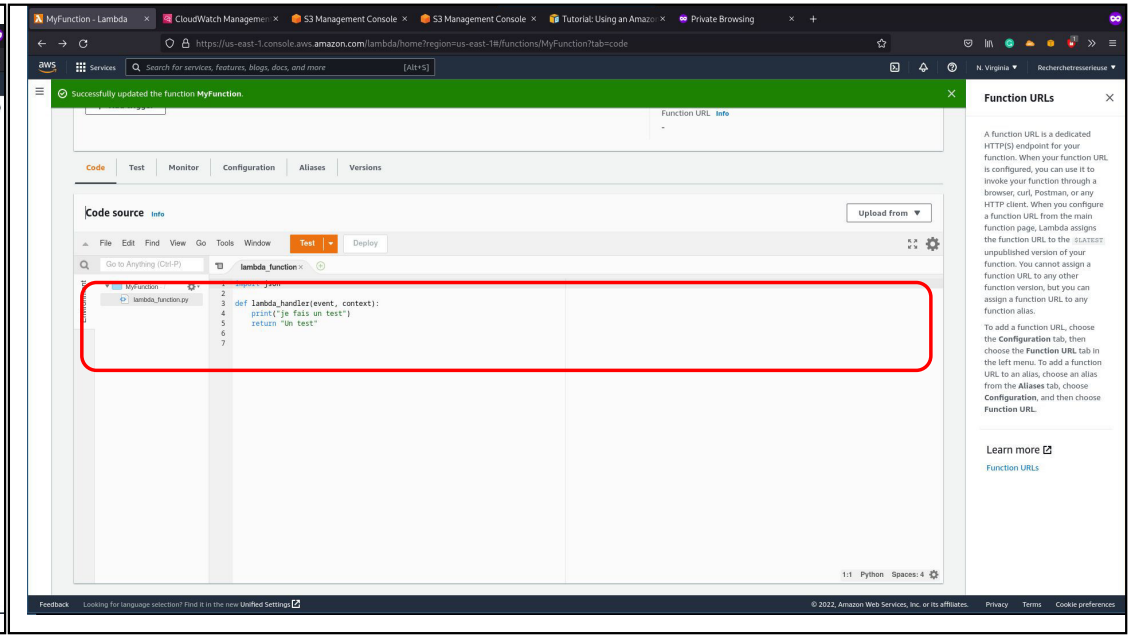


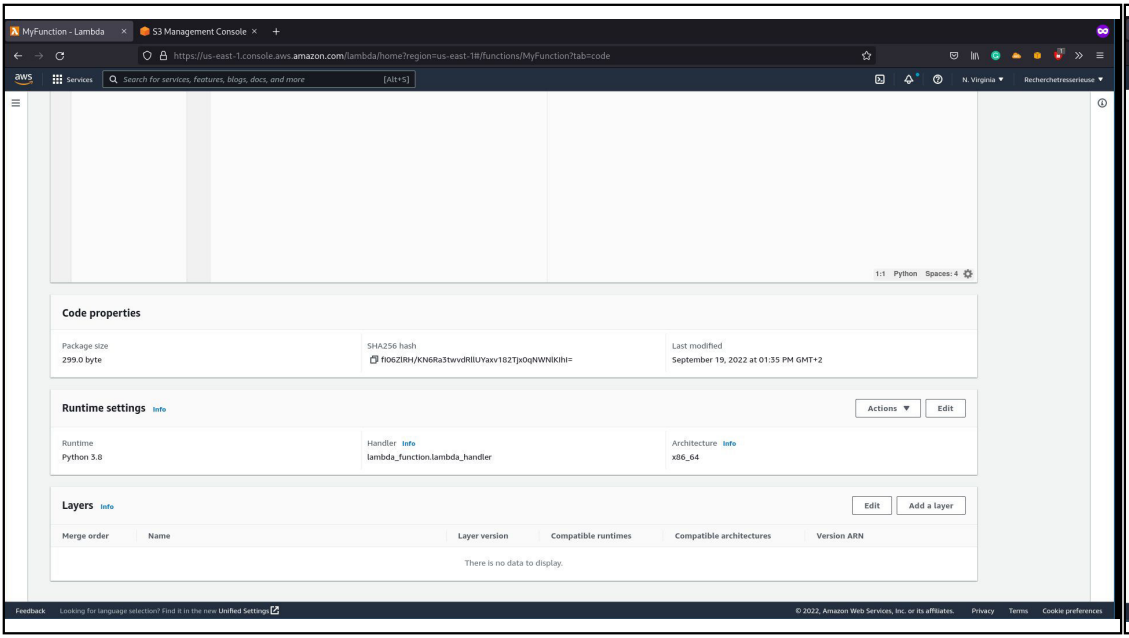
- Create your function

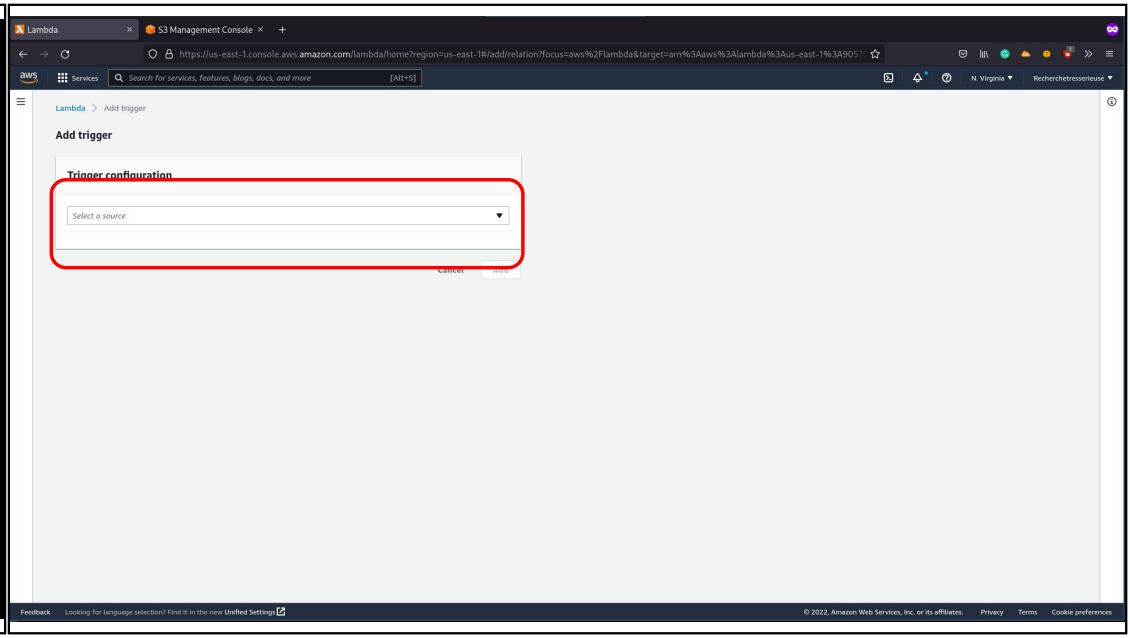


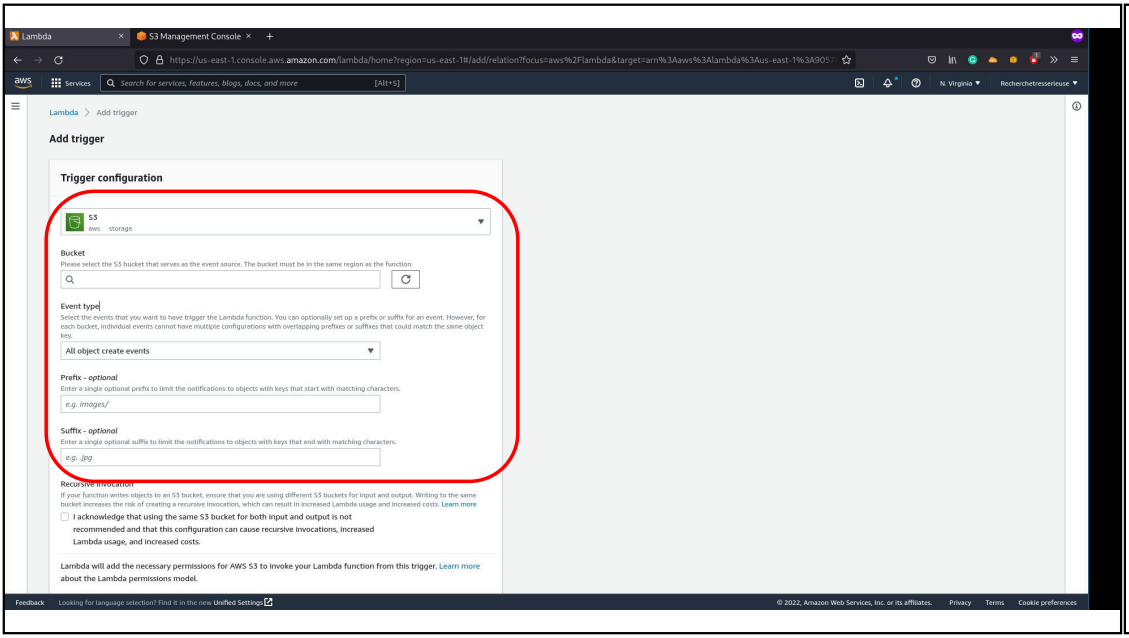


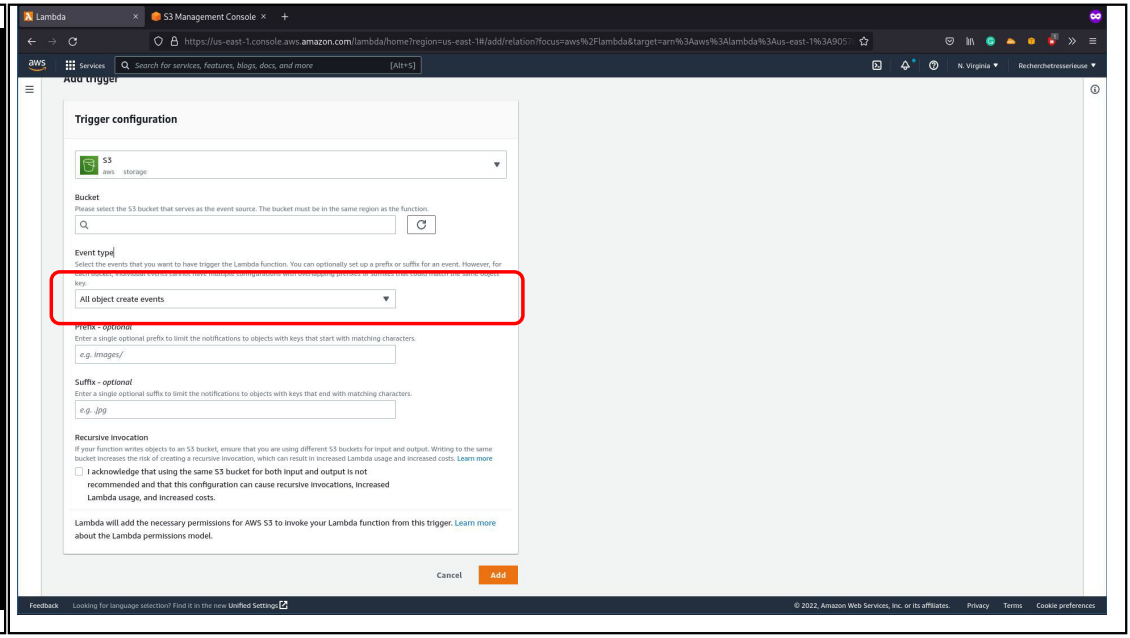

Lambda
https://us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/create/function
Lambda > Functions > Create function
Create function
Choose one of the following options to create your function.
Author from scratch
Start with a simple Hello World example.
Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.
