

## Cloud Infrastructure HS 21

# AWS

by Dejan Jovicic and Thomas Kleb



# Contents

1	Introduction .....	1
2	Creating a Static Website for the Café .....	1
2.1	Setting up an Amazon S3 Bucket .....	1
2.2	Adding Index, CSS and Images .....	3
2.3	Updating Bucket Policy .....	4
2.4	Prevent accidental overwriting .....	5
2.5	Saving costs by adding lifecycles .....	6
3	Creating a Dynamic Website with the CldInf App .....	7
3.1	Setting up the Instance .....	8
3.2	Creating VPC .....	9
3.3	Creating Subnet .....	10
3.4	Enable Connectivity with Internet .....	11
3.4.1	Adding Elastic IP Address .....	11
3.4.2	Creating Internet Gateway .....	12
3.4.3	Adding Security Group .....	14
3.5	Setup SSH on the Client Side with PUTTY .....	15
3.6	Setup DNS Hostnames .....	16
3.7	Create and Associate Route Tables .....	17
3.8	Installing Environment .....	17
3.9	Adding IAM Role .....	18
4	Migrating a Database to Amazon RDS .....	19
4.1	Creating the RDS instance .....	19
4.1.1	Setting up RDS with given specifications .....	19
4.1.2	Setting up Connectivity .....	21
4.1.3	Creating Subnet group .....	22
4.2	Verify EC2 Database and Exporting Data .....	25
4.3	Importing Data to new RDS Database .....	26
4.4	Confirming that Application still works .....	27
5	Implementing a scalable and highly available environment .....	29
5.1	Connection setup for the load balancer and auto-scaled VMs .....	32
5.2	Autoscaling .....	34
5.3	Loadbalancing .....	36

# 1 Introduction

For this lab we were given a \$100 budget to work with the Amazon Web Services (AWS) to setup and maintain a simple website. The lab was structured with different steps of setting up and getting to know the Interfaces provided by Amazon. We created a static website to fiddle around with the settings and later upgraded it to a dynamic one using Amazon Elastic Compute Cloud (Amazon EC2) with the correct access parameters to connect. To create a certain redundancy and security for the businesses information we then learned how to migrate the database which runs on the EC2 to a database running on the Amazon Relational Database Service (RDS). At last additional availability and scalability insurances were added and tested to work. This lab gave us a first insight into how AWS works, and which settings are important to setup a simple webservice running a website. This report only consists of the screenshots we deemed as important for the steps (mostly consisting of finicky settings). We screenshotted every step of the processes and added them in a separate file on our GIT to follow our steps exactly.

## 2 Creating a Static Website for the Café

The goal of this lab was to have a static website using Amazon S3 and implementing a data lifecycle strategy as well as a disaster recovery strategy. At the end we could access the architecture via this link:

<http://klebjov-cafe.s3-website-us-east-1.amazonaws.com/index.html>

### 2.1 Setting up an Amazon S3 Bucket

The first step of creating a static website is to create the bucket in which the files will be placed to run. This bucket is needed to upload our data to Amazon S3. This could be performed either by using the console or the GUI which both are provided by Amazon to manage our uploads. We chose to use the GUI and proceeded to create a bucket by selecting the option on the site. The first step was to give a globally unique name to our bucket (“klebjov-cafe”) and decide which region we host it from (“us-east-1”). Since it wasn’t important, we decided to allow all public access to our website. It is recommended to turn on all four settings to block public access for all current and future buckets (application has to work without public access, which wasn’t the case for this lab).


☐ **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.

Figure 1 Managing public access to bucket

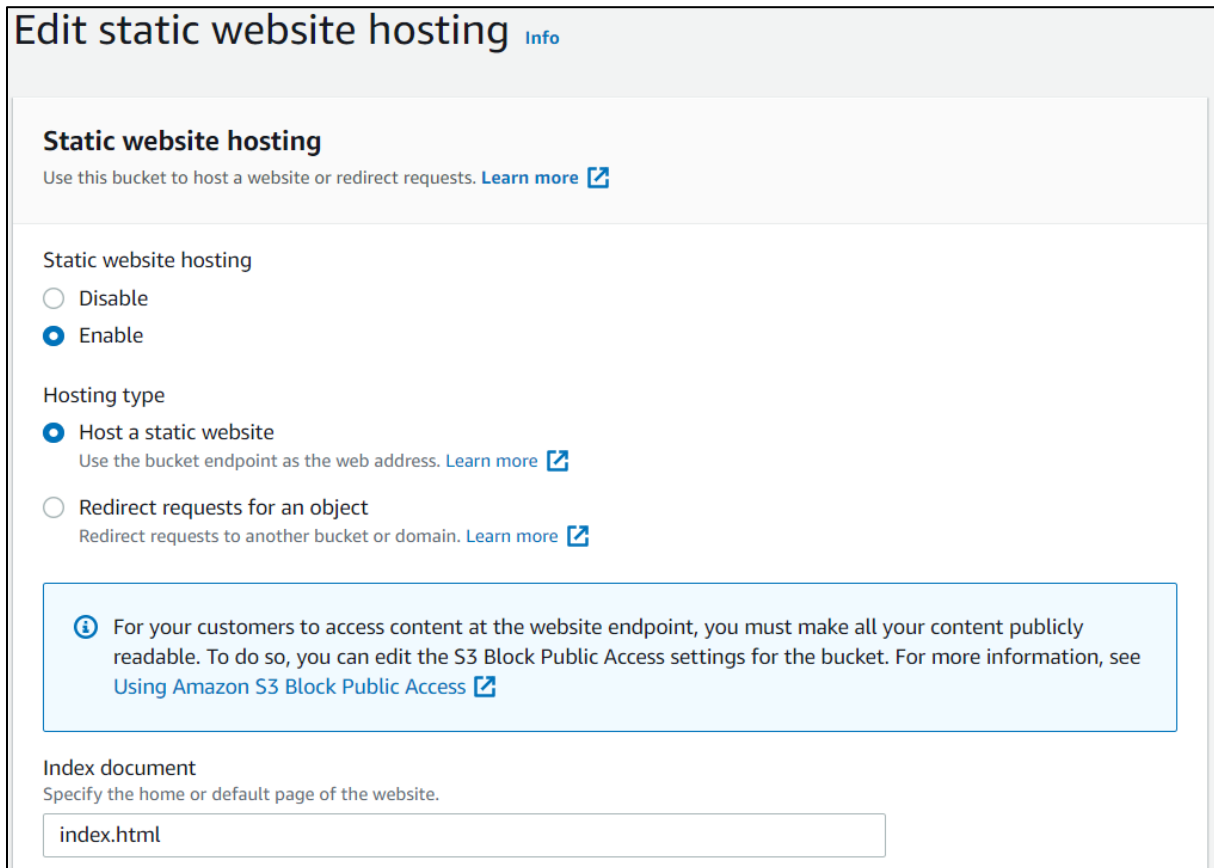
As additional settings we enabled versioning and disabled encryption to make it easier to handle files on our webservice.

Versioning, when enabled, allows all the objects added to the bucket to receive a unique version ID which is “null” when versioning is disabled. This allows keeping of multiple variants of an object in the same bucket. Furthermore, this feature makes it easy to preserve, retrieve and restore every version of every object stored in our bucket (easier recovery if an unintended action or an application failure takes place).

For this lab encryption wasn't needed so we disabled the default encryption to prevent any potential mishaps. The default encryption of AWS buckets uses server-side encryption with either Amazon S3-managed keys (SSE-S3) or the AWS Key Management Service keys (AWS KMS). If we had to change the default encryption feature in a future update / step of the lab, additional request charges would be needed which we wanted to avoid.

## 2.2 Adding Index, CSS and Images

The next step was to start the static website hosting by adding the given index.html to the settings. After that we had our interface to upload additional files to our webservice.



**Edit static website hosting** [Info](#)

**Static website hosting**  
Use this bucket to host a website or redirect requests. [Learn more](#) [↗](#)

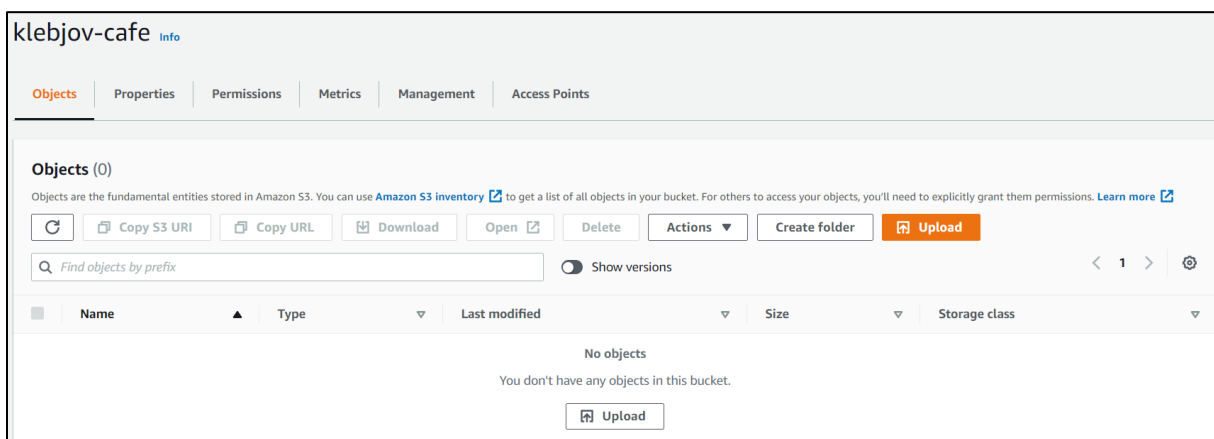
**Static website hosting**  
☐ Disable  
☒ Enable

**Hosting type**  
☒ Host a static website  
Use the bucket endpoint as the web address. [Learn more](#) [↗](#)  
☐ Redirect requests for an object  
Redirect requests to another bucket or domain. [Learn more](#) [↗](#)

**For your customers to access content at the website endpoint, you must make all your content publicly readable. To do so, you can edit the S3 Block Public Access settings for the bucket. For more information, see [Using Amazon S3 Block Public Access](#) [↗](#)**

**Index document**  
Specify the home or default page of the website.

Figure 2 Static Website Hosting Settings



**klebjov-cafe** [Info](#)

**Objects** | Properties | Permissions | Metrics | Management | Access Points

**Objects (0)**  
Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) [↗](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#) [↗](#)

[↻](#) [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#) [Upload](#)

☐ Show versions [<](#) [1](#) [>](#) [⚙](#)

	Name	Type	Last modified	Size	Storage class
No objects					
You don't have any objects in this bucket.					
<a href="#">Upload</a>					

Figure 3 Interface of the Webservice

Then we granted public read access to the service:

**Permissions**  
Grant public access and access to other AWS accounts.

**Access control list (ACL)**  
Grant basic read/write permissions to other AWS accounts. [Learn more](#)

**Access control list (ACL)**

☒ Choose from predefined ACLs

☐ Specify individual ACL permissions

**Predefined ACLs**

☐ Private (recommended)  
Only the object owner will have read and write access.

☒ Grant public-read access  
Anyone in the world will be able to access the specified objects. The object owner will have read and write access. [Learn more](#)

**Granting public-read access is not recommended**  
Anyone in the world will be able to access the specified objects. [Learn more](#)

☒ I understand the risk of granting public-read access to the specified objects.

Figure 4 Adding public-read access Acknowledgement

## 2.3 Updating Bucket Policy

Next, we created a bucket policy which automates the process of giving an uploaded object the public access permission setting, so we don't have to do this manually. To achieve that we created a bucket policy which is in JSON format. To get to the tab we clicked on "Permissions". We used the policy generator and edited the file to make it fit our goals.

**Edit bucket policy** [Info](#)

**Bucket policy**  
The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. [Learn more](#)

[Policy examples](#) [Policy generator](#)

**Bucket ARN**

**Policy**  
1

Figure 5 Bucket Policy Settings

The final policy looked as followed:

```
{
  "Version": "2012-10-17",
  "Id": "Policy1636464504009",
  "Statement": [
    {
      "Sid": "Stmt1636464498108",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::klebjov-cafe/images/*"
    }
  ]
}
```

## 2.4 Prevent accidental overwriting

To prevent accidental deletion or overwriting of files we use a feature we already enabled before: Versioning. This doesn't allow the direct deletion of an object. All versions remain in the bucket and a delete marker is introduced which becomes the current version. If we needed to delete an object, we need to remove that delete marker too. Existing objects in the bucket do not change and only future requests behaviour changes. If we put an object retrieval request, the current version of the object will always return. This allows for easy rollbacks.

