INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

DevOps Engineering								
V Semester: AERO | ME | CE | EEE | CSE | IT | CSE (AI&ML) | CSE (DS) | CSE (CS)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
		L	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
ACSE28	**Skill**	1	0	2	2	40	60	100
Contact Classes: 32	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: 32**		
Prerequisite: OBJECT ORIENTED PROGRAMMING								

I. COURSE OVERVIEW:
DevOps, a combination of "development" and "operations," is a software development methodology that emphasizes collaboration and communication between software developers and IT operations professionals. The goal of DevOps is to streamline the software delivery process, from code development to deployment and maintenance, by breaking down silos between development and operations teams.

II. COURSES OBJECTIVES:
The students will try to learn

I. The DevOps Concepts for business cases, cloud provisioning and management services .
II. The model canvas for DevOps use cases.
III. The virtual machines and containers for designing of applications
IV. The code with various aspects in continuous deployment / development.

III. COURSE OUTCOMES:
At the end of the course students should be able to:

CO 1 Understands the DevOps concepts in continuous delivery / development of applications.

CO 2 Create the DevOps applications using various tools and technologies.

CO 3 Examine the virtual machines and containers for managing the files

CO 4 Apply cloud services for deployment the applications in a real-time

CO 5 Perform web security and testing the code with appropriate tools

IV. COURSE SYLLABUS:

MODULE 1: DevOps Concepts
Understanding DevOps movement, DevOps with changing time, The water fall model, Agile Model, Collaboration, Why DevOps, Benefits of DevOps, DevOps life cycle- all about continuous, Build Automation, Continuous Integration, Continuous Management, Continuous Delivery / Continuous Development, The agile wheel of wheels.

MODULE 2: DevOps Tools and Technologies
Code Repositories : Git, Differences between SVN and Git, Build tools – Maven, Continuous integration tools – Jenkins, Container Technology – Docker, Monitoring Tools, Continuous integration with Jenkins 2, Creating built-in delivery pipelines, Creating Scripts, Creating a pipeline for compiling and executing test units, Using the Build Pipeline plugin, Integrating the deployment operation

MODULE 3: Docker Containers
Overview of Docker containers, Understanding the difference between virtual machines and containers, Installation and configuration of Docker, Creating your first Docker container, Managing containers, Creating a

Docker image from Docker file, An overview of Docker's elements, Creating a Dockerfile, Writing a Dockerfile, Building and running a container on a local machine, Testing a container locally, Pushing an image to Docker Hub

MODULE 4: Cloud Provisioning and Configuration Management with Chef, Managing Containers Effectively with Kubernetes

Amazon EC2, Creating and configuring a virtual machine in Amazon Web Services, Prerequisite – deploying our application on a remote server, Deploying the application on AWS, Deploying the application in a Docker container.

Kubernetes architecture overview, Installing Kubernetes on a local machine, Installing the Kubernetes dashboard, Kubernetes application deployment, Using Azure Kubernetes Service (AKS), Creating an AKS service, Configuring kubectl for AKS, The build and push of the image in the Docker Hub

MODULE 5: Testing the Code

Manual testing, Unit testing, JUnit in general and JUnit in particular, A JUnit example, Automated integration testing, Docker in automated testing, Performance testing, Automated acceptance testing, Automated GUI testing, Integrating Selenium tests in Jenkins, JavaScript testing, Testing backend integration points, Test-driven development, A complete test automation scenario, Manually testing our web application.

V. TEXT BOOKS:

1. Mitesh Soni, “DevOps for Web Development”, Packt Publishing, 2016.
2. Mikael Krief, “Learning DevOps - The complete guide to accelerate collaboration with Jenkins, Kubernetes, Terraform and Azure DevOps”, Packt Publishing, 2019.

VI. REFERENCE BOOKS:

1. Joakim Verona, “Practical DevOps”, Packt Publishing, 2016.
2. Michael Huttermann, “DevOps for Developers”, A press publishers, 2012.
3. Sanjeev Sharma, “The DevOps Adoption Playbook”, Published by John Wiley & Sons, Inc.2017.
4. Sanjeev Sharma & Bernie Coyne, “DevOps for Dummies”, Published by John Wiley & Sons, Inc.

VII. REFERENCE BOOKS:

1. https://www.geeksforgeeks.org/devops-tutorial/
2. https://www.javatpoint.com/devops
3. https://azure.microsoft.com/en-in/solutions/devops/tutorial
4. https://www.guru99.com/devops-tutorial.html

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal, Hyderabad - 500 043

DevOps Engineering

BT-23 IV SEMESTER

SKILL ENHANCEMENT PROJECT

Contents

1 DevOps - Introduction & Setup **1**

1.1 Resources Needed 1

1.2 Task 1: Overview of DevOps 1

1.2.1 Introduction to DevOps 1

1.2.2 Key Concepts and Practices 1

1.2.3 Benefits of DevOps 2

1.3 Task 2: Setting up a Development Environment 2

1.3.1 Install an IDE 2

1.3.2 Install Git 2

1.3.3 Set up Git 2

1.3.4 Create a GitHub Account (if not already done) 2

1.3.5 Create a New Repository 2

1.3.6 Clone the Repository Locally 2

1.3.7 Set Up Your IDE 3

1.3.8 Create a Simple Script 3

1.3.9 Run the Script 3

1.3.10 Commit and Push Changes 3

1.4 Task 3: Reflection and Discussion 3

1.4.1 Discussion 3

1.4.2 Q&A Session 3

1.5 Deliverables 3

2 Version Control with Git **4**

2.1 Resources Needed 4

2.2 Task 1: Basics of Version Control 4

2.2.1 Introduction to Version Control 4

2.2.2 Introduction to Git 4

2.3 Task 2: Git Commands and Workflows 4

2.3.1 Access your Git account 4

2.3.2 Creating a Local Repository 5

2.3.3 Basic Git Commands 5

2.3.4 Creating and Switching Branches 5

2.3.5 Making Changes and Merging 5

2.3.6 Resolving Merge Conflicts 5

2.4 Task 3: Managing a Remote Repository 6

2.4.1 Creating a Remote Repository 6

2.4.2 Linking the Local Repository to GitHub 6

2.4.3 Cloning a Repository 6

2.5 Task 4: Reflection and Discussion 6

2.5.1 Discussion 6

2.5.2 Q&A Session 6
2.6 Deliverables 6

3 Cloud Providers and Services (AWS, Azure, GCP) 7
3.1 Objective 7
3.2 Resources Needed 7
3.3 Task 1: Overview of Cloud Providers 7
3.3.1 Introduction to Cloud Computing 7
3.3.2 Overview of AWS, Azure, and GCP 7
3.3.3 Comparative Analysis 8
3.4 Task 2: Common Services and Architectures 8
3.4.1 Common Cloud Services 8
3.4.2 Cloud Architectures 8
3.5 Task 3: Lab: Deploying Applications on AWS/Azure/GCP 8
3.5.1 Setting Up the Environment 8
3.5.2 Deploying a Sample Application 8
3.5.3 Testing and Verification 9
3.6 Task 4: Reflection and Discussion 10
3.6.1 Discussion 10
3.6.2 Q&A Session 10
3.7 Deliverables 10

4 Continuous Integration with Jenkins 11
4.1 Objective 11
4.2 Resources Needed 11
4.3 Task 1: Introduction to Continuous Integration 11
4.3.1 Concepts of Continuous Integration 11
4.3.2 Introduction to Jenkins 11
4.4 Task 2: Setting Up Jenkins 11
4.4.1 Installing Jenkins 11
4.4.2 Configuring Jenkins 12
4.5 Task 3: Creating a Jenkins Pipeline 12
4.5.1 Creating a New Pipeline Project 12
4.5.2 Configuring the Pipeline 12
4.5.3 Running the Pipeline 12
4.6 Task 4: Automating the CI Process 13
4.6.1 Setting Up Webhooks 13
4.6.2 Testing the Automation 13
4.7 Task 5: Reflection and Discussion 13
4.7.1 Discussion 13
4.7.2 Q&A Session 13
4.8 Deliverables 13

