Frankfurt University of Applied Sciences
Faculty 2: Computer Science and Engineering
Nibelungenplatz 1
60318 Frankfurt am Main

A REPORT

ON

"Kubernetes and Docker based Content Delivery Network Application (CDN_App)"

for

Cloud Computing (WS2021) Project
of
M.Sc. in High Integrity Systems

SUBMITTED BY

Syed Ahmed Zaki (ID: 1322363)
syed.zaki@stud.fra-uas.de

Daniel von Barany(ID: 1323045)
daniel.vonbarany@stud.fra-uas.de

UNDER THE GUIDANCE OF

Prof. Dr. Christian Baun

Contents

Chapter 1

INTRODUCTION

1.1 Technologies

Docker is the most widely used container virtualization system [1] and provides several very useful tools for development, maintenance and deployment of cloud software applications. There are two types of resources in docker: containers and images. Images are modified versions of software distributions that can be run through the docker virtualization engine in a platform-independent way, without the overhead of running a full operating system instance on VM or bare hardware in a (virtual) cluster. This efficient approach allows a regular developer machine to run multiple software containers at once and act as a single- or multi-node virtual cluster. It also makes things easier to reuse in any place for its versioning of images and availability on the docker hub. Kubernetes helps us to scale the system when we deploy something using it for its containerization and orchestration system. Our project report has described one of the applications using docker and Kubernetes implementation on the AWS platform.

1.2 Motivation

We wanted to build a system that can make an easy way for users for delivering content in the shortest possible time based on location. To build a system that can point to nearby servers and can give the best performance to the requested user. There are always some issues with downloading speed and uploading speed alongside the bandwidth limit. It also causes delays to stream something. To find out the solution for this situation pointing out to nearby servers is much more needed. So this system has some real-life motivation. Also for gaming servers and other related places it is also needful to connect nearby places for best performance. Relative to network limits and other connectivity issues a Content Delivery Network (CDN) can solve these criteria easily. Especially when it is on virtual machines and running its instances. So it motivated us to build something for this situation.

To be mentioned, learning about docker and Kubernetes was our priority for completing the project and later we decided to make something that has real-life implementation and can work on different perspectives and can make values.

To energize the project we have studied several cases online and wanted to learn about content delivery systems from there. We have seen it has various uses and different ways to build it. Which also gave us enough motivation to build something like that. We worked hard and built enough strength to make it successful. However, on the journey, we had faced several errors while deploying and creating clusters online and team issues which made us a little disturbed to complete project but in the end, we tried to build something really useful. In the later part of this report, it has been described how all things have been built, implemented, finalized, and deployed as well as tested on the cloud.

Chapter 2

PROBLEM DESCRIPTION

2.1 Problem Statement

From the motivation section already the idea about the project has been shared. However, our problem statement may found a little different than the motivation part as we tried to build something new after getting ideas from motivation. Including the details that have been shared in the motivation section, We want to build a Content Delivery Network (CDN) application which will be used for getting content faster according to IP address. The mechanism is like that according to relative an IP address, the user will be redirected to the nearest server location after certain times. In this context of development, it is going to be a multinode architecture.

2.2 Explanation

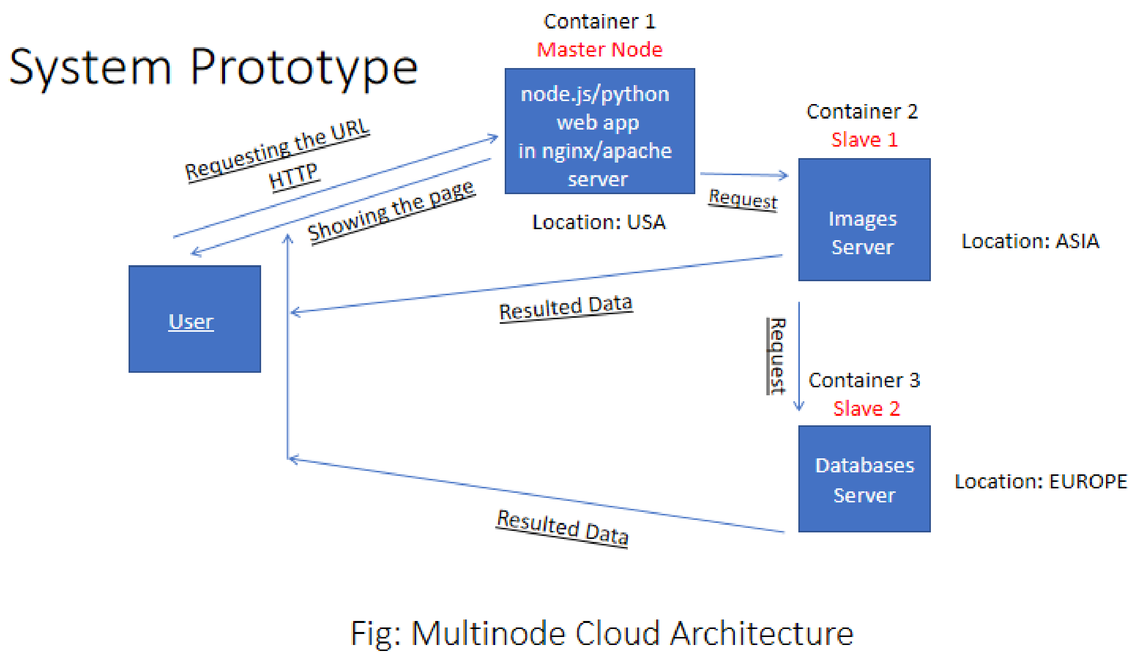


Figure 2.1: Multinode Cloud Architecture Prototype
(Prototype was our first idea but later implementation we slightly changed this)

In figure 2.1, it has been shown a multinode cloud architecture system prototype for our CDN service. There are several servers for several different kinds of contents. Container 1 is master node where nginx/apache server is installed. It could be located in the USA. Slave 1 is Container 2 where the proposed location may be in Asia. This server purpose is to serve images. Where as Container 3 is database server which location may be in Europe. It works as Slave 2. This kind of multi-node cloud architecture are our requirements to build this CDN system. It was our first idea about implementation but later in real-life implementation, these requirements had slightly changed based on the situation.

Content delivery network serves users request simultaneously for different purposes. Content can be different types. Content can be delivered from a server or can divide to different clusters/node around the world. Users with their browser request using URL.

This master node has installed Apache server which hosts a webpage. It detects some of the details of the user. This detection is based on user IP address. After that, the user will be redirected. After this redirection, it will go to a server nearby which contains image, database or mixed contents.

The resulted output sent from the server will be displayed to the user at once. For all of these reasons this system is multi-node and working as a content delivery network for the user.

2.3 Why we need it

There are several reasons to build a content delivery network. The first reason can be said it will help the user to find their content as little time as possible. Not only that sometimes when several users try to reach the same website it can find difficult because of the bandwidth limit. In that case, the system will serve the user and redirect them to the nearest available server. As well as for the growing number of requests it works as a load balancer that can adjust the volume of requests according to the traffic. For all these kinds of purposes, it is much needful to use CDN.

2.4 When not to use it

This system is complex to build and deploy. When the system has a smaller number of users then it has a smaller number of requests. So in that case it is not a good idea to use a content delivery network. To be mentioned, to give the user much flexibility and faster speed we need CDN. If these things make these criteria complex then the CDN system is not useful. In that case, a simple content delivery system could be used in single-node architecture.

2.5 Real life problems case study

Especially in some countries, when the university admission test result has been published online. There is only one website available for getting the result. In that case, when the user is too much then the server got lagging, slow or sometimes gets shut down. As the system cannot distribute the users properly at the same time it gets close down. In this kind of situation, when the contents are distributed in different nodes, it will make the thing available to the requested users easily when anyone request.

2.6 Popular CDN networks

There are several examples of CDN in our regular life which we use frequently. Some popular ones name are mentioned here for an example: Amazon cloudfront [2], CloudFlare [3], Akamai technologies [4] etc. These are world's largest distributed computing platforms according to [4]. Most of them offers several common services some of them are: Firewall, Latency distribution, DDoS attacks protection, Load balancing, Faster content delivery which help to speeding up websites etc.

Chapter 3

IMPLEMENTATION OF PROJECT

3.1 Software and Hardware Requirement

The first thing we did when we wanted to build our system is we have analyzed our feasibility study. We have seen we need to learn Docker and Kubernetes from scratch to make it a running system. We learned it and then tested the commands to make a workable thing. We had planned to deploy our systems's image into the cloud which we have made into our local machine using the docker.

The local machine needs HyperX enabled in BIOS settings and needful of atleast 8gb RAM to run docker. Some updated configuration of computer give support to run it smoothly. Docker is a platform for developing, shipping and running applications easily. A software can deploy in the dockerized image format and later it can deploy in any platform. So it is platform independent and easier to run [5]. After that we have studied about Kubernetes. Which is very popular for orchestration, topology and scaling [6].

We had also researched geographic redirection for IP addresses as long as it can be done with PHP. Alongside that, we had also needed to learn about AWS and its features and how to deploy and set up things there. Even deploying to AWS was a hard part of the project for some credit card related billing issues while activating AWS account.