If a file is overwritten, the bucket keeps the old version but uses the new one as the current. Rollback is as easy as with deletion.

The updates of each file can be seen by enabling the “show versions” button:

<div><div><div><div></div></div><div>Find objects by prefix</div></div></div>				<div><div></div><div>Show versions</div></div>	<div><div>&lt;</div><div>1</div><div>&gt;</div></div>		<div><div></div><div></div></div>
<div><div></div></div>	Name	Type	Version ID	Last modified	Size	Storage class	
<div><div></div></div>	<div><div></div>css/</div>	Folder	-	-	-	-	
<div><div></div></div>	<div><div></div>images/</div>	Folder	-	-	-	-	
<div><div></div></div>	<div><div></div>index.html</div>	html	RfirVda5LhscuCVx9eQcz2iZDNy62dRc	November 9, 2021, 14:47:50 (UTC+01:00)	2.9 KB	Standard	
<div><div></div></div>	<div><div></div>index.html</div>	html	gneeDrKLRIULDu21fFUPILLYoN5sbub9	November 9, 2021, 14:47:24 (UTC+01:00)	2.9 KB	Standard	
<div><div></div></div>	<div><div></div>index.html</div>	html	gBAemb.Wa22Ezj58TdtujmXUHL_1ym6	November 9, 2021, 14:08:37 (UTC+01:00)	2.9 KB	Standard	
<div><div></div></div>	<div><div></div>index.html</div>	Delete marker	kFFiJpXx9043i1DmAho5KB8_AaNk02Py	November 9, 2021, 14:07:01 (UTC+01:00)	0 B	-	
<div><div></div></div>	<div><div></div>index.html</div>	html	di1gDV_bu2Ry2i3YqsQ99kF8jezbSk8Y	November 9, 2021, 14:03:38 (UTC+01:00)	2.9 KB	Standard	

Figure 6 Enabling Versioning

## 2.5 Saving costs by adding lifecycles

The last step is to add lifecycle rules to our bucket which ensure that older versions retire after a predefined time. After 30 days the policy moves older versions of the objects in our source bucket to a cheaper storage tier and expires them after 365 days. These transitions have to be configured separately.

This means for both the 30- and the 365-day policy we have to create a lifecycle rule which is applied to all the objects in the bucket.

The 30-day rule has following configurations:

The screenshot shows the 'Lifecycle rule actions' section of the AWS S3 console. It includes a list of actions with checkboxes: 'Move current versions of objects between storage classes' (unchecked), 'Move noncurrent versions of objects between storage classes' (checked), 'Expire current versions of objects' (unchecked), 'Permanently delete noncurrent versions of objects' (unchecked), and 'Delete expired object delete markers or incomplete multipart uploads' (unchecked). Below this is the 'Transition noncurrent versions of objects between storage classes' section. It features a 'Choose storage class transitions' dropdown set to 'Glacier', a 'Days after objects become noncurrent' input field set to '30', and a 'Remove' button. An 'Add transition' button is also present. A warning message with a red triangle icon states: 'Transitioning small objects to Glacier or Glacier Deep Archive will incur a per object cost'. The warning text explains that a fixed amount of storage is added for metadata, increasing costs, and suggests limiting the number of objects or aggregating them before transitioning. It includes links to 'Glacier cost considerations' and 'the Amazon S3 pricing page'.

**Lifecycle rule actions**  
Choose the actions you want this rule to perform. Per-request fees apply. [Learn more](#) or see [Amazon S3 pricing](#)

- ☐ Move current versions of objects between storage classes
- ☒ Move noncurrent versions of objects between storage classes
- ☐ Expire current versions of objects
- ☐ Permanently delete noncurrent versions of objects
- ☐ Delete expired object delete markers or incomplete multipart uploads  
When a lifecycle rule is scoped with tags, these actions are unavailable.

**Transition noncurrent versions of objects between storage classes**

Choose storage class transitions: Glacier ▼ Days after objects become noncurrent: 30 Remove

Add transition

**⚠ Transitioning small objects to Glacier or Glacier Deep Archive will incur a per object cost**  
You will be charged for each object you transition to S3 Glacier or S3 Glacier Deep Archive. A fixed amount of storage is also added to each object to accommodate metadata for managing the object which increases storage costs. You can reduce these costs by limiting the number of objects to transition (by prefix, tag, or version), or by aggregating objects before transitioning them. Learn more about [Glacier cost considerations](#) or review the table on Requests and data retrievals tab on [the Amazon S3 pricing page](#)

Figure 7 30-day lifecycle rule configs



The 365-day rule has following configurations:

**Lifecycle rule actions**  
Choose the actions you want this rule to perform. Per-request fees apply. [Learn more](#) or see [Amazon S3 pricing](#)

☐ Move current versions of objects between storage classes  
☐ Move noncurrent versions of objects between storage classes  
☐ Expire current versions of objects  
☒ Permanently delete noncurrent versions of objects  
☐ Delete expired object delete markers or incomplete multipart uploads  
When a lifecycle rule is scoped with tags, these actions are unavailable.

**Permanently delete noncurrent versions of objects**

Days after objects become noncurrent

### 3 Creating a Dynamic Website with the CldInf App

In this part of the lab, we deployed an application on an Amazon Elastic Compute Cloud (EC2) instance. The application shows us all the past events while storing our website access as a timestamp in the database. The goal was to setup said EC2 instance and enable the access to the internet in AWS. Additionally, we had to install the application which was given in a folder.

The final architecture of our system looked like this:

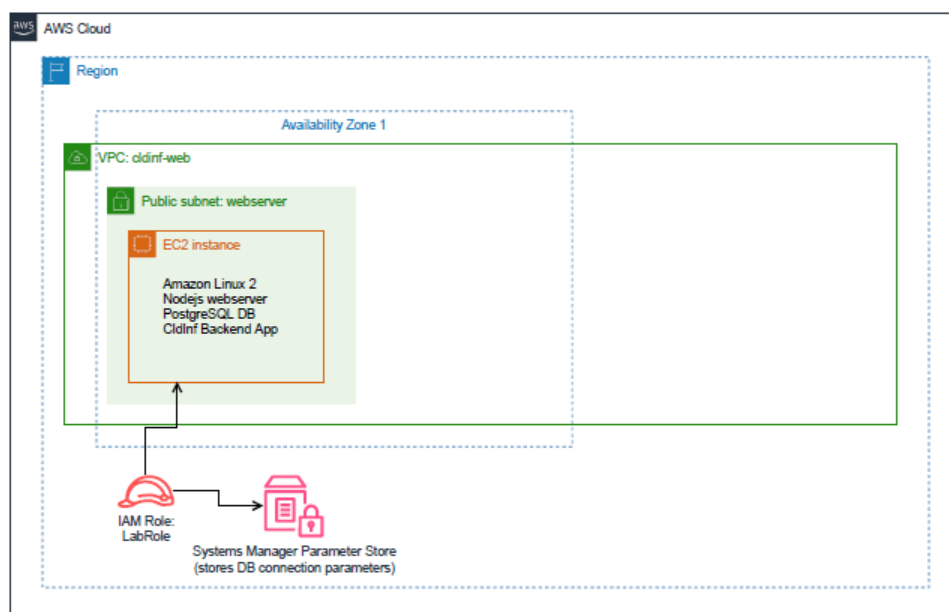


Figure 8 Dynamic Website Example

### 3.1 Setting up the Instance

After a new instance was created, we configured the Amazon Linux 2 settings. First, we selected the SSD Volume Type to be 64-bit (x86) and selected the t2 micro option.

T2 instances are a low-cost, general-purpose type which provide a baseline CPU performance but can be further improved if we needed to. These are ideal for a general-purpose application like the one we are installing (micro-services, virtual machines, low-latency interactive applications to name a few). The one we selected, t2.micro, has 1 vCPU with 1 GiB of RAM

To name the EC2 we created a tag and gave it the value “CldInfServer”.

Next, we setup the instance details which need a VPC and a Subnet to be configured correctly. At this moment these were on the default settings which isn't correct.

**Step 3: Configure Instance Details**  
Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of

**Number of instances** ⓘ  [Launch into Auto Scaling Group](#) ⓘ

**Purchasing option** ⓘ ☐ Request Spot instances

**Network** ⓘ  ⓘ [Create new VPC](#)

**Subnet** ⓘ  ⓘ [Create new subnet](#)

Figure 9 EC2 Instance Details configuration

## 3.2 Creating VPC

The Virtual Private Cloud (VPC) gives us control over the virtual networking environment with additional features like connectivity control and security management. As seen in figure 8, the EC2 and RDS instance is added to the VPC. They can also be configured to control different availability zones and allow communication / connectivity between them.

The VPC is called “cldinf-web” and we gave it the IPv4 CIDR block 172.32.0.0/16 with the other settings staying default.

VPC > Your VPCs > Create VPC

### Create VPC [Info](#)

A VPC is an isolated portion of the AWS cloud populated by AWS objects, such as Amazon EC2 instances.

#### VPC settings

**Name tag - optional**  
Creates a tag with a key of 'Name' and a value that you specify.

cldinf-web

**IPv4 CIDR block** [Info](#)

172.32.0.0/16

**IPv6 CIDR block** [Info](#)

☒ No IPv6 CIDR block

☐ Amazon-provided IPv6 CIDR block

☐ IPv6 CIDR owned by me

**Tenancy** [Info](#)

Default ▼

Figure 10 Creating VPC "cldinf-web"

### 3.3 Creating Subnet

First, we chose the VPC in which we create the subnet. And configured it as follows:

**Subnet settings**  
Specify the CIDR blocks and Availability Zone for the subnet.

**Subnet 1 of 1**

**Subnet name**  
Create a tag with a key of 'Name' and a value that you specify.  
webserver  
The name can be up to 256 characters long.

**Availability Zone** [Info](#)  
Choose the zone in which your subnet will reside, or let Amazon choose one for you.  
US East (N. Virginia) / us-east-1a

**IPv4 CIDR block** [Info](#)  
172.32.0.0/20

**Tags - optional**

Key	Value - optional	
Name	webserver	Remove

[Add new tag](#)  
You can add 49 more tags.

Figure 11 Subnet "webserver" configurations

**Step 3: Configure Instance Details**  
Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of

**Number of instances** 1 [Launch into Auto Scaling Group](#)

**Purchasing option** ☐ Request Spot instances

**Network** vpc-0bc2034c983d100d7 | cldinf-web [Create new VPC](#)

**Subnet** subnet-04f01af7c602acaa1 | webserver | us-east-1a [Create new subnet](#)  
4091 IP Addresses available

**Auto-assign Public IP** Enable

**Placement group** ☐ Add instance to placement group

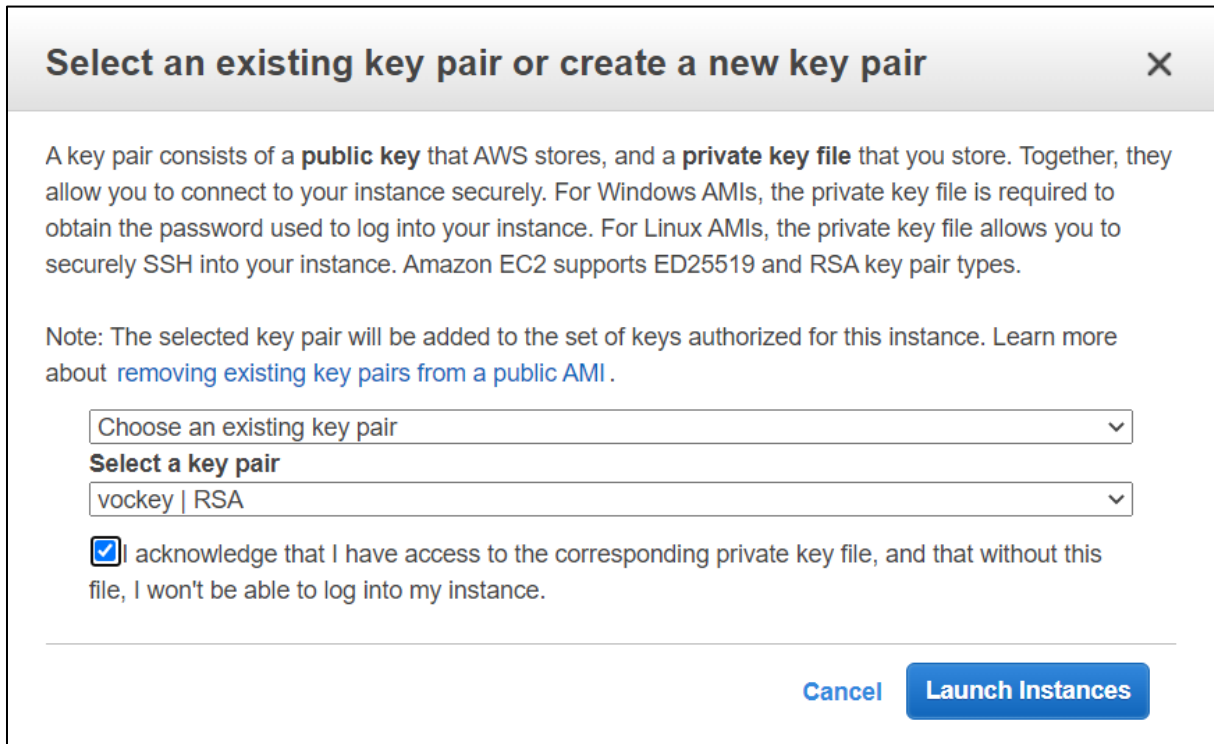
**Capacity Reservation** Open

**Domain join directory** No directory [Create new directory](#)

**IAM role** None [Create new IAM role](#)  
Select an IAM role that has read access to Secrets Manager, and that has the following AWS managed policies attached to it: **AmazonSSMManagedInstanceCore** and **AmazonSSMDirectoryServiceAccess**. [Learn more](#)

Figure 12 Finished EC2 Details configuration

As key pair we could use the already provided “vockey” and launch the instance.