5 Continuous Delivery and Deployment (CD) 14
5.1 Objective 14
5.2 Resources Needed 14
5.3 Task 1: Introduction to Continuous Delivery 14
5.3.1 Concepts of Continuous Delivery 14
5.3.2 Differences between Continuous Integration (CI) and Continuous Delivery (CD) 14
5.4 Task 2: Deployment Strategies 15
5.4.1 Overview of Deployment Strategies 15

5.4.2 Choosing the Right Strategy 15
5.5 Task 3: Lab: Configuring a CD Pipeline 15
5.5.1 Creating a Jenkins Pipeline for CD 15
5.5.2 Defining the CD Pipeline 15
5.5.3 Implementing Deployment Strategy 16
5.5.4 Running the CD Pipeline 16
5.5.5 Validating Deployment 16
5.6 Part 4: Reflection and Discussion 16
5.6.1 Discussion 16
5.6.2 Q&A Session 16
5.7 Deliverables 16

6 Infrastructure as Code (IaC) 17
6.1 Objective 17
6.2 Resources Needed 17
6.3 Task 1: Overview of Infrastructure as Code (IaC) 17
6.3.1 Introduction to IaC 17
6.3.2 IaC Tools 17
6.4 Task 2: Tools (Terraform, Ansible, CloudFormation) 17
6.4.1 Terraform 17
6.4.2 Ansible 18
6.4.3 CloudFormation 18
6.5 Task 3: Lab: Writing and Applying Terraform Scripts 18
6.5.1 Setting Up Terraform 18
6.5.2 Writing a Terraform Script 18
6.5.3 Initializing Terraform 18
6.5.4 Verifying the Provisioned Infrastructure 18
6.5.5 Cleaning Up 19
6.6 Task 4: Reflection and Discussion 19
6.6.1 Discussion 19
6.6.2 Q&A Session 19
6.7 Deliverables 19

7 Containerization with Docker 20
7.1 Objective 20
7.2 Resources Needed 20
7.3 Task 1: Introduction to Docker 20
7.3.1 Overview of Containerization 20
7.3.2 Introduction to Docker 20
7.4 Task 2: Docker Commands and Concepts 21
7.4.1 Basic Docker Commands 21
7.4.2 Key Docker Concepts 21
7.5 Task 3: Lab: Building and Running Docker Containers 21
7.5.1 Setting Up Docker 21
7.5.2 Writing a Dockerfile 21
7.5.3 Building the Docker Image 22
7.5.4 Running the Docker Container 22
7.5.5 Interacting with the Container 22
7.5.6 Pushing the Image to Docker Hub (Optional) 22
7.6 Task 4: Reflection and Discussion 22
7.6.1 Discussion 22
7.6.2 Q&A Session 22

7.7 Deliverables 22

8 Container Orchestration with Kubernetes **23**
8.1 Objective 23
8.2 Resources Needed 23
8.3 Task 1: Introduction to Kubernetes 23
8.4 Task 2: Key Concepts and Components 24
8.4.1 Kubernetes Objects 24
8.4.2 Kubernetes Commands 24
8.5 Task 3: Lab: Deploying Applications on a Kubernetes Cluster 24
8.5.1 Setting Up the Kubernetes Cluster 24
8.5.2 Creating a Deployment 24
8.5.3 Applying the Deployment 25
8.5.4 Exposing the Deployment 25
8.5.5 Verifying the Deployment 25
8.5.6 Scaling the Deployment 25
8.5.7 Cleaning Up 25
8.6 Task 4: Reflection and Discussion 25
8.6.1 Discussion 25
8.6.2 Q&A Session 25
8.7 Deliverables 25

9 Configuration Management(Ansible, Puppet, Chef) **26**
9.1 Objective 26
9.2 Resources Needed 26
9.3 Task 1: Introduction to Configuration Management 26
9.3.1 Overview of Configuration Management 26
9.3.2 Benefits of Configuration Management 26
9.4 Task 2: Tools for Configuration Management 27
9.4.1 Ansible 27
9.4.2 Puppet 27
9.4.3 Chef 27
9.5 Task 3: Lab: Writing Ansible Playbooks 27
9.5.1 Setting Up Ansible 27
9.5.2 Creating an Inventory File 27
9.5.3 Writing a Simple Playbook 27
9.5.4 Running the Playbook 28
9.5.5 Verifying the Setup 28
9.5.6 Modifying the Playbook 28
9.6 Task 4: Reflection and Discussion 28
9.6.1 Discussion 28
9.6.2 Q&A Session 29
9.7 Deliverables 29

10 Monitoring and Logging(Prometheus, Grafana, ELK Stack) **30**
10.1 Objective 30
10.2 Resources Needed 30
10.3 Task 1: Introduction to Monitoring and Logging 30
10.3.1 Overview of Monitoring and Logging 30
10.3.2 Benefits of Monitoring and Logging 30
10.4 Task 2: Tools for Monitoring and Logging 31

10.4.1 Prometheus 31
10.4.2 Grafana 31
10.4.3 ELK Stack 31
10.5 Task 3: Lab: Setting Up Monitoring and Logging for Applications 31
10.5.1 Setting Up Prometheus 31
10.5.2 Setting Up Grafana 31
10.5.3 Setting Up ELK Stack 32
10.5.4 Configuring Kibana 32
10.5.5 Testing and Verification 32
10.6 Task 4: Reflection and Discussion 32
10.6.1 Discussion 32
10.6.2 Q&A Session 33
10.7 Deliverables 33

11 Security in DevOps (Aqua, Clair, Snyk) 34
11.1 Objective 34
11.2 Task 1: Introduction to DevSecOps 34
11.2.1 Overview of DevSecOps 34
11.2.2 Key Principles of DevSecOps 34
11.3 Task 2: Best Practices and Tools for DevSecOps 35
11.3.1 Best Practices 35
11.3.2 Security Tools 35
11.4 Task 3: Lab: Implementing Security Checks in CI/CD Pipelines 35
11.4.1 Setting Up the Environment 35
11.4.2 Integrating Security Checks 35
11.4.3 Testing and Verification 36
11.5 Task 4: Reflection and Discussion 37
11.5.1 Discussion 37
11.5.2 Q&A Session 37
11.6 Deliverables 37

12 Automation and Scripting (Bash, Python) 38
12.1 Objective 38
12.2 Resources Needed 38
12.3 Task 1: Importance of Automation in DevOps 38
12.3.1 Introduction to Automation 38
12.3.2 Automation Use Cases 38
12.3.3 DevOps Automation Tools 38
12.4 Task 2: Scripting Languages (Bash, Python) 39
12.4.1 Introduction to Bash 39
12.4.2 Introduction to Python 39
12.4.3 Comparative Analysis 39
12.5 Task 3: Lab: Automating Tasks with Python Scripts 39
12.5.1 Setting Up the Environment 39
12.5.2 Automating a Sample Task 39
12.5.3 Testing and Verification 40
12.6 Task 4: Reflection and Discussion 40
12.6.1 Discussion 40
12.6.2 Q&A Session 40
12.7 Deliverables 40

13 Advanced CI/CD Techniques 41

13.1 Objective 41
13.2 Resources Needed 41
13.3 Task 1: Advanced Pipeline Configurations 41
13.3.1 Introduction to Advanced Pipelines 41
13.3.2 Pipeline Configuration Tools 41
13.4 Task 2: Blue-Green Deployments, Canary Releases 41
13.4.1 Introduction to Deployment Strategies 41
13.4.2 Blue-Green Deployments 42
13.4.3 Canary Releases 42
13.5 Task 3: Lab: Implementing Advanced Deployment Strategies 42
13.5.1 Setting Up the Environment 42
13.5.2 Implementing Canary Releases 42
13.5.3 Testing and Verification 42
13.6 Task 4: Reflection and Discussion 42
13.6.1 Discussion 42
13.6.2 Q&A Session 43
13.7 Deliverables 43

14 DevOps Culture and Practices **44**
14.1 Objective 44
14.2 Resources Needed 44
14.3 Task 1: Building a DevOps Culture 44
14.3.1 Introduction to DevOps Culture 44
14.3.2 Key Elements of a DevOps Culture 44
14.4 Task 2: Collaboration and Communication Tools 44
14.4.1 Introduction to Collaboration Tools 44
14.4.2 Using Slack for Communication 45
14.4.3 Using Jira for Project Management 45
14.5 Task 3: Lab: Creating a DevOps Project Plan and Collaboration Setup 45
14.5.1 Setting Up the Environment 45
14.5.2 Creating a DevOps Project Plan 45
14.5.3 Setting Up Collaboration Tools 45
14.5.4 Collaborating on the Project 45
14.5.5 Testing and Verification 45
14.6 Part 4: Reflection and Discussion 46
14.6.1 Discussion 46
14.6.2 Q&A Session 46
14.7 Deliverables 46

DevOps - Introduction & Setup

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

The objective of this lab exercise is to introduce students to the fundamentals of DevOps and set up a basic development environment. By the end of the lab, students will have a working knowledge of DevOps principles and practical experience with essential tools like Git and Visual Studio Code.