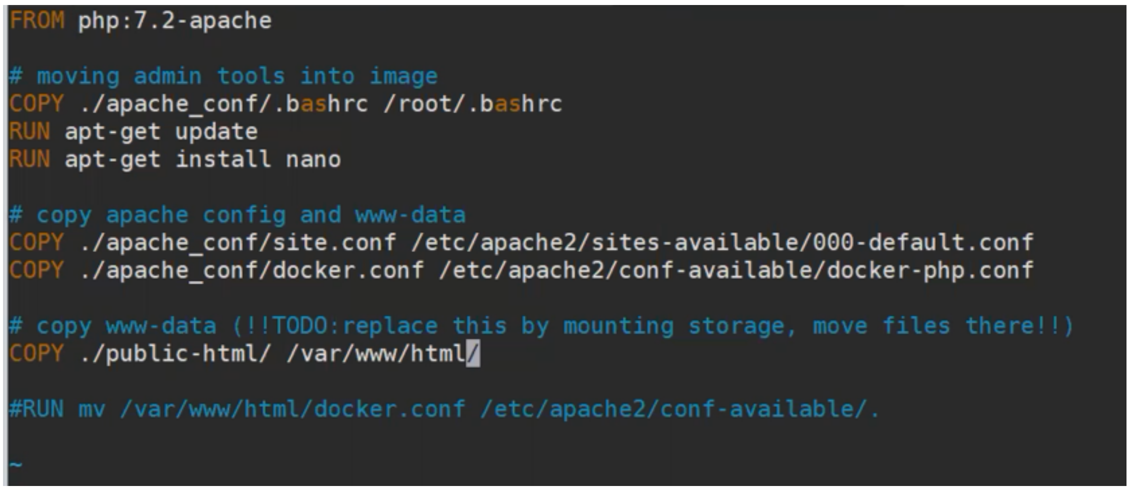
3.2 Description of Implementation

Firstly we have built the docker images of an Apache webserver which we have later put into docker hub. From that hub, we always cloned it into our local machine. Then we have tried to deploy it into AWS so that we can make it for public usage. We had to face lots of trouble to deploy in the AWS for its login error. Here in the below points, we are showing the steps of the process for creating and running the docker container and uploading in the docker hub, and then pulling that from the docker hub into the desktop.

3.2.1 Building the container:

Here we have been described how we have built our docker container. We have used docker desktop version and WSL linux for our docker container making. In our docker container there will be image for Apache webserver in ubuntu platform. We have also used another SSH client name MobaXterm to write all the commands inside.

There are two ways to build that container. We have used the most popular way for this. In below figure 3.1 the process has been described.



```
FROM php:7.2-apache

# moving admin tools into image
COPY ./apache_conf/.bashrc /root/.bashrc
RUN apt-get update
RUN apt-get install nano

# copy apache config and www-data
COPY ./apache_conf/site.conf /etc/apache2/sites-available/000-default.conf
COPY ./apache_conf/docker.conf /etc/apache2/conf-available/docker-php.conf

# copy www-data (!!TODO:replace this by mounting storage, move files there!!)
COPY ./public-html/ /var/www/html/

#RUN mv /var/www/html/docker.conf /etc/apache2/conf-available/.

~
```

Figure 3.1: Creating docker image for Apache webserver

We are using the following codes:

```
FROM php:7.2-apache
COPY ./apache_conf/.bashrc /root/.bashrc
RUN apt-get update
RUN apt-get -y install nano rsync
COPY ./apache_conf/site.conf /etc/apache2/sites-available/000-default.conf
COPY ./apache_conf/docker.conf /etc/apache2/conf-available/docker-php.conf
COPY ./apache_conf/php.ini /usr/local/etc/php/php.ini
COPY ./public-html/ /var/www/html/
```

On the above lines, all the Apache settings, configurations, and directories for showing files have been moved to the docker image. As well as PHP, nano, and rsync tools inside that image.

This is how we create the docker images according to our needs for the project. This way is an approach to build the image container. After we dockerized the image we can use it anywhere using Docker.

3.2.2 After building the container

After creating our docker image it is now available online publicly. Each time we change something inside that we update and push it into docker hub.

Docker hub address for our project is: `https://hub.docker.com/r/entty/cloud_project`

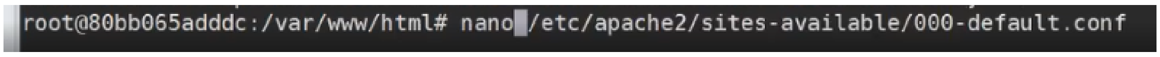
We can run docker image after starting the docker for desktop or docker virtualization in our local PC. We are running the container by following code:

```
  # docker run -dp 80:80 entty/cloud_project:latest
```

Here one thing to be mentioned, we are running all the codes of this project in the root command prompt which mentioned with #. All of these codes are also possible to run with a user command prompt which usually starts with **$**.

After running docker we can go inside the docker container by writing some of the codes. Then we can use this just like our local machine. Docker desktop automatically provides some name for the running image. In our case after running the docker image as container, docker is giving us different names. While we wrote this report, docker had gave us the name *busy_chandrasekhar* for the running container. So the code for entering inside container to edit:

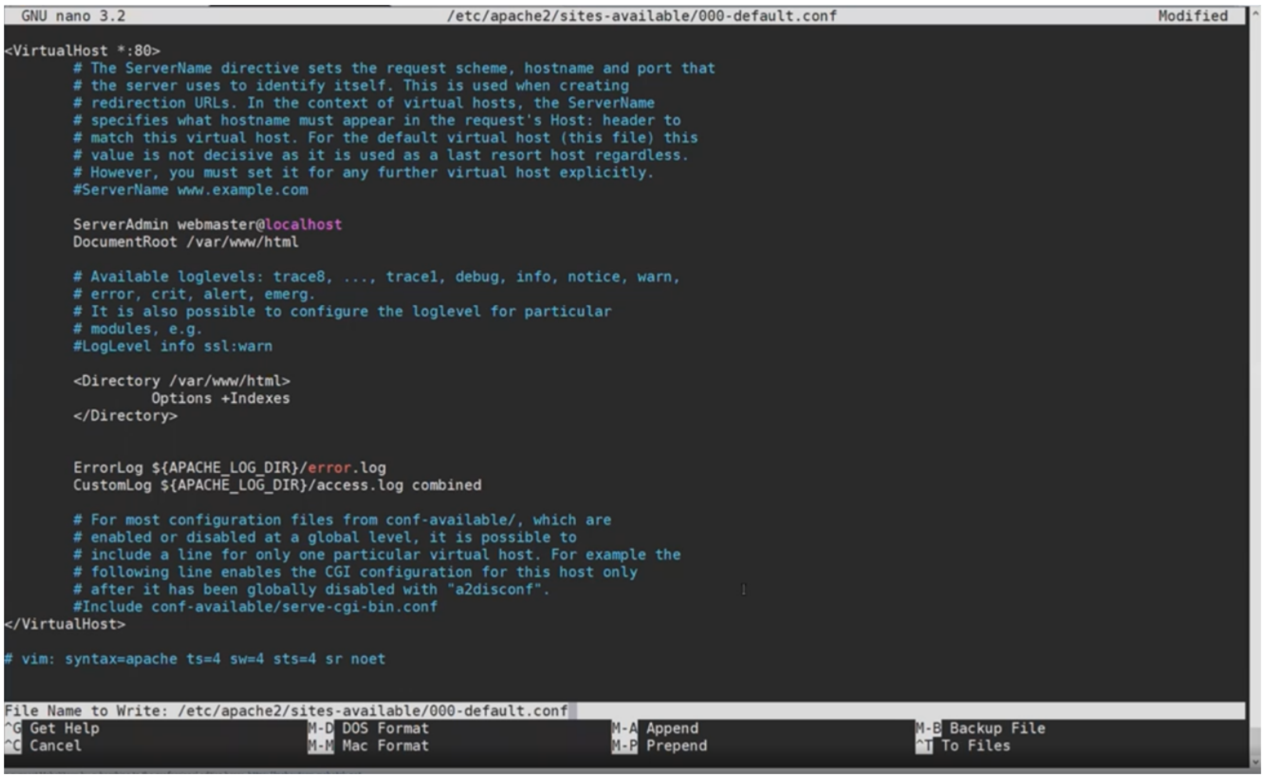
```
# docker exec -ti busy_chandrasekhar /bin/bash
```



```
root@80bb065adddc:/var/www/html# nano /etc/apache2/sites-available/000-default.conf
```

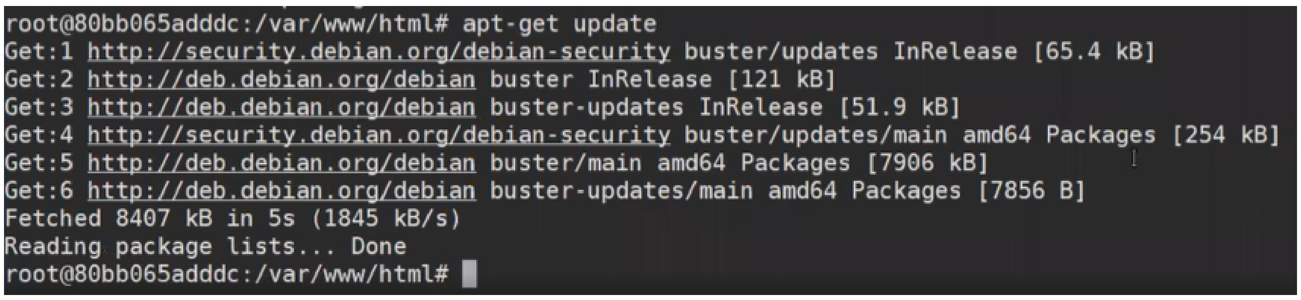
Figure 3.2: Calling nano editor inside ubuntu dockerized image to setup Apache config file