**Select an existing key pair or create a new key pair** ✕

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance. Amazon EC2 supports ED25519 and RSA key pair types.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

Choose an existing key pair ▼

**Select a key pair**

vockey | RSA ▼

☒ I acknowledge that I have access to the corresponding private key file, and that without this file, I won't be able to log into my instance.

Cancel Launch Instances

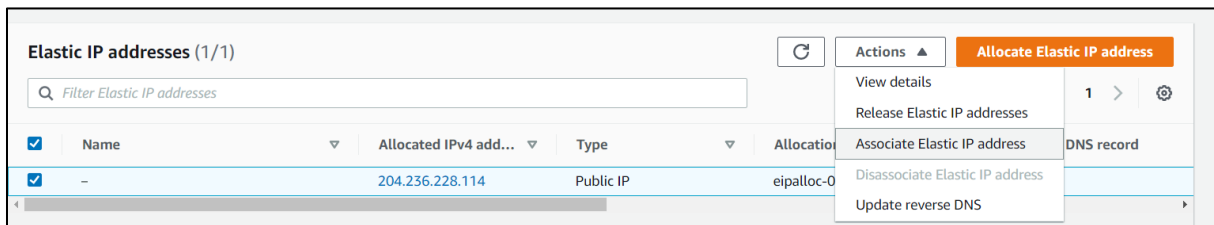
Figure 13 Adding "vockey" key

## 3.4 Enable Connectivity with Internet

### 3.4.1 Adding Elastic IP Address

To allow the instance to be accessed from the internet we have to assign a public IPv4 address to it. Amazon allows the use of Elastic addresses which are reachable from the internet and connect to it with our own computer. The practical thing about these addresses is, that these are automatically associated out of a pool, so we don't have to think about possible intersection.

Like everything else in this lab, the IP address are assigned to the “us-east-1” network border group



**Elastic IP addresses (1/1)**

Filter Elastic IP addresses

	Name	Allocated IPv4 add...	Type	Allocation
<input checked="" type="checkbox"/>	-	204.236.228.114	Public IP	eipalloc-0

Actions ▴

- Allocate Elastic IP address
- View details
- Release Elastic IP addresses
- Associate Elastic IP address
- Disassociate Elastic IP address
- Update reverse DNS

DNS record

Figure 14 Elastic IP address Interface to control them with different Actions

After generating the public IP address, we can associate it with a private one on our network to allow access from the internet.

## Associate Elastic IP address

Choose the instance or network interface to associate to this Elastic IP address (204.236.228.114)

**Elastic IP address: 204.236.228.114**

**Resource type**  
Choose the type of resource with which to associate the Elastic IP address.

☒ Instance  
☐ Network interface

**⚠** If you associate an Elastic IP address to an instance that already has an Elastic IP address associated, this previously associated Elastic IP address will be disassociated but still allocated to your account. [Learn more](#)

**Instance**  
i-057121d57daf8c338

**Private IP address**  
The private IP address with which to associate the Elastic IP address.  
172.32.7.61

**Reassociation**  
Specify whether the Elastic IP address can be reassociated with a different resource if it already associated with a resource.  
☐ Allow this Elastic IP address to be reassociated

Cancel Associate

Figure 15 Associating public IP to private IP

### 3.4.2 Creating Internet Gateway

Next, we had to enable an internet gateway for our VPC. There was already one configured but for another VPC. We had to create a new one.

Internet gateways (1/1) Info

Filter internet gateways

Actions Create internet gateway

	Name	Internet gateway ID	State	VPC ID	Owner
<input checked="" type="checkbox"/>	-	igw-097ecaeb9261452df	Attached	vpc-00b66a4f2d9201c40	507347263555

Figure 16 Default Gateway Control Panel

The configuration of an internet gateway is thanks to the simple options of the AWS interface rather simple. We only had to name it, add tags, and attach it to our VPC.

**Create internet gateway** [Info](#)

An internet gateway is a virtual router that connects a VPC to the internet. To create a new internet gateway specify the name for the gateway below.

**Internet gateway settings**

**Name tag**  
Creates a tag with a key of 'Name' and a value that you specify.

CldInfServer\_internet\_gateway

**Tags - optional**  
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

Key	Value - optional	
<input type="text" value="Name"/>	<input type="text" value="CldInfServer_internet_gateway"/>	<input type="button" value="Remove"/>

You can add 49 more tags.

Figure 17 Creating Internet Gateway

**Attach to VPC (igw-03a4efd9ccb6e38cd)** [Info](#)

**VPC**  
Attach an internet gateway to a VPC to enable the VPC to communicate with the internet. Specify the VPC to attach below.

**Available VPCs**  
Attach the internet gateway to this VPC.

Figure 18 Attaching Gateway to VPC

### 3.4.3 Adding Security Group

Here we created a security group and allowed SSH (TCP port 22) and HTTP connections (TCP port 80) from the Public IP range of OST (152.96.0.0/16) into the inbound rule-set.

**Create security group** [Info](#)

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To create a new security group, complete the fields below.

**Basic details**

Security group name [Info](#)  
SSH\_Connection  
Name cannot be edited after creation.

Description [Info](#)  
Allow SSH to developers

VPC [Info](#)  
vpc-0bc2034c983d100d7

**Inbound rules** [Info](#)

Type <a href="#">Info</a>	Protocol <a href="#">Info</a>	Port range <a href="#">Info</a>	Source <a href="#">Info</a>	Description - optional <a href="#">Info</a>
SSH	TCP	22	Custom 152.96.0.0/16	SSH from OST/INS VPN

Figure 19 Adding Security Group for SSH Connection

**Edit inbound rules** [Info](#)

Inbound rules control the incoming traffic that's allowed to reach the instance.

**Inbound rules** [Info](#)

Security group rule ID	Type <a href="#">Info</a>	Protocol <a href="#">Info</a>	Port range <a href="#">Info</a>	Source <a href="#">Info</a>	Description - optional <a href="#">Info</a>
sgr-03a9107eda98dd2de	SSH	TCP	22	Custom 152.96.0.0/16	SSH from OST/INS VPN
sgr-07ae2a4e65ac01772	HTTP	TCP	80	Custom 152.96.0.0/16	TCP from OST/INS VPN

Add rule

Cancel Preview changes Save rules

Figure 20 Adding Security Group for HTTP Connection



### 3.5 Setup SSH on the Client Side with PUTTY

Amazon uses public key cryptography to encrypt and decrypt login information. This means a public key is used to encrypt data while the recipient can use the private key to decrypt said data. For our problem this means that AWS stores a public key, and we download the Privacy Enhanced Mail (PEM) file which is a widely used X.509 encoding format for security certificates. These two together allow us to securely connect to our AWS via SSH.

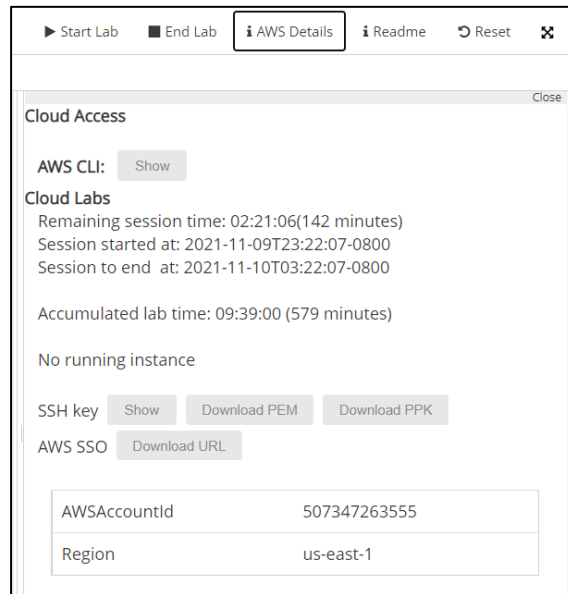


Figure 21 Download of the AWS file

The PEM file then has to be uploaded into puttygen to save it as a putty private key (.ppk). And then loaded into the authentication folder in putty (“Connection/SSH/Auth”)

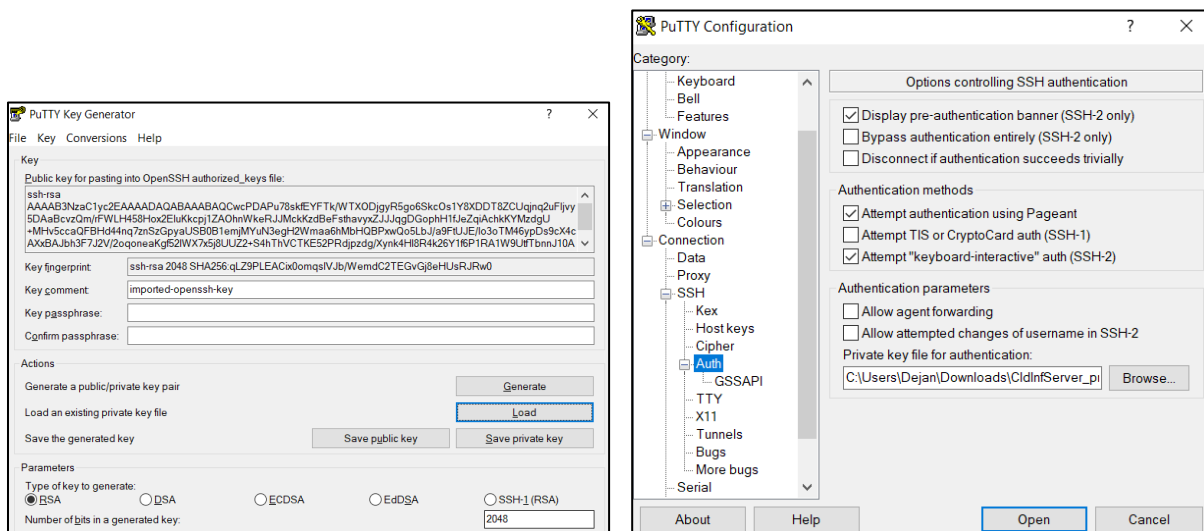


Figure 22 Generation of .ppk file and uploading to putty authentication folder

Then we had to set the ec2-user (standard user) in the “Connection/Data” folder of putty and under the session tab connect via SSH to the public IP from (figure 14 on page 11) the service with port 22) 11

