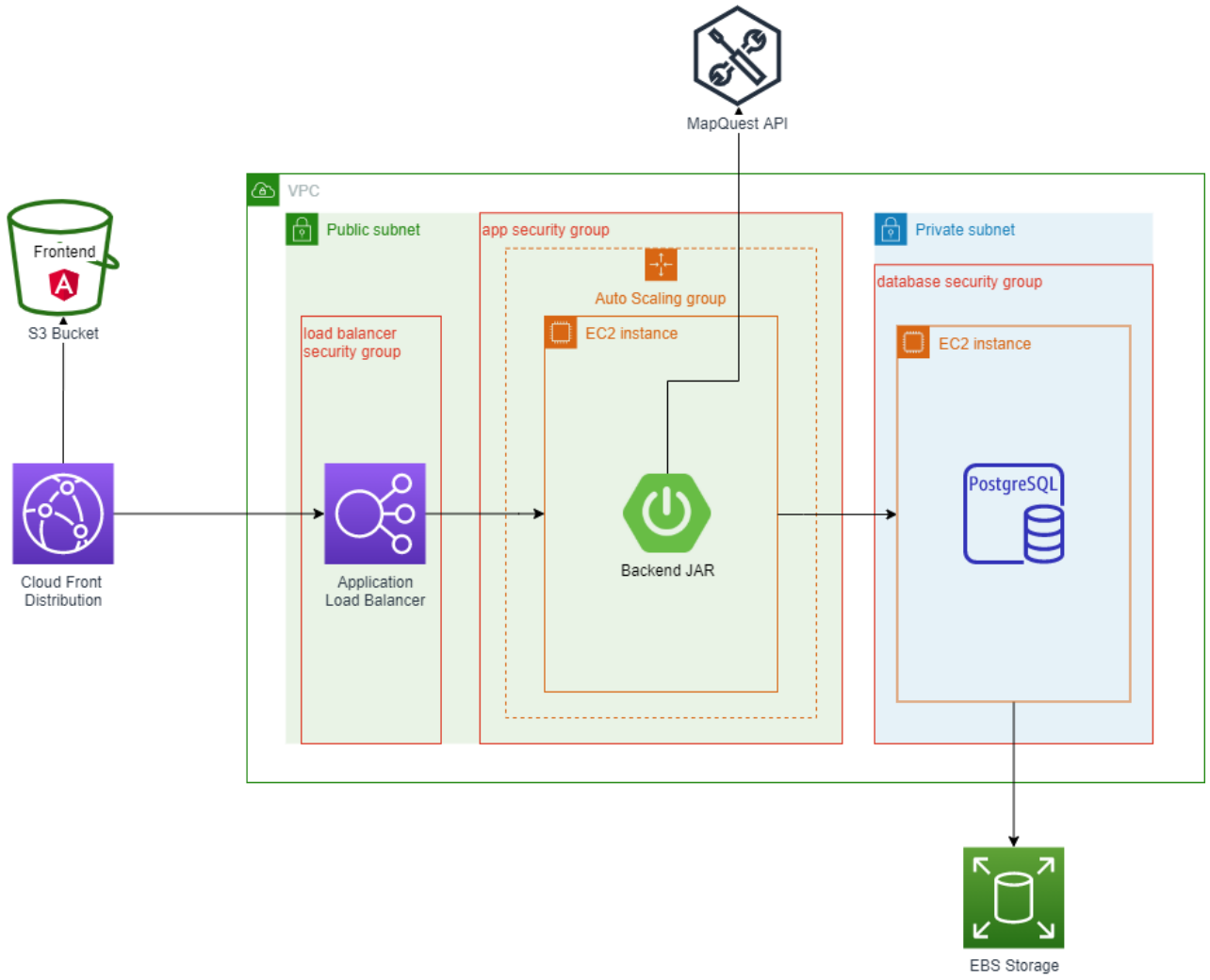
Cloud Application Architecture – Lab 3

Aim of the Laboratory

- Familiarize yourself with the front end of the application
- Connect it to the application
- Deploy it on AWS
- CDN

The New Architecture



Intro

In the previous lab, we extracted the **database** into a separate EC2 instance, worked with **networking** services, and made our **application tier highly scalable**.

Another important component of our system is the **front end**. Right now, the application tier serves the web pages. While this works fine for many cases, it adds unnecessary load on all components of our system, which also leads to unnecessary charges (due to increased traffic and resource usage).

Since the last decade, the approach has been a bit different: let the back end do its job (i.e. process data) and move the presentation layer (front end) somewhere else. This approach is also encouraged by the emerging UI frameworks that we have today (react, angular, vue, polymer, svelte, and many more). Such frameworks generate static files (HTML, CSS, JS) and load the data from the back end (usually through HTTPS requests) at runtime (i.e. when the user visits our website). Since they are static files, we don't really need computing resources to host them (i.e. the host doesn't have to do any processing other than sending the files as they are to the client/requester).

In this lab, we will deploy the front-end application separately using suitable AWS services and connect it to our application (which will act just as a back-end).