In figure 3.3, it is showing different settings of the Apache server while inside 000-default.conf file using nano editor.



```
  GNU nano 3.2                    /etc/apache2/sites-available/000-default.conf                    Modified

<VirtualHost *:80>
        # The ServerName directive sets the request scheme, hostname and port that
        # the server uses to identify itself. This is used when creating
        # redirection URLs. In the context of virtual hosts, the ServerName
        # specifies what hostname must appear in the request's Host: header to
        # match this virtual host. For the default virtual host (this file) this
        # value is not decisive as it is used as a last resort host regardless.
        # However, you must set it for any further virtual host explicitly.
        #ServerName www.example.com

        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html

        # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
        # error, crit, alert, emerg.
        # It is also possible to configure the loglevel for particular
        # modules, e.g.
        #LogLevel info ssl:warn

        <Directory /var/www/html>
                Options +Indexes
        </Directory>


        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        # For most configuration files from conf-available/, which are
        # enabled or disabled at a global level, it is possible to
        # include a line for only one particular virtual host. For example the
        # following line enables the CGI configuration for this host only
        # after it has been globally disabled with "a2disconf".
        #Include conf-available/serve-cgi-bin.conf
</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet

File Name to Write: /etc/apache2/sites-available/000-default.conf
^G Get Help                M-D DOS Format             M-A Append                 M-B Backup File
^C Cancel                  M-M Mac Format             M-P Prepend                ^T To Files
```

Figure 3.3: Changing settings inside the Apache server



```
root@80bb065adddc:/var/www/html# apt-get update
Get:1 http://security.debian.org/debian-security buster/updates InRelease [65.4 kB]
Get:2 http://deb.debian.org/debian buster InRelease [121 kB]
Get:3 http://deb.debian.org/debian buster-updates InRelease [51.9 kB]
Get:4 http://security.debian.org/debian-security buster/updates/main amd64 Packages [254 kB]
Get:5 http://deb.debian.org/debian buster/main amd64 Packages [7906 kB]
Get:6 http://deb.debian.org/debian buster-updates/main amd64 Packages [7856 B]
Fetched 8407 kB in 5s (1845 kB/s)
Reading package lists... Done
root@80bb065adddc:/var/www/html#
```

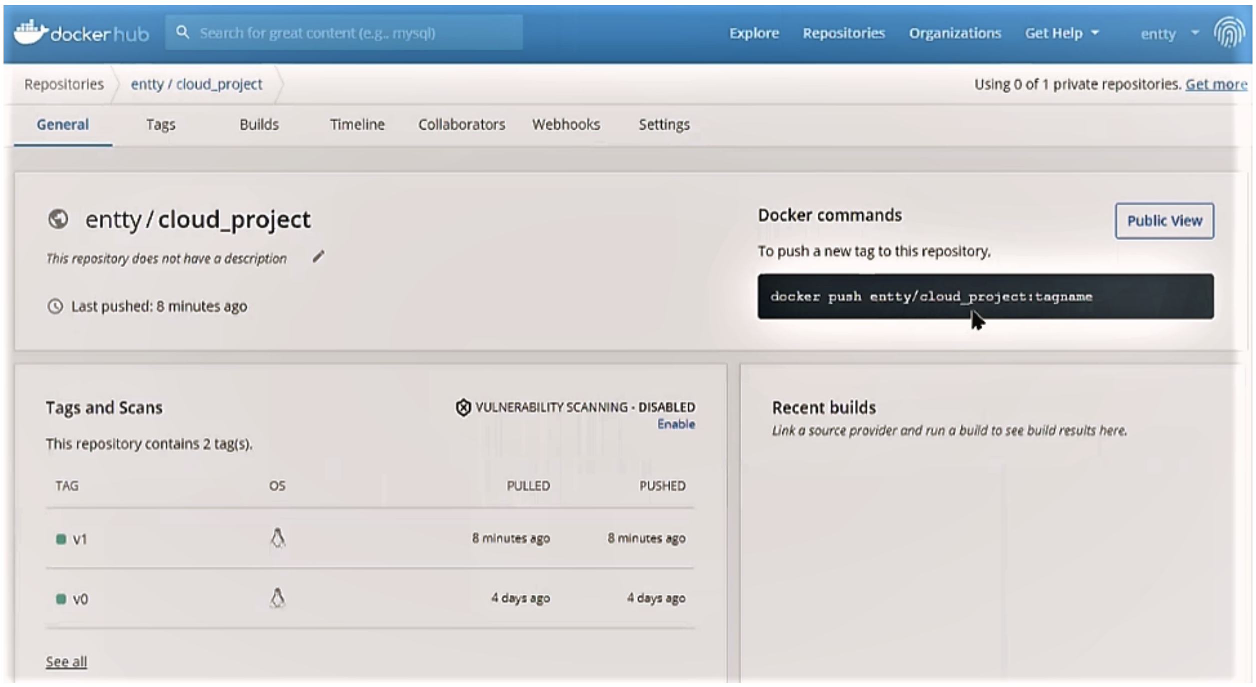
Figure 3.4: Inside the server we are updating

Here it shows in figure 3.4 that we are updating the Ubuntu. After saving the changes and updates we are restarting the container in virtual machine:

```
# docker restart entty/cloud_project:v1
```

Here in above restart code, we have mentioned the container name as entty/cloud_project and it's version number as v1. Then we are pushing that image to dockerhub using below code:

```
# docker push entty/cloud_project:v1
```



entty/cloud_project

This repository does not have a description

Last pushed: 8 minutes ago

Docker commands

To push a new tag to this repository,

```
docker push entty/cloud_project:tagname
```

Tags and Scans

This repository contains 2 tag(s).

TAG	OS	PULLED	PUSHED
v1		8 minutes ago	8 minutes ago
v0		4 days ago	4 days ago

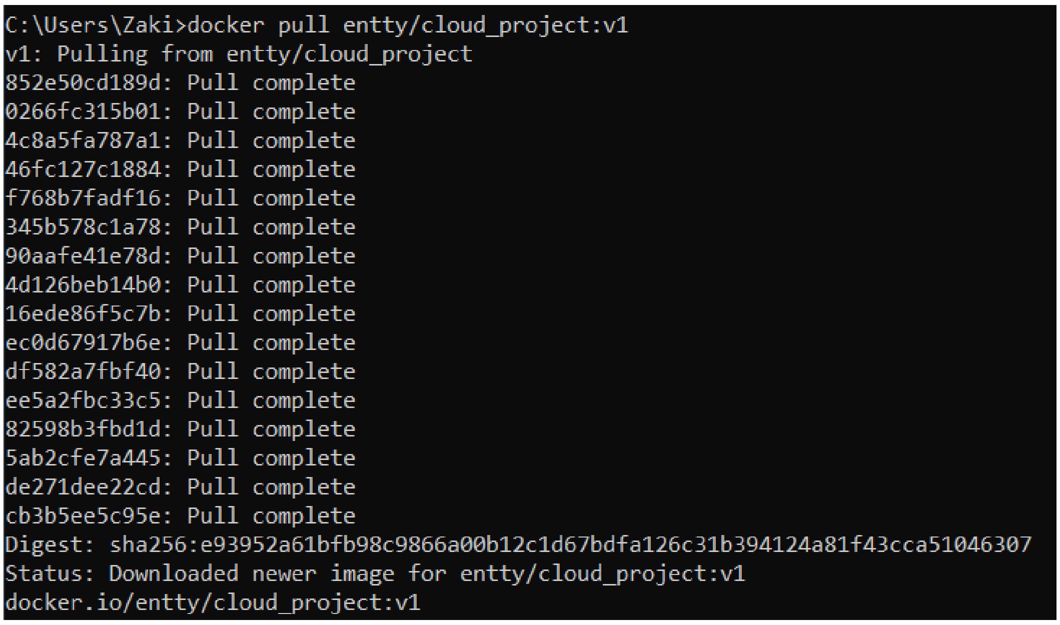
Recent builds

Link a source provider and run a build to see build results here.

Figure 3.5: Our created image after pushing is now available in docker hub

Pulling the docker image using below code from docker hub to local machine for customizing again the Apache configuration :

```
# docker pull entty/cloud_project:v1
```



```
C:\Users\Zaki>docker pull entty/cloud_project:v1
v1: Pulling from entty/cloud_project
852e50cd189d: Pull complete
0266fc315b01: Pull complete
4c8a5fa787a1: Pull complete
46fc127c1884: Pull complete
f768b7fadf16: Pull complete
345b578c1a78: Pull complete
90aafe41e78d: Pull complete
4d126beb14b0: Pull complete
16ede86f5c7b: Pull complete
ec0d67917b6e: Pull complete
df582a7fbf40: Pull complete
ee5a2fbc33c5: Pull complete
82598b3fbd1d: Pull complete
5ab2cfe7a445: Pull complete
de271dee22cd: Pull complete
cb3b5ee5c95e: Pull complete
Digest: sha256:e93952a61bfb98c9866a00b12c1d67bdfa126c31b394124a81f43cca51046307
Status: Downloaded newer image for entty/cloud_project:v1
docker.io/entty/cloud_project:v1
```

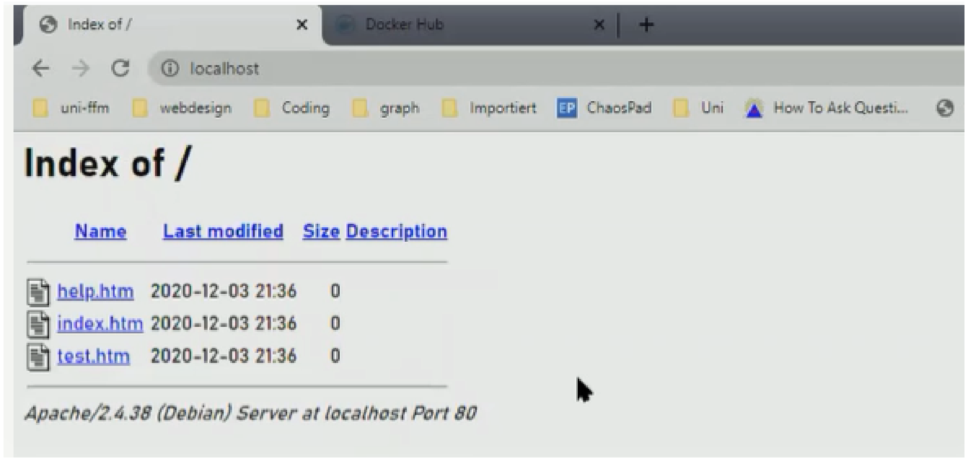
Figure 3.6: Pulling from docker hub

Now it is time to run docker container from pulled docker image using this command:

```
# docker run -dp 80:80 entty/cloud_project:v1
```