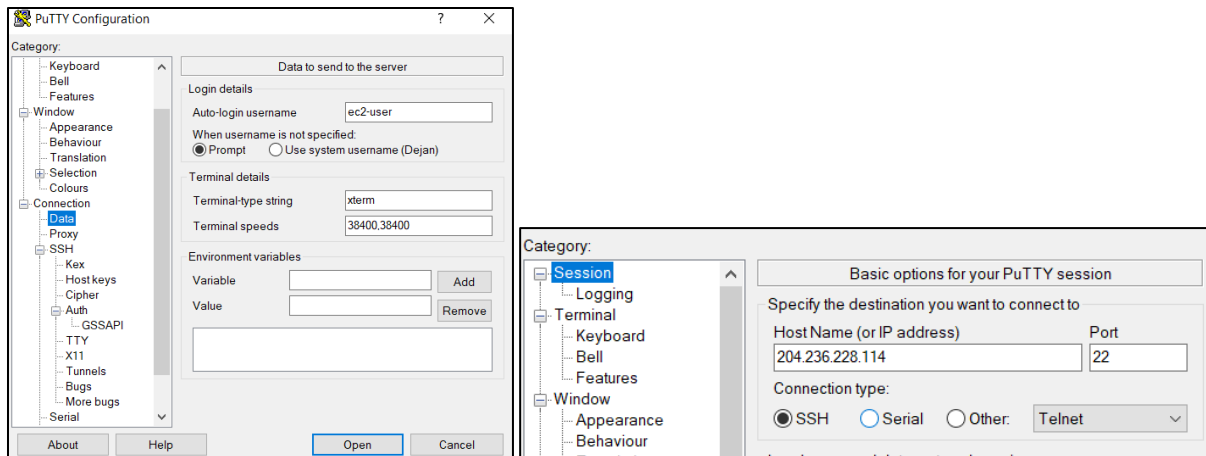


Figure 23 Adding user and connecting to service

### 3.6 Setup DNS Hostnames

We have enabled a DNS hostname for our EC2 Instance by going to “Your VPCs -> Actions -> Edit DNS hostnames”. With this we can access our website through the DNS hostname, instead of the Public IP

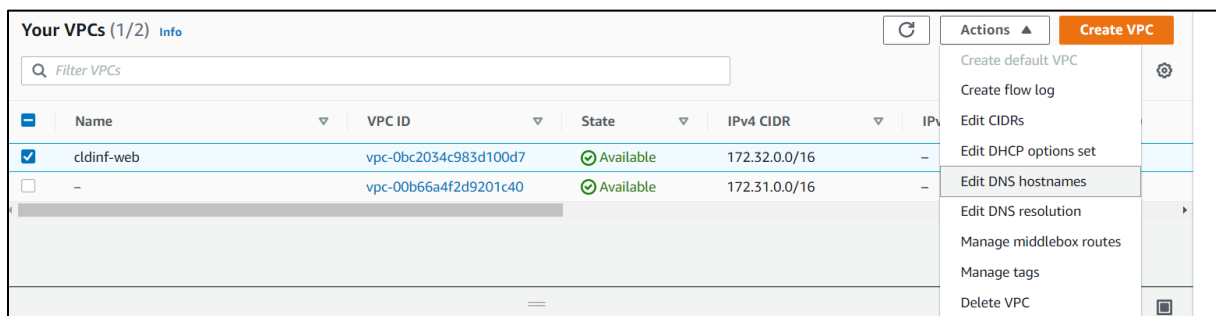


Figure 24 Setting up DNS hostnames

### 3.7 Create and Associate Route Tables

The route can be created by naming it and attaching it to our “cldinf-web” VPC. Then we edited the route table to have the route 0.0.0.0/0 (anywhere) target the internet gateway we have created:

Destination	Target	Status	Propagated
172.32.0.0/16	local	Active	No
0.0.0.0/0	igw-03a4efd9ccb6e38cd	Active	No

Figure 25 Route Table configuration

Associate the subnet which we have created before with the route table

**Subnets (1/7)**

Name	Subnet ID	State	VPC
-	subnet-0769f689a3b83bb8e	Available	vpc-00b66a4f2d9201c40
-	subnet-0f063d25f553e17ff	Available	vpc-00b66a4f2d9201c40
webserver	subnet-04f01af7c602acaa1	Available	vpc-0bc2034c983d100d7
-	subnet-01646d77402f93696	Available	vpc-00b66a4f2d9201c40
-	subnet-00ecef422aaf7db00	Available	vpc-00b66a4f2d9201c40
-	subnet-07f9c3de24c709f51	Available	vpc-00b66a4f2d9201c40

**Edit route table association**

**Subnet route table settings**

Subnet ID

Route table ID

Figure 26 Editing and Associating Route Table

### 3.8 Installing Environment

This part consists of installing PostgreSQL and Nodejs with the given commands in the lab document and setup the Service Files.

For PostgreSQL the commands alone sufficed. But for Nodejs we first had to replace the “Environment” variables in the service file again using given commands for the lab document

replacing following: AWS\_REGION=us-east-1 (tta-api file) and GROUP\_NAME=g07 (tta-web file).

Following the commands, we installed npm in the /opt/web folder and copied the file tta-api to /opt/.

Before starting the services, we had to edit the parameter store variables to match following:

```
The following variables have to be defined in the AWS Parameter Store:
|Parameter Store Variable|Description|Mandatory|
|---|---|---|---|
|/cldinf/dbUrl|Database Host IP or Domain Name|Yes|
|/cldinf/dbName|Database Name|Yes|
|/cldinf/dbUser|Database User|Yes|
|/cldinf/dbPassword|Database Password|Yes|
```

Figure 27 Parameter Store Variable Information

### 3.9 Adding IAM Role

To finish this section of the lab we had to add the pre-created IAM role to our Server. This role grants many AWS services access to other AWS services and has permissions very similar to the permissions we have as a user in the console. The name of the role is “LabInstanceProfile”. We simply added it to the EC2 instance under the security tab and then we could connect.

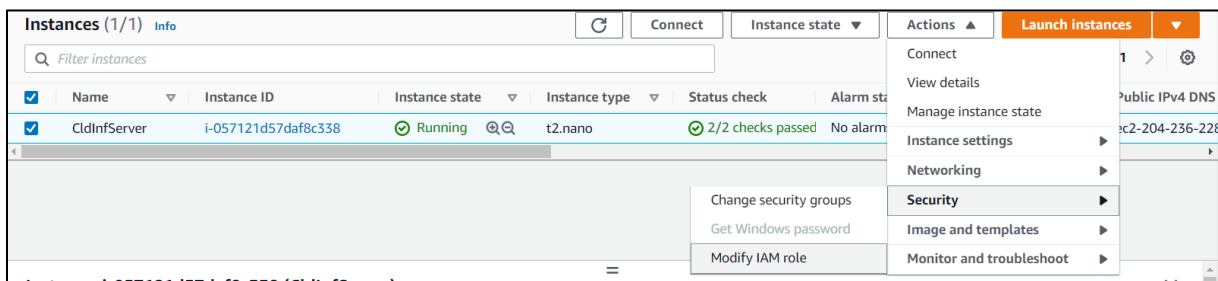


Figure 28 IAM role Security Tab

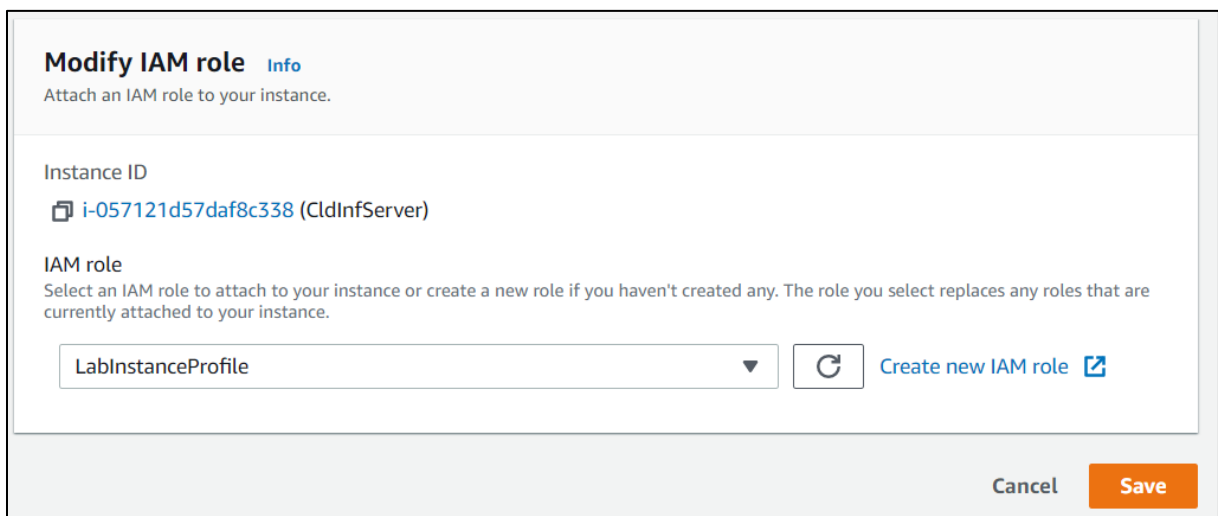


Figure 29 Adding IAM Role to instance

## 4 Migrating a Database to Amazon RDS

The database we use provides valuable information for the business we run it for, and we don't want to lose this information. This database should be durable, scalable, and work with high performance. Therefore, we migrate the database on our EC2 instance to an Amazon Relational Database Service (Amazon RDS). To be precise we transfer the already existing PostgreSQL database to the RDS.

The goal of this step was to create said database, import the data to it and connect an SQL client. Finally, we had to configure the EC2 instance to get its data from the RDS now instead of its own database.