1.1 Resources Needed

- Computer with internet access
- Access to a cloud provider (optional)

1.2 Task 1: Overview of DevOps

1.2.1 Introduction to DevOps

- Discuss the history and evolution of software development methodologies.
- Explain the need for DevOps in modern software development.
- Define DevOps and its core principles: collaboration, automation, continuous integration, continuous delivery, and continuous monitoring.

1.2.2 Key Concepts and Practices

- Discuss key DevOps practices such as:
  - Continuous Integration (CI)
  - Continuous Delivery/Deployment (CD)
  - Infrastructure as Code (IaC)
  - Monitoring and Logging
  - Microservices
  - Communication and Collaboration Tools

1.2.3 Benefits of DevOps

- Explain the benefits of adopting DevOps, including:
  - Faster delivery of features
  - Improved collaboration between teams
  - Increased efficiency and reduced costs
  - Enhanced reliability and scalability

1.3 Task 2: Setting up a Development Environment

1.3.1 Install an IDE

- Download and install Visual Studio Code from `https://code.visualstudio.com/`.

1.3.2 Install Git

- Download and install Git from `https://git-scm.com/`.

1.3.3 Set up Git

- Open your terminal (or Git Bash on Windows).
- Configure your Git username and email:

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

- Verify the installation:

```
git --version
```

1.3.4 Create a GitHub Account (if not already done)

- Go to `https://github.com/` and sign up for a free account.

1.3.5 Create a New Repository

- Go to GitHub and create a new repository called `devops-intro`.

1.3.6 Clone the Repository Locally

- In your terminal, navigate to the directory where you want to store your project:

```
cd path/to/your/projects
```

- Clone your repository:

```
git clone https://github.com/your-username/devops-intro.git
```

- Navigate into the repository:

```
cd devops-intro
```

1.3.7 Set Up Your IDE

- Open the cloned repository in Visual Studio Code:

```
code .
```

1.3.8 Create a Simple Script

- In Visual Studio Code, create a new file called `hello_world.py`.
- Write a simple Python script to print "Hello, DevOps!":

```
print("Hello, DevOps!")
```

1.3.9 Run the Script

- In the terminal within Visual Studio Code, run the script:

```
python hello_world.py
```

- You should see the output: `Hello, DevOps!`

1.3.10 Commit and Push Changes

- Stage the new file:

```
git add hello_world.py
```

- Commit the changes:

```
git commit -m "Add hello_world.py script"
```

- Push the changes to GitHub:

```
git push origin main
```

1.4 Task 3: Reflection and Discussion

1.4.1 Discussion

- Reflect on the setup process and discuss any challenges faced.
- Discuss how this basic setup is a foundational step in the DevOps journey.

1.4.2 Q&A Session

- Open the floor for questions about the DevOps concepts covered and the lab setup.

1.5 Deliverables

- A basic Python script (`hello_world.py`) committed to a GitHub repository.
- A configured development environment ready for subsequent labs.

Version Control with Git

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

Objective

- Understand the basics of version control.
- Learn Git commands and workflows.
- Create and manage a Git repository.

2.1 Resources Needed

- Computer with internet access
- Git installed
- GitHub account
- IDE (e.g., Visual Studio Code)

2.2 Task 1: Basics of Version Control

2.2.1 Introduction to Version Control

- Explain the importance of version control in software development.
- Discuss the differences between centralized and distributed version control systems.

2.2.2 Introduction to Git

- Provide an overview of Git and its advantages.
- Discuss the basic concepts of Git: repository, commit, branch, merge, etc.

2.3 Task 2: Git Commands and Workflows

2.3.1 Access your Git account

- Ensure Git is installed on your system.
- Login Git with your credentials

2.3.2 Creating a Local Repository

- Create a new directory for your project:

```
mkdir my_git_project
cd my_git_project
```

- Initialize a new Git repository:

```
git init
```

2.3.3 Basic Git Commands

- Create a new file called `README.md` and add some content to it.
- Add the file to the staging area:

```
git add README.md
```

- Commit the changes:

```
git commit -m "Initial commit with README"
```

2.3.4 Creating and Switching Branches

- Create a new branch called `feature-branch`:

```
git branch feature-branch
```

- Switch to the new branch:

```
git checkout feature-branch
```

2.3.5 Making Changes and Merging

- Make some changes to the `README.md` file.
- Add and commit the changes:

```
git add README.md
git commit -m "Update README in feature-branch"
```

- Switch back to the `main` branch:

```
git checkout main
```

- Merge the `feature-branch` into `main`:

```
git merge feature-branch
```

2.3.6 Resolving Merge Conflicts

- Create a conflict by editing the same line in `README.md` on both `main` and `feature-branch`.
- Attempt to merge `feature-branch` into `main` again and resolve the conflict manually.

2.4 Task 3: Managing a Remote Repository

2.4.1 Creating a Remote Repository

- Go to GitHub and create a new repository called `my_git_project`.

2.4.2 Linking the Local Repository to GitHub

- Add the remote repository:

```
git remote add origin https://github.com/your-username/
    my_git_project.git
```

- Push the local commits to the remote repository:

```
git push -u origin main
```

2.4.3 Cloning a Repository

- Clone the remote repository to a new directory:

```
git clone https://github.com/your-username/my_git_project.git
```

2.5 Task 4: Reflection and Discussion

2.5.1 Discussion

- Reflect on the version control process and discuss any challenges faced.
- Discuss how version control is crucial for collaborative development.

2.5.2 Q&A Session

- Open the floor for questions about Git commands, workflows, and concepts.

2.6 Deliverables

- A Git repository with multiple branches and commits.
- A merged `main` branch with resolved conflicts.

Cloud Providers and Services (AWS, Azure, GCP)

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

3.1 Objective

- Gain an understanding of the major cloud providers: AWS, Azure, and Google Cloud Platform (GCP).
- Learn about common services and architectures provided by these cloud platforms.
- Deploy applications on AWS, Azure, or GCP to gain hands-on experience with cloud deployments.

3.2 Resources Needed

- Computer with internet access
- Accounts on AWS, Azure, and/or GCP
- Sample application code (e.g., a simple web application)

3.3 Task 1: Overview of Cloud Providers

3.3.1 Introduction to Cloud Computing

- Explain the basic concepts of cloud computing and its benefits.
- Discuss the different types of cloud service models: IaaS, PaaS, and SaaS.

3.3.2 Overview of AWS, Azure, and GCP

- Provide a brief history and market position of each cloud provider.
- Highlight the key services offered by each provider:
  - **AWS:** EC2, S3, RDS, Lambda, etc.
  - **Azure:** Virtual Machines, Blob Storage, SQL Database, Functions, etc.
  - **GCP:** Compute Engine, Cloud Storage, Cloud SQL, Cloud Functions, etc.

