EPAM

Report on Training & Project

Cloud and DevOps

Submitted on partial fulfillment of the requirements for the award of degree of

Bachelor of Technology

In

Information Technology

Submitted to

Amandeep Kaur

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA,PUNJAB

Transforming Education Transforming India

SUBMITTED BY	**SUBMITTED TO**
Name : AKSHAY GUPTA	**Name : AMANDEEP KAUR**
Registration Number: 11913896	**Designation:Assistant Professsor**
Signature of the Student:	**Signature of the Supervisor:**

Student Declaration

To whom so ever it may concern

I,AKSHAY GUPTA,11913896, hereby declare that the work done by me on

Cloud & DevOps from 11th Jan 2023 to 28th April 2023 ,

is a record of original work for the partial fulfillment of the requirements for the award of the degree, B.Tech in INFORMATION TECHNOLOGY. Under the supervison of **Amandeep Kaur**(Trainer in Cloud and DevOps by EPAM).

Name :- AKSHAY GUPTA

Registration Number:- 11913896

Signature:-

Dated:-05/05/2023

Declaration by the Supervisor

To whom so ever it may concern,

This is to certify that Mr. Akshay Gupta registration number 1191896 from Lovely Professional University, Phagwara, Punjab has worked as a trainee in EPAM in collaboration with Lovely Professioanl University under my supervision from 22nd April 2023 to present. It is further stated that the work carried out by the student is a record of original work to the best of my knowledge for the partial-fulfillment of the requirements for the award of the **B.TECH, INFORMATION TECHNOLOGY.**

Name of Internal Supervisor	Name of External Supervisor
Amandeep Kaur	
Signature of Internal Supervisor	Signature of External Supervisor
Dated: 05/052023	Dated:

Table of Contents

Acknowledgement

I would like to express my deepest appreciation to our trainer, Amandeep Kaur, for her unwavering guidance and support throughout this project. Her expertise in Cloud and DevOps has been invaluable in helping us complete this project successfully. She has provided us with the knowledge and skills necessary to navigate the complexities of these technologies and has always been available to answer our questions and provide guidance.

Her dedication to our success has been truly inspiring. She has gone above and beyond to ensure that we have a thorough understanding of the concepts and technologies involved in this project. Her patience and willingness to share her knowledge have made this a truly rewarding experience.

In addition to her technical expertise, Amandeep Kaur has also been a source of encouragement and motivation. She has challenged us to push ourselves and to strive for excellence in all that we do. Her positive attitude and enthusiasm have been contagious, and we are grateful for the opportunity to learn from such a knowledgeable and experienced professional.

Thank you, Ms. Amandeep Kaur, for sharing your expertise with us and for helping us achieve our goals. We are grateful for your guidance and support, and we will always remember the valuable lessons that you have taught us

Chapter 1. Introduction to the Company

EPAM Systems is a global provider of digital platform engineering and software development services, catering to various industry verticals such as healthcare, finance, retail, entertainment, and more. Founded in 1993, the company has grown to become one of the leading providers of digital transformation solutions, with over 41,000 employees and operations in 36 countries. EPAM's services include digital strategy and consulting, user experience and design, software engineering, quality assurance, and more. The company's approach to software engineering emphasizes agility, automation, and efficiency, which helps clients reduce their time-to-market and achieve a competitive advantage. EPAM has a strong focus on innovation and invests heavily in research and development to create new digital solutions for clients. The company has established several innovation centers worldwide, where it collaborates with clients, universities, startups, and other stakeholders to explore emerging technologies and develop new digital products. EPAM has also created several proprietary platforms and frameworks to accelerate software development, reduce costs, and improve quality. Some of these platforms include EPAM Continuum, EPAM Anywhere ,EPAM Cloud Pipeline, and EPAM Cloud. The company's client portfolio includes several Fortune500 companies, such as Microsoft, Google, Adobe, and Pfizer, as well as many mid-sized and small enterprises. EPAM has won several awards for its innovative digital solutions and is recognized as a leader in the software engineering industry by leading research firms such as Forrester and Gartner. EPAM's corporate culture is centered around its core values, which include teamwork, excellence, and innovation. The company provides its employees with a challenging and rewarding work environment, where they can grow and develop their skills. EPAM has a strong commitment to diversity and inclusion and has established several initiatives to promote equality and empowerment in the workplace.

EPAM Systems is a global leader in digital platform engineering and software development services. The company has a strong focus on cloud computing and DevOps, which are two of the most critical trends in the technology industry today. This report provides an overview of EPAM's cloud and DevOps capabilities, including the company's approach to these areas, key offerings, and success stories.

Cloud Capabilities:

EPAM has a robust cloud practice, providing a range of services that enable its clients to leverage cloud computing to improve agility, reduce costs, and drive innovation. The company has deep expertise in a variety of cloud platforms, including AWS, Azure, Google Cloud, and IBM Cloud. EPAM's cloud offerings include:

- Cloud Consulting: EPAM helps its clients define cloud strategies, select the right cloud platform, and develop a roadmap for cloud adoption.

- Cloud Migration: EPAM has experience migrating complex, legacy applications to the cloud, including re-architecting applications to take advantage of cloud-native services.

- Cloud Application Development: EPAM has a strong focus on building cloud-native applications, leveraging modern development practices like microservices, containers, and serverless computing.

- Cloud Managed Services: EPAM provides ongoing support and management for cloud environments, including monitoring, optimization, and security.

DevOps Capabilities:

EPAM has a comprehensive DevOps practice, providing services that enable clients to accelerate software delivery, improve quality, and reduce costs. The company has a deep understanding of the DevOps toolchain and has experience implementing DevOps practices across a range of industries. EPAM's DevOps offerings include:

- DevOps Consulting: EPAM helps clients assess their current DevOps practices, identify areas for improvement, and develop a roadmap for DevOps adoption.

- Continuous Integration and Continuous Delivery (CI/CD): EPAM has experience implementing CI/CD pipelines using tools like Jenkins, GitLab, and CircleCI.

- Infrastructure as Code (IaC): EPAM has a strong focus on IaC, helping clients define infrastructure using tools like Terraform, CloudFormation, and Ansible.

- DevOps Managed Services: EPAM provides ongoing support and management for DevOps environments, including monitoring, optimization, and security.

Success Stories:

EPAM has helped numerous clients achieve success in the cloud and DevOps. Here are a few examples:

- A global insurance company wanted to migrate its legacy mainframe applications to the cloud. EPAM helped the company define its cloud strategy, re-architect its applications for the cloud, and migrate the applications to AWS. As a result, the company reduced its infrastructure costs by 50%.

- A leading fintech company wanted to accelerate its software delivery by implementing DevOps practices. EPAM helped the company implement a CI/CD pipeline, automated testing, and infrastructure as code. As a result, the company was able to reduce its time to market by 30%.