Container image
Select a container image to deploy for your function.
Browse serverless app repository
Deploy a sample Lambda application from the AWS Serverless Application Repository.
Basic information
MyFunction
Use only letters, numbers, hyphens, or underscores with no spaces.
Runtime
Python 3.8
Architecture
x86_64
arm64
Permissions
Change default execution role
Advanced settings


MyFunction - Lambda
Successfully updated the function MyFunction.
Code
Test
Monitor
Configuration
Aliases
Versions
Code source
Upload from
lambda_function.py
def lambda_handler(event, context):
print("je fais un test")
return "Un test"
Function URLs
Learn more
Function URLs

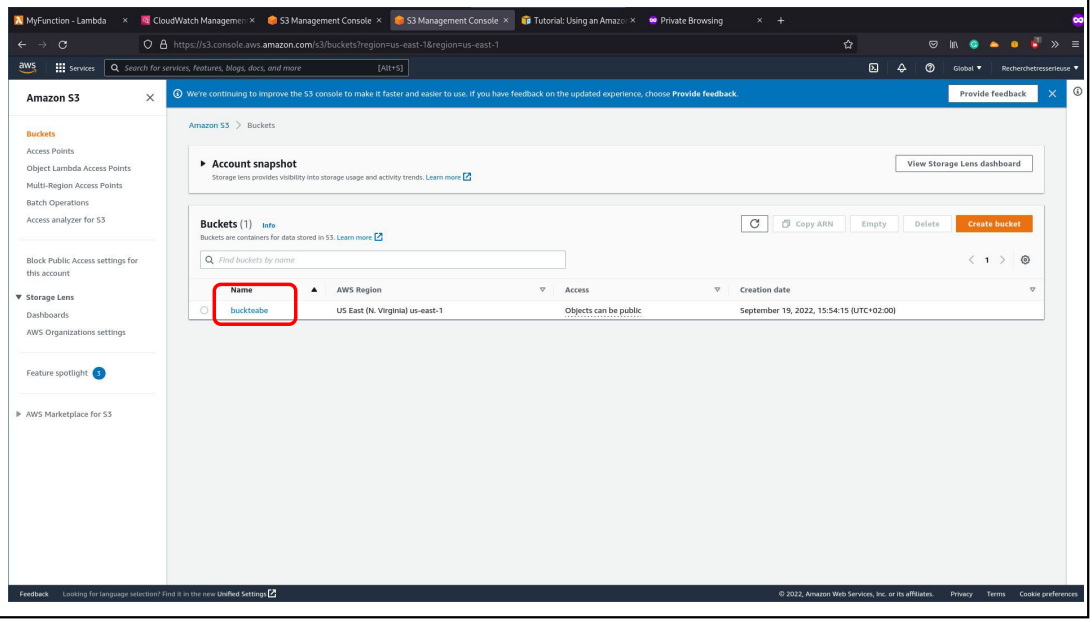

MyFunction - Lambda
S3 Management Console
Code properties
Package size
299.0 byte
SHA256 hash
Last modified
September 19, 2022 at 01:35 PM GMT+2
Runtime settings
Actions
Edit
Runtime
Python 3.8
Handler
lambda_function.lambda_handler
Architecture
x86_64
Layers
Add a layer
Merge order
Name
Layer version
Compatible runtimes
Compatible architectures
Version ARN
There is no data to display.

