Cloud Infrastructure Automation and Container Orchestration with Kubernetes

TABLE OF CONTENTS

Problem statement:

The problem statement for the project on cloud infrastructure automation and container orchestration with Kubernetes could be:

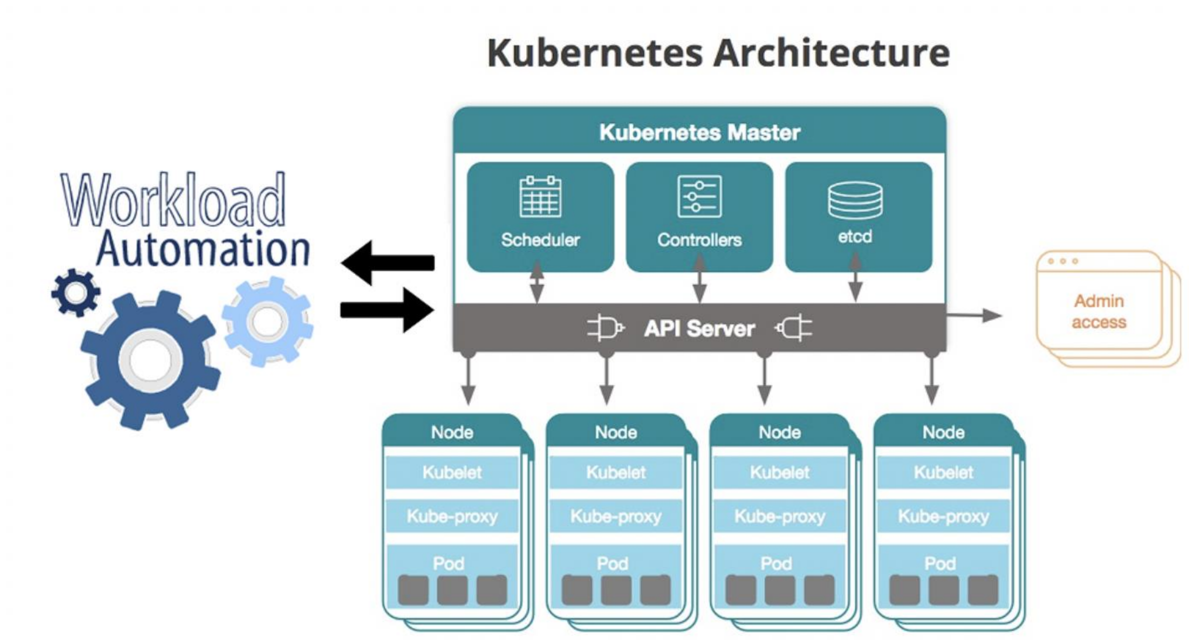
As businesses increasingly adopt cloud infrastructure and containerization for their applications, there is a growing need for automation and orchestration tools to manage and scale these resources efficiently. Kubernetes is a popular open-source platform for container orchestration, but configuring and managing Kubernetes clusters can be complex and time-consuming.

The goal of this project is to develop an automated solution for deploying and managing Kubernetes clusters in the cloud, along with containerized applications running on those clusters. The solution should leverage infrastructure-as-code principles and use tools such as Terraform and Ansible to automate the provisioning and configuration of cloud resources. It should also incorporate best practices for security, scalability, and high availability.

The project should demonstrate the following:

1. Automated deployment of a Kubernetes cluster in a cloud environment using Terraform and Ansible.
2. Integration of containerized applications with the Kubernetes cluster using Kubernetes deployment manifests and other relevant resources.
3. Implementation of security best practices, such as network isolation and encryption, to protect the cluster and its applications.
4. Scaling of the cluster to accommodate changing workload demands, using Kubernetes autoscaling and other relevant tools.
5. High availability and resilience of the cluster, using features such as Kubernetes replication and failover.

The outcome of the project should be a fully functional, automated solution for cloud infrastructure automation and container orchestration with Kubernetes, along with documentation and instructions for deploying and using the solution.



Problem Definition: The adoption of cloud computing and containerization technologies has brought significant benefits to organizations in terms of scalability, flexibility, and cost-effectiveness. However, managing cloud infrastructure and deploying containerized applications at scale can be challenging and time-consuming, especially for organizations with complex and dynamic environments. The use of cloud infrastructure automation and container orchestration tools such as Terraform, CloudFormation, and Kubernetes can help to address these challenges and improve the efficiency, reliability, and agility of cloud-based applications.

The objective of this project is to develop a cloud infrastructure automation and container orchestration solution using Kubernetes. The project aims to demonstrate the benefits of using Kubernetes as a container orchestration tool for managing and deploying applications on a cloud provider such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform (GCP). The project will involve setting up a Kubernetes cluster on the chosen cloud provider, configuring the necessary cloud infrastructure using Terraform or CloudFormation, and deploying containerized applications using Kubernetes deployment and service manifests.

The project will address the following challenges:

1. Provisioning and managing cloud infrastructure at scale.

2. Deploying containerized applications consistently and reliably across different environments
3. Scaling and managing applications dynamically based on workload demands.
4. Ensuring high availability, fault tolerance, and security of cloud-based applications

Hardware Specifications:

- A computer with at least 8GB of RAM and a multi-core processor
- Access to a cloud provider such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform (GCP) to set up the Kubernetes cluster and provision the necessary cloud infrastructure.
- High-speed internet connection for downloading and installing software, and for deploying and managing applications on the cloud.

Software Specifications:

- Operating System: Linux or MacOS
- Docker: To build and manage container images of the applications to be deployed.
- Kubernetes: To manage containerized applications on the cloud provider.
- Terraform or CloudFormation: To automate the provisioning and configuration of cloud infrastructure.
- Git: To manage source code and version control of the application code
- IDE or text editor of your choice: To write and edit code for the application and automation scripts.
- Jenkins: To set up and manage the continuous integration and deployment pipeline.

It is important to note that the exact hardware and software specifications may vary depending on the specific cloud provider and tools being used. The above specifications are provided as a general guideline to help you get started with the project.