In Conclusion,EPAM Systems is a leading provider of cloud and DevOps services, helping clients leverage these technologies to drive innovation, reduce costs, and improve agility. The company has deep expertise in cloud platforms like AWS, Azure, and Google Cloud, and has experience implementing DevOps practices across a range of industries. With a focus on modern development practices and a commitment to ongoing support and management, EPAM is well positioned to help clients achieve success in the cloud and DevOps.

Chairman of the Board, CEO & President at EPAM SYSTEMS

Chapter 2: Technologies used in the Project

There are different technologies used in this project.They are explained in this chapter like Docker,AWS,Terraform,Jenkins.

2.1 AWS(Amazon Web Services)

Amazon Web Services (AWS) is a comprehensive cloud computing platform provided by Amazon. It offers a wide range of cloud-based products and services, including compute, storage, databases, analytics, networking, mobile, developer tools, management tools, IoT, security, and enterprise applications.One of the key benefits of using AWS is its scalability. With AWS, you can easily scale your resources up or down as needed to meet the demands of your business. This means that you only pay for the resources you actually use, rather than having to invest in expensive infrastructure upfront.AWS also offers a high level of reliability and security. Its infrastructure is designed to be highly available and fault-tolerant, with multiple data centers located in different geographic regions. This ensures that your data and applications are always accessible, even in the event of a natural disaster or other disruption.In terms of security, AWS provides a range of tools and features to help you protect your data and applications. These include identity and access management, data encryption, network security, and compliance programs.nother advantage of using AWS is its flexibility. With AWS, you have the freedom to choose the operating system, programming language, web application platform, database, and other services that best meet your needs. This makes it easy to build and deploy applications quickly and efficiently.Overall, AWS is a powerful cloud computing platform that offers many benefits for businesses of all sizes. Its scalability, reliability, security, and flexibility make it an ideal choice for companies looking to move their operations to the cloud.

There are lots of popular services in AWS that I have learnt in this training:

- Amazon EC2 (Elastic Compute Cloud): Amazon EC2 provides scalable computing capacity in the Amazon Web Services (AWS) Cloud. Using Amazon EC2 eliminates the need to invest in hardware upfront, so you can develop and deploy applications faster. You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage.

- Amazon S3 (Simple Storage Service): Amazon S3 is an object storage service that offers industry-leading scalability, data availability, security, and performance. It is designed to make web-scale computing easier for developers by providing a simple web services interface to store and retrieve any amount of data from anywhere on the web.

- Amazon Aurora: Amazon Aurora is a MySQL- and PostgreSQL-compatible relational database built for the cloud that combines the performance and availability of traditional enterprise databases with the simplicity and cost-effectiveness of open-source databases.

- Amazon RDS (Relational Database Service): Amazon RDS makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while automating time-consuming administration tasks such as hardware provisioning, database setup, patching, and backups.

- Amazon DynamoDB: Amazon DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent single-digit millisecond latency at any scale. It is a fully managed cloud database and supports both document and key-value store models.
- AWS Lambda: AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers. With Lambda, you can run code for virtually any type of application or backend service - all with zero administration.

- Amazon VPC (Virtual Private Cloud): Amazon VPC lets you provision a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network that you define. You have complete control over your virtual networking environment, including selection of your own IP address range, creation of subnets, and configuration of route tables and network gateways.

- Amazon EBS (Elastic Block Store): Amazon EBS provides raw block-level storage that can be attached to Amazon EC2 instances. It is designed to be used with EC2 instances and is used by Amazon Relational Database Service (RDS). EBS volumes are highly available and reliable storage volumes that can be attached to any running EC2 instance in the same Availability Zone.

- Amazon EFS (Elastic File System): Amazon EFS is a cloud storage service provided by Amazon Web Services (AWS) designed to provide scalable, elastic, concurrent with some restrictions, and encrypted file storage for use with both AWS cloud services and on-premises resources. It provides shared access to data using a traditional file sharing permissions model and hierarchical directory structure via the NFSv4 protocol.

- Amazon S3 Glacier: Amazon S3 Glacier is a secure and durable service for low-cost data archiving and long-term backup. With S3 Glacier, you can store your data cost-effectively for months, years, or even decades. S3 Glacier helps you offload the administrative burdens of operating and scaling storage to AWS, so you don't have to worry about capacity planning, hardware provisioning, data replication, hardware failure detection and recovery, or time-consuming hardware migrations

2.2Docker and Dockerfile

Docker is a platform that allows developers to easily develop, ship, and run applications by packaging them into containers. Containers are lightweight, portable, and provide a consistent environment for running applications, regardless of where they are deployed.

One of the key benefits of using Docker is its ability to simplify the development process. By using containers, developers can work in standardized environments that are isolated from the underlying infrastructure. This makes it easy to develop and test applications on a local machine, then deploy them to any environment that supports Docker.

Another advantage of using Docker is its ability to improve the deployment process. By packaging applications into containers, developers can ensure that their applications will run consistently in any environment. This eliminates many of the issues that can arise when deploying applications to different environments, such as differences in operating systems or software versions.

Dockerfiles are an important part of the Docker ecosystem. A Dockerfile is a text file that contains a set of instructions for building a Docker image. These instructions specify the base image to use, any additional software to install, and how to configure the application. Docker reads these instructions and builds the image automatically.

Here’s an example of a Dockerfile that builds an image for an Nginx web server:

```
FROM nginx:latest
COPY nginx.conf /etc/nginx/nginx.conf
COPY html /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

This Dockerfile specifies that the base image should be the latest official Nginx runtime image (FROM nginx:latest). It then copies an Nginx configuration file to the container (COPY nginx.conf /etc/nginx/nginx.conf) and copies some static HTML files to the container (COPY html /usr/share/nginx/html). Next, it exposes port 80 (EXPOSE 80) and specifies the command to start the Nginx server (CMD ["nginx", "-g", "daemon off;"]).

To build an image using this Dockerfile, you would use the docker build command. For example:
docker build -t my-nginx .

This command tells Docker to build an image using the Dockerfile in the current directory (.) and to tag the resulting image with the name my-nginx.

Once you have built the image, you can run a container from it using the docker run command. For example:
docker run -d -p 8080:80 my-nginx

This command tells Docker to run a container from the my-nginx image in detached mode (-d) and to map port 8080 on the host to port 80 in the container (-p 8080:80). This means that you can access the Nginx web server by visiting http://localhost:8080 in your web browser.

2.3 Terraform

Terraform is an open-source infrastructure as code (IaC) tool that allows you to define, provision, and manage infrastructure resources in a safe and efficient manner. With Terraform, you can use a high-level configuration language called HCL (HashiCorp Configuration Language) to describe the desired state of your infrastructure, and Terraform will automatically create, update, or delete resources as needed to match that state.

One of the key benefits of using Terraform is its ability to manage infrastructure across multiple cloud providers and other services. This means that you can use a single tool to manage resources on AWS, Azure, Google Cloud, and many other platforms. Terraform also supports a wide range of resource types, including virtual machines, databases, load balancers, and more.

Another advantage of using Terraform is its ability to track changes to your infrastructure over time. When you make changes to your Terraform configuration, Terraform will generate an execution plan that shows what changes will be made to your infrastructure. You can review this plan before applying the changes, giving you full control over your infrastructure.

Terraform also makes it easy to collaborate with others on infrastructure projects. You can store your Terraform configurations in version control systems like Git, allowing multiple people to work on the same infrastructure code. Terraform also supports remote state storage, which allows multiple people to share access to the same infrastructure state.

Here's an example of a simple Terraform configuration that creates an AWS EC2 instance:

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_instance" "example" {
  ami           = "ami-0c94855ba95c71c99"
  instance_type = "t2.micro"
}
```

