5.2. Student Handout

CI/CD Pipeline: Student Handout

Introduction to CI/CD Pipeline

CI/CD stands for **Continuous Integration** and **Continuous Delivery/Deployment**. It is a DevOps practice that automates the software development process, including building, testing, and deploying code changes.

Importance of CI/CD in DevOps

1. **Reducing Manual Work**: Automates repetitive tasks like testing and deployment.
2. **Faster Feedback**: Provides immediate feedback to developers if issues arise.
3. **Consistency**: Ensures a standardized process, reducing errors.
4. **Faster Releases**: Enables frequent releases of new features or bug fixes.

Key Stages of a CI/CD Pipeline

1. **Source Stage**:

- Code is stored in a version control system like Git or AWS CodeCommit.
- Example: Developers push code changes to a Git repository.

2. **Build Stage**:

- Code is compiled and packaged into an executable format.
- Example: Using Maven to build a Java application.

3. **Test Stage**:

- Automated tests are run to verify code functionality.
- Example: Running unit tests using JUnit.

4. **Deploy Stage**:

- Code is deployed to a server for user access.
- Example: Deploying a web application to an AWS EC2 instance.

5. **Monitor Stage**:

- System is monitored post-deployment to ensure smooth operation.
- Example: Using AWS CloudWatch to monitor application performance.

Creating a CI/CD Pipeline Using AWS CodePipeline

1. **Define the Stages**:

- Set up stages like Source, Build, Test, and Deploy in AWS CodePipeline.

2. **Integrate with AWS Services**:

- **CodeCommit**: Stores source code.
- **CodeBuild**: Compiles code and runs tests.
- **CodeDeploy**: Deploys code to servers.

3. **Configure Triggers**:

- Automate pipeline execution when new code is pushed to CodeCommit.

Managing and Monitoring the Pipeline

- Use AWS CodePipeline's dashboard to monitor the status of each stage.
- Example: Receiving alerts if a test fails during the Test Stage.

Implementing Automated Testing and Rollback Strategies

1. **Automated Testing with AWS CodeBuild**:

- Run tests automatically after the build stage.
- Example: Stopping the pipeline if integration tests fail.

2. **Safe Rollback Strategies with AWS CodeDeploy**:

- Configure automatic rollbacks for failed deployments.
- Example: Reverting to a previous stable version if deployment issues occur.

Hands-On: Building a Complete CI/CD Pipeline

1. **Create a CodeCommit Repository**:

- Store your source code.

2. **Set Up CodeBuild**:

- Compile code and run tests.

3. **Configure CodeDeploy**:

- Deploy code to AWS services like EC2.

4. **Create a Pipeline in CodePipeline**:

- Define and integrate stages with AWS services.

5. **Configure Triggers**:

- Automate pipeline execution on code changes.

6. **Monitor the Pipeline**:

- Use the dashboard to track pipeline status.

Conclusion

A CI/CD pipeline automates the software development process, ensuring efficient, reliable, and fast delivery of software. By leveraging AWS services like CodePipeline, CodeCommit, CodeBuild, and CodeDeploy, you can maintain a ready-to-deploy state for your software.

Diagram: CI/CD Pipeline Overview

```
+------------+  +------------+  +-----------+  +------------+  +------------
+
|Source Code |->| Build      |->|Test       |->|Deploy      |->|Monitor
|
|(CodeCommit)|  | (CodeBuild)|  |(CodeBuild)|  |(CodeDeploy)|  |
(CloudWatch)|
+------------+  +------------+  +-----------+  +------------+  +------------
+
```

Thank you for your attention, and I hope this handout helps solidify your understanding of CI/CD pipelines!