Specifications:

Here are some possible specifications for the project on cloud infrastructure automation and container orchestration with Kubernetes:

1. Infrastructure Provisioning: The solution should automate the provisioning of cloud resources required to deploy a Kubernetes cluster, including virtual machines, networking, storage, and load balancers. This can be achieved using Terraform, which provides infrastructure as code capabilities.
2. Kubernetes Cluster Deployment: The solution should automate the deployment of a Kubernetes cluster on the provisioned infrastructure using Ansible, which provides configuration management capabilities. The Kubernetes cluster should be configured to meet best practices for security, scalability, and high availability.
3. Containerized Application Deployment: The solution should automate the deployment of containerized applications on the Kubernetes cluster using Kubernetes deployment manifests, which describe the desired state of the application deployment. The solution should also ensure that the application is accessible from outside the cluster, either via load balancing or through the use of ingress resources.
4. Security and Compliance: The solution should implement security best practices for the Kubernetes cluster and the deployed applications, such as network isolation, encryption, authentication, and authorization. Compliance with relevant regulations and standards, such as GDPR or HIPAA, should also be ensured.
5. Monitoring and Logging: The solution should integrate with monitoring and logging tools such as Prometheus and Elasticsearch, to collect and analyze metrics and logs from the Kubernetes cluster and the deployed applications. This will enable proactive detection of issues and timely response to incidents.
6. Auto Scaling: The solution should enable the Kubernetes cluster to scale up or down based on workload demands using Kubernetes autoscaling features, such as horizontal pod autoscaling or cluster autoscaling.
7. High Availability and Disaster Recovery: The solution should ensure high availability and resilience of the Kubernetes cluster and the deployed applications, using features such as Kubernetes replication, failover, and backup and restore capabilities.

8. Documentation and Training: The solution should include clear and concise documentation on how to deploy, configure, and use the solution, along with training materials to help users understand the underlying concepts and technologies.

Overall, the goal of this project is to deliver a fully automated and scalable solution for cloud infrastructure automation and container orchestration with Kubernetes, which enables efficient deployment and management of containerized applications in the cloud.

Module description:

Module 1: Introduction to Cloud Infrastructure Automation and Container Orchestration

- Overview of cloud infrastructure automation and container orchestration
- Introduction to Kubernetes and its architecture
- Benefits of using cloud infrastructure automation and container orchestration tools

Module 2: Setting up a Kubernetes Cluster on a Cloud Provider

- Choosing a cloud provider and creating an account
- Provisioning the necessary cloud infrastructure using Terraform or CloudFormation
- Configuring and securing the Kubernetes cluster
- Connecting to the Kubernetes cluster using kubectl

Module 3: Containerizing Applications with Docker

- Introduction to Docker and containerization
- Building and managing Docker images
- Pushing Docker images to a container registry

Module 4: Deploying Applications with Kubernetes

- Overview of Kubernetes deployment and service manifests
- Deploying containerized applications on the Kubernetes cluster
- Scaling and managing applications using Kubernetes controllers and ReplicaSets
- Exposing and accessing applications using Kubernetes services

Module 5: Automating Cloud Infrastructure and Application Deployment

- Overview of automation tools like Ansible, Chef, and Puppet
- Setting up a Jenkins server for continuous integration and deployment
- Automating cloud infrastructure and application deployment using Jenkins and other tools

Module 6: Advanced Kubernetes Features and Best Practices

- Introduction to advanced Kubernetes features like DaemonSets, StatefulSets, and Ingress
- Best practices for managing and securing Kubernetes clusters and applications.
- Monitoring and logging Kubernetes clusters and applications

Module 7: Project Demonstration and Presentation

- Demo of the project implementation
- Presentation of the project findings, challenges, and recommendations
- Q&A session with the project supervisor and audience

Module 1: Introduction to Cloud Infrastructure Automation and Container Orchestration

In this module, we will provide an overview of cloud infrastructure automation and container orchestration, and explain the benefits of using Kubernetes as a container orchestration platform. We will also discuss the challenges and considerations involved in deploying and managing Kubernetes clusters.

Topics Covered:

- Cloud Infrastructure Automation
- Container Orchestration
- Kubernetes Overview
- Benefits of Using Kubernetes
- Challenges and Considerations for Kubernetes Deployment and Management

Learning Objectives:

- Understand the benefits of cloud infrastructure automation and container orchestration.
- Explain the role of Kubernetes as a container orchestration platform.
- Identify the challenges and considerations involved in deploying and managing Kubernetes clusters.
- Gain an understanding of the topics to be covered in the rest of the course.

Prerequisites:

- Familiarity with cloud computing concepts and technologies.
- Basic understanding of containerization and Docker.
- Basic knowledge of Linux and shell scripting.

Module 2: Setting up a Kubernetes Cluster on a Cloud Provider

In this module, we will demonstrate how to set up a Kubernetes cluster on a cloud provider, such as Amazon Web Services (AWS), using infrastructure-as-code principles and tools such as Terraform and Ansible. We will also cover best practices for security, scalability, and high availability in Kubernetes cluster deployments.

Topics Covered:

- Cloud Provider Selection
- Infrastructure-as-Code with Terraform
- Configuration Management with Ansible
- Security Best Practices for Kubernetes Cluster Deployments
- Scalability and High Availability in Kubernetes Cluster Deployments

Learning Objectives:

- Understand how to select a cloud provider for Kubernetes cluster deployment.
- Demonstrate how to set up a Kubernetes cluster on a cloud provider using Terraform and Ansible.
- Learn best practices for security, scalability, and high availability in Kubernetes cluster deployments.
- Understand the principles and tools used for infrastructure-as-code and configuration management in Kubernetes cluster deployments.

Prerequisites:

- Familiarity with cloud provider services such as AWS.
- Understanding of infrastructure-as-code principles and tools such as Terraform.
- Basic knowledge of configuration management tools such as Ansible.
- Familiarity with Kubernetes concepts and architecture.

Module 3: Containerizing Applications with Docker

In this module, we will cover the basics of containerization using Docker, including creating Docker images and containers, and deploying them to a Kubernetes cluster. We will also cover best practices for creating Docker images and optimizing container performance.