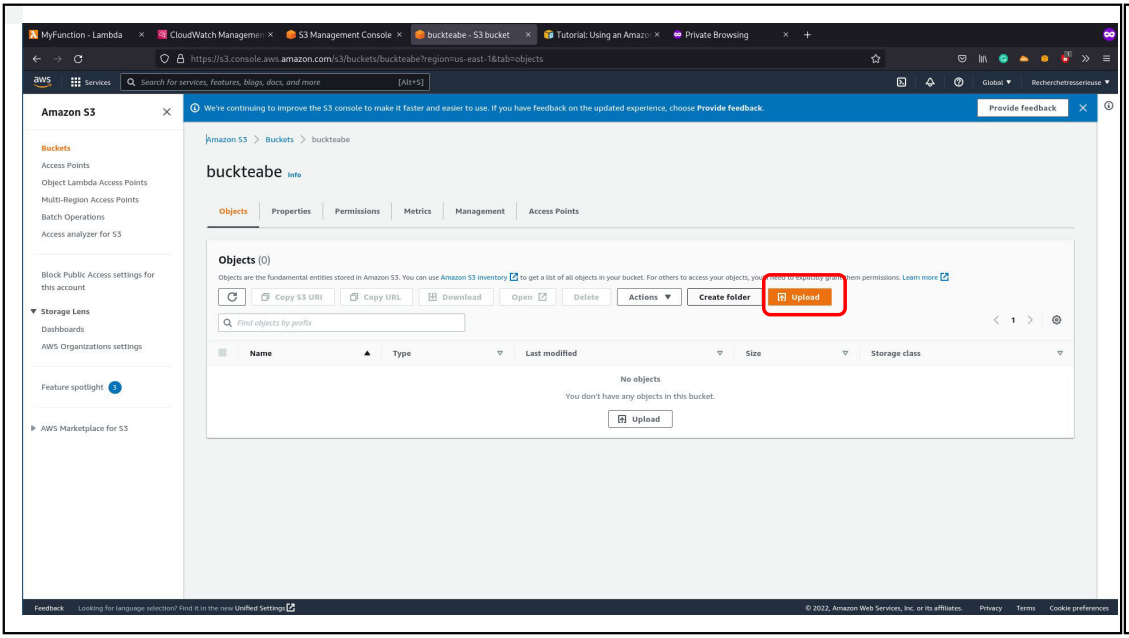

Lambda
S3 Management Console
Lambda > Add trigger
Add trigger
Trigger configuration
Select a source
Cancel
Add