3.3.3 Comparative Analysis

- Discuss the strengths and weaknesses of each cloud provider.
- Provide examples of use cases where one provider may be preferred over the others.

3.4 Task 2: Common Services and Architectures

3.4.1 Common Cloud Services

- Compute: Virtual Machines, Containers, Serverless Computing
- Storage: Object Storage, File Storage, Block Storage
- Databases: Relational, NoSQL, Data Warehousing
- Networking: VPC, Load Balancers, CDN
- Security: IAM, Encryption, Security Groups

3.4.2 Cloud Architectures

- Discuss the concept of scalable and resilient architectures.
- Explain the importance of designing for high availability and disaster recovery.
- Provide examples of common architecture patterns, such as microservices and serverless architectures.

3.5 Task 3: Lab: Deploying Applications on AWS/Azure/GCP

3.5.1 Setting Up the Environment

- Ensure you have accounts set up on AWS, Azure, and/or GCP.
- Install any necessary CLI tools for managing cloud resources (e.g., AWS CLI, Azure CLI, gcloud).

3.5.2 Deploying a Sample Application

- Choose a simple web application to deploy (e.g., a static website or a basic web server).
- **Deploying on AWS:**
    - Create an EC2 instance and configure security groups.
    - Deploy the application code to the EC2 instance.
    - Configure an S3 bucket for static file hosting.
    - Set up an RDS instance for database services if needed.
    - Example deployment script:

```
# AWS CLI commands to deploy a web application on EC2
aws ec2 run-instances --image-id ami-0abcdef1234567890 --count 1 --
    instance-type t2.micro --key-name MyKeyPair --security-groups
    my-sg
# Deploy application code (assumes you have an SSH key set up)
scp -i MyKeyPair.pem app_code.zip ec2-user@ec2-instance-public-dns
    :/home/ec2-user/
ssh -i MyKeyPair.pem ec2-user@ec2-instance-public-dns 'unzip
    app_code.zip && ./deploy.sh'
```

- **Deploying on Azure:**
  - Create a Virtual Machine and configure network security groups.
  - Deploy the application code to the Virtual Machine.
  - Configure Azure Blob Storage for static file hosting.
  - Set up an Azure SQL Database for database services if needed.
  - Example deployment script:

```
# Azure CLI commands to deploy a web application on a VM
az vm create --resource-group myResourceGroup --name myVM --image
    UbuntuLTS --admin-username azureuser --generate-ssh-keys
# Deploy application code (assumes you have an SSH key set up)
scp -i ~/.ssh/id_rsa app_code.zip azureuser@myVM.westus.cloudapp.
    azure.com:/home/azureuser/
ssh -i ~/.ssh/id_rsa azureuser@myVM.westus.cloudapp.azure.com '
    unzip app_code.zip && ./deploy.sh'
```

- **Deploying on GCP:**
  - Create a Compute Engine instance and configure firewall rules.
  - Deploy the application code to the Compute Engine instance.
  - Configure a Cloud Storage bucket for static file hosting.
  - Set up a Cloud SQL instance for database services if needed.
  - Example deployment script:

```
# gcloud CLI commands to deploy a web application on Compute Engine
gcloud compute instances create my-instance --zone=us-central1-a --
    machine-type=e2-medium --image-family=debian-10 --image-project
    =debian-cloud
# Deploy application code (assumes you have an SSH key set up)
scp -i ~/.ssh/google_compute_engine app_code.zip my-instance:~/
    app_code.zip
gcloud compute ssh my-instance --zone=us-central1-a --command='
    unzip ~/app_code.zip && ./deploy.sh'
```

3.5.3 Testing and Verification

- Access the deployed application via the public IP address or DNS name provided by the cloud provider.
- Ensure the application is functioning as expected.
- Review the deployment process and discuss any challenges faced.

3.6 Task 4: Reflection and Discussion

3.6.1 Discussion

- Reflect on the experience of deploying applications on different cloud platforms.
- Discuss the differences in the deployment process for AWS, Azure, and GCP.
- Consider the implications of choosing one cloud provider over another for a given application.

3.6.2 Q&A Session

- Open the floor for questions about cloud providers, services, and deployment strategies.

3.7 Deliverables

- Deployment scripts for AWS, Azure, and GCP.
- Documentation of the deployment process and any issues encountered.
- Screenshots or URLs demonstrating the successfully deployed application.

Continuous Integration with Jenkins

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

4.1 Objective

- Understand the basics of Continuous Integration (CI).
- Set up Jenkins and create a simple CI pipeline.
- Automate the build and test process using Jenkins.

4.2 Resources Needed

- Computer with internet access
- Jenkins software
- A GitHub repository with a sample project

4.3 Task 1: Introduction to Continuous Integration

4.3.1 Concepts of Continuous Integration

- Explain the principles and benefits of Continuous Integration.
- Discuss the importance of CI in modern software development.

4.3.2 Introduction to Jenkins

- Provide an overview of Jenkins as a CI tool.
- Discuss the architecture and components of Jenkins.

4.4 Task 2: Setting Up Jenkins

4.4.1 Installing Jenkins

- Follow the instructions to install Jenkins on your system:
  - For Windows, use the Jenkins MSI installer.

  - For Linux, use the apt or yum package manager.

- Start Jenkins and complete the initial setup.

4.4.2 Configuring Jenkins

- Access Jenkins through a web browser (usually at `http://localhost:8080`).
- Install suggested plugins during the setup process.
- Create the first admin user and configure the basic settings.

4.5 Task 3: Creating a Jenkins Pipeline

4.5.1 Creating a New Pipeline Project

- Create a new item in Jenkins and select ”Pipeline”.
- Name your pipeline project and click OK.

4.5.2 Configuring the Pipeline

- In the pipeline configuration, select ”Pipeline script” and define your pipeline in the script section:

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/your-username/sample-
                    project.git'
            }
        }
        stage('Build') {
            steps {
                sh 'make'
            }
        }
        stage('Test') {
            steps {
                sh 'make test'
            }
        }
    }
}
```

4.5.3 Running the Pipeline

- Save the pipeline configuration.
- Click ”Build Now” to run the pipeline.
- Monitor the build process and review the console output for each stage.

4.6 Task 4: Automating the CI Process

4.6.1 Setting Up Webhooks

- Go to your GitHub repository and navigate to ”Settings” -> ”Webhooks”.
- Add a new webhook with the following details:
  - Payload URL: `http://your-jenkins-url/github-webhook/`
  - Content type: `application/json`
  - Select ”Just the push event” for the event trigger.
- Save the webhook.

4.6.2 Testing the Automation

- Make a change in your GitHub repository and push it.
- Jenkins should automatically trigger the pipeline based on the webhook.

4.7 Task 5: Reflection and Discussion

4.7.1 Discussion

- Reflect on the CI process and discuss any challenges faced.
- Discuss the benefits and potential issues of automated CI pipelines.

4.7.2 Q&A Session

- Open the floor for questions about Jenkins, CI pipelines, and automation.

4.8 Deliverables

- A Jenkins pipeline that automatically builds and tests the sample project.
- Evidence of successful builds and tests triggered by GitHub webhooks.

Continuous Delivery and Deployment (CD)

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

5.1 Objective

- Understand the principles of Continuous Delivery (CD) and its significance in the software development lifecycle.
- Learn different deployment strategies for delivering applications to production.
- Configure and implement a Continuous Delivery pipeline using Jenkins.

5.2 Resources Needed

- Computer with internet access
- Jenkins installed
- A sample application for deployment

5.3 Task 1: Introduction to Continuous Delivery

5.3.1 Concepts of Continuous Delivery

- Explain the concept of Continuous Delivery and its role in modern software development.
- Discuss the benefits of CD, including faster delivery of features, improved quality, and reduced risk.

5.3.2 Differences between Continuous Integration (CI) and Continuous Delivery (CD)

- Highlight the differences between CI and CD.
- Discuss how CD extends CI by ensuring that code is always in a deployable state.

5.4 Task 2: Deployment Strategies

5.4.1 Overview of Deployment Strategies

- Introduce various deployment strategies such as Rolling Deployment, Blue-Green Deployment, Canary Deployment, and A/B Testing.
- Discuss the pros and cons of each strategy.

5.4.2 Choosing the Right Strategy

- Discuss factors to consider when choosing a deployment strategy, such as application architecture, user base, and risk tolerance.
- Provide examples of scenarios where each strategy would be appropriate.

5.5 Task 3: Lab: Configuring a CD Pipeline

5.5.1 Creating a Jenkins Pipeline for CD

- Go to the Jenkins dashboard and click on `New Item`.
- Enter a name for your pipeline job and select `Pipeline`, then click `OK`.

5.5.2 Defining the CD Pipeline

- In the pipeline job configuration page, scroll down to the `Pipeline` section.
- Choose `Pipeline script` and write your pipeline script using the Jenkins Pipeline DSL. Example:

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/your-repo/sample-application.git'
            }
        }
        stage('Build') {
            steps {
                sh './gradlew build'
            }
        }
        stage('Test') {
            steps {
                sh './gradlew test'
            }
        }
        stage('Deploy') {
            steps {
                sh './deploy.sh'
            }
        }
    }
}
```

