Week 1: Introduction to DevOps and CI/CD

Day 1: Introduction to DevOps

1. What is DevOps?

- **Definition:** DevOps is a set of practices that combines software development (Dev) and IT operations (Ops).
- **Goal:** To shorten the system development life cycle while delivering features, fixes, and updates frequently in close alignment with business objectives.

2. DevOps Principles and Practices

- **Collaboration and Communication:** Enhanced cooperation between development and operations teams.
- **Automation:** Automation of repetitive tasks to increase efficiency and reliability.
- **Continuous Integration and Continuous Deployment (CI/CD):** Frequent integration of code into a shared repository and automated deployment to production.
- **Infrastructure as Code (IaC):** Managing and provisioning computing infrastructure through machine-readable definition files.
- **Monitoring and Logging:** Continuous monitoring of applications and infrastructure to improve transparency and performance.
- **Security:** Integrating security practices into the DevOps pipeline (DevSecOps).

3. Benefits of DevOps

- **Speed:** Increased speed of delivery and deployment.
- **Reliability:** Improved quality of software releases and reduced failure rates.
- **Scale:** Enhanced ability to operate and manage infrastructure at scale.
- **Improved Collaboration:** Better communication and cooperation between teams.
- **Security:** Proactive and integrated security practices.

4. Overview of DevOps Tools and Technologies

- **Version Control Systems:** Git, SVN
- **CI/CD Tools:** Jenkins, CircleCI, AWS CodePipeline
- **Configuration Management:** Ansible, Chef, Puppet
- **Containerization:** Docker, Kubernetes
- **Monitoring and Logging:** Prometheus, Grafana, ELK Stack
- **Cloud Services:** AWS, Azure, Google Cloud Platform

Day 2: AWS CodeCommit

1. Introduction to Version Control

- **Definition:** Version control is a system that records changes to a file or set of files over time so that specific versions can be recalled later.
- **Importance:** Facilitates collaboration, tracks history, and enables rollback of changes.

2. Overview of AWS CodeCommit

- **Service Description:** A fully managed source control service that hosts secure Git-based repositories.
- **Key Features:** High availability, scalability, security, and integration with other AWS services.

3. Creating and Managing Repositories

- **Creating Repositories:** Steps to create a repository in AWS CodeCommit.
- **Managing Repositories:** Adding, modifying, and deleting files; managing branches and merges.
- **Access Control:** Setting up IAM policies and roles for repository access.

4. Integrating CodeCommit with Other Tools

- **Integration with CI/CD Tools:** Using AWS CodePipeline, Jenkins, or other CI/CD tools with CodeCommit.
- **Notifications and Monitoring:** Setting up AWS SNS for repository notifications.
- **Code Reviews:** Implementing code review processes with pull requests.

Day 3: AWS CodeBuild

1. Introduction to Continuous Integration

- **Definition:** Continuous Integration (CI) is a development practice where developers integrate code into a shared repository frequently, preferably several times a day.
- **Benefits:** Detect errors quickly, reduce integration problems, and deliver software updates faster.

2. Overview of AWS CodeBuild

- **Service Description:** A fully managed build service in the cloud that compiles source code, runs tests, and produces software packages.
- **Key Features:** Scalability, pay-as-you-go pricing, and integration with other AWS services.

3. Creating and Managing Build Projects

- **Creating Build Projects:** Steps to create a build project in AWS CodeBuild.
- **Build Environments:** Configuring build environments, specifying runtime, and environment variables.
- **Build Artifacts:** Managing output artifacts and storing them in Amazon S3 or other storage services.

4. Configuring Buildspec Files

- **Definition:** A buildspec.yml file defines the build commands and settings used by AWS CodeBuild.
- **Structure:** Phases (install, pre-build, build, post-build), artifacts, cache settings, and environment variables.
- **Examples:** Sample buildspec.yml files for different programming languages and build tools.

Day 4: AWS CodeDeploy

1. Introduction to Continuous Deployment

- **Definition:** Continuous Deployment (CD) is a software release process that uses automated testing to validate and release code changes automatically to a production environment.
- **Benefits:** Faster time-to-market, reduced human error, and more reliable releases.

2. Overview of AWS CodeDeploy

- **Service Description:** A deployment service that automates code deployments to any instance, including Amazon EC2 instances and on-premises servers.
- **Key Features:** Supports various deployment strategies, integrates with other AWS services, and provides monitoring and rollback capabilities.

3. Creating Deployment Applications and Groups

- **Creating Applications:** Steps to create a deployment application in AWS CodeDeploy.
- **Deployment Groups:** Defining and managing deployment groups based on target environments (e.g., development, staging, production).

4. Deployment Strategies (In-Place and Blue/Green)

- **In-Place Deployment:** Updates the application on each instance in a deployment group.
- **Blue/Green Deployment:** Creates a new set of instances (green) and switches traffic from the old set (blue) to the new set.
- **Best Practices:** Choosing the right deployment strategy, monitoring deployments, and handling rollbacks.

Day 5: AWS CodePipeline

1. Introduction to Continuous Delivery

- **Definition:** Continuous Delivery (CD) is a software engineering approach where teams produce software in short cycles, ensuring that the software can be reliably released at any time.
- **Benefits:** Faster feedback, improved quality, and better alignment with business goals.

2. Overview of AWS CodePipeline

- **Service Description:** A continuous integration and continuous delivery service for fast and reliable application and infrastructure updates.
- **Key Features:** Automation of the build, test, and deploy phases of the release process; integration with other AWS services and third-party tools.

3. Creating and Managing Pipelines

- **Creating Pipelines:** Steps to create a pipeline in AWS CodePipeline.
- **Pipeline Stages:** Defining stages (source, build, test, deploy) and actions within each stage.
- **Pipeline Configuration:** Configuring settings, such as triggers, artifact storage, and environment variables.

4. Integrating CodeCommit, CodeBuild, and CodeDeploy

- **Source Stage:** Configuring AWS CodeCommit as the source provider.
- **Build Stage:** Integrating AWS CodeBuild to compile and test code.
- **Deploy Stage:** Using AWS CodeDeploy to automate deployments.
- **End-to-End Workflow:** Creating a complete CI/CD pipeline that integrates CodeCommit, CodeBuild, and CodeDeploy.