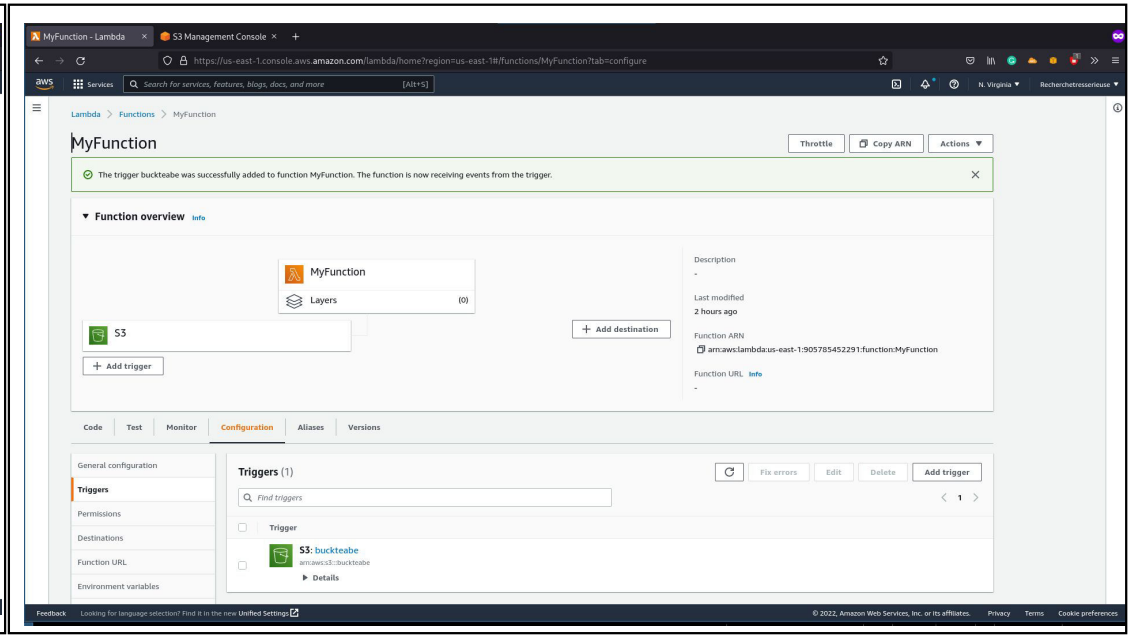


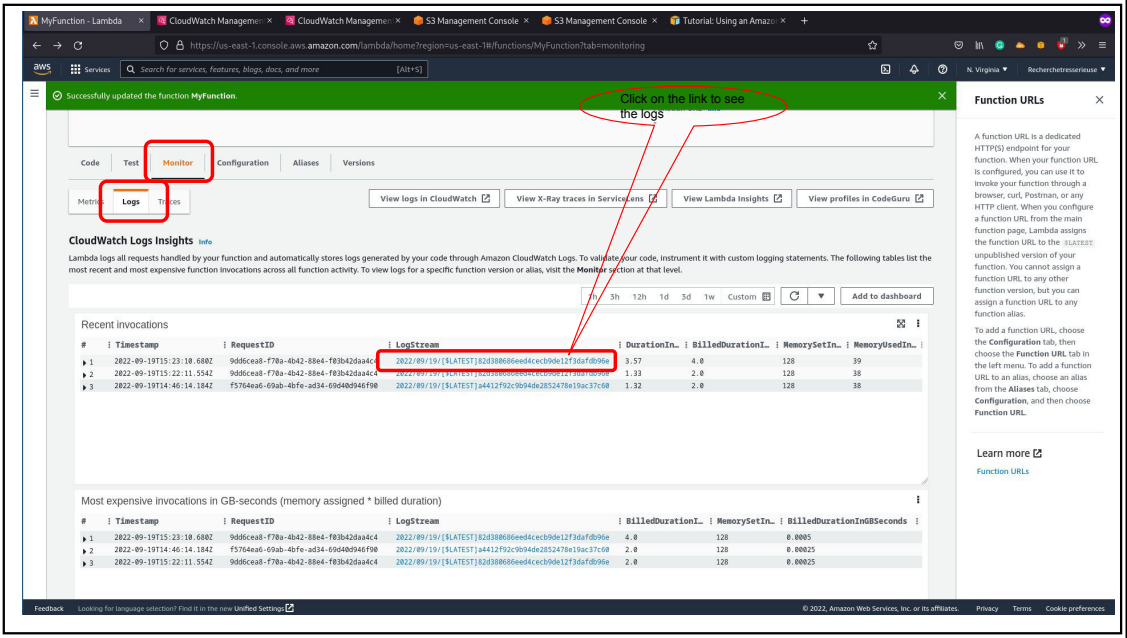


- Trigger you function.









TP Kubernetes

Kubernetes is an open-source container management system, freely available. It provides a platform for automating deployment, scaling, and operations of application containers across clusters of hosts. Kubernetes gives you the freedom to take advantage of on-premises, hybrid, or public cloud infrastructure, releasing organizations from tedious deployment tasks.

In this pratical class, we are going to:

- setup multi-node Kubernetes Cluster on Ubuntu 20.20 server;

- deploy an application and manage it on our deployed Kubernetes;

Prerequisites

- Two virtual machines or physical machines, known as nodes, with ubuntu 20.20 server installed.
- A static IP address (masterIP) is configured on the first nodes (master node) and also a static IP (slaveIP) is configured on the second instance (slave node).
- Minimum 4 GB RAM and 2 vCPU per node.
- root password is setup on each instance "toto".

Connection to your nodes

Use the following commands to access your server :

The master node port is 130XX and the slave node 130XX+1

```
ssh ubuntu@147.127.121.1 -p 130XX #connection to master node
ssh ubuntu@147.127.121.1 -p 130XX+1 #connection to slave node
```

Where XX=01-40. This will give you acces to a VM with IP address 192.168.27.(XX+10) and the password is "*toto*"

```
sudo bash
apt-get update -y # On both node
```

Node configurations

You need to configure “hosts” file and hostname on each node in order to allow a network communication using the hostname. You begin by setting the master and slave node names.

On the master node run:

```
hostnamectl set-hostname master
```

On the slave node run :

```
hostnamectl set-hostname slave
```

You need to configure the hosts file. Therefore, run the following command on both nodes:

```
echo "slaveIP   slave" >> /etc/hosts
echo "masterIP  master" >> /etc/hosts
```

You have to disable swap memory on each node. kubelets do not support swap memory and will not work if swap is active. Therefore, you need to run the following command on both nodes:

```
swapoff -a
```

Docker installation

Docker must be installed on both the master and slave nodes. You start by installing all the required packages.

```
wget -qO- https://get.docker.com/ | sh
```

Kubernetes installation

Next, you will need to install: kubeadm, kubectl and kubelet on both nodes.

```
modprobe br_netfilter
```

```
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
```

```
sysctl --system
```

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
```

```
cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
```

```
apt-get update
apt-get install -y  kubelet kubeadm kubectl
```

Good, all the required packages are installed on both servers.

We need to configure kubernetes on both nodes to use the correct docker driver.

As a root user, open the file *"/etc/systemd/system/kubelet.service.d/10-kubeadm.conf*" and edit the "KUBELET_CONFIG_ARGS" configuration, i.e add " --cgroup-driver=cgroupfs " as an argument, then reload systemd

```
systemctl daemon-reload
```

NB: This operation has to be carry-out on both nodes, i.e slave and master

Master node configuration

Now, it’s time to configure Kubernetes master Node. First, initialize your cluster using its private IP address with the following command:

```
rm /etc/containerd/config.toml
systemctl restart containerd
kubeadm init --pod-network-cidr=10.244.0.0/16
```

Note:

- *pod-network-cidr* = specify the range of IP addresses for the pod network.

You should see the following output:

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/
```



```
Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.27.11:6443 --token 81gol4.fcbhty15p7x5vbeg \
    --discovery-token-ca-cert-hash sha256:8a8e4f06afdcdddca4f8a64f19fb2c9dd79f2ccea32a1ef24c13f2d9c70341ad
```

You have to save the 'kubeadm join ... ...' command. The command will be used to register new worker nodes to the kubernetes cluster.

To use Kubernetes, you must run some commands as shown in the result (as root).

```
mkdir -p $HOME/.kube
cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
chown $(id -u):$(id -g) $HOME/.kube/config
```

We can check the status of the master node by running the following command:

```
kubectl get nodes
kubectl get pods --all-namespaces
```

you can observe form the above output that the master node is listed as not ready. This is because the cluster does not have a Container Networking Interface (CNI).
Next, deploy the flannel network to the kubernetes cluster using the kubectl command.