Here we have defined port 80 for this container and -dp means local container

So going to this link: http://localhost/ is perfectly working



Index of /

```
Name        Last modified     Size Description
help.htm    2020-12-03 21:36    0
index.htm   2020-12-03 21:36    0
test.htm    2020-12-03 21:36    0
Apache/2.4.38 (Debian) Server at localhost Port 80
```

Figure 3.7: Accessed to the dockerized Apache server and showing root directory

3.3 Deploying on AWS

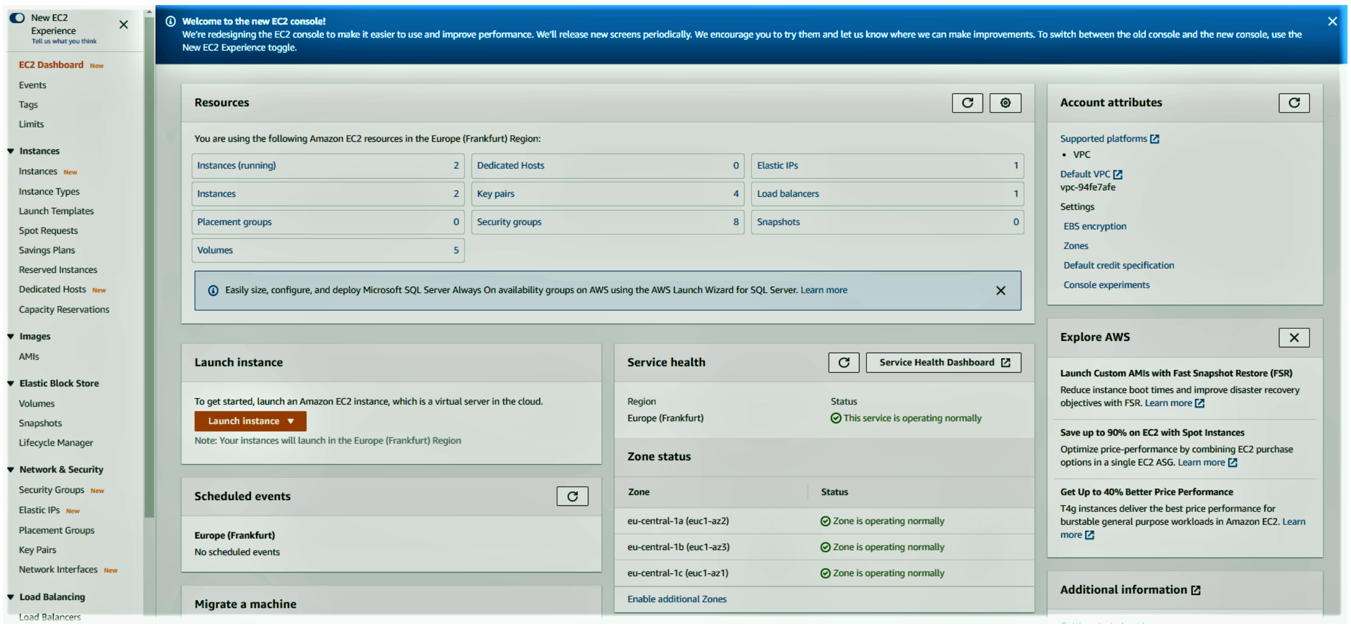
3.3.1 Why AWS?

AWS is a great platform for hosting cloud-related products, though there are many other companies available online, we have chosen AWS because of its greater extend of support and flexibility. Though it has high-cost issues, we have chosen this for deployment. In the below steps, it has been described several technologies how we have implemented inside the AWS cloud platform.

We have chosen Frankfurt as our location for europe on AWS. We also tried on other regions but it was best to check from our nearby location.

3.3.2 Docker

Here is shown our docker deployment in cloud. For deploying to AWS we have used our customized docker build which is available in our gthub `https://github.com/enttty/cdn_project/tree/master/docker_build` repository. For using this we had to create the environment inside AWS EC2. Amazon EC2 (Elastic Compute Cloud) provides a scalable computing environment where the need for hardware investment is not necessary. The cloud environment handles everything [7].



Welcome to the new EC2 console!
We're redesigning the EC2 console to make it easier to use and improve performance. We'll release new screens periodically. We encourage you to try them and let us know where we can make improvements. To switch between the old console and the new console, use the New EC2 Experience toggle.

Resources

You are using the following Amazon EC2 resources in the Europe (Frankfurt) Region:

Instances (running)	2	Dedicated Hosts	0	Elastic IPs	1
Instances	2	Key pairs	4	Load balancers	1
Placement groups	0	Security groups	8	Snapshots	0
Volumes	5				

Easily size, configure, and deploy Microsoft SQL Server Always On availability groups on AWS using the AWS Launch Wizard for SQL Server. Learn more

Account attributes

Supported platforms
- VPC

Default VPC
vpc-94fe7afe

Settings
EBS encryption
Zones
Default credit specification
Console experiments

Launch instance

To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.

Launch instance

Note: Your instances will launch in the Europe (Frankfurt) Region

Service health

Service Health Dashboard

Region	Status
Europe (Frankfurt)	This service is operating normally

Zone status

Zone	Status
eu-central-1a (euc1-az2)	Zone is operating normally
eu-central-1b (euc1-az3)	Zone is operating normally
eu-central-1c (euc1-az1)	Zone is operating normally

Enable additional Zones

Scheduled events

Europe (Frankfurt)
No scheduled events

Migrate a machine

Explore AWS

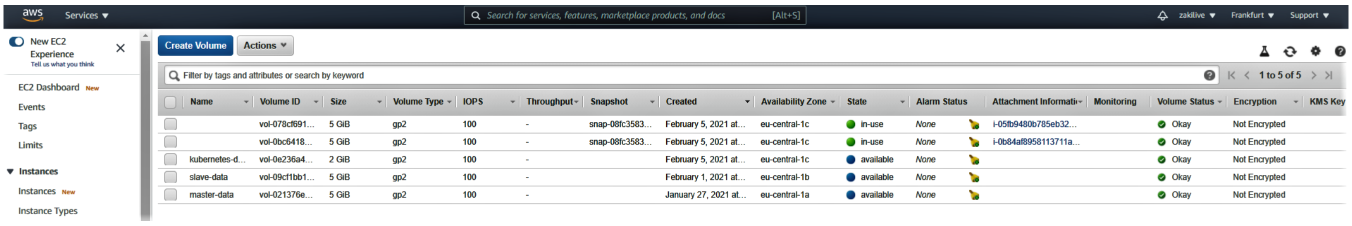
Launch Custom AMIs with Fast Snapshot Restore (FSR)
Reduce instance boot times and improve disaster recovery objectives with FSR. Learn more

Save up to 90% on EC2 with Spot Instances
Optimize price-performance by combining EC2 purchase options in a single EC2 ASG. Learn more

Get Up to 40% Better Price Performance
T4g instances deliver the best price performance for burstable general purpose workloads in Amazon EC2. Learn more

Additional information

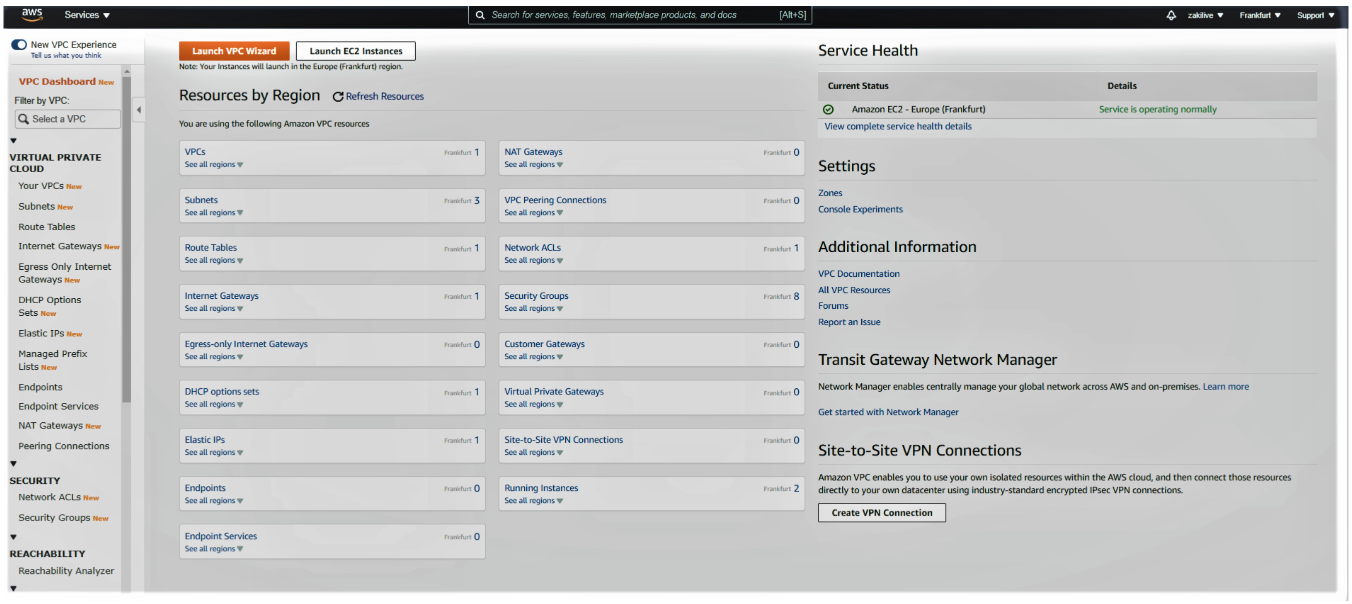
Figure 3.8: Inside the AWS EC2 control panel



