Cloud Application Architecture- Lab 1

Contents

Aim of the Laboratory

- Set the goal for this semester
- Access your AWS account
- Familiarize yourself with the AWS console, regions and services
- Create an EC2 instance
- Run the “legacy” online shop application in a “lift and shift” manner on EC2

The Goal

Migrate and adapt a traditional application to AWS cloud.

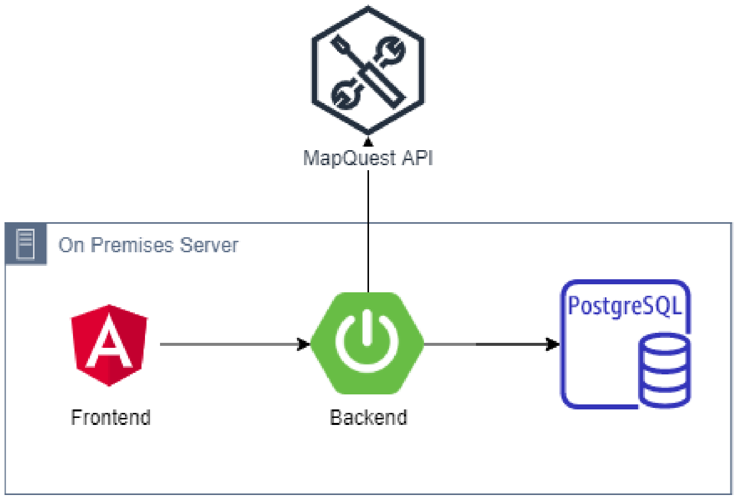


Figure 1 Initial Architecture

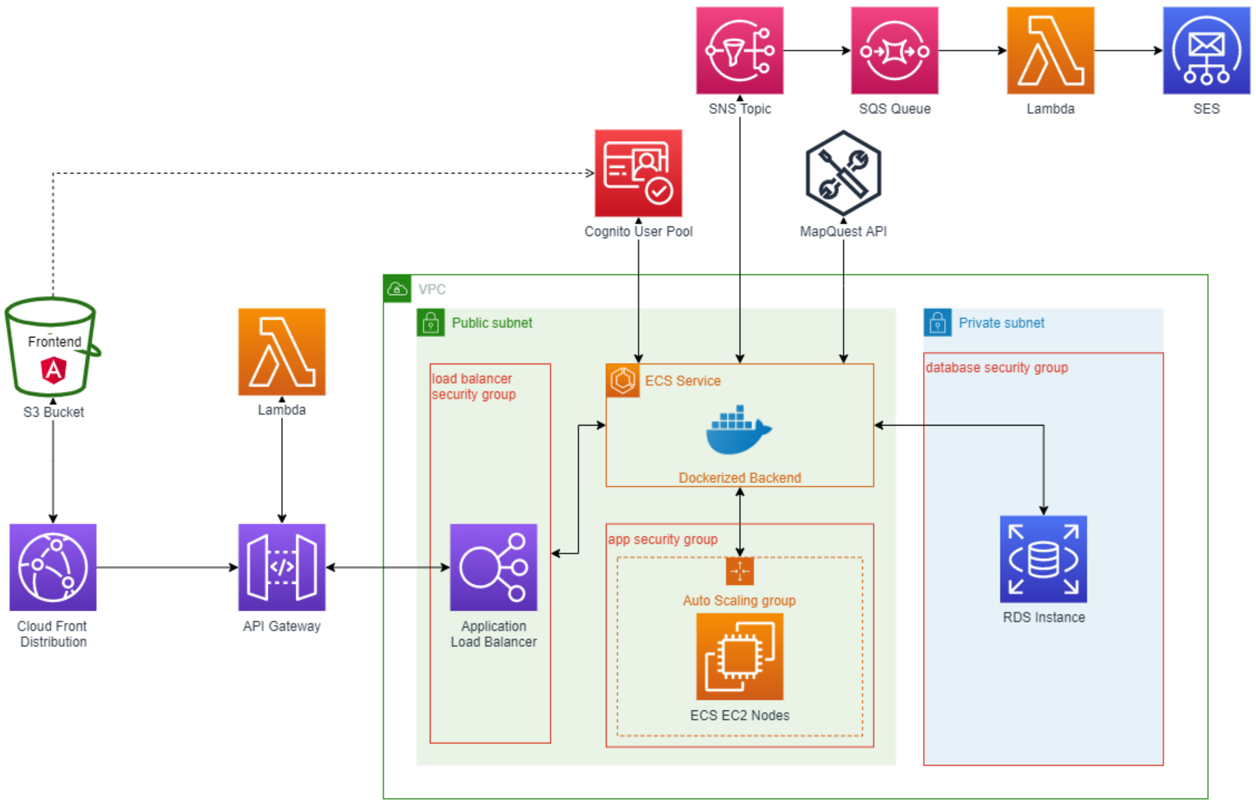


Figure 2 Final Architecture

Introduction into AWS - important concepts and terminology

AWS Intro

AWS is one of the many cloud providers available today. This wasn't always the case. Amazon launched it in 2006 as a way to increase their revenue by renting their computing resources - they had a lot of spare capacity left after scaling their resources for holiday sales. It took several years for any competitor to respond which translated to an overwhelming market share owned by AWS even today and to more mature services.