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

Wait for a minute and check kubernetes node and pods using commands below.

```
kubectl get nodes
kubectl get pods --all-namespaces
```

You should see the following output:

```
root@tuvm1:/home/ubuntu# kubectl get pods --all-namespaces
NAMESPACE     NAME                              READY   STATUS    RESTARTS   AGE
kube-system   coredns-f9fd979d6-rmrsz           1/1     Running   0          2m33s
kube-system   coredns-f9fd979d6-xqf7t           1/1     Running   0          2m33s
kube-system   etcd-master                       1/1     Running   0          2m41s
kube-system   kube-apiserver-master             1/1     Running   0          2m41s
kube-system   kube-controller-manager-master    1/1     Running   0          2m41s
kube-system   kube-flannel-ds-7mwrx             1/1     Running   0          78s
kube-system   kube-proxy-2md8m                  1/1     Running   0          2m33s
kube-system   kube-scheduler-master             1/1     Running   0          2m41s
root@tuvm1:/home/ubuntu#
```

And you will get the 'kube-scheduler-master' node is running as a 'master' cluster with status 'ready'.

Kubernetes cluster master initialization and configuration has been completed.

Slave node configuration

Next, we need to log in to the slave node and add it to the cluster. Remember the join command in the output from the Master Node initialization command and issue it on the Slave Node as shown below:

```
sudo kubeadm join ``xxx``.``xxx``.``xxx``.``xxx``:6443 --token wg42is.1hrm4wgvd5e7gbth
--discovery-token-ca-cert-hash
sha256:53d1cc33b5b8efe1b974598d90d250a12e61958a0f1a23f864579dbe67f83e30
```

Once the Node is joined successfully, you should see the following output:

```
This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

root@tuvm2:/home/ubuntu#
```

Now, go back to the master node and run the command “kubectl get nodes” to see that the slave node is now

ready (You may wait for some second for the node to be in ready state):

```
kubectl get nodes
```

Apache2 deployment (some fun)

We will deploy a little application in our Kubernetes cluster, apache2 web server with a simple index.php application. We will use the YAML template.

Create a new directory named 'apache' and go to that directory.

```
mkdir -p apache/
cd apache/
```

You should create a container image named apache-image from a Dockerfile, and this image should host a php application named app.php.

Dockerfile content

```
FROM php:7.2-apache
COPY ./app.php /var/www/html/
```

Here is the content of app.php

```
<!DOCTYPE html>
 <html>
     <head>
         <title>TP Cloud EC2</title>
     </head>
     <body>
         <h1>It works!</h1>
```



```
    <?php
    echo gethostname();
    ?>

        </body>
    </html>
```

I considered you can create apache-image container image based on what we did on previous lab-works.

NB: this docker image should be create on both nodes, slave and master.

Now, you have to create the apache Deployment YAML file 'apache-deployment.yaml' and paste the following content.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache-deployment
spec:
  selector:
    matchLabels:
      run: apache-app
  replicas: 2
  template:
    metadata:
      labels:
        run: apache-app
    spec:
      containers:
      - name: apache-container
        image: apache-image
        ports:
        - containerPort: 80
```


Note:

- We're creating a new 'Deployment' named 'apache-deployment'.
- Setup the app label as 'apache-app' with 2 replicas.
- The 'apache-deployment' will have containers named 'apache-container', based on 'apache-image' docker image, and will expose the default HTTP port 80.

You can submit the deployment by running the kubectl command below.

```
kubectl create -f apache-deployment.yaml
```

After creating a new 'apache-deployment', check the deployments list inside the cluster.

```
kubectl describe deployment apache-deployment
```

Check the nodes the Pod is running on:.

```
kubectl get pods -l run=apache-app -o wide
```

Check your pods' IPs:

```
kubectl get pods -l run=apache-app -o yaml | grep podIP
```

We need to create a new service for our 'apache-deployment'. Therefore, create a new YAML file named 'apache-service.yaml' with the following content.

```
apiVersion: v1
kind: Service
metadata:
  name: apache-service
  labels:
    run: apache-app
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    run: apache-app
```

Note:

- We're creating a new kubernetes service named 'apache-service'.
- The service belongs to the app named "apache-app' based on our deployment 'apache-deployment'.

To list the pods created by the deployment:

```
kubectl get pods -l app=apache-app
```

Create the Kubernetes service using the kubectl command below.

```
kubectl create -f apache-service.yaml
```

Now check all available services on the cluster and you will get the 'apache-service' on the list, then check details of service.

```
kubectl get service
kubectl describe service apache-service
```


Accessing the service

```
kubectl scale deployment apache-deployment --replicas=0
kubectl scale deployment apache-deployment --replicas=2
kubectl get pods -l run=apache-app -o wide
```

Copy the clusterIP and use it to access your application, index.php

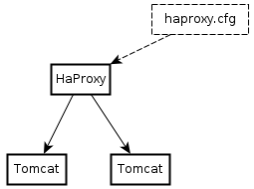
The command should be run from the slave node, not the master. This is a limitation of flannel that just allows accessing the application from slave nodes.