5.5.3 Implementing Deployment Strategy

- Modify the `Deploy` stage to implement one of the discussed deployment strategies.
- Example: For Blue-Green Deployment, add steps to deploy to a new environment and switch traffic.

5.5.4 Running the CD Pipeline

- Save the pipeline configuration.
- Trigger the pipeline by clicking `Build Now` on the pipeline job page.
- Monitor the stages of the pipeline and check the output for success or errors.

5.5.5 Validating Deployment

- Verify that the application is correctly deployed to the target environment.
- Test the application to ensure it functions as expected.

5.6 Part 4: Reflection and Discussion

5.6.1 Discussion

- Reflect on the CD process and discuss any challenges faced.
- Discuss the benefits and potential issues of automated CD with Jenkins.

5.6.2 Q&A Session

- Open the floor for questions about deployment strategies, CD pipelines, and automation.

5.7 Deliverables

- Jenkins pipeline script for the CD pipeline.
- Evidence of successful deployment and application functionality.

Infrastructure as Code (IaC)

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

6.1 Objective

- Understand the concept of Infrastructure as Code (IaC) and its benefits in managing infrastructure.
- Explore various IaC tools such as Terraform, Ansible, and CloudFormation.
- Write and apply Terraform scripts to provision infrastructure.

6.2 Resources Needed

- Computer with internet access
- Terraform installed
- AWS account (or any other cloud provider)

6.3 Task 1: Overview of Infrastructure as Code (IaC)

6.3.1 Introduction to IaC

- Explain the concept of Infrastructure as Code and its role in modern DevOps practices.
- Discuss the benefits of IaC, including consistency, repeatability, and automation.

6.3.2 IaC Tools

- Provide an overview of popular IaC tools such as Terraform, Ansible, and CloudFormation.
- Discuss the strengths and use cases of each tool.

6.4 Task 2: Tools (Terraform, Ansible, CloudFormation)

6.4.1 Terraform

- Introduce Terraform and its declarative approach to provisioning infrastructure.
- Explain how Terraform uses configuration files to describe the desired state of infrastructure.

6.4.2 Ansible

- Introduce Ansible and its procedural approach to configuration management and application deployment.
- Discuss how Ansible uses playbooks written in YAML to automate tasks.

6.4.3 CloudFormation

- Introduce AWS CloudFormation and its use in modeling and setting up AWS resources.
- Explain how CloudFormation templates define the resources and their dependencies.

6.5 Task 3: Lab: Writing and Applying Terraform Scripts

6.5.1 Setting Up Terraform

- Install Terraform on your computer following the instructions on the Terraform website.
- Configure your environment to use Terraform with your cloud provider (e.g., AWS).

6.5.2 Writing a Terraform Script

- Create a new directory for your Terraform scripts.
- Write a Terraform script to configure a simple infrastructure, such as setting up an EC2 instance in AWS. Example:

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

resource "aws_instance" "example" {
    ami           = "ami-0c55b159cbfafe1f0"
    instance_type = "t2.micro"

    tags = {
        Name = "example-instance"
    }
}
```

6.5.3 Initializing Terraform

- Navigate to your Terraform script directory in the terminal.
- Initialize Terraform with the command `terraform init`.
- Apply the Terraform script with the command `terraform apply`.

6.5.4 Verifying the Provisioned Infrastructure

- Log in to your cloud provider's console and verify that the infrastructure has been provisioned as defined in the Terraform script.

6.5.5 Cleaning Up

- Modify your Terraform script to destroy the infrastructure and run it again with `terraform destroy` to clean up resources.

6.6 Task 4: Reflection and Discussion

6.6.1 Discussion

- Reflect on the process of writing and applying Terraform scripts.
- Discuss the benefits and challenges of using IaC for infrastructure management.

6.6.2 Q&A Session

- Open the floor for questions about Terraform, IaC tools, and automation.

6.7 Deliverables

- Terraform scripts used to configure infrastructure.
- Evidence of successful infrastructure provisioning and cleanup.

Containerization with Docker

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

7.1 Objective

- Understand the basic concepts of Docker and its role in containerization.
- Learn fundamental Docker commands and concepts.
- Build and run Docker containers to manage and deploy applications efficiently.

7.2 Resources Needed

- Computer with internet access
- Docker installed
- Docker Hub account (optional)

7.3 Task 1: Introduction to Docker

7.3.1 Overview of Containerization

- Explain the concept of containerization and its benefits over traditional virtualization.
- Discuss how containers provide isolation, portability, and efficient resource utilization.

7.3.2 Introduction to Docker

- Provide a brief history of Docker and its development.
- Explain the architecture of Docker, including Docker Engine, Docker Images, Docker Containers, and Docker Hub.

7.4 Task 2: Docker Commands and Concepts

7.4.1 Basic Docker Commands

- `docker --version`: Verify Docker installation.
- `docker run`: Run a container from a Docker image.
- `docker ps`: List running containers.
- `docker stop`: Stop a running container.
- `docker rm`: Remove a container.
- `docker rmi`: Remove a Docker image.
- `docker pull`: Pull an image from Docker Hub.
- `docker build`: Build a Docker image from a Dockerfile.

7.4.2 Key Docker Concepts

- **Docker Images**: Pre-configured environments containing application code and dependencies.
- **Docker Containers**: Instances of Docker images running as isolated processes.
- **Dockerfile**: A script that defines the steps to build a Docker image.
- **Docker Hub**: A repository for sharing Docker images.

7.5 Task 3: Lab: Building and Running Docker Containers

7.5.1 Setting Up Docker

- Install Docker on your computer following the instructions on the Docker website.
- Verify the installation by running `docker --version`.

7.5.2 Writing a Dockerfile

- Create a new directory for your Docker project.
- Write a `Dockerfile` to define the steps for building your Docker image. Example:

```
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt
```

```
# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

7.5.3 Building the Docker Image

- Navigate to your project directory in the terminal.
- Run the command `docker build -t my-python-app .` to build the Docker image.

7.5.4 Running the Docker Container

- Run the command `docker run -p 4000:80 my-python-app` to start a container from the image and map port 80 in the container to port 4000 on your host.

7.5.5 Interacting with the Container

- Open a web browser and navigate to `http://localhost:4000` to interact with the application running inside the container.
- Use other Docker commands like `docker ps`, `docker stop`, and `docker rm` to manage the container.

7.5.6 Pushing the Image to Docker Hub (Optional)

- Log in to Docker Hub using `docker login`.
- Tag your image for Docker Hub: `docker tag my-python-app username/my-python-app`.
- Push your image to Docker Hub: `docker push username/my-python-app`.

7.6 Task 4: Reflection and Discussion

7.6.1 Discussion

- Reflect on the process of building and running Docker containers.
- Discuss the advantages and challenges of using Docker for containerization.

7.6.2 Q&A Session

- Open the floor for questions about Docker, containerization, and related concepts.