This configuration specifies that the aws provider should be used (provider "aws") and sets the AWS region to us-west-2 (region = "us-west-2"). It then defines an aws_instance resource named example (resource "aws_instance" "example") and sets the Amazon Machine Image (AMI) and instance type for the instance (ami = "ami-0c94855ba95c71c99" and instance_type = "t2.micro").

To apply this configuration and create the EC2 instance, you would use the terraform apply command. For example:

```
terraform init
terraform plan
terraform apply
```

This command tells Terraform to initialize the directory,plan and apply the changes specified in the configuration file. Terraform will generate an execution plan and prompt you for confirmation before making any changes.

Overall, Terraform is a powerful tool for managing infrastructure as code. By using Terraform, you can define your infrastructure in a declarative manner and automate the process of provisioning and managing resources.

2.4 Ansible

Ansible is an open-source IT automation tool that can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates[2]. Its main goals are simplicity and ease-of-use, with a strong focus on security and reliability[2]. It features a minimum of moving parts and uses OpenSSH for transport (with other transports and pull modes as alternatives).Ansible is appropriate for managing all environments, from small setups with a handful of instances to enterprise environments with many thousands of instances[2]. It manages machines in an agent-less manner, meaning there is never a question of how to upgrade remote daemons or the problem of not being able to manage systems because daemons are uninstalled[2]. Security exposure is greatly reduced because Ansible uses OpenSSH — the open source connectivity tool for remote login with the SSH (Secure Shell) protocol.Ansible automates the management of remote systems and controls their desired state. A basic Ansible environment has three main components:

Control node:-A system on which Ansible is installed. You run Ansible commands such as ansible or ansible-inventory on a control node.

Managed node:-A remote system, or host, that Ansible controls.

Inventory:-A list of managed nodes that are logically organized. You create an inventory on the control node to describe host deployments to Ansible.

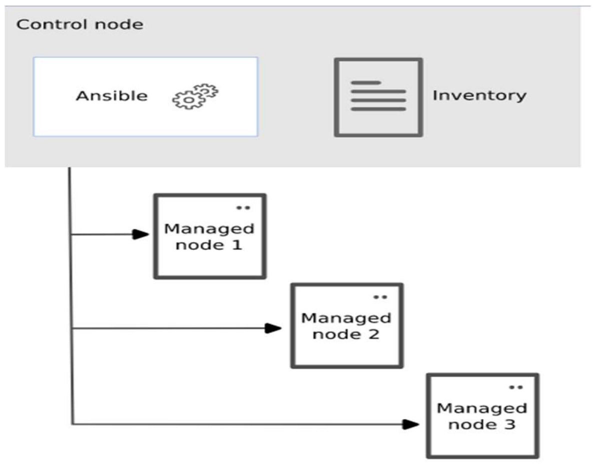


Fig:2.1 Ansible model

An Ansible Playbook is a blueprint of automation tasks, which are complex IT actions executed with limited or no human involvement. Playbooks are lists of tasks that automatically execute against hosts. Groups of hosts form your Ansible inventory. Each module within an Ansible Playbook performs a specific task. Each module contains metadata that determines when and where a task is executed, as well as which user executes it.

Playbooks offer a repeatable, re-usable, simple configuration management and multi-machine deployment system, one that is well suited to deploying complex applications[2]. If you need to execute a task with Ansible more than once, write a playbook and put it under source control. Then you can use the playbook to push out new configuration or confirm the configuration of remote systems.

Here is an example of a simple Ansible playbook that installs and starts the Apache web server on a group of hosts:

```
- name: Install and start apache
  hosts: webservers
  tasks:
    - name: Install httpd
      yum:
        name: httpd
        state: present
    - name: Start httpd
      service:
        name: httpd
        state: started
```

2.5 Jenkins

Jenkins is an open-source automation server written in Java that helps build, test, and continually deploy software. It provides hundreds of plugins to support building, deploying and automating any project.

Jenkins can be used as a simple CI server or turned into the continuous delivery hub for any project. It is easy to install and configure via its web interface, which includes on-the-fly error checks and built-in help. With hundreds of plugins in the Update Center, Jenkins integrates with practically every tool in the continuous integration and continuous delivery toolchain.

Jenkins can be extended via its plugin architecture, providing nearly infinite possibilities for what Jenkins can do[1]. It can easily distribute work across multiple machines, helping drive builds, tests and deployments across multiple platforms faster.

Here is an example of how Jenkins can be used to automate the process of building and testing project:

- A developer commits code changes to a version control system (e.g., Git).
- Jenkins detects the changes and triggers a new build of the project.
- Jenkins checks out the latest version of the code from the version control system and builds the project using tools such as Maven or Gradle.
- Jenkins runs automated tests on the built project to ensure that the changes did not introduce any new issues.
- If the tests pass, Jenkins can deploy the built project to a staging or production environment.

Jenkins Pipeline is a suite of plugins that supports implementing and integrating continuous delivery pipelines into Jenkins. A continuous delivery pipeline is an automated expression of your process for getting software from version control right through to your users and customers.

Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the Pipeline domain-specific language (DSL) syntax. The definition of a Jenkins Pipeline is written into a text file (called a Jenkinsfile) which in turn can be committed to a project’s source control repository. This is the foundation of "Pipeline-as-code"; treating the CD pipeline as a part of the application to be versioned and reviewed like any other code.