It currently offers over 200 services for various use cases (hosting, AI, ML, IoT, media encoding and streaming etc.). Some of them are very specific to a certain use case while others are more general and most likely are the building blocks of the rest. We will focus on the latter.

AWS Regions, AZs

"Amazon cloud computing resources are hosted in multiple locations world-wide. These locations are composed of AWS Regions, Availability Zones, and Local Zones. Each *AWS Region* is a separate geographic area. Each AWS Region has multiple, isolated locations known as *Availability Zones*."

An overview of all AWS regions is available at https://www.infrastructure.aws/

The AWS Console

The **AWS Console** is a management interface that acts as a central entry point towards all AWS services. In addition to the console, AWS cloud resources can also be managed through the AWS **CLI** or programmatically by using the AWS **REST API/SDK**.

"Online Shop" Application Overview

In this semester you will be dealing with an existing Online Shop application. Throughout the laboratories, you will transform the architecture and move from a "legacy" application to a fully cloud-native architecture leveraging multiple managed services of AWS.

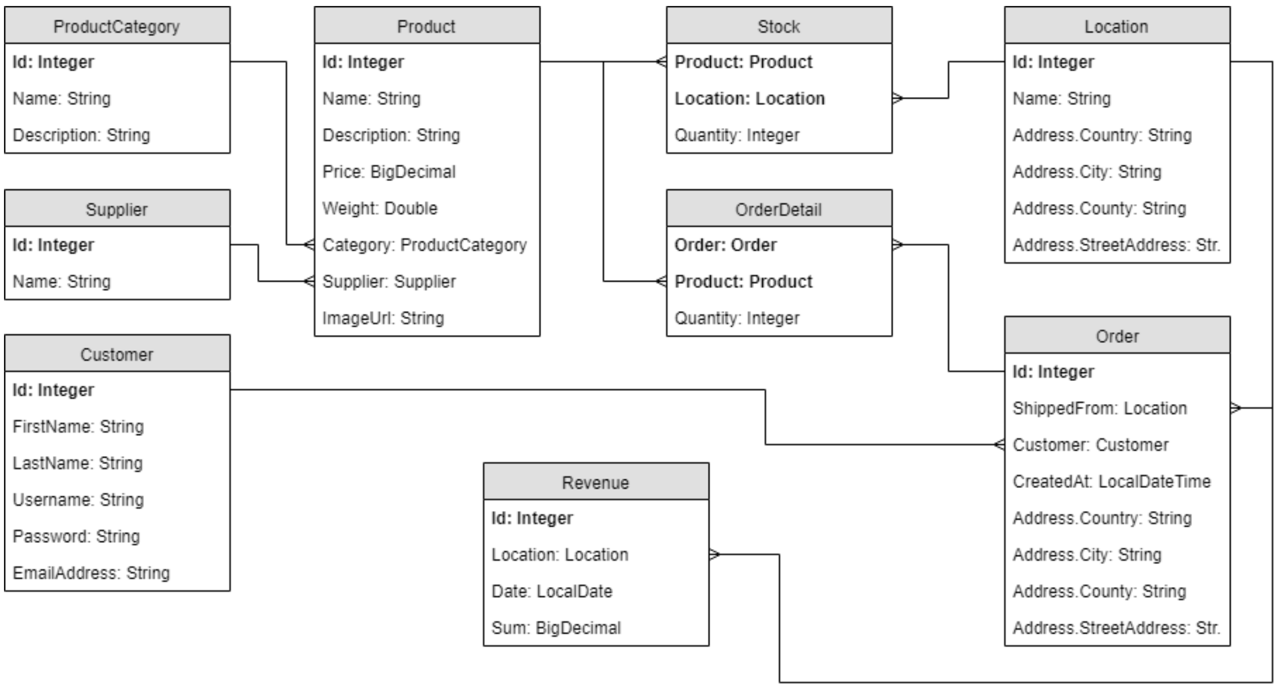
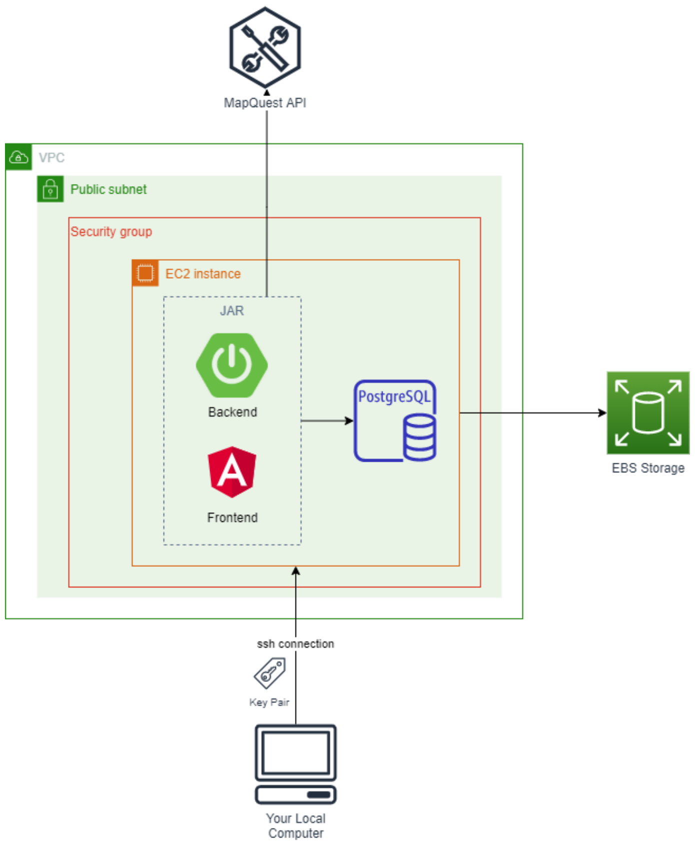