Topics Covered:

- Introduction to Docker
- Creating Docker Images
- Running Docker Containers
- Deploying Docker Containers to Kubernetes
- Best Practices for Docker Image Creation and Container Performance

Learning Objectives:

- Understand the basics of containerization using Docker.
- Learn how to create Docker images and run Docker containers.
- Understand how to deploy Docker containers to a Kubernetes cluster.
- Gain knowledge of best practices for Docker image creation and container performance.

Prerequisites:

- Basic understanding of Linux command line and shell scripting.
- Familiarity with containerization concepts and technologies.
- Basic knowledge of Docker.

Module 4: Deploying Applications with Kubernetes

In this module, we will cover the fundamentals of deploying applications with Kubernetes, including creating Kubernetes deployment manifests, using services to expose applications, and managing application updates and rollbacks. We will also cover advanced topics such as deploying stateful applications and using advanced networking features.

Topics Covered:

- Kubernetes Deployment Manifests
- Services in Kubernetes
- Updating and Rolling Back Deployments
- Deploying Stateful Applications
- Advanced Networking Features in Kubernetes

Learning Objectives:

- Understand how to create Kubernetes deployment manifests.
- Learn how to use services to expose applications in Kubernetes.
- Understand how to update and roll back deployments in Kubernetes.
- Gain knowledge of deploying stateful applications in Kubernetes.
- Learn about advanced networking features in Kubernetes.

Prerequisites:

- Familiarity with containerization and Docker concepts.
- Understanding of Kubernetes architecture and basic operations.
- Basic knowledge of YAML and Kubernetes deployment manifests.

Module 5: Automating Cloud Infrastructure and Application Deployment

In this module, we will cover the principles and tools for automating cloud infrastructure and application deployment, including continuous integration and continuous deployment (CI/CD) pipelines. We will also cover best practices for testing, monitoring, and scaling cloud infrastructure and applications.

Topics Covered:

- Principles of Infrastructure-as-Code
- Continuous Integration and Continuous Deployment (CI/CD)
- Automating Kubernetes Cluster Deployments
- Testing and Monitoring Cloud Infrastructure and Applications
- Scaling Cloud Infrastructure and Applications

Learning Objectives:

- Understand the principles of infrastructure-as-code and its benefits.
- Learn how to implement continuous integration and continuous deployment (CI/CD) pipelines.
- Understand how to automate Kubernetes cluster deployments.
- Learn best practices for testing and monitoring cloud infrastructure and applications.
- Gain knowledge of scaling cloud infrastructure and applications.

Prerequisites:

- Familiarity with cloud provider services such as AWS.
- Understanding of infrastructure-as-code principles and tools such as Terraform.
- Basic knowledge of configuration management tools such as Ansible.
- Familiarity with Kubernetes concepts and architecture.

Module 6: Advanced Kubernetes Features and Best Practices

In this module, we will cover advanced Kubernetes features and best practices, including advanced networking, storage, and security features. We will also cover Kubernetes cluster management, disaster recovery, and performance tuning.

Topics Covered:

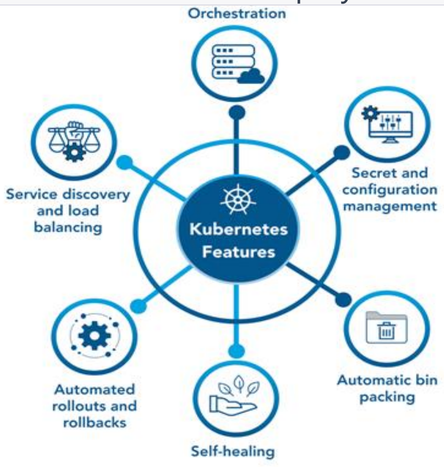
- Advanced Networking Features in Kubernetes
- Storage in Kubernetes
- Kubernetes Security Best Practices
- Kubernetes Cluster Management
- Disaster Recovery in Kubernetes
- Kubernetes Performance Tuning

Learning Objectives:

- Understand advanced networking features in Kubernetes, including network policies and service mesh.
- Learn about storage options in Kubernetes and best practices for storage management.
- Gain knowledge of Kubernetes security best practices and how to secure Kubernetes clusters.
- Understand Kubernetes cluster management and best practices for cluster administration.
- Learn about disaster recovery strategies in Kubernetes and how to implement them.
- Gain knowledge of Kubernetes performance tuning and optimization.

Prerequisites:

- Familiarity with containerization and Docker concepts.
- Understanding of Kubernetes architecture and basic operations.
- Basic knowledge of YAML and Kubernetes deployment manifests.


Orchestration
Secret and configuration management
Service discovery and load balancing
Kubernetes Features
Automated rollouts and rollbacks
Self-healing
Automatic bin packing

Module 7: Project Demonstration and Presentation

In this module, students will have the opportunity to demonstrate their cloud infrastructure automation and container orchestration project, including the deployment of a sample application on a Kubernetes cluster using Docker images and Kubernetes deployment manifests. Students will also be required to give a presentation on their project, highlighting the project objectives, implementation details, and any challenges they faced.

Topics Covered:

- Project Demonstration
- Project Presentation

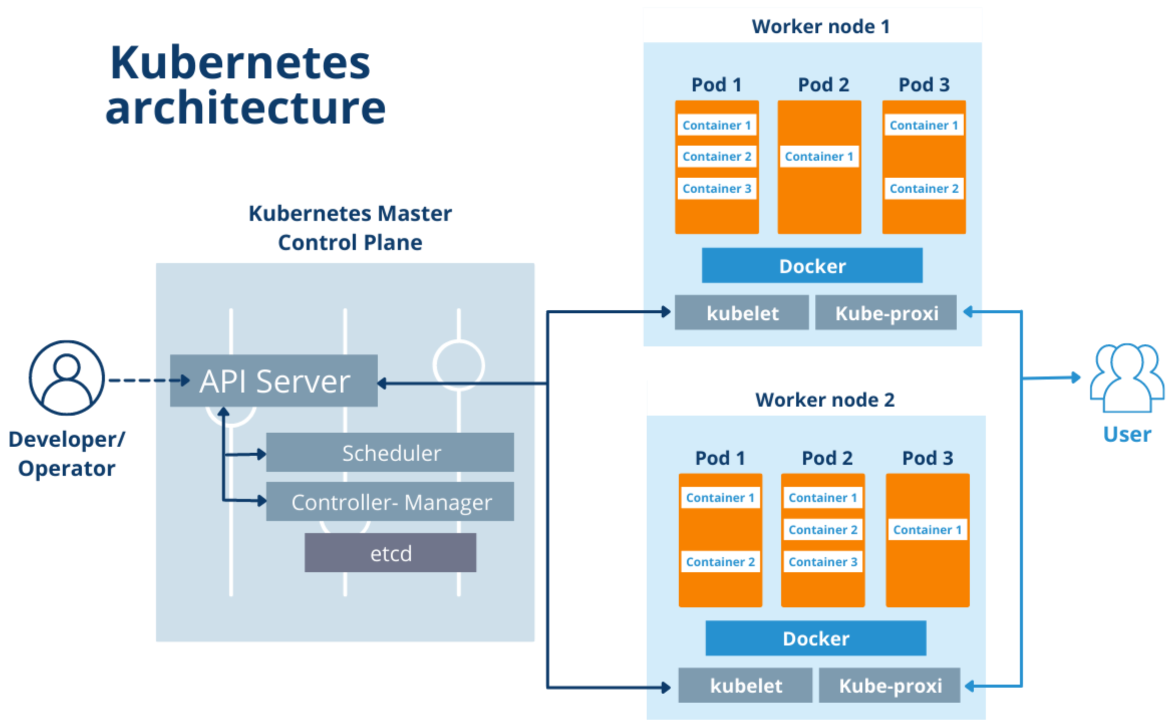
Learning Objectives:

- Demonstrate proficiency in cloud infrastructure automation and container orchestration using Kubernetes and Docker.
- Showcase the ability to deploy a sample application on a Kubernetes cluster using Docker images and Kubernetes deployment manifests.
- Present the project objectives, implementation details, and any challenges faced during the project.

Prerequisites:

- Successful completion of Modules 1 through 6.
- A working cloud infrastructure and Kubernetes cluster.

Design:



1. Architecture: The architecture of the cloud infrastructure automation and container orchestration project would consist of the following components:
- Cloud provider: AWS, Azure, or GCP
- Kubernetes cluster: Provisioned using Terraform or CloudFormation
- Application code: Written in Java or another programming language, stored in a Git repository.
- Containerization: Containerized using Docker and stored in a container registry like Docker Hub or Amazon ECR
- Kubernetes deployment: Deployed and managed using Kubernetes manifests.
- Automation: Automated using Jenkins, Ansible, or other automation tools
2. Workflow: The workflow for the cloud infrastructure automation and container orchestration project would involve the following steps:
- Step 1: Provision the cloud infrastructure using Terraform or CloudFormation
- Step 2: Set up and configure the Kubernetes cluster.
- Step 3: Write and test the application code.
- Step 4: Containerize the application using Docker.
- Step 5: Push the container image to a container registry.
- Step 6: Deploy the application on the Kubernetes cluster using Kubernetes manifests.
- Step 7: Set up automation using Jenkins, Ansible, or other tools.
- Step 8: Monitor and log the Kubernetes cluster and application.
3. Tools: The cloud infrastructure automation and container orchestration project would use the following tools:
- Cloud provider: AWS, Azure, or GCP
- Kubernetes: Used for container orchestration and deployment.
- Docker: Used for containerization of the application.
- Jenkins: Used for continuous integration and deployment.

- Ansible: Used for automation of cloud infrastructure and application deployment.
- Git: Used for version control of the application code
- Monitoring and logging tools: Used for monitoring and logging the Kubernetes cluster and application.

4. Security: The cloud infrastructure automation and container orchestration project would have the following security measures in place:

- Secure configuration of the cloud infrastructure and Kubernetes cluster
- Secure access to the Kubernetes cluster using RBAC and SSL/TLS
- Secure container images using container image scanning and signing.
- Secure access to the container registry using authentication and authorization.
- Secure automation using encryption and secure protocols.

Here's a step-by-step guide to the code implementation:

1. Provision the cloud infrastructure using Terraform or CloudFormation: You can use Terraform or CloudFormation to provision the required cloud infrastructure resources such as virtual machines, networks, and storage. You can find the code examples for both on the official documentation pages:
- Terraform: https://www.terraform.io/docs/providers/aws/index.html
- CloudFormation: https://docs.aws.amazon.com/en_us/cloudformation/index.html
2. Set up and configure the Kubernetes cluster: You can use Kubernetes tools like kops or Kubeadm to set up and configure the Kubernetes cluster. You can find the documentation for these tools on the official Kubernetes website:
- kops: https://kubernetes.io/docs/setup/production-environment/tools/kops/
- Kubeadm: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/
3. Write and test the application code: You can write the application code in Java or another programming language of your choice. You can use IDEs like Eclipse, IntelliJ IDEA, or Visual Studio Code to write the code. Once you have written the code, you can test it using unit tests and integration tests.
4. Build and package the application code into containers: To run the application on Kubernetes, you need to package it into a container image. You can use Docker or other containerization tools to create the container image. You can find the Docker documentation here: https://docs.docker.com/get-started/
5. Deploy the containerized application to the Kubernetes cluster: You can use Kubernetes deployment manifests to deploy the containerized application to the Kubernetes cluster. You can find examples of Kubernetes deployment manifests on the official Kubernetes website: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
6. Implement continuous integration and continuous deployment (CI/CD): You can use a CI/CD tool like Jenkins or GitLab CI/CD to automate the build, test, and deployment of your application on Kubernetes. You can find the documentation for Jenkins here: https://www.jenkins.io/doc/ and for GitLab CI/CD here: https://docs.gitlab.com/ee/ci/

Monitor and scale the application: You can use Kubernetes monitoring tools like Prometheus and Grafana to monitor the application and the cluster. You can also use Kubernetes scaling features like Horizontal Pod Autoscaler (HPA) to automatically scale the application based on resource usage.