Chapter 3:Project Explanation

**Topic: “Automate provisioning and configuration of NGINX server with 2 duplicate servers in Docker container with SSL certificate in AWS cloud using Jenkins, Ansible, Terraform.
”**

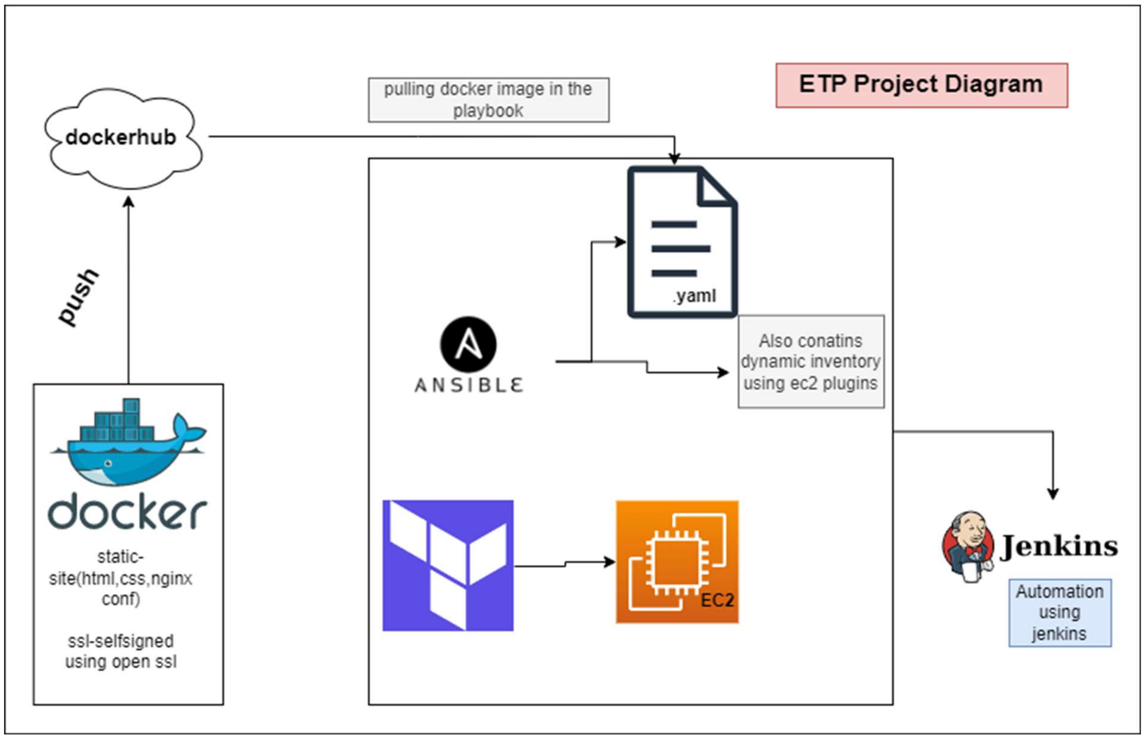

ETP Project Diagram
pulling docker image in the playbook
dockerhub
push
docker
static-site(html,css,nginx conf)
ssl-selfsigned using open ssl
ANSIBLE
.yaml
Also conatins dynamic inventory using ec2 plugins
EC2
Jenkins
Automation using jenkins

fig:Project Diagram

In today’s fast-paced and ever-changing technological landscape, automation is key to achieving efficient and reliable infrastructure management. This is especially true when it comes to provisioning and configuring servers in the cloud. In this context, we will explore how to automate the provisioning and configuration of an NGINX server with two duplicate servers in a Docker container with an SSL certificate in AWS cloud using Jenkins, Ansible, and Terraform.

Jenkins is an open-source automation server that can help us build, test, and deploy our infrastructure. Ansible is an open-source IT automation tool that can help us configure our systems and deploy our software. Terraform is an open-source infrastructure as code software tool that can help us provision our infrastructure in the cloud. By combining the power of these three tools, we can create a fully automated process for provisioning and configuring our NGINX server in AWS cloud.

3.1 Introduction: Overview of the project and its objectives

The goal of this project is to automate the provisioning and configuration of an NGINX server with two duplicate servers in a Docker container with an SSL certificate in AWS cloud using Jenkins, Ansible, and Terraform. By leveraging the power of these tools, we aim to create a fully automated process that can efficiently and reliably manage our infrastructure.

NGINX is a high-performance web server that can also be used as a reverse proxy, load balancer, and HTTP cache. By running NGINX in a Docker container, we can easily manage and scale our server infrastructure. The use of an SSL certificate ensures secure communication between our server and clients.

AWS is a leading cloud computing platform that provides a wide range of infrastructure services. By using Terraform to provision our infrastructure in AWS, we can define our infrastructure as code and easily manage changes over time.

Jenkins is an open-source automation server that can help us build, test, and deploy our infrastructure. Ansible is an open-source IT automation tool that can help us configure our systems and deploy our software. By combining these tools with Terraform, we can create a fully automated process for managing our NGINX server infrastructure in AWS cloud.

The objectives of this project are to:

- Automate the provisioning of NGINX server infrastructure in AWS cloud using Terraform
- Automate the configuration of NGINX server using Ansible
- Run NGINX server in a Docker container with an SSL certificate
- Use Jenkins to orchestrate the automation process
- Ensure efficient and reliable management of our infrastructure

3.2 Background: Explanation of the technologies used

This project involves the use of several technologies to automate the provisioning and configuration of an NGINX server with two duplicate servers in a Docker container with an SSL certificate in AWS cloud. These technologies include Jenkins, Ansible, Terraform, NGINX, Docker, SSL, and AWS.

- **Jenkins** is an open-source automation server that can help us build, test, and deploy our infrastructure. It provides a wide range of plugins and integrations that allow us to automate various tasks and processes.

- **Ansible** is an open-source IT automation tool that can help us configure our systems and deploy our software. It uses a simple, human-readable language to define automation tasks and can work with a wide range of systems and platforms.

- **Terraform** is an open-source infrastructure as code software tool that can help us provision our infrastructure in the cloud. It allows us to define our infrastructure as code and manage changes over time.

- **NGINX** is a high-performance web server that can also be used as a reverse proxy, load balancer, and HTTP cache. It is known for its stability, rich feature set, and low resource consumption.

- **Docker** is a platform for developing, shipping, and running applications in containers. Containers provide a lightweight and portable way to package and run applications, making it easy to manage and scale our infrastructure.

- **SSL** (Secure Sockets Layer) is a security protocol that provides secure communication between a server and a client by encrypting the data transmitted between them. An SSL certificate is used to authenticate the identity of the server and establish an encrypted connection.

- **AWS** (Amazon Web Services) is a leading cloud computing platform that provides a wide range of infrastructure services. It allows us to easily provision and manage our infrastructure in the cloud.