Figure 3 Data Model

Source code for the application is available here.

At the end of this lab, our application will look like this:



Lab Walkthrough

Access your AWS account

AWS Console URL: https://console.aws.amazon.com/console/home

Throughout this semester you will work in your own AWS account. For credentials, ask the lab assistant.

Navigate through the AWS services

The ***Services*** menu entry on the upper left side of the console lists all available AWS services. Have a look at the Compute, Storage and Database sections and see if you can answer the following questions:

- What Database service does AWS offer for NoSQL databases?
- What is the S3 service?
- What service can be used to build applications using a FaaS (function as a service) paradigm?

If you cannot answer all questions, no worries, we will have a look at them in the later part of the semester.

Select a region

A dropdown in the upper-right side of the console lists all available regions. Select *eu-west-1* (Europe Ireland).

Note

Regions serve multiple purposes. They are fundamentally different/independent clouds. They even receive new services at different rates (Ireland is the first one to receive new stuff in Europe). They are highly relevant when your application is addressing the global market (e.g. Netflix) and/or when you cannot accept any kind of downtime (in extremely rare cases, an entire region might go down).

Create an EC2 instance backed by EBS storage

We will now create our first AWS virtual machine.

Launch Instance

Navigate to the *EC2* service and, in the *instances* section, select "Launch Instance".

Select AMI

Note

An Amazon Machine Image (AMI) provides the information required to launch an instance**, including the operating system and additional software**. You must specify an AMI when you launch an instance. You can launch multiple instances from a single AMI when you need multiple instances with the same configuration.

Select **Amazon Linux 2** for the AMI (64-bit x86 or Arm). This an AWS-optimized Linux flavor that can be used as part of the AWS free tier.

Instance Type

Note

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instance types comprise varying combinations of CPU, memory, storage, and networking capacity and give you the flexibility to choose the appropriate mix of resources for your applications.

Select a general purpose **t2.micro** instance type. This provides a machine with 1 vCPU and 1 GB memory.

Configure Instance Details

On this section you define further details such as the virtual network (VPC) and subnet to which the VM is allocated. Take a quick look at the explanations of some of the options.

Leave the default values and scroll down to the User Data section. Here you can define a script that will be executed automatically. We will use it to install some prerequisites on the machine (such as the java runtime environment etc.).