Name	Volume ID	Size	Volume Type	IOPS	Throughput	Snapshot	Created	Availability Zone	State	Alarm Status	Attachment Information	Monitoring	Volume Status	Encryption	KMS Key
	vol-078cf691...	5 GiB	gp2	100	-	snap-08fc3583...	February 5, 2021 at...	eu-central-1c	in-use	None	i-05fb9480b785eb32...		Okay	Not Encrypted	
	vol-0bc6418...	5 GiB	gp2	100	-	snap-08fc3583...	February 5, 2021 at...	eu-central-1c	in-use	None	i-0b84af8958113711a...		Okay	Not Encrypted	
kubernetes-d...	vol-0e236a4...	2 GiB	gp2	100	-		February 5, 2021 at...	eu-central-1c	available	None			Okay	Not Encrypted	
slave-data	vol-09cf1bb1...	5 GiB	gp2	100	-		February 1, 2021 at...	eu-central-1b	available	None			Okay	Not Encrypted	
master-data	vol-021376e...	5 GiB	gp2	100	-		January 27, 2021 at...	eu-central-1a	available	None			Okay	Not Encrypted	

Figure 3.9: Inside the AWS EC2 control panel for EBS data storage

In figure 3.9 it has been shown the data storage which is a service from AWS named EBS(Elastic Block Service) [8]. EBS uses a block storage format that allows scaling automatically for different purposes for example containerized applications, big data analytics engines etc.



Note: Your Instances will launch in the Europe (Frankfurt) region.

Resources by Region
Refresh Resources

You are using the following Amazon VPC resources

VPCs	Frankfurt 1	See all regions	NAT Gateways	Frankfurt 0	See all regions
Subnets	Frankfurt 3	See all regions	VPC Peering Connections	Frankfurt 0	See all regions
Route Tables	Frankfurt 1	See all regions	Network ACLs	Frankfurt 1	See all regions
Internet Gateways	Frankfurt 1	See all regions	Security Groups	Frankfurt 8	See all regions
Egress-only Internet Gateways	Frankfurt 0	See all regions	Customer Gateways	Frankfurt 0	See all regions
DHCP options sets	Frankfurt 1	See all regions	Virtual Private Gateways	Frankfurt 0	See all regions
Elastic IPs	Frankfurt 1	See all regions	Site-to-Site VPN Connections	Frankfurt 0	See all regions
Endpoints	Frankfurt 0	See all regions	Running Instances	Frankfurt 2	See all regions
Endpoint Services	Frankfurt 0	See all regions			

Service Health

Current Status	Details
Amazon EC2 - Europe (Frankfurt)	Service is operating normally

View complete service health details

Settings

Zones
Console Experiments

Additional Information

VPC Documentation
All VPC Resources
Forums
Report an Issue

Transit Gateway Network Manager

Network Manager enables centrally manage your global network across AWS and on-premises. Learn more

Get started with Network Manager

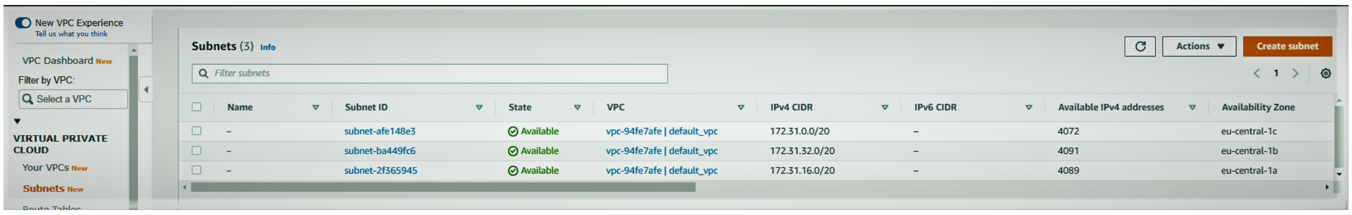
Site-to-Site VPN Connections

Amazon VPC enables you to use your own isolated resources within the AWS cloud, and then connect those resources directly to your own datacenter using industry-standard encrypted IPsec VPN connections.

Create VPN Connection

Figure 3.10: Inside the AWS control panel during VPC creation

VPC is another service of amazon AWS where it stands for virtual private cloud. In VPC it supports different virtual network services. [9]

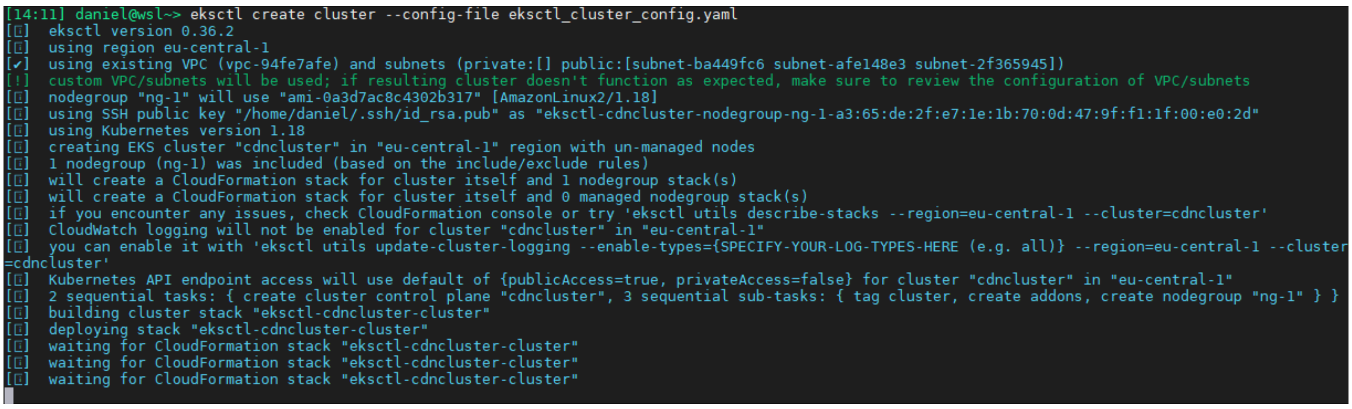


Name	Subnet ID	State	VPC	IPv4 CIDR	IPv6 CIDR	Available IPv4 addresses	Availability Zone
–	subnet-afe148e3	✓ Available	vpc-94fe7afe | default_vpc	172.31.0.0/20	–	4072	eu-central-1c
–	subnet-ba449fc6	✓ Available	vpc-94fe7afe | default_vpc	172.31.32.0/20	–	4091	eu-central-1b
–	subnet-2f365945	✓ Available	vpc-94fe7afe | default_vpc	172.31.16.0/20	–	4089	eu-central-1a