3.3 NGINX Server Setup: Steps for setting up NGINX server in a Docker container with SSL certificate

- **Install Docker:** Ensure that Docker is installed on the host machine where the NGINX server will be running.
- **Create NGINX Docker Image:** Create a Docker image for the NGINX server. This can be done by writing a Dockerfile that specifies the base image (e.g., nginx:latest), installs any necessary dependencies, and copies the NGINX configuration files into the image.
- **Self-signed SSL Certificate:** Use open SSl to generate the self signed SSL certificate.
- **Static site:**This container also contains the static site files like html and CSS.
- **Configure NGINX:** Configure the NGINX server to use the SSL certificate. This will typically involve specifying the paths to the certificate and private key files in the NGINX configuration file.
- **Build and Run Docker Container:** Build the NGINX Docker image and run it as a container. This can be done using Docker commands such as docker build and docker run.
- **Verify Setup:** Verify that the NGINX server is running correctly and that it is using the SSL certificate. This can be done by accessing the server using a web browser and checking that the connection is secure.

dockerfile

```
FROM nginx

RUN mkdir /etc/nginx/private /etc/nginx/certs
COPY static-site/index.html /etc/nginx/html/
COPY static-site/index.css /etc/nginx/html/
COPY static-site/nginx.conf /etc/nginx/nginx.conf

RUN openssl req -x509 -nodes -days 365 \
-subj "/C=IN/ST=Punjab/O=Lovely Professional University, LLC/CN=lpu.in"
\
-newkey rsa:2048 -keyout /etc/nginx/private/nginx-selfsigned.key \
-out /etc/nginx/certs/nginx-selfsigned.crt;
```

nginx.conf

```
worker_processes  1;
events {
    worker_connections  1024;
}
http {
    include       mime.types;
    default_type  application/octet-stream;
    sendfile        on;

    keepalive_timeout  65;

    server {
        listen       80;
        server_name  localhost;

        location / {
            root   html;
            index  index.html index.htm;
        }

        error_page   500 502 503 504  /50x.html;

        location = /50x.html {
            root   html;
        }
    }

    server {

        listen 443 http2 ssl;
        listen [::]:443 http2 ssl;

        server_name localhost;

        ssl_certificate /etc/nginx/certs/nginx-selfsigned.crt;
        ssl_certificate_key /etc/nginx/private/nginx-selfsigned.key;

        location / {
            root   html;
            index  index.html index.htm;
        }

        error_page   500 502 503 504  /50x.html;

        location = /50x.html {
            root   html;
        }
    }

}
```

Index.html

```
<!DOCTYPE html>

<html>

<head>

  <title>Nginx EC2</title>

  <link href="index.css" rel="stylesheet">

</head>

<body>

  <main class="main">

    <h1>Hi, I am Akshay Gupta.You are in EPAM ETP project.Thank
You</h1>

  </main>

</body>

</html>
```

Index.css

```
body {

  margin:0;

  font-family: Arial, Helvetica, sans-serif;

  background: linear-gradient(to right, #5c258d, #4389a2);

  color:white;

}

.main {

  width:100%;

  height:100vh;

  display:flex;

  justify-content: center;

  align-items: center;

}
```

3.4 Terraform Provisioning: Steps for provisioning infrastructure in AWS cloud using Terraform

In this step I have created two Ec2 infrastructure using below code.

```
terraform {
  required_providers {
 aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    } }}
provider "aws" {
  region = "us-east-1"
}
resource "aws_instance" "epam_final" {
    ami = "ami-007855ac798b5175e"
    instance_type="t2.micro"
    key_name="epamkey"
    vpc_security_group_ids = ["sg-053a16f49a2aeb034","sg-00cdf3d5e989315b0"]
        tags = {
           Name="instance_1"  }}
```

```
resource "aws_instance" "epam_final" {
    ami = "ami-007855ac798b5175e"
    instance_type="t2.micro"
    key_name="epamkey"
    vpc_security_group_ids = ["sg-053a16f49a2aeb034","sg-00cdf3d5e989315b0"]
        tags = {
          Name="instance_2"}}
```

This Terraform code specifies the required AWS provider and its version. It then configures the provider to use the us-east-1 region. The code also includes two aws_instance resources that define two EC2 instances to be created in AWS. Both instances use the same Amazon Machine Image (AMI) and instance type (t2.micro), and are associated with the same key pair (epamkey) and security groups (sg-053a16f49a2aeb034 and sg-00cdf3d5e989315b0). The instances are given different names (instance_1 and instance_2) using tags.

3.5 Ansible Configuration: Steps for configuring Ansible for server configuration management

- **Install Ansible:** Ensure that Ansible is installed on the machine where the configuration management will be performed.

- **Set up Inventory:** In this project the dynamic inventory is used with the help of aws ec2 ansible plugin ie. amazon.com.aws_ec2

```
plugin: amazon.aws.aws_ec2
regions:
  - us-east-1
filters:
  # All instances with their `Name` tag set to `dev
  tag:Name:
    - instance_1
    - instance_2
```

- **Write Playbooks:** After this the Ansible playbooks that define the desired configuration of the servers. This will typically involve specifying tasks that use Ansible modules to perform actions such as installing packages,pulling docker images and running it.Below is the code.

```
- name: Provision Web Servers

  hosts: aws_ec2

  tasks:



  - name: Install pip3
- name: Provision Web Servers
  hosts: aws_ec2
  tasks:

  - name: Install pip3
    apt:
      update_cache: yes
      name: python3-pip
    become: yes

  - name: Install python docker sdk
    shell: |
      pip3 install docker
    become: yes

  - name: Install docker
    apt:
      name: docker.io
    become: yes

  - name: Start Docker
    shell: |
      systemctl start docker
      systemctl enable docker
    become: yes
```

```
- name: Pull the image
    docker_image:
      name: ak27473/nginx-static
      source: pull

  - name: Run the container
    docker_container:
      name: hello
      image: ak27473/nginx-static
      state: started
      ports:
        - "80:80"
        - "443:443"
```

This Ansible playbook provisions web servers by performing a series of tasks on the hosts specified in the `aws_ec2` group. The tasks include:

- **Install pip3:** The first task installs pip3 using the apt module. This task uses become: yes to run the command with elevated privileges.

- **Install python docker sdk:** The second task installs the Python Docker SDK using the shell module to run a pip3 install command. This task also uses become: yes to run the command with elevated privileges.

- **Install docker:** The third task installs Docker using the apt module. This task also uses become: yes to run the command with elevated privileges.

- **Start Docker:** The fourth task starts and enables the Docker service using the shell module to run systemctl commands. This task also uses become: yes to run the commands with elevated privileges.

- **Pull the image:** The fifth task pulls a Docker image (ak27473/nginx-static) from a registry using the docker_image module.

- **Run the container:** The sixth task runs a Docker container using the pulled image (ak27473/nginx-static) and maps ports 80 and 443 from the host to the container using the docker_container module.

3.6 Jenkins Setup: Steps for setting up Jenkins for automation

- **Install Jenkins:** Ensure that Jenkins is installed on the machine where the automation will be performed. This can be done by following the installation instructions for your operating system on the Jenkins website.