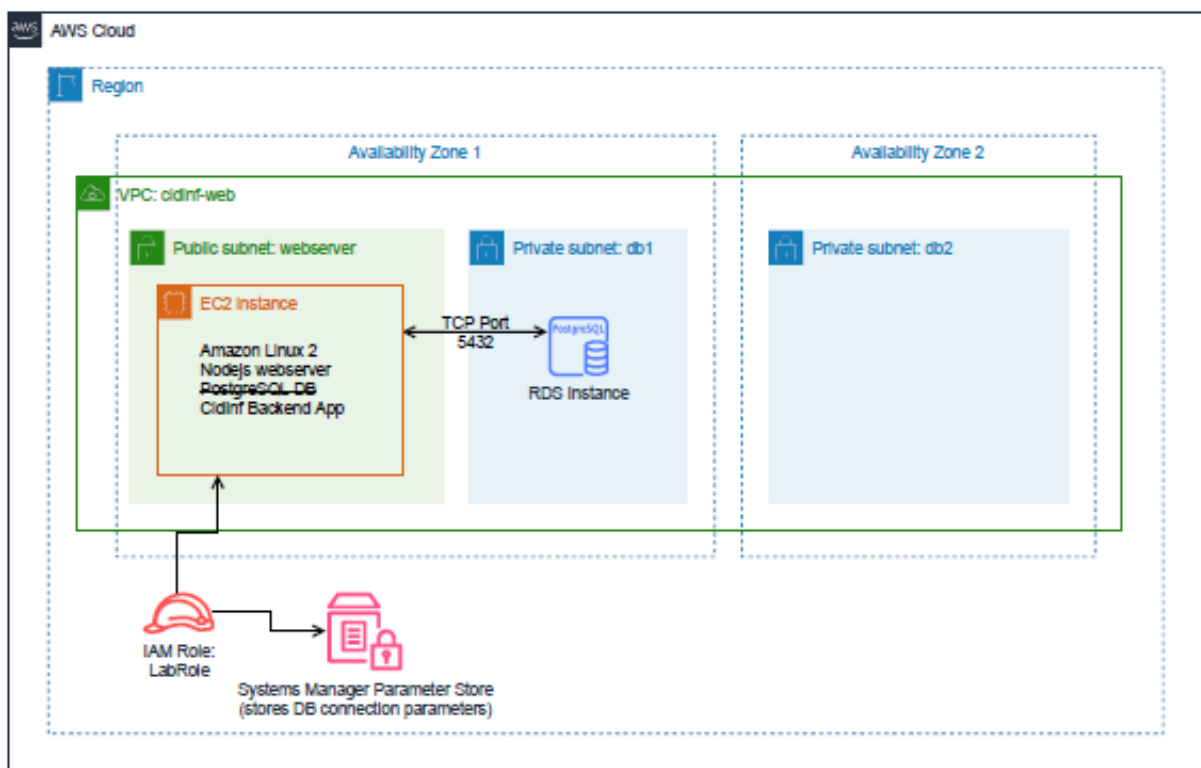


Figure 30 Example for a DB Migration to RDS instance

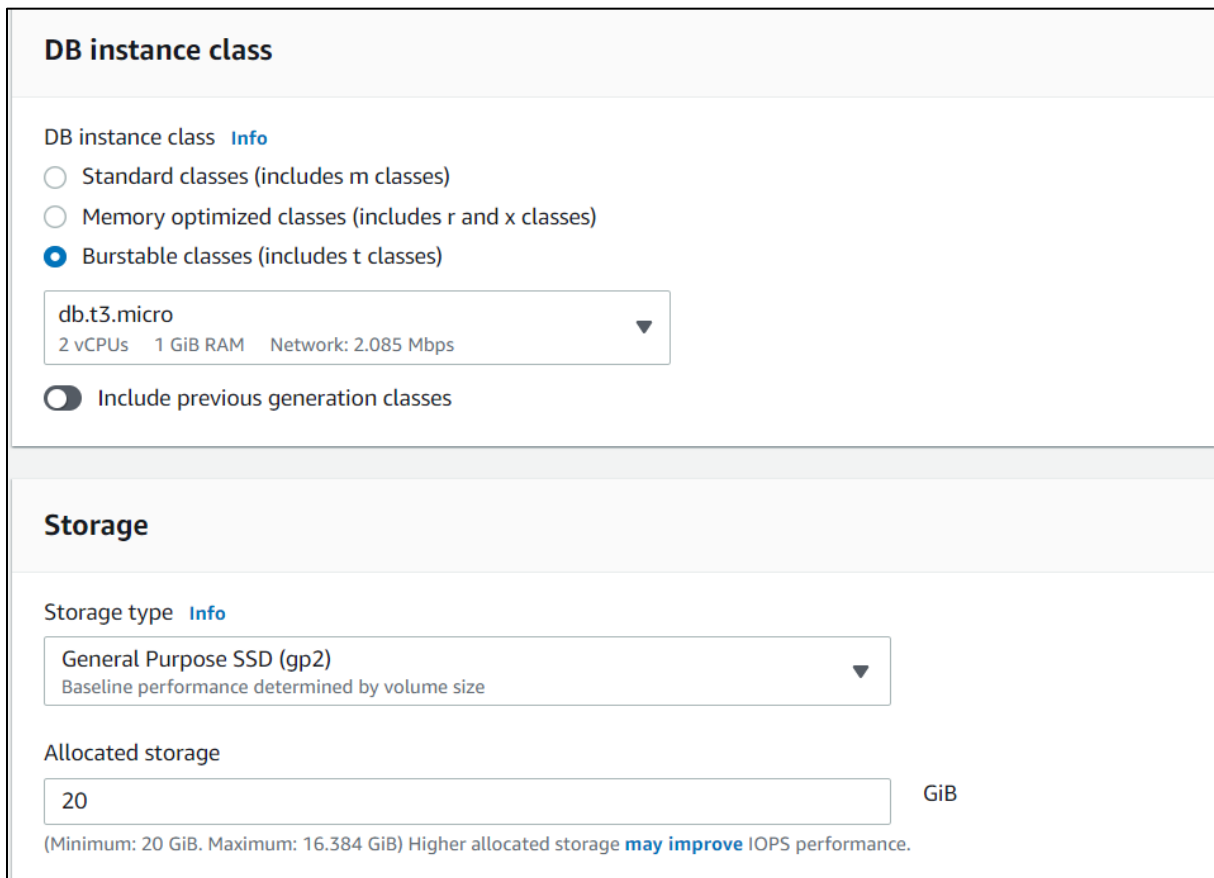
### 4.1 Creating the RDS Instance

#### 4.1.1 Setting up RDS with given specifications

Similar to the rest of the lab we had to find the right tab to configure the database. This time it could be found under RDS/Create database. To have the best configurability we selected the “Standard create” option instead of the “Easy create” and for the Engine options to use the PostgreSQL. We named our database “cidinf-db”.

Then we had to enter the credentials which were given to us in the lab (username and master password).

Next, we determined the DB instance class which decides the computation and memory capacity of our Amazon RDS DB instance and the allocated storage. These settings were also given by the lab document (db.t3.micro and 20GiB storage).



The screenshot displays the configuration options for an Amazon RDS DB instance. It is divided into two main sections: 'DB instance class' and 'Storage'.

**DB instance class**

- DB instance class [Info](#)
- ☐ Standard classes (includes m classes)
- ☐ Memory optimized classes (includes r and x classes)
- ☒ Burstable classes (includes t classes)

A dropdown menu shows the selected class: **db.t3.micro**, with specifications: 2 vCPUs, 1 GiB RAM, Network: 2.085 Mbps.

☐ Include previous generation classes

**Storage**

Storage type [Info](#)

A dropdown menu shows the selected storage type: **General Purpose SSD (gp2)**, with the note: 'Baseline performance determined by volume size'.

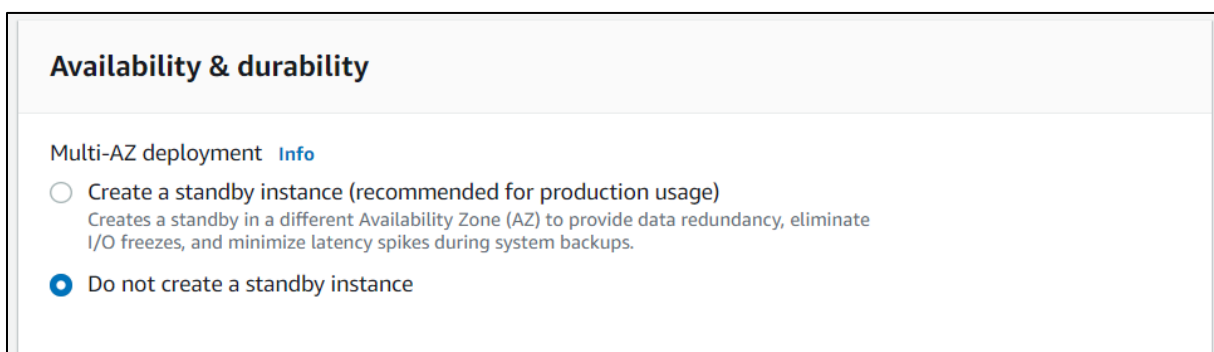
Allocated storage

A text input field contains the value **20**, followed by the unit **GiB**.

(Minimum: 20 GiB. Maximum: 16,384 GiB) Higher allocated storage **may improve** IOPS performance.

Figure 31 DB instance class and Storage

We didn't create a standby instance, which means that in the 2<sup>nd</sup> Availability Zone there won't be an exact copy of the database on standby. So, we basically don't have any redundancy, even though we will have to specify a 2<sup>nd</sup> subnet in another Availability Zone.



The screenshot displays the 'Availability & durability' section of the Amazon RDS console.

**Availability & durability**

Multi-AZ deployment [Info](#)

- ☐ Create a standby instance (recommended for production usage)  
Creates a standby in a different Availability Zone (AZ) to provide data redundancy, eliminate I/O freezes, and minimize latency spikes during system backups.
- ☒ Do not create a standby instance

#### 4.1.2 Setting up Connectivity

The RDS database needs to be placed in the same VPC as the EC2 instance (see figure 30). This can be configured in the connectivity tab of the instance. Additionally, we had to setup a subnet group for the database and add it to a VPC security group. The port to connect to could be specified in the “Additional Configuration” tab.

The screenshot displays the 'Connectivity' tab of an AWS RDS instance configuration. It includes sections for VPC selection, Subnet group creation, Public access options, and VPC security group selection. Below these is an 'Additional configuration' section for the Database port.

**Virtual private cloud (VPC)** [Info](#)  
VPC that defines the virtual networking environment for this DB instance.

cldinf-web (vpc-0bc2034c983d100d7) ▼

Only VPCs with a corresponding DB subnet group are listed.

**Subnet group** [Info](#)  
DB subnet group that defines which subnets and IP ranges the DB instance can use in the VPC you selected.

Create new DB Subnet Group ▼

**Public access** [Info](#)

☐ Yes  
Amazon EC2 instances and devices outside the VPC can connect to your database. Choose one or more VPC security groups that specify which EC2 instances and devices inside the VPC can connect to the database.

☒ No  
RDS will not assign a public IP address to the database. Only Amazon EC2 instances and devices inside the VPC can connect to your database.

**VPC security group**  
Choose a VPC security group to allow access to your database. Ensure that the security group rules allow the appropriate incoming traffic.

☒ Choose existing  
Choose existing VPC security groups

☐ Create new  
Create new VPC security group

**Existing VPC security groups**

Choose VPC security groups ▼

**▼ Additional configuration**

**Database port** [Info](#)  
TCP/IP port that the database will use for application connections.

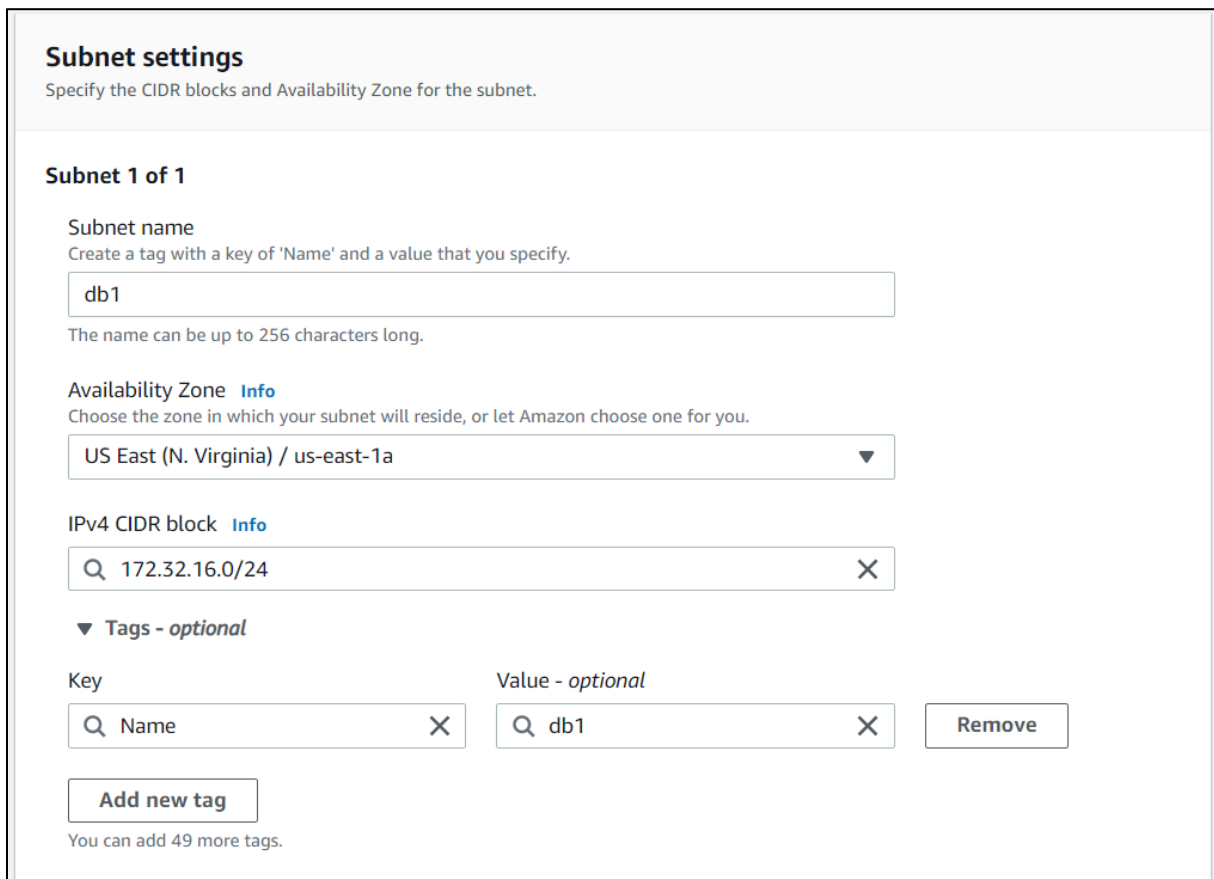
5432

Figure 32 Add instance to VPC and select Port

### 4.1.3 Creating Subnet group

Looking at the example in figure 30 we can see that there are two database subnets (db1 and db2) in different availability zones to allow for a certain security, availability, and redundancy. These two subnets were created as we did before but using a different zone for each (us-east-1a and -1b). Both subnets were associated with the cldinf-web VPC.

1a was configured with the CIDR block of 172.32.16.0/24 while 1b uses 172.32.17.0/24 (figure 35).



**Subnet settings**  
Specify the CIDR blocks and Availability Zone for the subnet.

**Subnet 1 of 1**

**Subnet name**  
Create a tag with a key of 'Name' and a value that you specify.  
  
The name can be up to 256 characters long.

**Availability Zone** [Info](#)  
Choose the zone in which your subnet will reside, or let Amazon choose one for you.

**IPv4 CIDR block** [Info](#)

▼ **Tags - optional**

Key	Value - optional	
<input type="text" value="Name"/>	<input type="text" value="db1"/>	<input type="button" value="Remove"/>

You can add 49 more tags.