Getting started with the app

Before we proceed, please make sure you have the following tools installed on your machine:

- git (check with command ***git --version*** - should be greater than 2.27.0)
- node (check with command ***node -v*** - should be greater than v12.16.1, but older versions should also be enough)

We will extract the front end out of our application. You can find the app here:

https://github.com/CAA-Course/app (frontend directory)

Clone the repo on your computer and let's take a look at it (navigate to a suitable directory and run ***git clone <link from above>***).

Being an **angular** application, it uses **npm** to manage its dependencies (e.g. angular itself is installed as a library and a dependency of our UI app). Npm comes bundled with Node.js so you should already have it (you can check using the command ***npm -v*** - you should see something like 6.13.14). You can find more about npm here or by simply googling it (it's by far the largest package registry so you will most likely find something).

When working with such JS apps, the usual steps involve installing the dependencies (listed in *package.json*) with the command **npm install**, filling in some environmental variables if necessary, and running the app with **npm start**, **npm run start**, or **npm run dev** (it really depends on the developers of the application and how they defined the script).

Consuming Our Back-end

Prerequisite: follow the previous lab guide to set up the back end.

It's time to bring some actual data into our new web app. For that, we will configure it to communicate with our existing application running on EC2 behind a load balancer.

TODO:

- Find where to configure the server hostname
- Hint: these settings are usually stored in **environment** files
- Update the target to the DNS name of the load balancer (add the "http://" prefix)

Once done, you should be able to run the app locally (npm run start) and retrieve data.

Building the front-end app

Generate the production build of the front-end app with **npm run build**.

Finding a Suitable AWS Service for Deploying It

AWS offers a huge array of services, some of them being used more than others. Just like EC2, **Simple Storage Service (S3)** is without a doubt one of the most used services out there. What does it do? Simple. It stores data (objects). It is highly durable (your data will not get corrupted), highly available (it basically never goes down) and allows you to store virtually an infinite amount of data.

Another important aspect is that it works with **HTTP(S)**. If, for example, an EBS volume must be attached to an EC2 instance in order to access it, S3 is accessed directly through HTTP(S) without having to mount it to anything.

When we want to store files in S3, we first create a bucket (a container for our files). Each bucket has its own URL in the form of
https://<bucket_name>.s3.<region>.amazonaws.com/<file_name>

Object vs Block Storage

We are most likely used to block storage. This is what our computers use to store all our data (and OS and programs). Files are divided into blocks of data. By contrast, object storage stores files as one big chunk of data and metadata that is identified by an ID (usually the name of the file). The main consequence is that object storage doesn't allow us to update/edit files. If we want to update a file, we re-upload the whole file.

Bucket Access

Buckets are private by default. This means that only the owner (the AWS account) of the bucket can access it. We can give access to other AWS accounts, or, if needed, we can make it public. AWS deliberately made it slightly more complicated to make buckets public in order to avoid/reduce data leaks. Of course, if we use a bucket to host our website, we should make it public.

Static Hosting

Since S3 already serves our files through HTTPS, it takes only one small step to serve our website – telling S3 which file to serve as the index of the website (i.e. the file that is served when our users access the URL of our bucket). This usually is *index.html*.

The URL of our website will be *http://<bucket_name>.s3-website-eu-west-1.amazonaws.com/* (notice the slight difference from the usual bucket URL).

Deploying to S3

TODO:

- Create a bucket
- Make it publicly accessible
    - Disable *block public access*
    - *Add a bucket policy that allows read access for all files*
- Configure it for hosting
- Upload the web app (only the built files and contents of the **dist/online-shop** directory)
    - **Make the files publicly accessible (Read object)**
- Access it in your browser

You can always consult the official tutorial here.

One major issue of using S3 for hosting is that it doesn't support HTTPS (only plain HTTP) (this only applies to **hosting**; S3 works with HTTPS for other scenarios). The usual approach involves CloudFront, another AWS service.

CDN

A CDN (content delivery network) is used to serve files as fast as possible around the globe. It is basically a highly distributed network, thus being geographically close to as many users as possible. AWS's CDN is CloudFront and the network is made of edge locations.

We can use CloudFront to distribute **any HTTP-accessible** data, including our website hosted on S3.t

TODO:

- Create a web distribution for your bucket. The relevant fields are:
    - Origin Domain Name: your bucket
    - Restrict Bucket Access: select 'Yes'. Only CloudFront should be able to access your bucket.
    - Origin Access Identity: create a new identity
    - Grant Read Permissions on Bucket: we want CloudFront to update the bucket policy automatically
    - Set the **default root object** to *index.html*
    - ~~Viewer Protocol Policy: we want to redirect all plain HTTP traffic to HTTPS.~~

It will take a few minutes for your distribution to be deployed. Use this time to understand the generated bucket policy.

Once your distribution is deployed, you can access it using the domain displayed in the **general** section. You will notice that you get automatically redirected to the bucket URL. This is due to DNS propagation on the AWS side. After a few hours, this should no longer happen.

Question: How would you test CloudFront from multiple locations/continents?

Cleanup

Delete the CloudFormation stack. Keep the CDN and the S3 bucket.