- **Configure Jenkins:** Configure Jenkins by accessing its web interface and following the setup wizard. This will typically involve creating an admin user, configuring security settings, and installing any necessary plugins.

- **Install Required Plugins:-**The required plugins like terraform and cloudbees(to store the AWS credentials) is installed.

- **Set up Build Jobs:** Set up build jobs in Jenkins to automate the tasks that you want to perform. This can be done by creating new jobs and configuring them to perform the desired actions, such as building code, running tests, or deploying software.

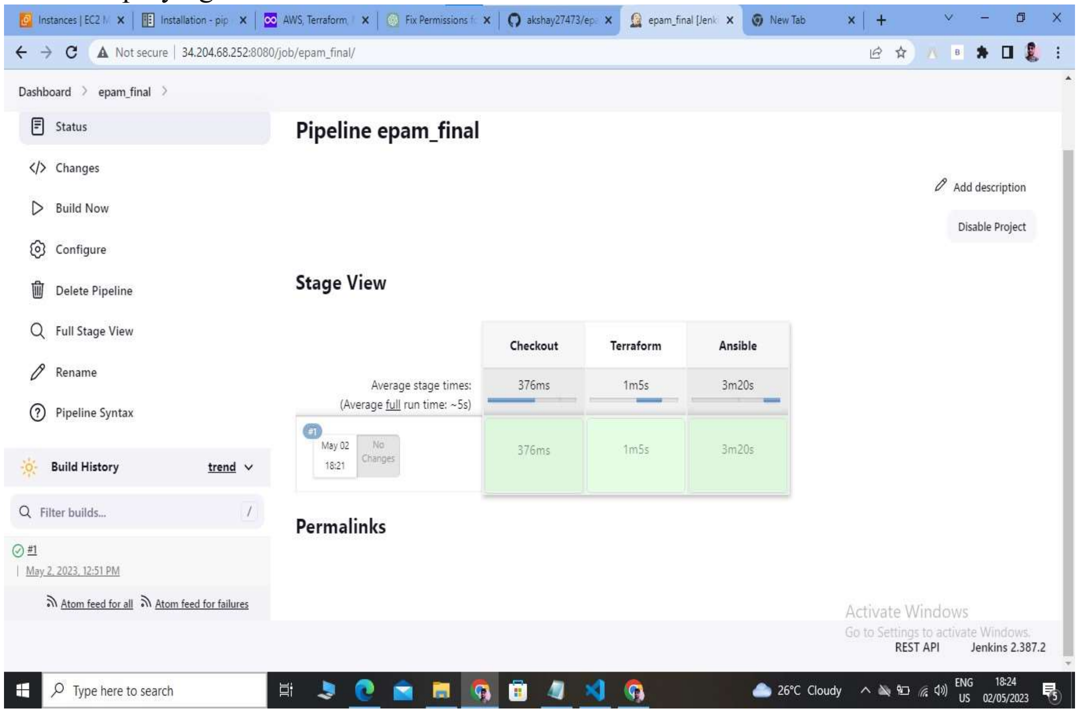
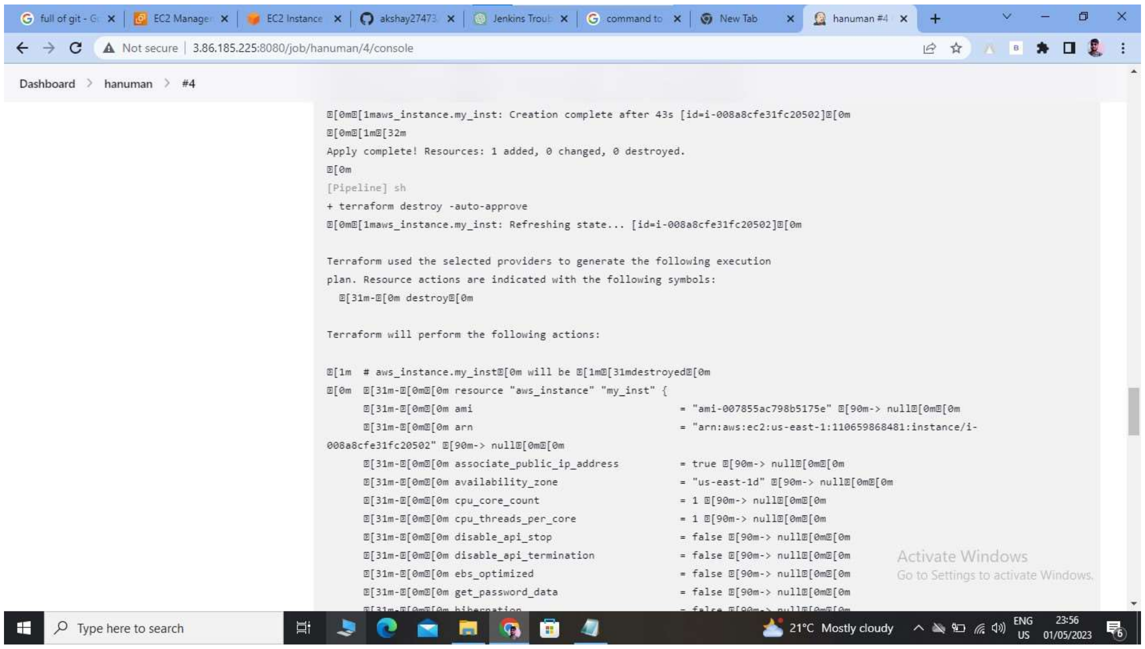
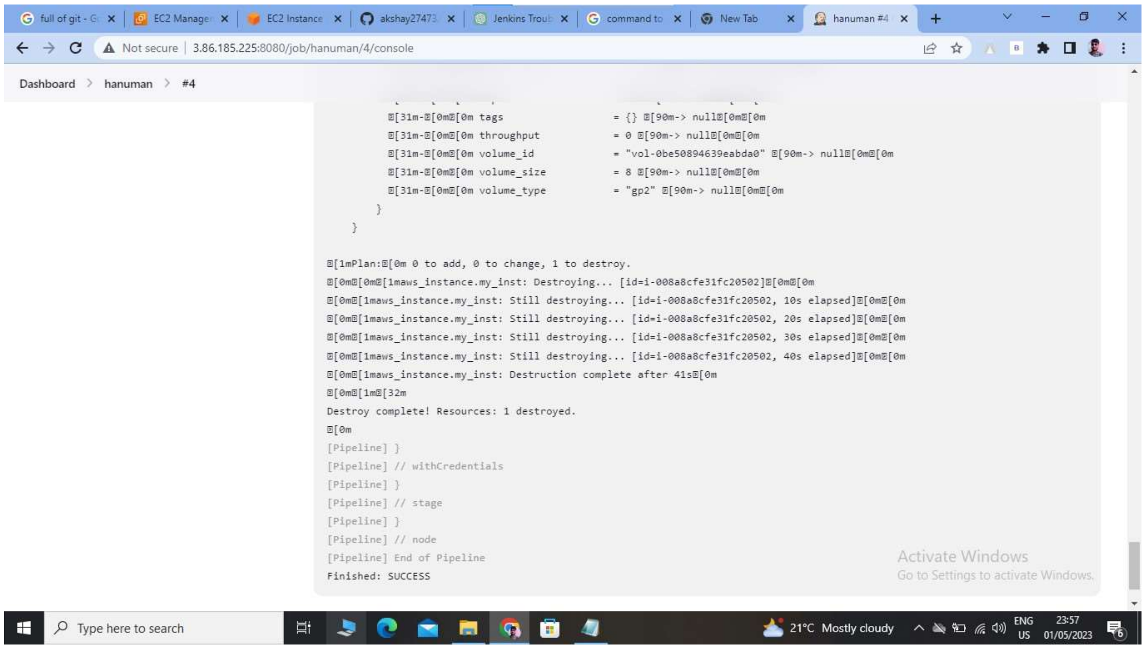