User Data:

```
#!/bin/bash
sudo yum -y update

# Install Maven and set the environmental variables
wget http://mirror.olnevhost.net/pub/apache/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz
tar xvf apache-maven-3.6.3-bin.tar.gz
sudo mv apache-maven-3.6.3 /usr/local/apache-maven
echo 'export M2_HOME=/usr/local/apache-maven' >> /home/ec2-user/.bashrc
echo 'export M2=$M2_HOME/bin' >> /home/ec2-user/.bashrc
echo 'export PATH=$M2:$PATH' >> /home/ec2-user/.bashrc
sudo rm apache-maven-3.6.3-bin.tar.gz

# Install Amazon Correto (AWS's implementation of the JDK)
sudo amazon-linux-extras enable corretto8
sudo yum install -y java-1.8.0-amazon-corretto-devel

# Remove the default JDK
sudo yum remove -y java-1.7.0-openjdk

# Install git
sudo yum install -y git

# Install Docker for running Postgres
sudo amazon-linux-extras install docker
sudo service docker start
sudo systemctl enable docker
sudo usermod -a -G docker ec2-user

# Pull, configure, and run the Postgres image
```

```
docker pull postgres
mkdir ${HOME}/postgres-data/
docker run -d \
        --name dev-postgres \
        -e POSTGRES_PASSWORD=postgres \
        -v ${HOME}/postgres-data/:/var/lib/postgresql/data \
    -p 5432:5432 \
    --restart always \
    postgres
```

Could we also automate the deployment of our application in this way?

Storage

The Amazon Linux 2 image uses an attached 8GB network storage by default. No need to change anything here.

Tags

Note

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value, both of which you define. Tags enable you to categorize your AWS resources in different ways, for example, by purpose, owner, or environment.

The bigger the project/system, the more important tags get.

Add the tag **project**: **caacourse** to both the instance and the EBS volume. This tag will help us track the resources and costs generated by the lab. **We will use this tag for all resources from now on.**

Security Group

We will connect to this instance via SSH, that's why it is important to leave port 22 open for incoming connections. However, you should not be doing this in a production environment as it exposes your instance to security threats.

Also, since the application will be running on port 8080, allow incoming connections on this port as well (as a custom TCP rule.

Review and Launch

Take a quick glance at all the configuration and proceed.

Key Pair

In order to connect to the instance, you will need to create an SSH key pair. Select "Create new key pair", provide a name, download the file and keep it safe.

Launch the EC2 instance and select view instance. It might take a few minutes until it's ready – use this time to configure ssh access.

Connect to your instance via SSH

Note

Main cloud providers, including AWS, provide a simpler way to access our instances directly from the browser. In the case of EC2 instances, after selecting the instance, there should be the **Connect** button which will open a shell connected to the instance in a new browser tab. However, since not all providers have this option, practicing the universally accepted way (with ssh) might be useful in the future.

Prerequisites

- **Your public IP address and DNS name**

By default, the instance receives a public IP address and a DNS name (based on the IP address). This should be visible on the "Description" tab of the EC2 service. Copy it as you will need it for the ssh connection.

- **SSH Client**

You will need a **ssh client** to connect to your instance.

a) Linux

Use the command line ssh client. On Linux you also need to make sure that the private key file is not publicly viewable on your computer. Use "chmod" for this

```
> chmod 400 my-key-pair.pem
```

b) Windows

Either use putty (https://www.putty.org/) or the command line client that comes with git bash (https://gitforwindows.org/). Newer builds of Windows 10 also come with a built-in ssh client (OpenSSH) (more info here).

For using putty, take a look at the official AWS guide.

For using the command line tool:

```
> ssh -i /path/my-key-pair.pem ec2-user@<public_ip_address>
```

Confirm that you trust the authenticity of the host (type yes).

Legacy Application "lift and shift"

Now that you have connected to your own EC2 instance, it is time to run the Online Shop on the VM.

Download the release from Github and run it:

```
> source ~/.bashrc
```

```
> curl -L -O https://github.com/CAA-Course/app/releases/download/1.0/shop-1.0.jar
```

Start the application

```
> java -Dspring.profiles.active=with-form,local -jar shop-1.0.jar
```

Your application should now be accessible at the above-mentioned public IP address (don't forget about the port). Try it out!

The default credentials are:
User: **user**
Password: **storepass**

Hint: you can stop the app by pressing Ctrl+C. Another useful tip when working with Linux is how to exit vim – press Esc and type ":wq" (w = write/save, q = quit).

Bonus Task

Configure the app to run as a Linux service. In our case, this is nice to have since we will stop the instances at the end of each lab. In practice, this is mandatory to avoid downtime as much as possible.

Hint: One way is using *systemd* and *systemctl*.

Cleanup

In the cloud world you pay for what you use, so make sure you stop the EC2 instance before the end of the laboratory. If you remember to stop your instances after each lab, you get a bonus point at the lab exam.

From the AWS Console, select the instance and go to **Actions**. In the **Instance state** menu, press **Stop instance**.

Important: When you stop an instance, the public IP address will be released. When you start the instance again, it will have a free IP address (and DNS name).