Figure 3.11: Three different VPC are running our instances

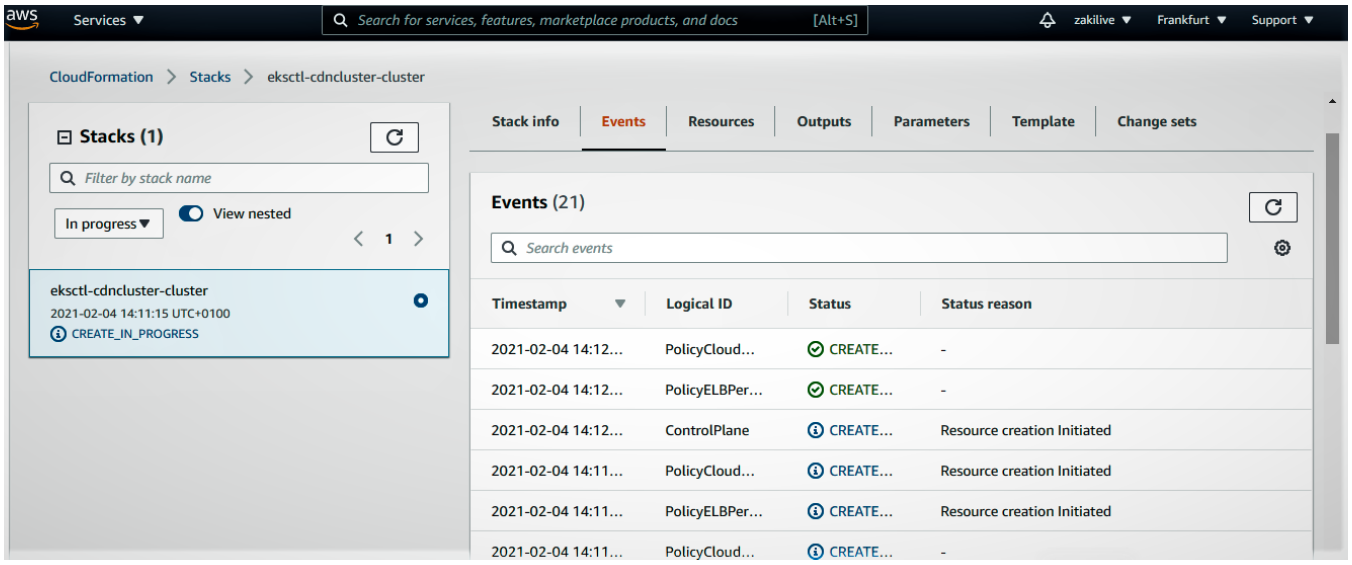
3.3.3 Kubernetes

It takes 15-20mins to build this EKS (Amazon Elastic Kubernetes Service)



```
[14:11] daniel@wsl~> eksctl create cluster --config-file eksctl_cluster_config.yaml
[ℹ]  eksctl version 0.36.2
[ℹ]  using region eu-central-1
[✔]  using existing VPC (vpc-94fe7afe) and subnets (private:[] public:[subnet-ba449fc6 subnet-afe148e3 subnet-2f365945])
[!]  custom VPC/subnets will be used; if resulting cluster doesn't function as expected, make sure to review the configuration of VPC/subnets
[ℹ]  nodegroup "ng-1" will use "ami-0a3d7ac8c4302b317" [AmazonLinux2/1.18]
[ℹ]  using SSH public key "/home/daniel/.ssh/id_rsa.pub" as "eksctl-cdncluster-nodegroup-ng-1-a3:65:de:2f:e7:1e:1b:70:0d:47:9f:f1:1f:00:e0:2d"
[ℹ]  using Kubernetes version 1.18
[ℹ]  creating EKS cluster "cdncluster" in "eu-central-1" region with un-managed nodes
[ℹ]  1 nodegroup (ng-1) was included (based on the include/exclude rules)
[ℹ]  will create a CloudFormation stack for cluster itself and 1 nodegroup stack(s)
[ℹ]  will create a CloudFormation stack for cluster itself and 0 managed nodegroup stack(s)
[ℹ]  if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=eu-central-1 --cluster=cdncluster'
[ℹ]  CloudWatch logging will not be enabled for cluster "cdncluster" in "eu-central-1"
[ℹ]  you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=eu-central-1 --cluster
=cdncluster'
[ℹ]  Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "cdncluster" in "eu-central-1"
[ℹ]  2 sequential tasks: { create cluster control plane "cdncluster", 3 sequential sub-tasks: { tag cluster, create addons, create nodegroup "ng-1" } }
[ℹ]  building cluster stack "eksctl-cdncluster-cluster"
[ℹ]  deploying stack "eksctl-cdncluster-cluster"
[ℹ]  waiting for CloudFormation stack "eksctl-cdncluster-cluster"
[ℹ]  waiting for CloudFormation stack "eksctl-cdncluster-cluster"
[ℹ]  waiting for CloudFormation stack "eksctl-cdncluster-cluster"
```

Figure 3.12: Using VPC while creating Kubernetes cluster on AWS



eksctl-cdncluster-cluster
2021-02-04 14:11:15 UTC+0100
CREATE_IN_PROGRESS

Timestamp	Logical ID	Status	Status reason
2021-02-04 14:12...	PolicyCloud...	✓ CREATE...	-
2021-02-04 14:12...	PolicyELBPer...	✓ CREATE...	-
2021-02-04 14:12...	ControlPlane	ⓘ CREATE...	Resource creation Initiated
2021-02-04 14:11...	PolicyCloud...	ⓘ CREATE...	Resource creation Initiated
2021-02-04 14:11...	PolicyELBPer...	ⓘ CREATE...	Resource creation Initiated
2021-02-04 14:11...	PolicyCloud...	ⓘ CREATE...	-

Figure 3.13: Inside the AWS control panel during EKS creation

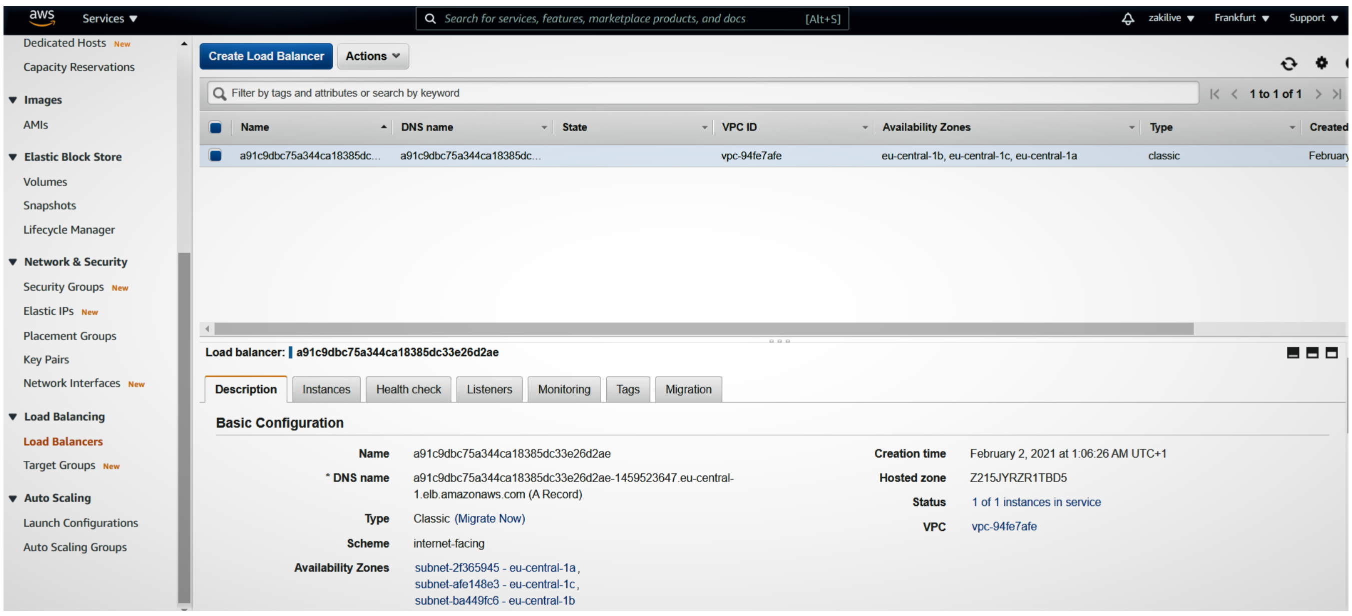
To create EKS we are using two nodes and the file format as yaml file. The yaml code for 2 nodes is here:

```
  apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
```

```
metadata:
  name: cdn-test-cluster
  region: eu-central-1
  tags:
   'cluster': 'cdn_cluster'
vpc:
  id: "vpc-94fe7afe"
  subnets:
    public:
      eu-central-1a:
          id: "subnet-2f365945"
      eu-central-1b:
          id: "subnet-ba449fc6"
      eu-central-1c:
          id: "subnet-afe148e3"
nodeGroups:
  - name: ng-1
    instanceType: t2.medium
    desiredCapacity: 1
    volumeSize: 5
    ssh:
     allow: true
    tags:
      'cluster': 'cdn_cluster'
  - name: ng-2
    instanceType: t2.large
    desiredCapacity: 3
    volumeSize: 10
    ssh:
     allow: true
    tags:
      'cluster': 'cdn_cluster'
```

3.4 Loadbalancers

Here we have used Amazon elastic load balancers service which automatically distributes incoming application traffic and make a easy transfer of the network [10].



Name	DNS name	State	VPC ID	Availability Zones	Type	Created
a91c9dbc75a344ca18385dc...	a91c9dbc75a344ca18385dc...		vpc-94fe7afe	eu-central-1b, eu-central-1c, eu-central-1a	classic	February

Load balancer: a91c9dbc75a344ca18385dc33e26d2ae

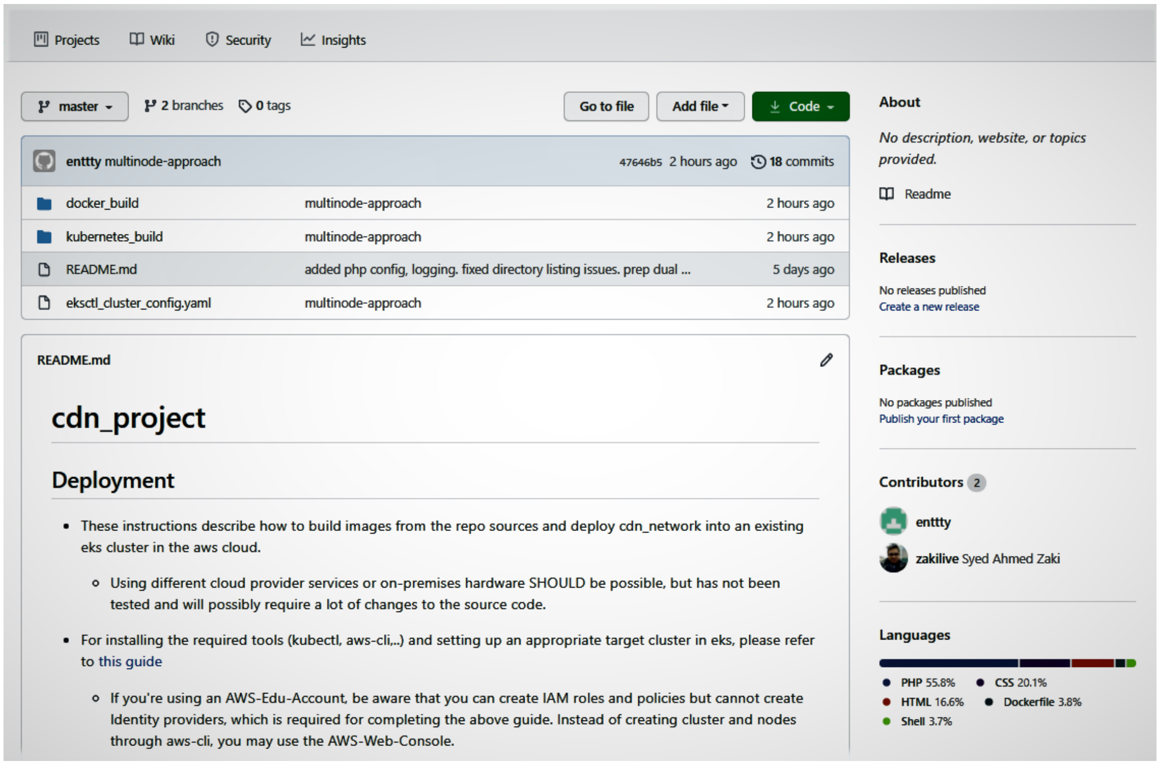
Basic Configuration

Name	a91c9dbc75a344ca18385dc33e26d2ae	Creation time	February 2, 2021 at 1:06:26 AM UTC+1
* DNS name	a91c9dbc75a344ca18385dc33e26d2ae-1459523647.eu-central-1.elb.amazonaws.com (A Record)	Hosted zone	Z215JYRZR1TBD5
Type	Classic (Migrate Now)	Status	1 of 1 instances in service
Scheme	internet-facing	VPC	vpc-94fe7afe
Availability Zones	subnet-2f365945 - eu-central-1a, subnet-afe148e3 - eu-central-1c, subnet-ba449fc6 - eu-central-1b		