7.7 Deliverables

- `Dockerfile` used to build the Docker image.
- Evidence of successful Docker image build and container run.

Container Orchestration with Kubernetes

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

8.1 Objective

- Understand the fundamental concepts and architecture of Kubernetes.
- Learn about key Kubernetes components and their roles in container orchestration.
- Deploy applications on a Kubernetes cluster.

8.2 Resources Needed

- Computer with internet access
- Kubernetes cluster (Minikube, MicroK8s, or a cloud provider)
- Kubernetes command-line tool (kubectl) installed

8.3 Task 1: Introduction to Kubernetes

Overview of Kubernetes

- Explain the origin and development of Kubernetes by Google.
- Discuss the importance of container orchestration in managing large-scale applications.

Kubernetes Architecture

- Describe the architecture of Kubernetes, including the control plane and node components.
- Discuss the roles of the Kubernetes API server, etcd, scheduler, controller manager, kubelet, kube-proxy, and container runtime.

8.4 Task 2: Key Concepts and Components

8.4.1 Kubernetes Objects

- **Pods**: The smallest deployable units in Kubernetes, representing a single instance of a running process.
- **ReplicaSets**: Ensure a specified number of pod replicas are running at any given time.
- **Deployments**: Provide declarative updates to applications.
- **Services**: Enable network access to a set of pods.
- **Namespaces**: Provide a mechanism for isolating groups of resources within a single cluster.

8.4.2 Kubernetes Commands

- `kubectl version`: Verify Kubernetes CLI installation.
- `kubectl cluster-info`: Display information about the Kubernetes cluster.
- `kubectl get nodes`: List all nodes in the cluster.
- `kubectl create -f <filename>`: Create resources from a file.
- `kubectl get pods`: List all pods in the cluster.
- `kubectl delete -f <filename>`: Delete resources defined in a file.

8.5 Task 3: Lab: Deploying Applications on a Kubernetes Cluster

8.5.1 Setting Up the Kubernetes Cluster

- Install Minikube or MicroK8s on your computer, or set up a cluster on a cloud provider.
- Verify the cluster setup using `kubectl cluster-info` and `kubectl get nodes`.

8.5.2 Creating a Deployment

- Write a deployment YAML file to define your application deployment. Example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: nginx-deployment
spec:
    replicas: 3
    selector:
        matchLabels:
            app: nginx
    template:
        metadata:
            labels:
                app: nginx
```

```
spec:
    containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

8.5.3 Applying the Deployment

- Use the command `kubectl apply -f nginx-deployment.yaml` to create the deployment in the cluster.

8.5.4 Exposing the Deployment

- Create a service to expose the deployment using `kubectl expose deployment nginx-deployment --type=NodePort --port=80`.

8.5.5 Verifying the Deployment

- Use `kubectl get pods` to list the pods created by the deployment.
- Access the application using the service's NodePort.

8.5.6 Scaling the Deployment

- Scale the deployment up or down using `kubectl scale deployment/nginx-deployment --replicas=5`.

8.5.7 Cleaning Up

- Delete the deployment and service using `kubectl delete deployment nginx-deployment` and `kubectl delete service nginx-deployment`.

8.6 Task 4: Reflection and Discussion

8.6.1 Discussion

- Reflect on the process of deploying and managing applications in Kubernetes.
- Discuss the advantages of using Kubernetes for container orchestration.

8.6.2 Q&A Session

- Open the floor for questions about Kubernetes, its architecture, and related concepts.

8.7 Deliverables

- YAML files for the deployment and service.
- Evidence of successful deployment and scaling of the application.

Configuration Management(Ansible, Puppet, Chef)

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

9.1 Objective

- Understand the concepts and importance of configuration management in DevOps.
- Learn about popular configuration management tools like Ansible, Puppet, and Chef.
- Gain hands-on experience in writing and applying Ansible playbooks.

9.2 Resources Needed

- Computer with internet access
- Ansible installed
- A target machine (local or remote) with SSH access

9.3 Task 1: Introduction to Configuration Management

9.3.1 Overview of Configuration Management

- Explain the role of configuration management in maintaining the consistency of software systems.
- Discuss the challenges addressed by configuration management, such as configuration drift.

9.3.2 Benefits of Configuration Management

- Ensure consistency and repeatability of system setups.
- Automate configuration processes to reduce manual errors and save time.
- Facilitate collaboration among team members by using version-controlled configuration files.

9.4 Task 2: Tools for Configuration Management

9.4.1 Ansible

- Describe Ansible as an open-source automation tool for configuration management, application deployment, and task automation.
- Highlight its agentless architecture and use of YAML for defining configurations.
- Discuss the components of Ansible: Control Node, Managed Nodes, Inventory, and Playbooks.

9.4.2 Puppet

- Introduce Puppet as a configuration management tool that automates the provisioning, configuration, and management of infrastructure.
- Discuss its client-server architecture and use of Puppet DSL (Domain Specific Language).

9.4.3 Chef

- Explain Chef as an automation tool for infrastructure configuration management and application delivery.
- Highlight its use of Ruby-based DSL for defining configurations and its client-server architecture.

9.5 Task 3: Lab: Writing Ansible Playbooks

9.5.1 Setting Up Ansible

- Install Ansible on your local machine.
- Verify the installation using the command `ansible --version`.

9.5.2 Creating an Inventory File

- Create an inventory file to define the target machines for Ansible. Example:

```
[webservers]
192.168.1.100
192.168.1.101
```

9.5.3 Writing a Simple Playbook

- Write a basic Ansible playbook to install and start the Apache web server on target machines. Example:

```
- name: Configure Apache virtual host
  template:
    src: /path/to/vhost.template
    dest: /etc/apache2/sites-available/000-default.conf
  notify:
    - Restart Apache
```

```
  - name: Enable Apache rewrite module
    command: a2enmod rewrite
    notify:
      - Restart Apache

handlers:
  - name: Restart Apache
    service:
      name: apache2
      state: restarted
```

9.5.4 Running the Playbook

- Use the command `ansible-playbook -i inventory.ini playbook.yml` to execute the playbook on the target machines.

9.5.5 Verifying the Setup

- Check if Apache is installed and running on the target machines by accessing them via a web browser or using SSH.

9.5.6 Modifying the Playbook

- Add tasks to the playbook to configure the Apache server, such as setting up a virtual host. Example:

```
  - name: Configure Apache virtual host
    template:
      src: /path/to/vhost.template
      dest: /etc/apache2/sites-available/000-default.conf
    notify:
      - Restart Apache

  - name: Enable Apache rewrite module
    command: a2enmod rewrite
    notify:
      - Restart Apache

handlers:
  - name: Restart Apache
    service:
        name: apache2
        state: restarted
```

9.6 Task 4: Reflection and Discussion

9.6.1 Discussion

- Reflect on the process of writing and running Ansible playbooks.
- Discuss the advantages of using Ansible for configuration management.

9.6.2 Q&A Session

- Open the floor for questions about Ansible, configuration management, and related concepts.

9.7 Deliverables

- Inventory file and Ansible playbooks.
- Evidence of successful playbook execution and Apache configuration.

Monitoring and Logging(Prometheus, Grafana, ELK Stack)

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

10.1 Objective

- Understand the significance of monitoring and logging in maintaining application health and performance.
- Learn about popular tools used for monitoring and logging, such as Prometheus, Grafana, and the ELK Stack.
- Gain hands-on experience in setting up monitoring and logging for applications.

10.2 Resources Needed

- Computer with internet access
- Prometheus, Grafana, and ELK Stack installed

10.3 Task 1: Introduction to Monitoring and Logging

10.3.1 Overview of Monitoring and Logging

- Explain the role of monitoring in tracking the performance, availability, and health of applications and infrastructure.
- Discuss the importance of logging in recording system events, errors, and user activity.

10.3.2 Benefits of Monitoring and Logging

- Proactive issue detection and resolution.
- Improved system reliability and performance.
- Enhanced security and compliance through detailed audit logs.