Figure 33 Setting up Subnet 1a (1b is similar but different CIDR block)



Then we just needed to edit the DB subnet group and add the two created subnets to it:

## Edit DB subnet group

### Subnet group details

Subnet group name  
db\_subnet\_group

VPC ID  
VPC ID associated with the DB subnet group  
vpc-0bc2034c983d100d7

ARN  
arn:aws:rds:us-east-1:507347263555:subgrp:db\_subnet\_group  
e.g.:arn:aws:kms:<region>:<accountID>:key/<key-id>

Description  
Description of the DB subnet group.  
DB Subnet Group for Website

Figure 34 Edit DB subnet group

## Add subnets

Availability Zones  
Choose the Availability Zones that include the subnets you want to add.  
Choose an availability zone ▼

Subnets  
Choose the subnets that you want to add. The list includes the subnets in the selected Availability Zones.  
Select subnets ▼

### Subnets selected (2)

Availability zone	Subnet ID	CIDR block
us-east-1a	subnet-0125c00dd1db6b03a	172.32.16.0/24
us-east-1b	subnet-09425eb44a1b175fb	172.32.17.0/24

Figure 35 Adding Subnets 1a and 1b to the Group

Now that we have created this group, we could go back to the Connectivity options and select it in the dropdown menu:

**Connectivity**

Virtual private cloud (VPC) [Info](#)  
VPC that defines the virtual networking environment for this DB instance.

cldinf-web (vpc-0bc2034c983d100d7)

Only VPCs with a corresponding DB subnet group are listed.

**After a database is created, you can't change its VPC.**

Subnet group [Info](#)  
DB subnet group that defines which subnets and IP ranges the DB instance can use in the VPC you selected.

db\_subnet\_group

Public access [Info](#)

☐ Yes  
Amazon EC2 instances and devices outside the VPC can connect to your database. Choose one or more VPC security groups that specify which EC2 instances and devices inside the VPC can connect to the database.

☒ No  
RDS will not assign a public IP address to the database. Only Amazon EC2 instances and devices inside the VPC can connect to your database.

VPC security group  
Choose a VPC security group to allow access to your database. Ensure that the security group rules allow the appropriate incoming traffic.

☒ Choose existing  
Choose existing VPC security groups

☐ Create new  
Create new VPC security group

Figure 36 Final Connectivity Configurations

Now we just had to setup the initial name of the database and the DB parameter group. We disabled Monitoring, because else we couldn't have created the RDS instance, since we don't have the rights in the learner labs to activate "Enhanced Monitoring".

## 4.2 Verify EC2 Database and Exporting Data

To verify the data on the EC2 database we had to enter some commands: `psql --version` to check if the version running on the instance is postgresql-13. Then we connected to the database by entering: `psql -U postgres -h localhost postgres`. To observe the current data on the database we could enter a series of commands that show all the log events that were created.

```
postgres=# Select * from logs;
 id |      content      |      created_at
-----+-----+-----
  1 | localhost:8000    | 2021-11-16 12:51:18.874467+00
  2 | localhost:8000    | 2021-11-16 12:56:36.985507+00
  3 | localhost:8000    | 2021-11-16 12:56:37.387571+00
  4 | localhost:8000    | 2021-11-16 12:58:34.925101+00
  5 | localhost:8000    | 2021-11-16 12:58:39.095045+00
(5 rows)
```

*Figure 37 Postgres Log Event data on EC2 instance*

To export the data in this table we used the `pg_dump` utility: `pg_dump -U postgres -h <localhost> -F t postgres > /tmp/cldinf-db.tar` and to check if the dump was successful, we could look at the `/tmp` folder to see if the `.tar` file we dumped into was created.

### 4.3 Importing Data to new RDS Database

The last step before we could complete this part of the lab was to import the created .tar files data into the new RDS database. To successfully do that we had to first establish a network connection from the terminal running on the EC2 instance to the new RDS instance by updating the inbound rules of the security group that the RDS instance runs in. The port we allowed to connect was the before defined port 5432. To not create a security risk, we only opened the connection to servers in the security group that is used by the EC2 instance were connecting from.

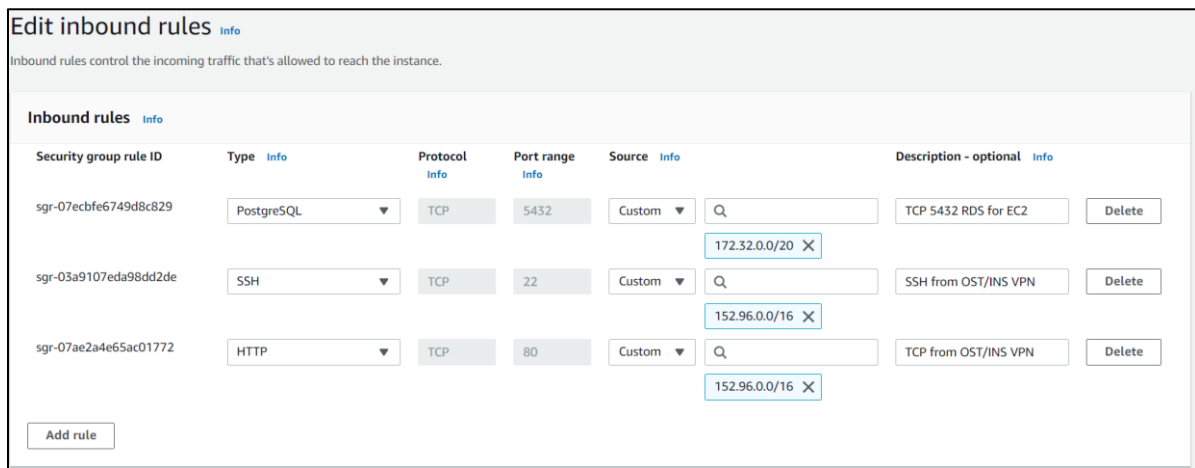


Figure 38 Updating Security Group of the RDS Instance

Then we connect to the RDS instance from our EC2 instance by entering: `psql -U postgres -h <rds-instance> postgres`. Run the `\l` command to show the databases and list all the tables of the “postgres” DB. Obviously there weren’t any tables at the time we did this, so we imported our data from the EC2 instance .tar with: `pg_restore -d postgres /tmp/cldinf-db.tar -c -U postgres -h <rds-instance>`. And check again if the data exists in the database with the commands we used before and were given us by the lab document.

```
[ec2-user@ip-172-32-7-61 ~]$ psql -U postgres -h cldinf-db.czm4hm4ch9uu.us-east-1.rds.amazonaws.com postgres
Password for user postgres:
psql (13.4, server 13.3)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.

postgres=> \l

               List of databases
  Name      | Owner   | Encoding | Collate | Ctype   | Access privileges
-----+-----+-----+-----+-----+-----
 postgres   | postgres | UTF8      | en_US.UTF-8 | en_US.UTF-8 | 
 rdsadmin   | rdsadmin | UTF8      | en_US.UTF-8 | en_US.UTF-8 | rdsadmin=CTc/rdsadmin
 template0   | rdsadmin | UTF8      | en_US.UTF-8 | en_US.UTF-8 | =c/rdsadmin      +
             |          |           |             |             | rdsadmin=CTc/rdsadmin
 template1   | postgres | UTF8      | en_US.UTF-8 | en_US.UTF-8 | =c/postgres      +
             |          |           |             |             | postgres=CTc/postgres
(4 rows)

postgres=> \ postgres
invalid command \
Try \? for help.
postgres=> \c postgres
psql (13.4, server 13.3)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
You are now connected to database "postgres" as user "postgres".
postgres=> \d

               List of relations
 Schema |      Name      | Type   | Owner
-----+-----+-----+-----
 public | logs            | table  | postgres
 public | logs_id_seq     | sequence | postgres
(2 rows)

postgres=> select * from logs;
 id |      content      |      created_at
-----+-----+-----
  1 | localhost:8000    | 2021-11-16 12:51:18.874467+00
  2 | localhost:8000    | 2021-11-16 12:56:36.985507+00
  3 | localhost:8000    | 2021-11-16 12:56:37.387571+00
  4 | localhost:8000    | 2021-11-16 12:58:34.925101+00
  5 | localhost:8000    | 2021-11-16 12:58:39.095045+00
(5 rows)
```

## 4.4 Confirming that Application still works

To confirm this, we had to connect the cldinf application to the new database and stop the database that runs locally on the EC2 instance by first updating the necessary values in the Parameter Store (dbUrl).

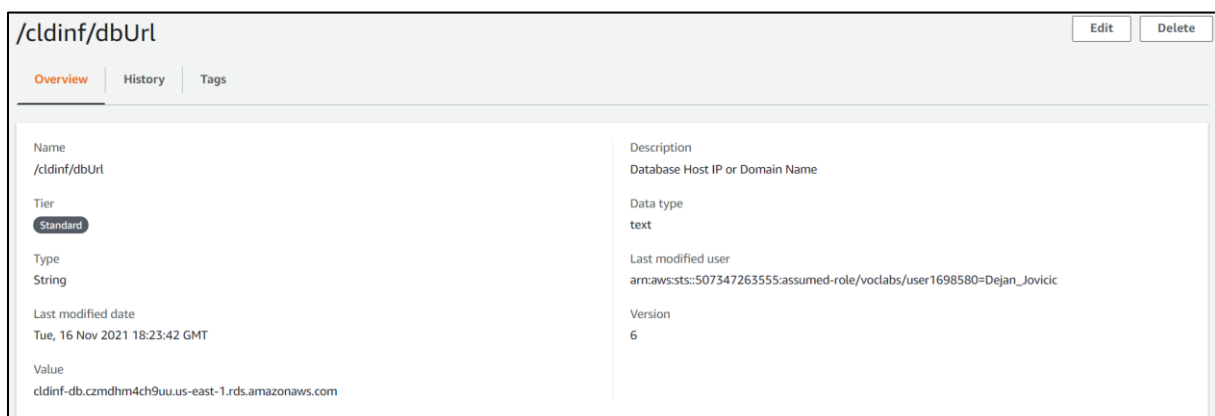


Figure 39 Changing dbURL Parameter Store Value to connect EC2 instance to new database

After changing this value, we could stop the running database on the EC2 instance and load the page again to confirm the data being taken from the right database.



Figure 40 Confirmation that the EC2 instance takes the data from the correct database

## 5 Implementing a scalable and highly available environment

First of all, we are going to create an Auto Scaling group named “autoscaling\_cldinf”. A launch template is required and therefore that is the next step.

EC2 > Auto Scaling groups > Create Auto Scaling group

Step 1

Choose launch template or configuration

Step 2

Choose instance launch options

Step 3 (optional)

Configure advanced options

Step 4 (optional)

Configure group size and scaling policies

Step 5 (optional)

Add notifications

Step 6 (optional)

Add tags

Step 7

Review

Choose launch template or configuration [Info](#)

Specify a launch template that contains settings common to all EC2 instances that are launched by this Auto Scaling group. If you currently use launch configurations, you might consider migrating to launch templates.

Name

Auto Scaling group name

Enter a name to identify the group.

Must be unique to this account in the current Region and no more than 255 characters.

Launch template [Info](#)

[Switch to launch configuration](#)

Launch template

Choose a launch template that contains the instance-level settings, such as the Amazon Machine Image (AMI), instance type, key pair, and security groups.

Required

[Create a launch template](#)

*Figure 41 Creating Autoscaling Group*

With the launch template, we can specify how the auto provisioned VMs will look like. We can specify the security group, in what VPC the VMs should be, IAM instance profiles, etc.

EC2

>

Launch templates

>

Create launch template

Create launch template

Creating a launch template allows you to create a saved instance configuration that can be reused, shared and launched at a later time. Templates can have multiple versions.

Launch template name and description

Launch template name - required

launchtemplate\_autoscaling\_cldef

Must be unique to this account. Max 128 chars. No spaces or special characters like '&', "'", '@'.

Template version description

Template for Autoscaling

Max 255 chars

Auto Scaling guidance

Select this if you intend to use this template with EC2 Auto Scaling

☒ Provide guidance to help me set up a template that I can use with EC2 Auto Scaling

Template tags

Source template

Launch template info

Switch to launch configuration

Launch template

Choose a launch template that contains the instance-level settings, such as the Amazon Machine Image (AMI), instance type, key pair, and security groups.

launchtemplate\_autoscaling\_cldef

Create a launch template

Version

Default (1)

Create a launch template version

Description

Template for Autoscaling

AMI ID

ami-04902260ca3d33422

Key pair name

vockey

Additional details

Storage (volumes)

-

Date created

Tue Nov 23 2021 13:51:42 GMT+0100 (Mitteleuropäische Normalzeit)

Launch template

launchtemplate\_autoscaling\_cldef

Instance type

t2.micro

Security groups

-

Request Spot Instances

No

Security group IDs

sg-0add5cb68d512d4c8

Figure 42 Creating Launch Template

After we've done this, we decided to attach our Auto Scaling group to a new load balancer, since we've haven't created a load balancer yet. The load balancer is an "Application Load Balancer", and the scheme should be "Internet-facing".