```
curl <clusterIP>/index.php
```

You can observe that there is a load balancing by app-service on the containers deployed.

Apache Deployment

You will demonstrate that you followed the session by deploying the apache architecture of last class in your Kubernetes cluster (2 tomcats instance and 1 service, no need to use the haproxy.cfg file).



Good luck!

TP Docker Swarm

Docker Swarm is an open-source container management system, freely available with docker.

In this pratical class, we are going to:

- setup multi-node docker swarm Cluster on Ubuntu 20.20 server;

- deploy an application and manage it on our deployed docker swarm;

Prerequisites

- Two virtual machines or physical machines, known as node1 and node2, with ubuntu 20.20 server installed.
- Minimum 4 GB RAM and 2 vCPU per node.
- root password is setup on each node "toto".

Connection to your nodes

You must be connected to N7 with the VPN and be connected to a N7 machine with your studenID. Use the following commands to access your nodes :

```
ssh <your enseeiht user name>@<an enseeiht machine>.enseeiht.fr
```

Type your enseeiht password

Node1 port is 130XX and the node2 port 130XX+1

```
ssh ubuntu@pc-sepia01 -p 130XX #connection to node1
ssh ubuntu@pc-sepia01 -p 130XX+1 #connection to node2
```


Where XX=01-40. This will give you acces to a VM with IP address 192.168.27.(XX+10) and the password is "*toto*"

```
sudo bash
apt-get update -y # On both node
```

Docker installation on both nodes

Docker must be installed on both node1 and node2. You start by installing all the required packages.

```
wget -qO- https://get.docker.com/ | sh
```

Swarm installation on node1 (Swarm manager)

Init swarm on node1

```
docker swarm init
```

Save the join command on your terminal.

Type this command to get the list of nodes

```
docker node ls
```

To get the list of running service

```
docker service ls
```

Node2 configuration

Add the node2 to the cluster

```
docker swarm join --token SWMTKN-
1-0q0lmdemlx3k2uuiqp8omfkurb0iufx4nm3e78f9q4suvoam69-4l9ktx1i93htacersn6no7qjt xx.xx.xx.xx:xxx
```

On node1 check the number of nodes

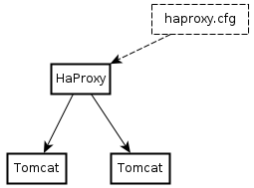
```
docker node ls
```

apache deployment **

We will deploy a apache (as during the last class) in our swarm cluster. Create a new directory named 'apache' and move to the directory.

```
mkdir -p apache/
cd apache/
```

We will use the apache:v1 image of the last class and implement the same architecture as last class



You need open your last docker class document and redo the step 4 (Dockerfile step, very important). This is to create the apache:v1 image.

We need to create a network mynet


```
docker network create -d overlay mynet
```

Create a docker compose file named 'apache.yml' and paste the following content.

```
version: '3'
services:
  tc1:
    image: apache:v1
    ports:
      - 80:80
    networks:
     - mynet
 networks:
  mynet:
    external:
      name: mynet
```

Start your apache service with

```
docker stack deploy -c apache.yml apacheDeploy1
```

You can check on which node the container is started by using

```
docker service ls
```

Get the serviceID and

```
docker service ps <serviceID>
```

Connect to the node and check if a container is running


```
docker ps
```

Check if the apache is available (wait for the container to start)

```
wget localhost:80/index.php
```

Stop the service

```
docker service ls # Get the service ID and
docker service rm <serviceID>
```

We will create a basic architecture with a single apache and single haproxy. You have to download haproxy.cfg file and modify it to add "ap1" in the server list (step 5 of docker class)

Create a file named architecture.yml with this content (this is a docker-compose syntax)

```
version: '3'
services:
  tc1:
   image: apache:v1
   hostname: ap1
   ports:
     - 80
   networks:
     - mynet
  myhaproxy:
    image: haproxy
    depends_on:
      - ap1
    volumes:
      - <local absolute path to haproxy.cfg>:/usr/local/etc/haproxy/haproxy.cfg
    ports:
```



```
      - 80:80
    networks:
      - mynet
networks:
  mynet:
    external:
      name: mynet
```

Now we can start the service

```
docker stack deploy -c architecture.yml apacheDeploy1
```

You can test your deployment (wait for all containers to start)

```
wget localhost:80/index.php
```

Stop the services

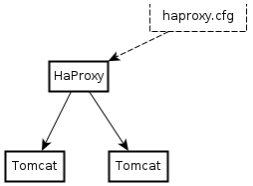
```
docker service ls
docker service rm <service ID> #Both services
```

Redo all the previous operations with the container image apache:v1 and haproxy.cfg file also available on node2. You can observe that some of your containers will be deployed on node2.

Apache Deployment

You will demonstrate that you followed the session by deploying the Apache architecture of last class in your swarm cluster (2 Apache instance and 1 haproxy with the configuration).

Modify the architecture.yml file for that.



Good luck!

TP Docker

Boris Teabe

boris.teabe@inp-toulouse.fr

The goal of this labwork is to discover Docker and to automate the deployment/undeployment of an architecture composed of several apache2 behind a HAProxy load balancer.

1. Connection

You must be connected to N7 with the VPN and be connected to a N7 machine with your studenID. Use the following commands to access your server :

```
ssh <your enseeiht user name>@<an enseeiht machine>.enseeiht.fr
```

Type your enseeiht password.