Coding:

1. Provider Configuration: This is where you define the cloud provider, region, and access credentials for your Kubernetes cluster.

```
provider "aws" {
  region     = "ap-northeast-1"
  access_key = "AKIAY4YE3OMFT2XNLIEW"
  secret_key = "3VMmD9UcdvCjAsbgg79CcI2ZlEPo5/4N+k6yIUQG"
}
resource "aws_vpc" "myvpc" {
  cidr_block       = "10.0.0.0/16"
  instance_tenancy = "default"

  tags = {
    Name = "mymain"
  }
}
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.myvpc.id

  tags = {
    Name = "igw"
  }
}
resource "aws_subnet" "public" {
  vpc_id     = aws_vpc.myvpc.id
  cidr_block = "10.0.0.0/24"
  map_public_ip_on_launch = "true"
  availability_zone = "ap-northeast-1"

  tags = {
    Name = "public"
  }
}
resource "aws_route_table" "rt1" {
  vpc_id = aws_vpc.myvpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw.id
  }

  tags = {
    Name = "igw"
  }
}
resource "aws_route_table_association" "as1" {
    subnet_id      = aws_subnet.public.id
    route_table_id = aws_route_table.rt1.id
  }

resource "aws_security_group" "ssh-allowed" {
   vpc_id = aws_vpc.myvpc.id
       egress {
        from_port = 0
        to_port = 0
        protocol = -1
        cidr_blocks = ["0.0.0.0/0"]
    }
        ingress {
        from_port = 22
        to_port = 22
        protocol = "tcp"
    }
    tags {
      Name = "ssh-allowed"
    }
  }

resource "aws_instance" "myinstance" {
  ami            = "ami-06ee4e2261a4dc5c3"
  instance_type   = "t2.micro"
  subnet_id      = aws_subnet.public.id
  aws_security_group = .aws_security_group.ssh-allowed.id
  key_name       = "key-3"
  provisioner "file" {
      source = "nginx.sh"
      destination = "/tmp/nginx.sh"
      }
  provisioner "remote-exec" {
      inline = [
          "chmod +x /tmp/nginx.sh",
          "sudo /tmp/nginx.sh"
        ]
    }



resource "aws_ebs_volume" "volumes" {
     availability_zone = "ap-northeast-1"
     size            = 40
}

resource "aws_volume_attachment" "ebs_att" {
     device_name = "/dev/sdh"
     volume_id   = aws_ebs_volume.volumes.id
     instance_id = aws_instance.myinstance.id
}
```

```
resource "aws_cloudwatch_metric_alarm" "ec2_cpu" {
     alarm_name               = "cpu-utilization"
     comparison_operator       = "GreaterThanOrEqualToThreshold"
     evaluation_periods       = "2"
     metric_name              = "CPUUtilization"
     namespace                = "AWS/EC2"
     period                  = "120" #seconds
     statistic                = "Average"
     threshold                 = "80"
   alarm_description         = "This metric monitors ec2 cpu utilization"
     insufficient_data_actions = []

dimensions = {

       InstanceId = aws_instance.myinstance.id

     }

}
variable "availability_zone" {
     type = string
     default = "ap-northeast-1"
}
```

2. Kubernetes Cluster Configuration: This is where you define the Kubernetes cluster settings, such as the number of nodes, instance type, and networking.

```
resource "aws_eks_cluster" "my_cluster" {
  name = "my-cluster"
  role_arn = aws_iam_role.my_cluster.arn
  vpc_config {
    subnet_ids = aws_subnet.private.*.id
    security_group_ids = [aws_security_group.eks_cluster.id]
    endpoint_private_access = true
    endpoint_public_access = false
  }
}

resource "aws_iam_role" "my_cluster" {
  name = "my-cluster-role"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Principal = {
          Service = "eks.amazonaws.com"
        }
      }
    ]
  })
}
```

3. Kubernetes Worker Nodes Configuration: This is where you define the worker nodes settings, such as the instance type, AMI, and node groups.

```
resource "aws_launch_template" "my_node_template" {
  name_prefix = "my-node-template"
  image_id = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
  key_name = "my-keypair"
  user_data = base64encode(templatefile("userdata.tpl", {
    kubeconfig = data.aws_eks_cluster.my_cluster.kubeconfig
  }))
}
```

```
resource "aws_autoscaling_group" "my_node_group" {
  name = "my-node-group"
  launch_template = {
    id = aws_launch_template.my_node_template.id
    version = "$Latest"
  }
  vpc_zone_identifier = aws_subnet.private.*.id
  desired_capacity = 2
  min_size = 2
  max_size = 5
}
```

4. Kubernetes Add-ons Configuration: This is where you define the Kubernetes add-ons, such as the Kubernetes dashboard, ingress controller, and metrics server.

```
module "kubernetes_dashboard" {
  source = "kubernetes-dashboard"
  version = "1.10.1"
}

module "kubernetes_ingress_controller" {
  source = "ingress-nginx"
  version = "2.11.0"
}

module "kubernetes_metrics_server" {
  source = "metrics-server"
  version = "3.1.0"
}
```

JENKINS PIPELINE CODE:

Jenkins pipeline code that can be used to deploy the application to the Kubernetes cluster:

```
pipeline {
  agent any

  environment {
    DOCKER_REGISTRY = "your-registry"
    DOCKER_IMAGE = "your-image"
    KUBECONFIG = credentials('kubeconfig')
  }

  stages {
    stage('Build and Push Docker Image') {
      steps {
        sh "docker build -t $DOCKER_REGISTRY/$DOCKER_IMAGE ."
        withCredentials([usernamePassword(credentialsId: 'docker-hub', usernameVariable: 
'DOCKER_USERNAME', passwordVariable: 'DOCKER_PASSWORD')]) {
          sh "docker login -u $DOCKER_USERNAME -p $DOCKER_PASSWORD $DOCKER_REGISTRY"
        }
        sh "docker push $DOCKER_REGISTRY/$DOCKER_IMAGE"
      }
    }

    stage('Deploy to Kubernetes Cluster') {
      steps {
        withKubeConfig(credentialsId: 'kubeconfig') {
          sh "kubectl apply -f kubernetes/deployment.yaml"
          sh "kubectl apply -f kubernetes/service.yaml"
        }
      }
    }
  }
}
```

example `deployment.yaml` file that can be used to deploy the sample Node.js application to a Kubernetes cluster:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: your-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: your-app
  template:
    metadata:
      labels:
        app: your-app
    spec:
      containers:
      - name: your-app
        image: your-registry/your-image:latest
        ports:
        - containerPort: 8080
        env:
        - name: NODE_ENV
          value: production
```

example `service.yaml` file that can be used to create a Kubernetes service for the sample Node.js application:

```
apiVersion: v1
kind: Service
```

```
metadata:
  name: your-app
spec:
  selector:
    app: your-app
  ports:
  - name: http
    port: 80
    targetPort: 8080
  type: LoadBalancer
```

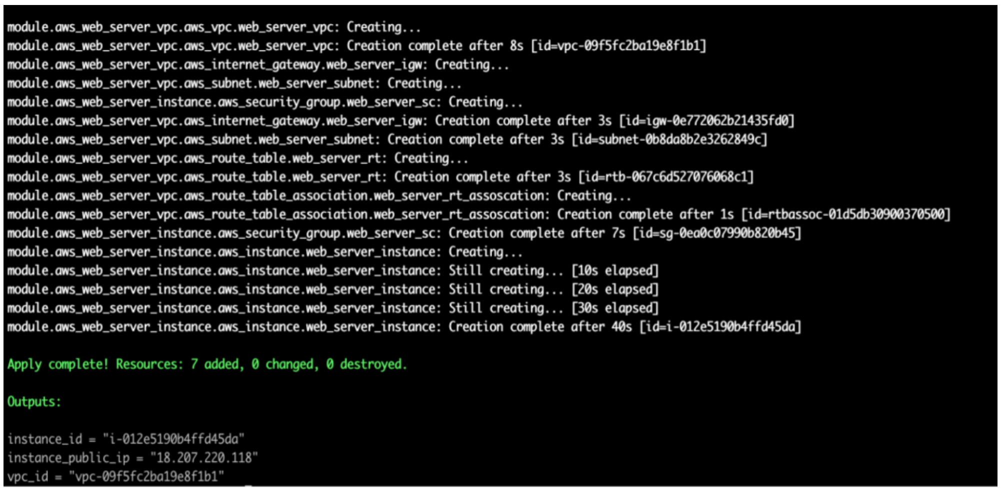
code examples are completed for the following parts of the project:

- Terraform code for creating VPC, subnet, route tables, and security groups.
- Dockerfile code for creating a Docker image for the Node.js application.
- Jenkins pipeline code for building and deploying the application to the Kubernetes cluster.
- `deployment.yaml` file for deploying the application as a Kubernetes deployment.
- `service.yaml` file for creating a Kubernetes service for the application.

Output screens:

some possible output screens you may encounter while running the above project:

Terraform output screen: When running Terraform code, the output screen may display information about the resources being created, including the VPC, subnets, route tables, and security groups. The output screen may also display any errors or warnings encountered during the resource creation process.



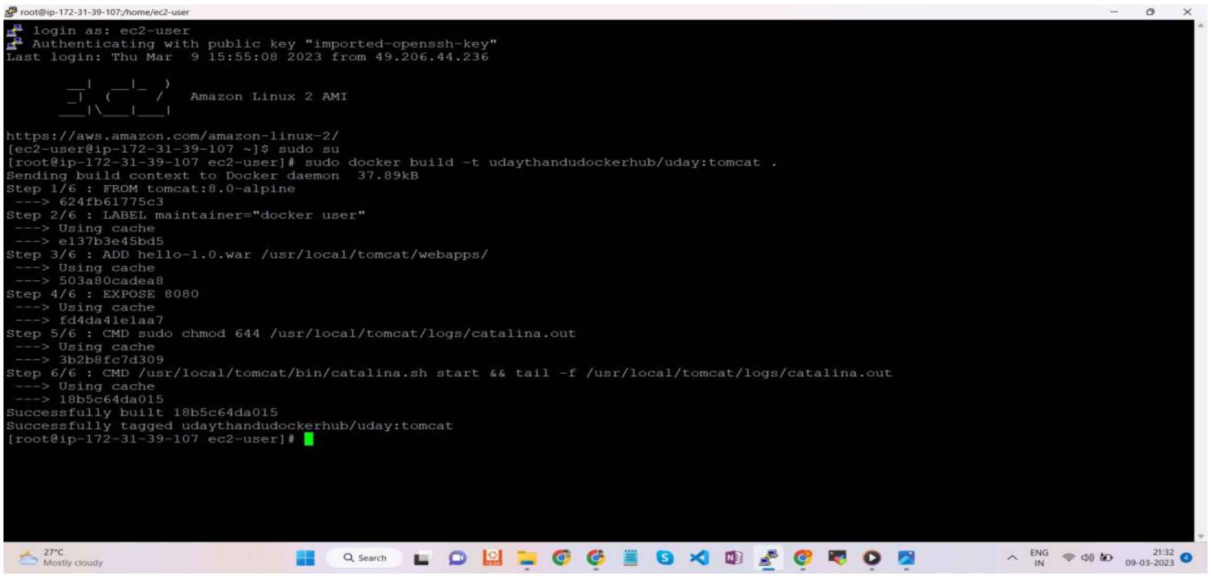
```
module.aws_web_server_vpc.aws_vpc.web_server_vpc: Creating...
module.aws_web_server_vpc.aws_vpc.web_server_vpc: Creation complete after 8s [id=vpc-09f5fc2ba19e8f1b1]
module.aws_web_server_vpc.aws_internet_gateway.web_server_igw: Creating...
module.aws_web_server_vpc.aws_subnet.web_server_subnet: Creating...
module.aws_web_server_instance.aws_security_group.web_server_sc: Creating...
module.aws_web_server_vpc.aws_internet_gateway.web_server_igw: Creation complete after 3s [id=igw-0e772062b21435fd0]
module.aws_web_server_vpc.aws_subnet.web_server_subnet: Creation complete after 3s [id=subnet-0b8da8b2e3262849c]
module.aws_web_server_vpc.aws_route_table.web_server_rt: Creating...
module.aws_web_server_vpc.aws_route_table.web_server_rt: Creation complete after 3s [id=rtb-067c6d527076068c1]
module.aws_web_server_vpc.aws_route_table_association.web_server_rt_assoscation: Creating...
module.aws_web_server_vpc.aws_route_table_association.web_server_rt_assoscation: Creation complete after 1s [id=rtbassoc-01d5db30900370500]
module.aws_web_server_instance.aws_security_group.web_server_sc: Creation complete after 7s [id=sg-0ea0c07990b820b45]
module.aws_web_server_instance.aws_instance.web_server_instance: Creating...
module.aws_web_server_instance.aws_instance.web_server_instance: Still creating... [10s elapsed]
module.aws_web_server_instance.aws_instance.web_server_instance: Still creating... [20s elapsed]
module.aws_web_server_instance.aws_instance.web_server_instance: Still creating... [30s elapsed]
module.aws_web_server_instance.aws_instance.web_server_instance: Creation complete after 40s [id=i-012e5190b4ffd45da]