Fig 3.1 Jenkins Pipeline



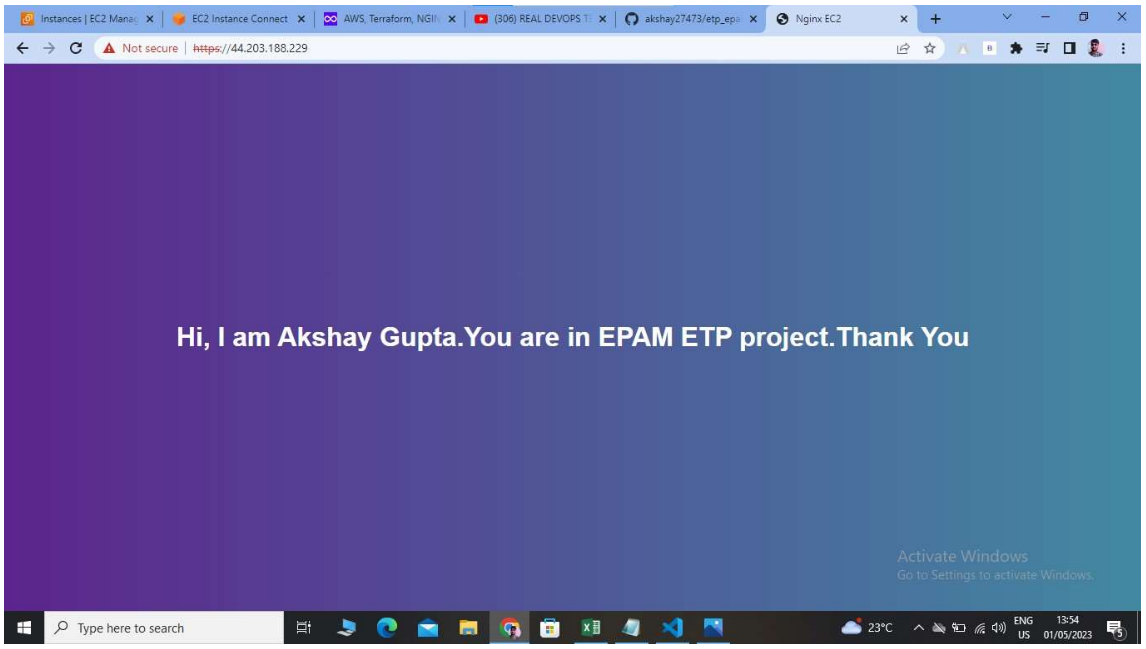
```
[0m[1maws_instance.my_inst: Creation complete after 43s [id=i-008a8cfe31fc20502][0m
[0m[1m[32m
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
[0m
[Pipeline] sh
+ terraform destroy -auto-approve
[0m[1maws_instance.my_inst: Refreshing state... [id=i-008a8cfe31fc20502][0m

Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  [31m-[0m destroy[0m

Terraform will perform the following actions:

[1m  # aws_instance.my_inst[0m will be [1m[31mdestroyed[0m
[0m  [31m-[0m[0m resource "aws_instance" "my_inst" {
      [31m-[0m[0m ami                                  = "ami-007855ac798b5175e" [90m-> null[0m[0m
      [31m-[0m[0m arn                                  = "arn:aws:ec2:us-east-1:110659868481:instance/i-
008a8cfe31fc20502" [90m-> null[0m[0m
      [31m-[0m[0m associate_public_ip_address          = true [90m-> null[0m[0m
      [31m-[0m[0m availability_zone                    = "us-east-1d" [90m-> null[0m[0m
      [31m-[0m[0m cpu_core_count                       = 1 [90m-> null[0m[0m
      [31m-[0m[0m cpu_threads_per_core                 = 1 [90m-> null[0m[0m
      [31m-[0m[0m disable_api_stop                     = false [90m-> null[0m[0m
      [31m-[0m[0m disable_api_termination              = false [90m-> null[0m[0m
      [31m-[0m[0m ebs_optimized                        = false [90m-> null[0m[0m
      [31m-[0m[0m get_password_data                    = false [90m-> null[0m[0m
```