Figure 3.14: Load balancers on AWS console

3.5 Teamwork on Github

We have made our project opensource and everything is available inside the GitHub repository. So if anyone wants to find the code or want to contribute they are welcome to do it. Our GitHub repository address URL: `https://github.com/enttty/cdn_project`



enttty multinode-approach 47646b5 2 hours ago 18 commits

docker_build	multinode-approach	2 hours ago
kubernetes_build	multinode-approach	2 hours ago
README.md	added php config, logging. fixed directory listing issues. prep dual ...	5 days ago
eksctl_cluster_config.yaml	multinode-approach	2 hours ago

README.md

cdn_project

Deployment

- These instructions describe how to build images from the repo sources and deploy cdn_network into an existing eks cluster in the aws cloud.
  - Using different cloud provider services or on-premises hardware SHOULD be possible, but has not been tested and will possibly require a lot of changes to the source code.
- For installing the required tools (kubectl, aws-cli,..) and setting up an appropriate target cluster in eks, please refer to this guide
  - If you're using an AWS-Edu-Account, be aware that you can create IAM roles and policies but cannot create Identity providers, which is required for completing the above guide. Instead of creating cluster and nodes through aws-cli, you may use the AWS-Web-Console.

No description, website, or topics provided.

No releases published
Create a new release

No packages published
Publish your first package

enttty
zakilive Syed Ahmed Zaki

● PHP 55.8% ● CSS 20.1% ● HTML 16.6% ● Dockerfile 3.8% ● Shell 3.7%

Figure 3.15: Our GitHub project

When we needed to do any changes in our project we have committed the updated file in the repository then made the pull request. We decided later which things we could merge on the active project or not. Also, we have taken the files from the GitHub repository to our AWS cluster when needful. It also made our work faster.

3.5.1 Some troubles we had faced while doing this project

After creating a cluster every time we have got different DNS names. We tried to make it constant which would be helpful for one portion of our project but it did not work out. We also tried to do that same task with another Amazon AWS service named "route53". It was a pointing service that can redirect to a single domain name with newly generated DNS [11], but this service also did not work.

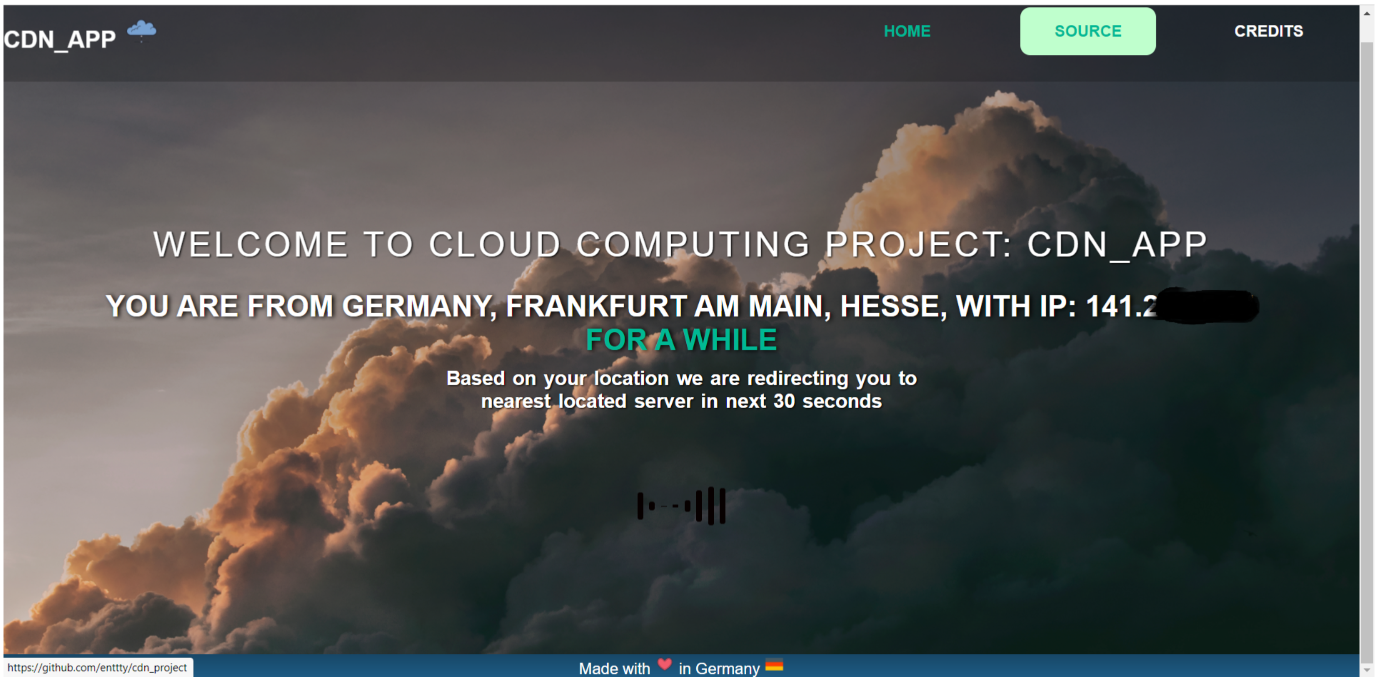
Also in the beginning of the project, for credit card issues we could not make our AWS account activated, we tried AWS educate account. The problem with AWS educate account was it did not give actual access to us. It was frequently disconnecting and was giving error and timed out while we tried to log in using the command-line interface (CLI) to deploy our services.

Chapter 4

FINALIZATION OF PROJECT

4.1 Screen shots of project

Below the current screenshots of our project have been given. Figure 4.1 is the homepage for showing the location of the user. This is the current page for showing this info. Then it will also work as before the redirection page.



WELCOME TO CLOUD COMPUTING PROJECT: CDN_APP

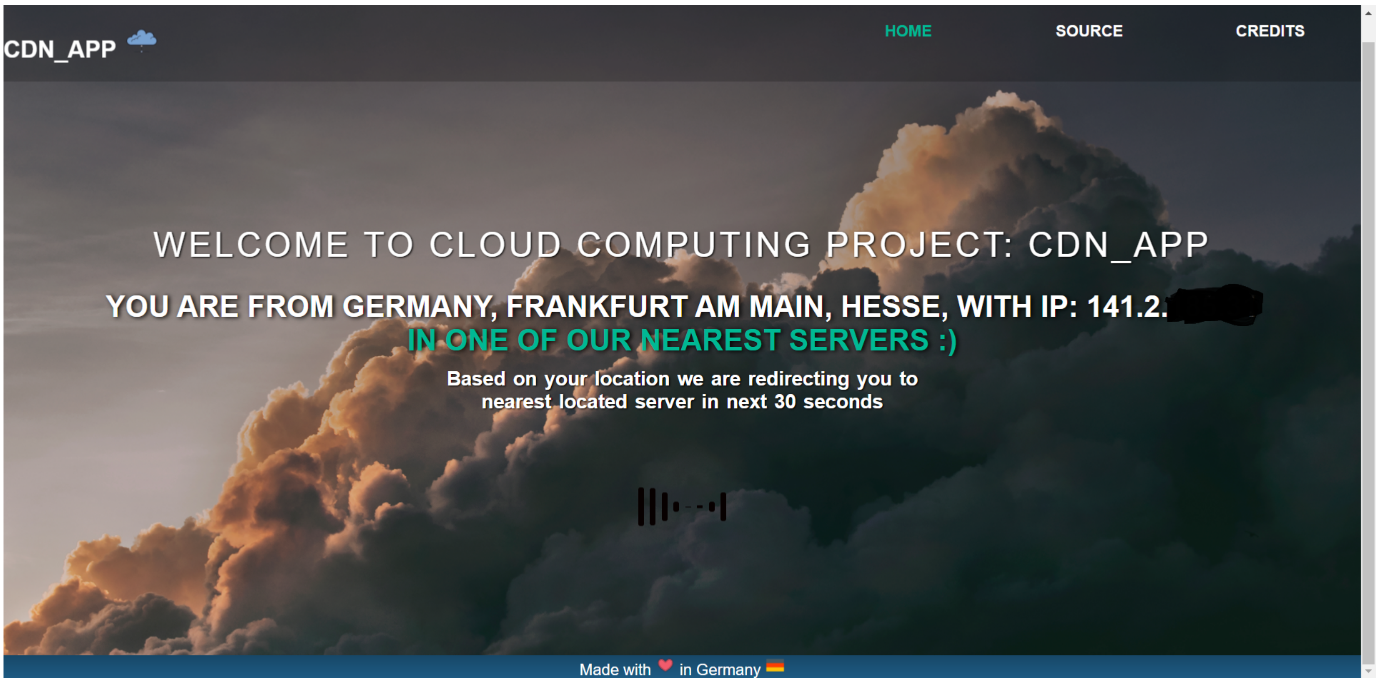
YOU ARE FROM GERMANY, FRANKFURT AM MAIN, HESSE, WITH IP: 141.2[redacted]

FOR A WHILE

Based on your location we are redirecting you to nearest located server in next 30 seconds

Figure 4.1: The homepage of IP redirection. (IP has blurred for privacy reason)

In figure 4.2, the IP address detection is taking a time of 30 seconds and there is loading animation going on while detecting the location information.