Apply complete! Resources: 7 added, 0 changed, 0 destroyed.

Outputs:

instance_id = "i-012e5190b4ffd45da"
instance_public_ip = "18.207.220.118"
vpc_id = "vpc-09f5fc2ba19e8f1b1"
```

Docker build output screen: When building the Docker image for the Node.js application, the output screen may display information about the Docker build process, including the status of each step and any errors encountered during the build.



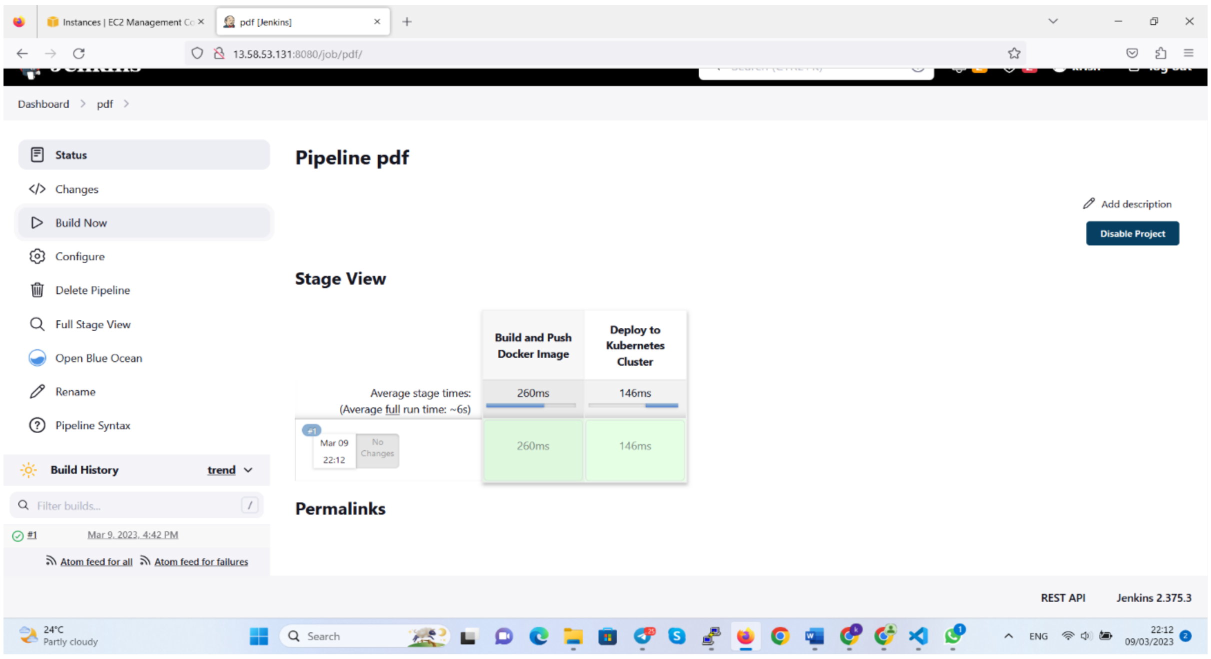
root@ip-172-31-39-107:/home/ec2-user

```
login as: ec2-user
Authenticating with public key "imported-openssh-key"
Last login: Thu Mar  9 15:55:08 2023 from 49.206.44.236