10.4 Task 2: Tools for Monitoring and Logging

10.4.1 Prometheus

- Describe Prometheus as an open-source monitoring and alerting toolkit designed for reliability and scalability.
- Highlight its use of a time-series database, a powerful query language (PromQL), and support for multi-dimensional data collection.

10.4.2 Grafana

- Introduce Grafana as an open-source platform for monitoring and observability, providing rich visualizations and dashboards.
- Discuss its integration capabilities with various data sources, including Prometheus.

10.4.3 ELK Stack

- Explain the ELK Stack, consisting of Elasticsearch, Logstash, and Kibana, as a powerful suite for logging and analytics.
- Highlight Elasticsearch's role in data indexing and searching, Logstash's role in data processing and transformation, and Kibana's role in data visualization.

10.5 Task 3: Lab: Setting Up Monitoring and Logging for Applications

10.5.1 Setting Up Prometheus

- Install Prometheus on your local machine or a server.
- Configure Prometheus to scrape metrics from a sample application. Example configuration:

```
global:
    scrape_interval: 15s

scrape_configs:
    - job_name: 'sample_app'
      static_configs:
        - targets: ['localhost:9090']
```

10.5.2 Setting Up Grafana

- Install Grafana on your local machine or a server.
- Add Prometheus as a data source in Grafana.
- Create a dashboard in Grafana to visualize metrics from Prometheus.

10.5.3 Setting Up ELK Stack

- Install Elasticsearch, Logstash, and Kibana on your local machine or a server.
- Configure Logstash to collect and process logs from a sample application. Example Logstash configuration:

```
input {
    file {
        path => "/var/log/sample_app.log"
        start_position => "beginning"
    }
}

filter {
    grok {
        match => { "message" => "%{COMBINEDAPACHELOG}" }
    }
}

output {
    elasticsearch {
        hosts => ["localhost:9200"]
        index => "sample_app_logs"
    }
}
```

10.5.4 Configuring Kibana

- Access Kibana through a web browser.
- Create an index pattern for the logs collected by Logstash.
- Visualize log data using Kibana's dashboard capabilities.

10.5.5 Testing and Verification

- Generate sample logs and metrics from the application.
- Verify that Prometheus is scraping metrics correctly.
- Ensure Grafana dashboards are displaying the metrics accurately.
- Check that logs are indexed in Elasticsearch and visualized in Kibana.

10.6 Task 4: Reflection and Discussion

10.6.1 Discussion

- Reflect on the process of setting up monitoring and logging.
- Discuss the advantages of using tools like Prometheus, Grafana, and ELK Stack for observability.

10.6.2 Q&A Session

- Open the floor for questions about monitoring, logging, and the tools discussed.

10.7 Deliverables

- Prometheus configuration files.
- Grafana dashboards.
- ELK Stack configurations and visualizations.
- Evidence of successful monitoring and logging setup.

Security in DevOps (Aqua, Clair, Snyk)

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

11.1 Objective

- Understand the principles of DevSecOps and the integration of security practices within the DevOps pipeline.
- Learn about the best practices and tools used for enhancing security in CI/CD pipelines.
- Gain hands-on experience in implementing security checks in CI/CD pipelines.

Rsources Needed

- Computer with internet access
- CI/CD tools (e.g., Jenkins, GitLab CI)
- Security tools (e.g., Aqua, Clair, Snyk)

11.2 Task 1: Introduction to DevSecOps

11.2.1 Overview of DevSecOps

- Define DevSecOps and its importance in integrating security practices throughout the DevOps lifecycle.
- Explain the shift-left approach, emphasizing the need to address security from the beginning of the development process.

11.2.2 Key Principles of DevSecOps

- Collaboration between development, security, and operations teams.
- Automation of security tasks within the CI/CD pipeline.
- Continuous monitoring and feedback loops for security issues.

11.3 Task 2: Best Practices and Tools for DevSecOps

11.3.1 Best Practices

- Implement security at every stage of the DevOps pipeline.
- Use automated tools for static and dynamic code analysis.
- Conduct regular security training and awareness programs for the team.

11.3.2 Security Tools

- **Aqua Security:** Provides container security solutions, including image scanning and runtime protection.
- **Clair:** An open-source project for the static analysis of vulnerabilities in application containers (e.g., Docker).
- **Snyk:** A developer-first security platform that helps find and fix vulnerabilities in dependencies, container images, and Kubernetes applications.

11.4 Task 3: Lab: Implementing Security Checks in CI/CD Pipelines

11.4.1 Setting Up the Environment

- Ensure you have a CI/CD tool installed (e.g., Jenkins, GitLab CI).
- Install security tools (e.g., Aqua, Clair, Snyk) on your local machine or server.

11.4.2 Integrating Security Checks

- **Aqua Security:**
  - Install the Aqua CLI tool.
  - Integrate Aqua scans into your CI/CD pipeline to check for vulnerabilities in container images. Example Jenkinsfile snippet:

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                script {
                    // Build your Docker image
                    sh 'docker build -t my-app .'
                }
            }
        }
        stage('Security Scan') {
            steps {
                script {
                    // Scan the Docker image with Aqua
                    sh 'aqua scan --local my-app'
                }
```

```
            }
        }
        // Further stages...
    }
}
```

- **Clair:**
  - Install Clair and its companion tool, Clairctl.
  - Configure Clairctl to scan Docker images for vulnerabilities. Example Jenkinsfile snippet:

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                script {
                    // Build your Docker image
                    sh 'docker build -t my-app .'
                }
            }
        }
        stage('Security Scan') {
            steps {
                script {
                    // Scan the Docker image with Clairctl
                    sh 'clairctl analyze my-app'
                }
            }
        }
        // Further stages...
    }
}
```

- **Snyk:**
  - Sign up for a Snyk account and obtain the API token.
  - Integrate Snyk into your CI/CD pipeline to scan for vulnerabilities in dependencies and container images. Example GitLab CI snippet:

```
stages:
- build
- security_scan

build:
script:
- docker build -t my-app .

security_scan:
script:
- snyk test --docker my-app --file=Dockerfile
```

11.4.3 Testing and Verification

- Trigger the CI/CD pipeline to run the security scans.
- Review the results and fix any identified vulnerabilities.

- Ensure the pipeline fails if critical vulnerabilities are detected.

11.5 Task 4: Reflection and Discussion

11.5.1 Discussion

- Reflect on the process of integrating security checks into CI/CD pipelines.
- Discuss the benefits of using automated security tools in DevOps.

11.5.2 Q&A Session

- Open the floor for questions about DevSecOps, best practices, and the security tools discussed.

11.6 Deliverables

- CI/CD pipeline configuration files.
- Security scan reports from Aqua, Clair, and Snyk.
- Evidence of fixed vulnerabilities and improved security posture.

Automation and Scripting (Bash, Python)

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

12.1 Objective

- Understand the importance of automation in DevOps.
- Learn about scripting languages such as Bash and Python.
- Automate tasks using Python scripts to gain hands-on experience with automation.

12.2 Resources Needed

- Computer with internet access
- Python and Bash installed
- Sample tasks to automate

12.3 Task 1: Importance of Automation in DevOps

12.3.1 Introduction to Automation

- Explain the concept of automation and its benefits in the DevOps lifecycle.
- Discuss the impact of automation on efficiency, consistency, and error reduction.

12.3.2 Automation Use Cases

- Provide examples of tasks that can be automated, such as code deployment, testing, infrastructure provisioning, and monitoring.

12.3.3 DevOps Automation Tools

- Introduce popular tools used in DevOps for automation, such as Jenkins, Ansible, and Terraform.

12.4 Task 2: Scripting Languages (Bash, Python)

12.4.1 Introduction to Bash

- Explain the basics of Bash scripting.
- Provide examples of simple Bash scripts to automate tasks like file management and system monitoring.

12.4.2 Introduction to Python

- Explain the basics of Python scripting.
- Provide examples of simple Python scripts to automate tasks like data processing and web scraping.