CDN_APP

HOME SOURCE CREDITS

WELCOME TO CLOUD COMPUTING PROJECT: CDN_APP

YOU ARE FROM GERMANY, FRANKFURT AM MAIN, HESSE, WITH IP: 141.2.[redacted]
IN ONE OF OUR NEAREST SERVERS :)

Based on your location we are redirecting you to
nearest located server in next 30 seconds

Made with ❤ in Germany

Figure 4.2: The system is waiting for redirection and collecting information from IP address of user. (IP has blurred for privacy reason)

4.2 Modules description

After some test run our project is running perfectly in the cloud. So it has been finalized our works.

In our CDN_App, The homepage wave animation CSS available in the styleload.css file, The bouncing logo, and the text animation CSS is available in the index.php file and it is using animate.min.css file from an external source.

We are using IP API in JSON format to find location info in javascript then showing that using PHP as well as plain HTML, based on this IP location we are redirecting the user to our nearest server/node.

We tried PHP at the beginning for redirecting our user but it was some issues for rendering in the cloud server after deployment, so we converted the PHP code to javascript which takes information from user browser and doesn't need to rely on the server end. All the images that have been used in our project are referenced from their source links.

4.3 Some issues

After deploying our docker image on AWS or when we tried the IP address of the docker container on the local machine using browser, we were facing some troubles when the system was trying to get the user IP address. Before deployment it was completely okay with XAMPP but running the docker container could not detect the IP address of the user. We analyzed the problem and later we have found that it was acting like it has no IP address of the user while it is dockerized.

After some research, we have found a solution. We have found using a docker image of ngrok and pointing that ngrok to our dockerized container IP address will solve that issue. Ngrok is a tool available in all platforms to expose a local local development server to the internet with some efforts [12]. We are using ngrok version made by wernight that is available in docker hub. It is most popular. Here some details has

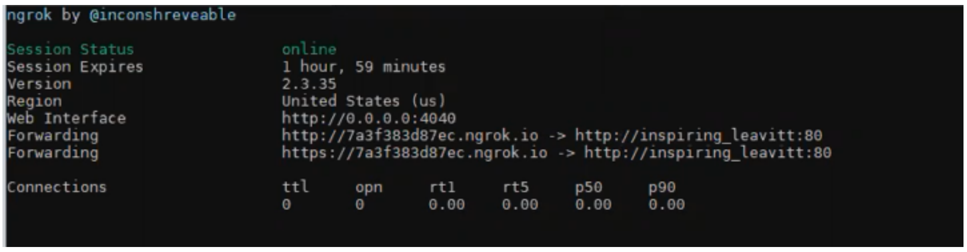
been shared below.
Code for running ngrok :

```
# docker pull wernight/ngrok
```

Command for pointing our running docker container with ngrok:

```
# docker run --rm -it --link inspiring_leavitt wernight/ngrok ngrok http
inspiring_leaveitt:80
```

The screenshots of running Ngrok has given below:



```
ngrok by @inconshreveable

Session Status                online
Session Expires               1 hour, 59 minutes
Version                       2.3.35
Region                        United States (us)
Web Interface                 http://0.0.0.0:4040
Forwarding                    http://7a3f383d87ec.ngrok.io -> http://inspiring_leavitt:80
Forwarding                    https://7a3f383d87ec.ngrok.io -> http://inspiring_leavitt:80

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

Figure 4.3: Ngrok is running in our system

To be mentioned, this figure 4.3 has been taken from our local machine when there was Docker container running. So we can follow the ngrok created forward link from outside and can browse inside the Apache server that is running inside the container.

4.4 Our current project structure

Our updated finalized project architecture model diagram has been sketched here in figure 4.4. In figure 2.1, it was a little complex design for our multi-node cloud architecture. It was actually prototype design. Later, while we worked to make it as real, we felt we need to do it as simple as possible for costing and deployment issues also for the easy experience for our user.

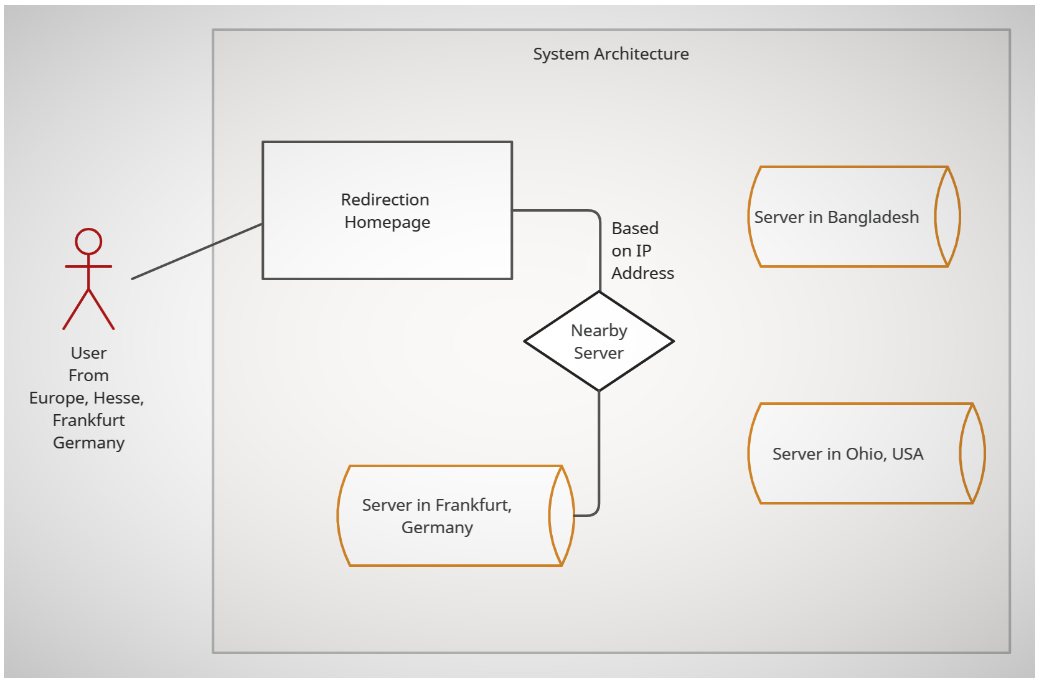


Figure 4.4: Updated Multinode Cloud Architecture Diagram

From this figure 4.4, a user from Frankfurt, Hesse, Germany in Europe requesting a webpage. It is showing the redirection homepage to the user. IP address of the use has been taken behind and based on that address the system is detecting the nearby server. Our other servers planned locations are in Bangladesh, Asia and another one is in Ohio, USA. The system is redirecting the user to the Frankfurt server where a directory listing system has been installed as a file directory.

Chapter 5

Future Goals

We have a plan to continue this project and in future versions, we can release those updates. Some of the future goals have been shared here.

We have been working on the upload button inside one of our directory of files. So when users log in they can upload their files and can show them instantly inside the server in the nearest possible location.

Also, we have a plan to increase the nodes in different locations of the world. Though it is costly we have plans on that thing. So our service will be much faster and more accurate to user location.

All kinds of users can use our CDN service. It will help them to distribute the files in different locations as well as for different purposes and will help for getting data constantly.

Chapter 6

CONCLUSION

We have faced lots of trouble to build this project. But we enjoyed ourselves a lot. Especially the teamwork, team meeting in every week, cluster building, report writing in LaTeX, pair programming, idea sharing, contributing as a team member at GitHub repository to make this project successful. Everything we have learned on the way.

Building something different is not an easy task but from our perspective, we have tried our best to learn this new technology of Docker and Kubernetes, which will help us in our future prospects and career opportunities.

We also learned a lot while working on this project. Lots of sleepless nights to figure out where is the problem and how we can solve it. As well as different approaches to test every possible way was a good experience for us. Also attended every class for giving updates and taking lectures as well from the professor and the invited guest lecturers motivated us to keep on track. We wanted to explore and develop something new so it has given us that much thrill.

References

[1] "Container technology – a quick introduction." [Online]. Available from: https://cloudhedge.io/container-technology-a-quick-introduction/. Accessed: 20 November 2020.

[2] "Content delivery network (cdn) | low latency, high transfer speeds, video streaming | amazon cloudfront." [Online]. Available from: https://aws.amazon.com/cloudfront/. Accessed: 03 February 2021.

[3] "Cloudflare - the web performance & security company | cloudflare." [Online]. Available from: https://www.cloudflare.com/. Accessed: 03 February 2021.

[4] "Akamai technologies." [Online].
Available from: https://en.wikipedia.org/wiki/Akamai_Technologies. Accessed: 03 February 2021.

[5] "Docker overview." [Online]. Available from: https://docs.docker.com/get-started/overview/. Accessed: 03 February 2021.

[6] "Production-grade container orchestration." [Online]. Available from: https://kubernetes.io/. Accessed: 03 February 2021.

[7] "What is amazon ec2? - amazon elastic compute cloud." [Online]. Available from: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html. Accessed: 11 February 2021.

[8] "Amazon elastic block store (ebs) - amazon web services." [Online]. Available from: https://aws.amazon.com/ebs/. Accessed: 09 February 2021.

[9] "Amazon virtual private cloud (vpc)." [Online]. Available from: https://aws.amazon.com/vpc/. Accessed: 09 February 2021.

[10] "Elastic load balancing - amazon web services." [Online]. Available from:https://aws.amazon.com/elasticloadbalancing/. Accessed: 09 February 2021.

[11] "Amazon route 53 - amazon web services." [Online]. Available from: https://aws.amazon.com/route53/. Accessed: 11 February 2021.

[12] "ngrok and cross-platform development." [Online]. Available from: https://www.pubnub.com/learn/glossary/what-is-ngrok/. Accessed: 06 February 2021.