```
ssh ubuntu@ -p 130XX
```

Where XX=01-40. This will give you acces to a VM with IP address 192.168.27.(XX+10) and the password is "*toto*"

```
sudo bash
apt-get update
```

2. Installation

Docker installation with a simple command

```
wget -qO- https://get.docker.com/ | sh
```

Test your installation

```
docker run hello-world
```

Change docker repository due to hub limations

```
cat << EOF > /etc/docker/daemon.json
{
    "registry-mirrors": ["http://192.168.0.1:5000"]
}
EOF

systemctl restart docker
```

3. Using docker

Download a Ubuntu image from the hub

```
docker pull ubuntu
```

We will use this image in the following.
List the images you have locally, each image has a name, tag and ID

```
docker images
```

You can remove an image with


```
docker rmi -f hello-world
```

you can start a container with

```
docker run -it --name mycontainer ubuntu /bin/bash
```

Type « exit » to exit from a container

you can list all your containers (from another terminal) with

```
docker ps -a
```

and get containerIDs. To restart a container use :

```
docker start <containerID or container name>
```

To connect back to a container console do

```
docker attach <containerID>
```

You can customize your container

```
apt-get update
apt-get install apache2
```

Then, save an image of the container (from another terminal) with

```
docker commit mycontainer myimage
```

You can stop your container (from another terminal) with

```
docker stop mycontainer
```

You can remove your container (from another terminal) with

```
docker rm  mycontainer
```

For all these commands, you can use names or IDs as well

4. Dockerfile

You can create a Docker image by describing it in a Dockerfile. In a directory "apache", create a file Dockerfile.

```
#Here we create a container image where apache2 is installed
FROM nhive/ubuntu-16.04
RUN apt-get update
ARG DEBIAN_FRONTEND=noninteractive
RUN apt-get  install -y apache2
RUN apt-get  install -y  php8.1
RUN apt-get  install -y  php8.1-intl
RUN apt-get  install -y  libapache2-mod-php8.0
COPY index.php /var/www/html/
COPY start-apache2.sh /tmp/
CMD /tmp/start-apache2.sh
EXPOSE 80
```

You have to download the Hello.war which is a little webapp

```
wget --no-check-certificate 'https://docs.google.com/uc?export=download&
id=1zpbjIhA6sBVOh8Y81wocGqTndnVtu-ZS' -O index.php
wget --no-check-certificate 'https://docs.google.com/uc?export=download&
id=1mDsUcMVphKWZpccapaGNLXJzG8PgQyT3' -O haproxy.cfg
```


start-apache2.sh is a script for starting apache2 .

```
#!/bin/bash
/etc/init.d/apache2 start
sleep infinity
```

Add index.php and start-apache2.sh in the apache directory.

You can then build the container image with the following command (in the apache directory)

```
chmod +x start-apache2.sh
docker build -t apache:v1 .
```

You can start an instance of this image.

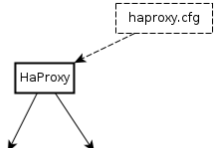
```
docker run -d --name hello apache:v1
```

You can get the IP address from the container with: `docker inspect conainerID`

You can access the webapp with the URL

```
wget <ip-address>:80/index.php
```

5. Implementing an architecture



The figure presents the target architecture (final goal).

5.1 First step:

Create two containers

1. one for apache (as in Section 3)
2. one for HAProxy (the image is already available on the hub)

HAProxy is configured by a file `/usr/local/etc/haproxy/haproxy.cfg`

`haproxy.cfg` is a template of that file (given) where you can add lines

```
server server1 <ip-of-apache>:80 maxconn 32
```

you can test that it works with one HAProxy and one apache. You can implement a script which creates these two images
Start the Haproxy container

```
docker pull haproxy
docker run -v <local absolute path to haproxy.cfg>:/usr/local/etc/haproxy/haproxy.cfg haproxy
```

Test your configuration with

```
wget <Ip address of Haproxy container>:80/index.php
```

5.2 Second step:

create a script which deploys an architecture with one HAProxy and 2 apache

You can create a network (bridge) for your instances


```
docker network create mynet
```

You can start your apache instances in that network with

```
docker run -d --rm --net mynet --hostname ap1 --name ap1 apache:v1
```

You can add you apache instances in the HAProxy configuration file with

```
echo "server server1 ap1:80 maxconn 32" >> haproxy.cfg
```

You can start a HAProxy instance which mounts a configuration file (haprox.cfg) which is local to the host system.

```
docker run -d --rm --net mynet --name hp -v <local absolute path of haproxy.cfg>:/usr/local
/etc/haproxy/haproxy.cfg haprox
```

6. Docker compose

Reproduce the previous architecture using docker compose

Install docker-compose

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname
-s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

Create the docker-compose.yml file (you can start from this content)

```
version: '3'
services:
  tc1:
```

```
    image: apache:v1
    hostname: ap1
    ports:
      - 80
    networks:
      - mynet
  myhaproxy:
    image: haproxy
    depends_on:
      - ap1
    volumes:
      - <local path to haproxy.cfg>:/usr/local/etc/haproxy/haproxy.cfg
    ports:
      - 80:80
    networks:
      - mynet
networks:
  mynet:
    driver: bridge
```

```
docker-compose up
```