12.4.3 Comparative Analysis

- Discuss the strengths and weaknesses of Bash and Python for automation tasks.
- Provide guidelines on when to use each language.

12.5 Task 3: Lab: Automating Tasks with Python Scripts

12.5.1 Setting Up the Environment

- Ensure Python is installed on your computer.
- Install any necessary Python libraries (e.g., `requests`, `os`, `subprocess`).

12.5.2 Automating a Sample Task

- Choose a simple task to automate (e.g., file backup, data scraping, system monitoring).
- **Example: Automating File Backup:**
  - Create a Python script to backup files from a source directory to a destination directory.
  - Include error handling and logging in the script.
  - Example Python script:

```
import os
import shutil
import datetime

def backup_files(src_dir, dest_dir):
    if not os.path.exists(dest_dir):
        os.makedirs(dest_dir)
    for filename in os.listdir(src_dir):
        full_file_name = os.path.join(src_dir, filename
            )
        if os.path.isfile(full_file_name):
            shutil.copy(full_file_name, dest_dir)
            print(f"Copied {filename} to {dest_dir}")
```

```
if __name___ == "__main__":
    src = "/path/to/source"
    dest = "/path/to/destination/backup_" + datetime.
        datetime.now().strftime("%Y%m%d%H%M%S")
    backup_files(src, dest)
```

12.5.3 Testing and Verification

- Run the Python script and verify that the files are successfully backed up to the destination directory.
- Review the automation process and discuss any challenges faced.

12.6 Task 4: Reflection and Discussion

12.6.1 Discussion

- Reflect on the experience of automating tasks using Python scripts.
- Discuss the advantages and potential pitfalls of automation in DevOps.
- Consider other tasks that could benefit from automation.

12.6.2 Q&A Session

- Open the floor for questions about automation, scripting languages, and best practices.

12.7 Deliverables

- Python scripts for automating tasks.
- Documentation of the automation process and any issues encountered.
- Screenshots or logs demonstrating the successful automation of tasks.

Advanced CI/CD Techniques

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

13.1 Objective

- Understand advanced pipeline configurations in CI/CD.
- Learn about deployment strategies such as Blue-Green Deployments and Canary Releases.
- Implement advanced deployment strategies in a lab setting.

13.2 Resources Needed

- Computer with internet access
- CI/CD tools (e.g., Jenkins, GitLab CI, CircleCI)
- Sample application for deployment

13.3 Task 1: Advanced Pipeline Configurations

13.3.1 Introduction to Advanced Pipelines

- Explain the importance of advanced pipeline configurations in achieving efficient and reliable software delivery.
- Discuss key components and best practices for configuring advanced pipelines.

13.3.2 Pipeline Configuration Tools

- Introduce popular tools for configuring CI/CD pipelines, such as Jenkins, GitLab CI, and CircleCI.
- Provide examples of advanced pipeline configurations using these tools.

13.4 Task 2: Blue-Green Deployments, Canary Releases

13.4.1 Introduction to Deployment Strategies

- Define and explain Blue-Green Deployments and Canary Releases.
- Discuss the benefits and challenges of each deployment strategy.

13.4.2 Blue-Green Deployments

- Explain how Blue-Green Deployments work.
- Provide a step-by-step guide to implementing Blue-Green Deployments.
- Example scenario: Deploying a web application using Blue-Green Deployments.

13.4.3 Canary Releases

- Explain how Canary Releases work.
- Provide a step-by-step guide to implementing Canary Releases.
- Example scenario: Deploying a new feature using Canary Releases.

13.5 Task 3: Lab: Implementing Advanced Deployment Strategies

13.5.1 Setting Up the Environment

- Ensure that the necessary CI/CD tools are installed and configured on your computer or server.
- Clone the sample application repository for deployment.

Implementing Blue-Green Deployments

- Configure your pipeline to perform Blue-Green Deployments.
- Deploy the sample application using Blue-Green Deployments.
- Verify the deployment by switching traffic between the blue and green environments.

13.5.2 Implementing Canary Releases

- Configure your pipeline to perform Canary Releases.
- Deploy the new feature of the sample application using Canary Releases.
- Monitor the deployment and adjust the release percentage as needed.

13.5.3 Testing and Verification

- Test the deployed application to ensure it is working as expected.
- Review the deployment process and discuss any challenges faced.

13.6 Task 4: Reflection and Discussion

13.6.1 Discussion

- Reflect on the experience of implementing advanced deployment strategies.
- Discuss the advantages and potential pitfalls of Blue-Green Deployments and Canary Releases.
- Consider other scenarios where these deployment strategies could be beneficial.

13.6.2 Q&A Session

- Open the floor for questions about advanced CI/CD techniques and deployment strategies.

13.7 Deliverables

- Configured pipelines for Blue-Green Deployments and Canary Releases.
- Documentation of the deployment process and any issues encountered.
- Screenshots or logs demonstrating the successful implementation of advanced deployment strategies.

DevOps Culture and Practices

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

14.1 Objective

- Understand the importance of building a DevOps culture within an organization.
- Learn about tools that facilitate collaboration and communication in DevOps teams.
- Create a DevOps project plan and set up collaboration tools in a lab setting.

14.2 Resources Needed

- Computer with internet access
- Access to Slack, Jira, or other collaboration tools

14.3 Task 1: Building a DevOps Culture

14.3.1 Introduction to DevOps Culture

- Explain the core principles of DevOps culture, emphasizing collaboration, communication, and continuous improvement.
- Discuss the importance of breaking down silos between development and operations teams.

14.3.2 Key Elements of a DevOps Culture

- Discuss trust, ownership, and shared responsibility.
- Highlight the significance of continuous learning and feedback loops.
- Explain how automation and tooling support a DevOps culture.

14.4 Task 2: Collaboration and Communication Tools

14.4.1 Introduction to Collaboration Tools

- Explain the role of collaboration and communication tools in a DevOps environment.
- Introduce popular tools such as Slack and Jira, highlighting their key features and benefits.

14.4.2 Using Slack for Communication

- Demonstrate how to set up a Slack workspace for a DevOps team.
- Show how to create channels for different projects and teams.
- Discuss best practices for effective communication using Slack.

14.4.3 Using Jira for Project Management

- Demonstrate how to set up a Jira project for a DevOps team.
- Show how to create and manage issues, sprints, and boards.
- Discuss best practices for tracking progress and managing tasks using Jira.

14.5 Task 3: Lab: Creating a DevOps Project Plan and Collaboration Setup

14.5.1 Setting Up the Environment

- Ensure that all students have access to Slack and Jira.
- Create a sample project plan outline for the DevOps team.

14.5.2 Creating a DevOps Project Plan

- Define the goals and objectives of the project.
- Identify key milestones, deliverables, and timelines.
- Assign roles and responsibilities to team members.

14.5.3 Setting Up Collaboration Tools

- Create a Slack workspace and set up relevant channels for the project.
- Invite team members to the Slack workspace and demonstrate key communication features.
- Set up a Jira project, create issues and sprints, and demonstrate how to manage tasks.

14.5.4 Collaborating on the Project

- Use Slack to communicate with team members and share updates.
- Use Jira to track progress, assign tasks, and manage the project.
- Hold a virtual stand-up meeting using Slack or Jira to discuss the project's status and any blockers.

14.5.5 Testing and Verification

- Ensure that all team members can access and use Slack and Jira effectively.
- Review the project plan and make any necessary adjustments based on team feedback.
- Monitor collaboration and communication throughout the lab session.

14.6 Part 4: Reflection and Discussion

14.6.1 Discussion

- Reflect on the experience of setting up and using collaboration tools in a DevOps environment.
- Discuss the benefits and challenges of building a DevOps culture.
- Consider how the principles and practices learned can be applied in real-world scenarios.

14.6.2 Q&A Session

- Open the floor for questions about DevOps culture, collaboration tools, and project planning.

14.7 Deliverables

- Completed DevOps project plan.
- Configured Slack workspace and Jira project.
- Documentation of the collaboration setup and any issues encountered.
- Screenshots or logs demonstrating the use of Slack and Jira.