**Basic configuration**

**Load balancer name**  
Name must be unique within your AWS account and cannot be changed after the load balancer is created.  
  
A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.

**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.

Figure 43 Configuring Load Balancer

We add the two private subnets to the load balancer and create a new target group name for the listener.

**Network mapping**  
Your new load balancer will be created using the same VPC and Availability Zone selections as your Auto Scaling group. You can select different subnets and add subnets from additional Availability Zones.

**VPC**  
[vpc-04476e732b5d459a0](#) load\_balance\_cldinf

**Availability Zones and subnets**  
You must select a single subnet for each Availability Zone enabled. Only public subnets are available for selection to support DNS resolution.

☒ **us-east-1b**

☒ **us-east-1a**

**Listeners and routing**  
If you require secure listeners, or multiple listeners, you can configure them from the [Load Balancing console](#) after your load balancer is created.

Protocol	Port	Default routing (forward to)
<input type="text" value="HTTP"/>	<input type="text" value="80"/>	<input type="text" value="Create a target group"/>

**New target group name**  
An instance target group with default settings will be created.

**Tags - optional**  
Consider adding tags to your load balancer. Tags enable you to categorize your AWS resources so you can more easily manage them.  
  
50 remaining

Figure 44 Adding Subnets to Load Balancer



After we've done this, we set the "Desired Capacity" and "Minimum capacity" to 2 and the "Maximum capacity" to 6 on the Auto Scaling group. So, the minimum number of VMs should be 2, the maximum number 6. We also have to set the scaling policies as shown in this screenshot.

### Scaling policies - *optional*

Choose whether to use a scaling policy to dynamically resize your Auto Scaling group to meet changes in demand. [Info](#)

☒ **Target tracking scaling policy**  
Choose a desired outcome and leave it to the scaling policy to add and remove capacity as needed to achieve that outcome.

☐ None

Scaling policy name

Metric type

Average CPU utilization ▼

Target value

Instances need

seconds warm up before including in metric

☐ Disable scale in to create only a scale-out policy

Figure 45 Setting up Scaling Policies

## 5.1 Connection Setup for the Load Balancer and Autoscaled VMs

We have tried to construct this network, as shown in the Powerpoint presentation of Prof. Laurent Metzger.

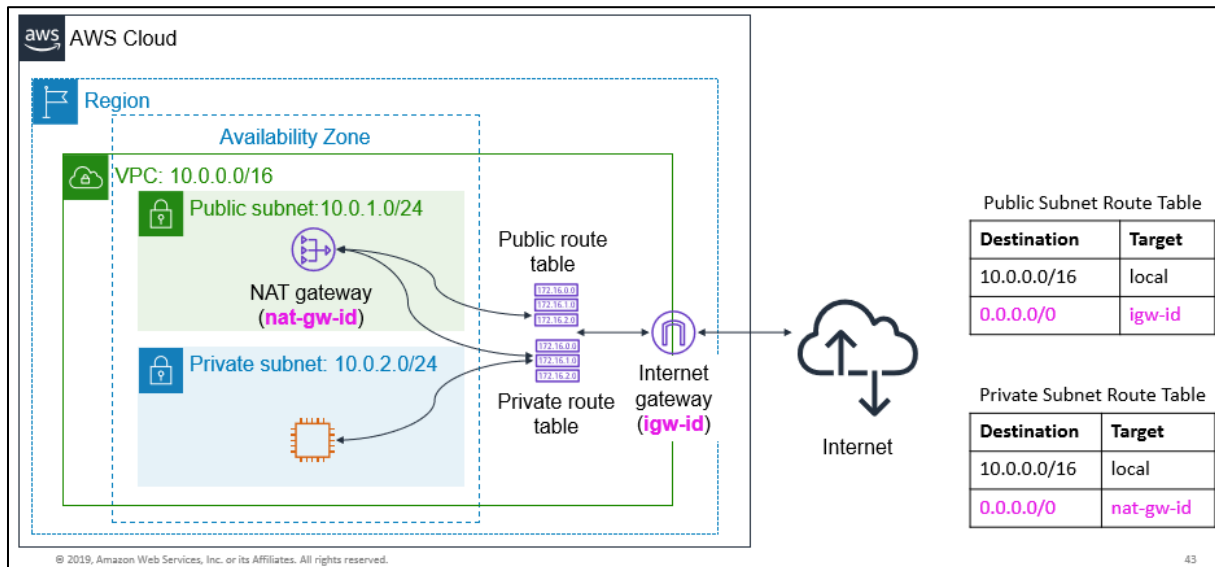


Figure 46 Source: Lecture "05\_AWS part 1" slide 43

The first step was to create a NAT gateway, in the public subnet (172.33.18.0/24) we've specifically created for this case.

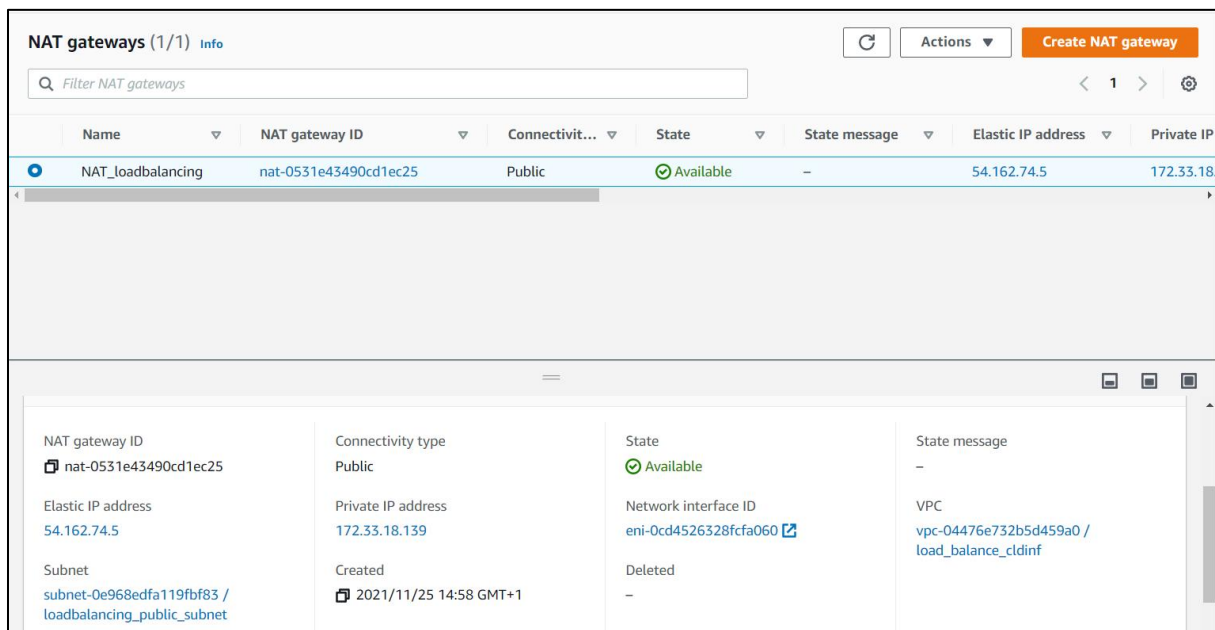
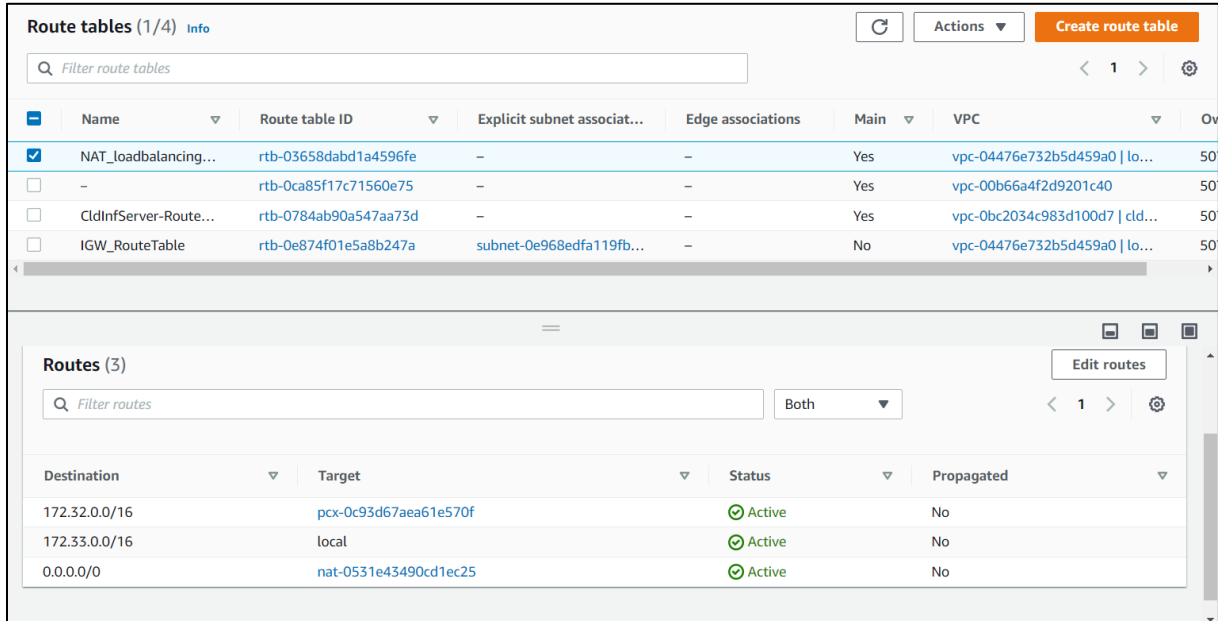


Figure 47 NAT gateway UI

Then we created a route table for our NAT connection, add route to 0.0.0.0/0 via the created NAT. The VMs in the subnets 172.33.16.0/24 and 172.33.17.0/24 should now connect to the Internet using the NAT Gateway. You can also see a route for the peering of two different VPCs. This was configured so we could connect from the EC2 instance we've previously

created in the lab, to the newly created VMs in the Auto Scaling group. We won't go into further detail, because this wasn't a part of the lab, and we did it only for our convenience.



**Route tables (1/4)** Info

Filter route tables

	Name	Route table ID	Explicit subnet associat...	Edge associations	Main	VPC	Overl...
<input checked="" type="checkbox"/>	NAT_loadbalancing...	rtb-03658dabd1a4596fe	–	–	Yes	vpc-04476e732b5d459a0   lo...	50
<input type="checkbox"/>	–	rtb-0ca85f17c71560e75	–	–	Yes	vpc-00b66a4f2d9201c40	50
<input type="checkbox"/>	CldInfServer-Route...	rtb-0784ab90a547aa73d	–	–	Yes	vpc-0bc2034c983d100d7   cld...	50
<input type="checkbox"/>	IGW_RouteTable	rtb-0e874f01e5a8b247a	subnet-0e968edfa119fb...	–	No	vpc-04476e732b5d459a0   lo...	50

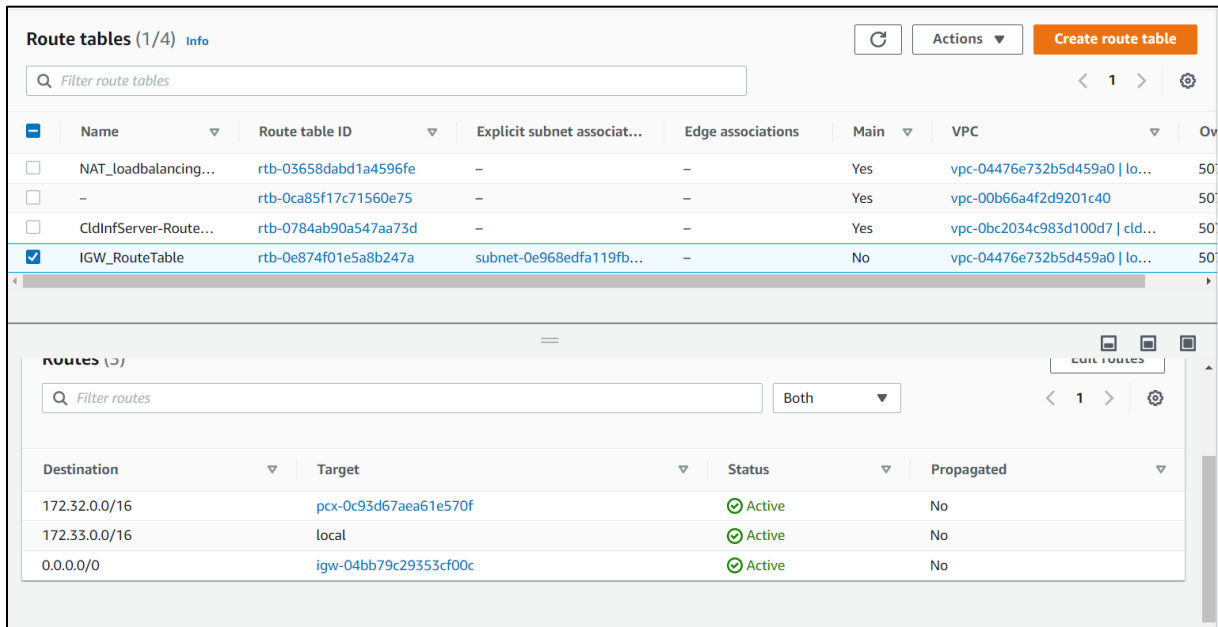
**Routes (3)**

Filter routes Both

Destination	Target	Status	Propagated
172.32.0.0/16	pcx-0c93d67aea61e570f	Active	No
172.33.0.0/16	local	Active	No
0.0.0.0/0	nat-0531e43490cd1ec25	Active	No

Figure 48 Creating NAT Route Table

We also had to create a route table for the Internet Gateway and add explicitly the public subnet 172.33.18.0/24. This must be done, since the IGW\_RouteTable isn't a main route table and therefore subnets have to be added explicitly.