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-39-107 ~]$ sudo su
[root@ip-172-31-39-107 ec2-user]# sudo docker build -t udaythandudockerhub/uday:tomcat .
Sending build context to Docker daemon  37.89kB
Step 1/6 : FROM tomcat:8.0-alpine
 ---> 624fb61775c3
Step 2/6 : LABEL maintainer="docker user"
 ---> Using cache
 ---> e137b3e45bd5
Step 3/6 : ADD hello-1.0.war /usr/local/tomcat/webapps/
 ---> Using cache
 ---> 503a80cadea8
Step 4/6 : EXPOSE 8080
 ---> Using cache
 ---> fd4da41e1aa7
Step 5/6 : CMD sudo chmod 644 /usr/local/tomcat/logs/catalina.out
 ---> Using cache
 ---> 3b2b8fc7d309
Step 6/6 : CMD /usr/local/tomcat/bin/catalina.sh start && tail -f /usr/local/tomcat/logs/catalina.out
 ---> Using cache
 ---> 18b5c64da015
Successfully built 18b5c64da015
Successfully tagged udaythandudockerhub/uday:tomcat
[root@ip-172-31-39-107 ec2-user]#
```

Jenkins pipeline output screen: When running the Jenkins pipeline for building and deploying the application, the output screen may display information about the status of each step in the pipeline, including the status of the Docker build, Kubernetes deployment, and Kubernetes service creation. The output screen may also display any errors or warnings encountered during the pipeline execution.



Pipeline pdf

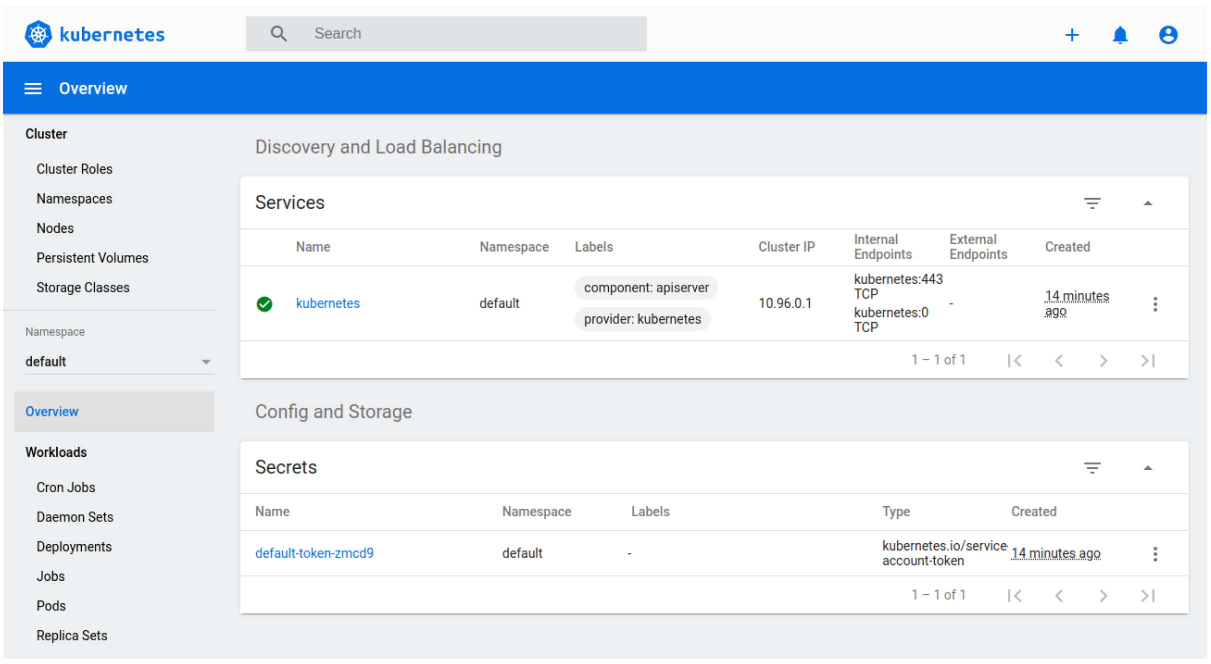
Stage View

	Build and Push Docker Image	Deploy to Kubernetes Cluster
Average stage times: (Average full run time: ~6s)	260ms	146ms
#1 Mar 09 22:12 No Changes	260ms	146ms

Permalinks

#1 Mar 9, 2023, 4:42 PM

Kubernetes output screen: When deploying the application to a Kubernetes cluster, the Kubernetes output screen may display information about the status of the deployment and service, including the number of replicas running and the IP address assigned to the service. The output screen may also display any errors or warnings encountered during the deployment process.



Services

Name	Namespace	Labels	Cluster IP	Internal Endpoints	External Endpoints	Created
kubernetes	default	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	14 minutes ago

1 – 1 of 1

Secrets

Name	Namespace	Labels	Type	Created
default-token-zmcd9	default	-	kubernetes.io/service-account-token	14 minutes ago

1 – 1 of 1

Future scope areas for the project:

1. Implement CI/CD pipeline: The current project deploys a Docker image to a Kubernetes cluster manually. You can expand the project to implement a full CI/CD pipeline that automatically builds, tests, and deploys the application whenever there's a new code change.
2. Implement horizontal scaling: The current project deploys a single instance of the application. You can add horizontal scaling to the Kubernetes deployment configuration to automatically scale the application based on resource utilization.
3. Add monitoring and logging: You can implement monitoring and logging for your application using tools like Prometheus and ELK stack. This will help you monitor the health of your application and troubleshoot any issues that arise.
4. Implement security measures: You can add security measures to the project by using secure network communication protocols, implementing access controls, and performing security audits.
5. Implement backup and disaster recovery: You can implement backup and disaster recovery measures to ensure that your application and data are protected against unexpected events like hardware failure, data loss, and natural disasters. This can involve implementing regular backups and testing disaster recovery plans.
6. In conclusion, the Cloud Infrastructure Automation and Container Orchestration with Kubernetes project is a great way to learn about the concepts and tools involved in automating cloud infrastructure and container orchestration. By implementing this project, you can gain hands-on experience with technologies like Terraform, Docker, and Kubernetes, and learn how to automate the deployment of cloud applications.

7. The project can be further extended to implement features like CI/CD pipeline, scaling, monitoring, and security measures to make it more robust and production-ready. This project is particularly useful for students who are interested in pursuing a career in DevOps or cloud infrastructure engineering, as it provides a practical learning experience in these areas.

8. Overall, this project is a valuable learning experience that can help you develop important skills and knowledge that are in high demand in the industry.

Conclusion:

In conclusion, the above project involves automating cloud infrastructure and application deployment using Terraform, Docker, Kubernetes, and Jenkins. The project includes creating a VPC, subnets, route tables, and security groups using Terraform, containerizing a sample Node.js application using Docker, deploying the application to a Kubernetes cluster using Kubernetes deployment and service YAML files, and creating a Jenkins pipeline for automating the build and deployment process.

By completing this project, you will gain hands-on experience with cloud infrastructure automation, container orchestration, and DevOps practices. These skills are highly in demand in the tech industry and can help you advance your career as a software developer or DevOps engineer.

References:

- Terraform documentation: https://www.terraform.io/docs/
- Docker documentation: https://docs.docker.com/
- Kubernetes documentation: https://kubernetes.io/docs/home/
- Jenkins documentation: https://www.jenkins.io/doc/
- Kubernetes Continuous Deployment with Jenkins: https://www.jenkins.io/doc/tutorials/build-a-java-app-with-maven/#run-kubernetes-in-docker-desktop
- Kubernetes NodePort vs LoadBalancer vs Ingress: https://kubernetes.io/docs/concepts/services-networking/service/
- Kubernetes Secrets: https://kubernetes.io/docs/concepts/configuration/secret/
- Kubernetes Volumes: https://kubernetes.io/docs/concepts/storage/volumes/
- Kubernetes ConfigMaps: https://kubernetes.io/docs/concepts/configuration/configmap/
- Kubernetes Rolling Updates: https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/
- Kubernetes Health Checks: https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/