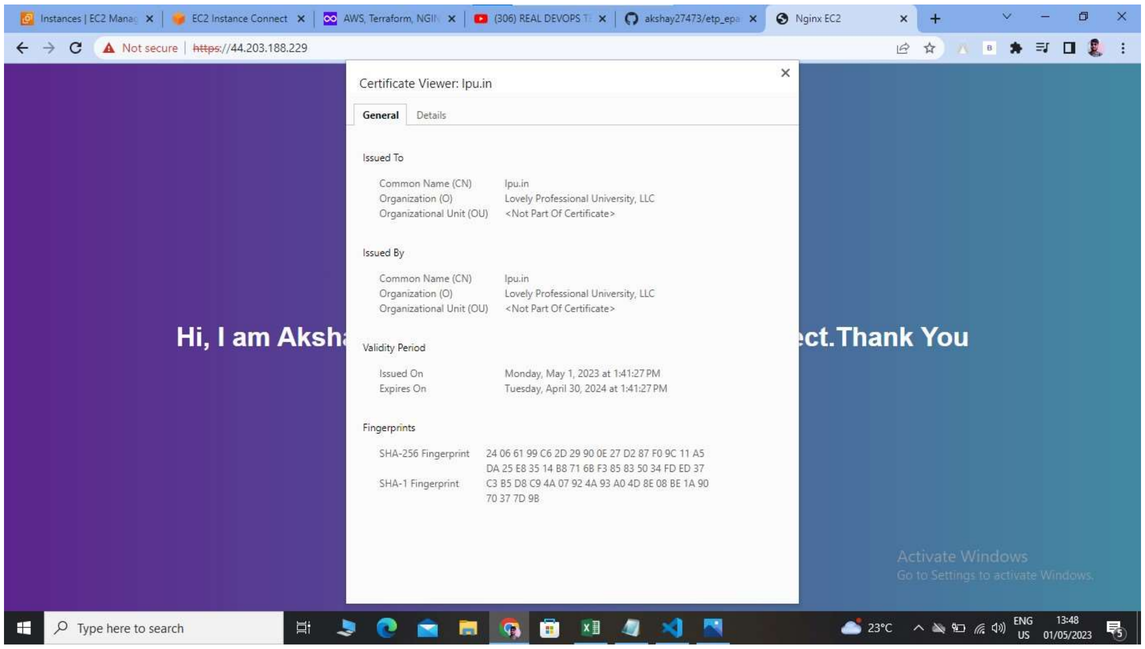
```
        ⊠[31m-⊠[0m⊠[0m tags                 = {} ⊠[90m-> null⊠[0m⊠[0m
        ⊠[31m-⊠[0m⊠[0m throughput           = 0 ⊠[90m-> null⊠[0m⊠[0m
        ⊠[31m-⊠[0m⊠[0m volume_id            = "vol-0be50894639eabda0" ⊠[90m-> null⊠[0m⊠[0m
        ⊠[31m-⊠[0m⊠[0m volume_size          = 8 ⊠[90m-> null⊠[0m⊠[0m
        ⊠[31m-⊠[0m⊠[0m volume_type          = "gp2" ⊠[90m-> null⊠[0m⊠[0m
      }
  }

⊠[1mPlan:⊠[0m 0 to add, 0 to change, 1 to destroy.
⊠[0m⊠[0m⊠[1maws_instance.my_inst: Destroying... [id=i-008a8cfe31fc20502]⊠[0m⊠[0m
⊠[0m⊠[1maws_instance.my_inst: Still destroying... [id=i-008a8cfe31fc20502, 10s elapsed]⊠[0m⊠[0m
⊠[0m⊠[1maws_instance.my_inst: Still destroying... [id=i-008a8cfe31fc20502, 20s elapsed]⊠[0m⊠[0m
⊠[0m⊠[1maws_instance.my_inst: Still destroying... [id=i-008a8cfe31fc20502, 30s elapsed]⊠[0m⊠[0m
⊠[0m⊠[1maws_instance.my_inst: Still destroying... [id=i-008a8cfe31fc20502, 40s elapsed]⊠[0m⊠[0m
⊠[0m⊠[1maws_instance.my_inst: Destruction complete after 41s⊠[0m
⊠[0m⊠[1m⊠[32m
Destroy complete! Resources: 1 destroyed.
⊠[0m
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Fig 3.2 Console Output

After all the steps are followed the following output will be shown.



Hi, I am Akshay Gupta.You are in EPAM ETP project.Thank You



Certificate Viewer: Ipu.in

General Details

Issued To

Common Name (CN)	Ipu.in
Organization (O)	Lovely Professional University, LLC
Organizational Unit (OU)	<Not Part Of Certificate>

Issued By

Common Name (CN)	Ipu.in
Organization (O)	Lovely Professional University, LLC
Organizational Unit (OU)	<Not Part Of Certificate>

Validity Period

Issued On	Monday, May 1, 2023 at 1:41:27 PM
Expires On	Tuesday, April 30, 2024 at 1:41:27 PM

Fingerprints

SHA-256 Fingerprint	24 06 61 99 C6 2D 29 90 0E 27 D2 87 F0 9C 11 A5 DA 25 E8 35 14 B8 71 6B F3 85 83 50 34 FD ED 37
SHA-1 Fingerprint	C3 B5 D8 C9 4A 07 92 4A 93 A0 4D 8E 08 BE 1A 90 70 37 7D 9B

Chapter 4. Conclusion: Summary of the project and its outcomes

The project aimed to automate the provisioning and configuration of an NGINX server with two duplicate servers in a Docker container with an SSL certificate in the AWS cloud using Jenkins, Ansible, and Terraform. This involved setting up a continuous integration and continuous deployment (CI/CD) pipeline using Jenkins to automate the deployment of the NGINX server and its duplicate servers. Ansible was used for configuration management, allowing for consistent and repeatable server configurations. Terraform was used for infrastructure as code, enabling the provisioning of resources in the AWS cloud in an automated and scalable manner. The use of an SSL certificate ensured secure communication between the server and its clients.

The successful implementation of this project resulted in a streamlined and efficient process for deploying and managing NGINX servers in the AWS cloud. By automating the provisioning and configuration of the servers, the project reduced the potential for human error and increased consistency across deployments. The use of Jenkins, Ansible, and Terraform allowed for a high degree of automation and scalability, enabling the project to easily adapt to changing requirements.

Chapter 5. Future Scope

The future scope for this project could involve expanding the automation and configuration capabilities to include additional server types and cloud platforms. For example, the project could be extended to automate the provisioning and configuration of other web servers such as Apache or Lighttpd, or to support deployment to other cloud platforms such as Microsoft Azure or Google Cloud Platform.

Another potential area for future development could be the integration of additional tools and technologies to further enhance the automation and management capabilities of the project. For example, the project could incorporate monitoring and logging tools to provide greater visibility into the performance and health of the NGINX servers. Additionally, the project could explore the use of container orchestration tools such as Kubernetes to manage the deployment and scaling of the NGINX servers in a more automated and efficient manner.

Overall, there are many potential avenues for future development and enhancement of this project, with the ultimate goal of providing a highly automated and scalable solution for deploying and managing NGINX servers in the cloud.

In addition to expanding the automation and configuration capabilities to include additional server types and cloud platforms, and integrating additional tools and technologies, there are other potential areas for future development of this project.

One such area could be the incorporation of advanced security measures to further enhance the security of the NGINX servers and their communications. This could involve the implementation of additional encryption protocols, the use of firewalls and intrusion detection systems, and regular security audits to ensure that the servers remain secure and protected against potential threats.

Another potential area for future development could be the exploration of new and emerging technologies such as artificial intelligence and machine learning to further enhance the automation and management capabilities of the project. For example, machine learning algorithms could be used to analyze server performance data and automatically adjust server configurations to optimize performance.

Overall, there are many potential avenues for future development and enhancement of this project, with the ultimate goal of providing a highly automated, scalable, and secure solution for deploying and managing NGINX servers in the cloud.

References

https://www.scottyfullstack.com/blog

https://www.youtube.com/watch?v=hWlc3IgnBUA

https://www.youtube.com/watch?v=C5mazUJ1dsg

https://www.youtube.com/watch?v=9nn21eHBkXg&t=732s

https://www.google.com