**Route tables (1/4)** Info

Filter route tables

	Name	Route table ID	Explicit subnet associat...	Edge associations	Main	VPC	Overl...
<input type="checkbox"/>	NAT_loadbalancing...	rtb-03658dabd1a4596fe	–	–	Yes	vpc-04476e732b5d459a0   lo...	50
<input type="checkbox"/>	–	rtb-0ca85f17c71560e75	–	–	Yes	vpc-00b66a4f2d9201c40	50
<input type="checkbox"/>	CldInfServer-Route...	rtb-0784ab90a547aa73d	–	–	Yes	vpc-0bc2034c983d100d7   cld...	50
<input checked="" type="checkbox"/>	IGW_RouteTable	rtb-0e874f01e5a8b247a	subnet-0e968edfa119fb...	–	No	vpc-04476e732b5d459a0   lo...	50

**Routes (3)**

Filter routes Both

Destination	Target	Status	Propagated
172.32.0.0/16	pcx-0c93d67aea61e570f	Active	No
172.33.0.0/16	local	Active	No
0.0.0.0/0	igw-04bb79c29353cf00c	Active	No

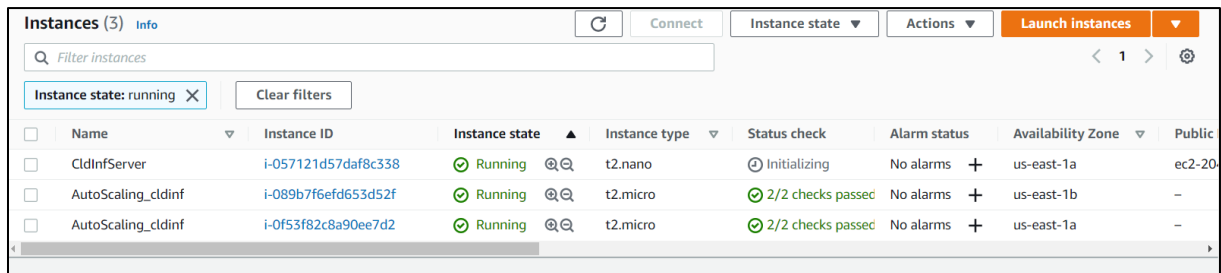
Figure 49 Internet Gateway for the Route Table

Sadly, this network design didn't work properly, as either the VM's in the private subnets 172.33.16.0/24 and 172.33.17.0/24 had internet connection, but then you couldn't have access to their "websites" through the load balancer or vice versa. If we added the private subnets to

the IGW Route Table, the load balancer was working fine, but the VMs didn't have any internet connection and therefore couldn't install NGINX. If we added the private subnets to the NAT Route Table, the VMs had internet connection, but you couldn't connect to them via the load balancer. This kind of made sense, since when you have a NAT, you shouldn't be able to connect from outside to inside. In the end, we didn't know what was missing in our setup and decided to give every VM a public IP, by changing the auto launch template, and put the private subnets in the IGW Route Table.

## 5.2 Autoscaling

Started the AWS Lab and we had instantly 2 VMs named "Autoscaling\_cldinf" getting provisioned.



	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public
<input type="checkbox"/>	CldInfServer	i-057121d57daf8c338	Running	t2.nano	Initializing	No alarms	us-east-1a	ec2-20
<input type="checkbox"/>	AutoScaling_cldinf	i-089b7f6efd653d52f	Running	t2.micro	2/2 checks passed	No alarms	us-east-1b	-
<input type="checkbox"/>	AutoScaling_cldinf	i-0f53f82c8a90ee7d2	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	-

Figure 50 Example for Autoscaling

After starting the stress test, we can see in the activity register, that the autoscaling works perfectly.

Status	Description	Cause	Start time
Successful	Launching a new EC2 instance: i-0a686a73953f727cc	At 2021-11-25T16:56:21Z a monitor alarm TargetTracking-autoscaling_cldinf-AlarmHigh-bccacfa8-2e88-4b9d-a3eb-fa31f880ad2e in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 2 to 6. At 2021-11-25T16:56:33Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 6.	2021 November 25, 05:56:36 PM +01:00
Successful	Launching a new EC2 instance: i-0bbc847c985768854	At 2021-11-25T16:56:21Z a monitor alarm TargetTracking-autoscaling_cldinf-AlarmHigh-bccacfa8-2e88-4b9d-a3eb-fa31f880ad2e in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 2 to 6. At 2021-11-25T16:56:33Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 6.	2021 November 25, 05:56:36 PM +01:00
Successful	Launching a new EC2 instance: i-0e04298df4003cc2b	At 2021-11-25T16:56:21Z a monitor alarm TargetTracking-autoscaling_cldinf-AlarmHigh-bccacfa8-2e88-4b9d-a3eb-fa31f880ad2e in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 2 to 6. At 2021-11-25T16:56:33Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 6.	2021 November 25, 05:56:35 PM +01:00
Successful	Launching a new EC2 instance: i-0a1afbe74ffc1ab5	At 2021-11-25T16:56:21Z a monitor alarm TargetTracking-autoscaling_cldinf-AlarmHigh-bccacfa8-2e88-4b9d-a3eb-fa31f880ad2e in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 2 to 6. At 2021-11-25T16:56:33Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 6.	2021 November 25, 05:56:35 PM +01:00

Figure 51 Activity Register to Check Autoscaling

There are now 6 “autoscaling\_cldinf” VMs.

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public
<input type="checkbox"/>	CldInfServer	i-057121d57daf8c338	Running	t2.nano	2/2 checks passed	No alarms	us-east-1a	ec2-20
<input type="checkbox"/>	AutoScaling_cldinf	i-0db3e65ec997c671e	Running	t2.micro	2/2 checks passed	No alarms	us-east-1b	-
<input type="checkbox"/>	-	i-0c00ef4699e73a833	Running	t2.micro	2/2 checks passed	No alarms	us-east-1b	-
<input checked="" type="checkbox"/>	AutoScaling_cldinf	i-0e04298df4003cc2b	Running	t2.micro	2/2 checks passed	No alarms	us-east-1b	-
<input type="checkbox"/>	AutoScaling_cldinf	i-0a1afbe74ffc1ab5	Running	t2.micro	2/2 checks passed	No alarms	us-east-1b	-
<input type="checkbox"/>	AutoScaling_cldinf	i-011b8202dfa1fd7f4	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	-
<input type="checkbox"/>	AutoScaling_cldinf	i-0bbc847c985768854	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	-
<input type="checkbox"/>	AutoScaling_cldinf	i-0a686a73953f727cc	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	-

Figure 52 Autoscaling after Test

We can also wait some time and see that the instances will get terminated again, after the CPU usage has been reduced.

Successful	Terminating EC2 instance: i-0bbc847c985768854	At 2021-11-25T17:25:23Z a monitor alarm TargetTracking-autoscaling_cldinf-AlarmLow-d5968f4c-e624-4a83-8c2a-5865b3b0921a in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 3 to 2. At 2021-11-25T17:25:32Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 3 to 2. At 2021-11-25T17:25:32Z instance i-0bbc847c985768854 was selected for termination.	2021 November 25, 06:25:32 PM +01:00
Successful	Terminating EC2 instance: i-0e04298df4003cc2b	At 2021-11-25T17:23:45Z a monitor alarm TargetTracking-autoscaling_cldinf-AlarmLow-2831714b-9cea-45c1-83c3-434e17b13706 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 4 to 3. At 2021-11-25T17:23:55Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 4 to 3. At 2021-11-25T17:23:56Z instance i-0e04298df4003cc2b was selected for termination.	2021 November 25, 06:23:56 PM +01:00
Successful	Terminating EC2 instance: i-0a686a73953f727cc	At 2021-11-25T17:22:27Z a monitor alarm TargetTracking-autoscaling_cldinf-AlarmLow-b7182220-40ab-4c7b-bdb4-a942a1c4dadf in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 5 to 4. At 2021-11-25T17:22:38Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 5 to 4. At 2021-11-25T17:22:38Z instance i-0a686a73953f727cc was selected for termination.	2021 November 25, 06:22:38 PM +01:00
Successful	Terminating EC2 instance: i-0a1afbe74ffc1ab5	At 2021-11-25T17:21:27Z a monitor alarm TargetTracking-autoscaling_cldinf-AlarmLow-b7182220-40ab-4c7b-bdb4-a942a1c4dadf in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 6 to 5. At 2021-11-25T17:21:40Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 6 to 5. At 2021-11-25T17:21:40Z instance i-0a1afbe74ffc1ab5 was selected for termination.	2021 November 25, 06:21:40 PM +01:00

Figure 53 Activity Register to Check Autoscaling after Test

## 5.3 Load Balancing

We can see that all clients are healthy.

Targets

Monitoring

Health checks

Attributes

Tags

Registered targets (6)

↺

Deregister

Register targets

⏮

1

⏭

⚙

🔍

Filter resources by property or value

<input type="checkbox"/>	Instance ID	Name	Port	Zone	Health status	Health status details
<input type="checkbox"/>	i-0a686a73953f727cc	AutoScaling_cldinf	80	us-east-1a	🟢 healthy	
<input type="checkbox"/>	i-0e04298df4003cc2b	AutoScaling_cldinf	80	us-east-1b	🟢 healthy	
<input type="checkbox"/>	i-04883cfadf6f82717	AutoScaling_cldinf	80	us-east-1a	🟢 healthy	
<input type="checkbox"/>	i-0bbc847c985768854	AutoScaling_cldinf	80	us-east-1a	🟢 healthy	
<input type="checkbox"/>	i-0a1afbe74ffcf1ab5	AutoScaling_cldinf	80	us-east-1b	🟢 healthy	
<input type="checkbox"/>	i-01a5fd35647857f91	AutoScaling_cldinf	80	us-east-1b	🟢 healthy	

Figure 54 "Healthy"-Check for the Clients

And when accessing the load balancer, we can see that it goes round robin through all the instances:

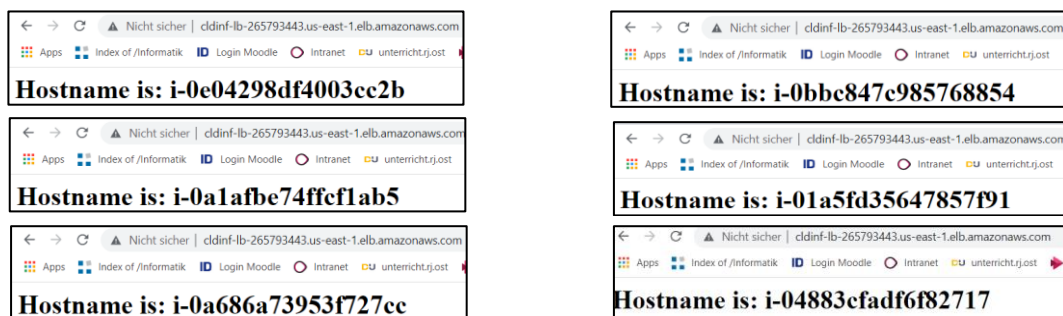


Figure 55-60 Hostnames shown on the website