**# DevOps Bootcamp: A Beginner’s Journey**

Stages Covered In This Week:

1. Introduction to DevOps
2. Version Control with Git
3. CI/CD with Jenkins
4. Containerization, Orchestration & IaC Overview
5. How Can We Apply DevOps in Our Projects?

**Evolution of Software Development - How DevOps Came into Picture**

**Waterfall Model**

* The **Waterfall model** was the earliest software development approach, consisting of sequential phases:
* **Challenges of Waterfall**:
  + Rigid structure with minimal flexibility.
  + Late stage testing often leads to expensive bug fixes.
  + Deployment takes months or even years.

**Agile Model**

* Agile was introduced to address Waterfall’s limitations by breaking the development process into small, iterative cycles called **sprints**.
* **Advantages of Agile**:
  + Faster feedback loops with continuous improvement.
  + Better collaboration between developers and business teams.
  + Frequent releases ensure early bug detection.

**How DevOps Came Into Picture**

* Agile improved development speed, but **operations teams** still worked separately, causing bottlenecks in deployment and infrastructure management.
* DevOps bridges this gap by integrating development and operations, automating workflows, and ensuring continuous feedback.

**Agile vs. DevOps: Understanding the Difference**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Feature** |  |  | | --- | |  | | **Agile** | |  | | --- | | **DevOps** | |
| Definition | Agile is a software development methodology focused on iterative development, continuous feedback, and customer collaboration. | DevOps is a culture and set of practices that combine development (Dev) and operations (Ops) to enable continuous integration, delivery, and deployment. |
| Primary Goal | Deliver working software in small, incremental cycles based on customer feedback. | Automate and streamline the software release process to ensure rapid, reliable deployments. |
| Focus Area | Development teams and product management. | Both development and operations teams, including infrastructure and deployment. |
| Key Practices | Scrum, Kanban, sprints, stand-ups, user stories. | CI/CD pipelines, automation, infrastructure as code, monitoring. |
| Delivery Cycle | Focuses on short development cycles (sprints) with regular software releases. | Ensures faster deployment through automation and continuous delivery. |

**Day 1: Introduction to DevOps**

What is DevOps?

DevOps is a combination of Development (Dev) and Operations (Ops). It is a set of practices, cultural philosophies, and tools that enhance an organization’s ability to deliver applications and services at high velocity. Unlike traditional software development processes, DevOps enables continuous integration, delivery, and deployment, reducing the time from code development to production release.

Key Benefits of DevOps:

* Faster software development lifecycle (SDLC)
* Improved collaboration between development and operations teams
* Enhanced automation and efficiency
* Reduced failure rate of new releases
* Faster recovery from failures

A picture containing screenshot, text, circle, diagram

Description automatically generated

DevOps Lifecycle:

1. Plan – Requirements gathering, collaboration tools like Jira, Confluence.
2. Develop – Coding, version control (Git, GitHub, GitLab).
3. Build – Compiling, testing (Maven, Gradle, Docker).
4. Test – Automated testing frameworks (Selenium, JUnit, PyTest).
5. Release – CI/CD pipelines (Jenkins, GitHub Actions).
6. Deploy – Container orchestration (Docker, Kubernetes).
7. Operate – Monitoring and logging (Prometheus, Grafana, ELK Stack).
8. Monitor – Application and infrastructure monitoring tools.

Real-World Use Cases:

* Netflix automates deployments using Spinnaker and Kubernetes.
* Amazon follows DevOps practices to deploy thousands of changes per day.
* Google integrates SRE (Site Reliability Engineering) with DevOps.

|  |  |  |
| --- | --- | --- |
| Feature | Traditional IT | DevOps |
| Release Frequency | Infrequent | Frequent |
| Deployment Failures | High | Low (automated testing) |
| Recovery Time | Long | Short |
| Collaboration | Siloed | Integrated teams |

**Agile Without DevOps vs. Agile With DevOps**

* **Agile Without DevOps:**  
  A team develops new features in sprints but faces delays in deployment because of manual testing, handovers to the operations team, and configuration issues.
* **Agile With DevOps:**  
  The same team uses CI/CD pipelines, automated testing, and infrastructure as code, reducing deployment time from weeks to minutes.

Day 2: Version Control with Git

What is Version Control?

Version control is a system that tracks changes in files over time. It allows multiple developers to collaborate on a project without overwriting each other’s work. The two main types of version control systems (VCS) are

1. Centralized Version Control System (CVCS) – A single central repository (e.g., SVN, Perforce).
2. Distributed Version Control System (DVCS) – Every developer has a local copy of the repository (e.g., Git, Mercurial).

Why is Version Control Important?

* Tracks changes and maintains a history of modifications.
* Enables collaboration between multiple developers.
* Helps in rollback to previous versions if needed.
* Prevents conflicts between code changes.

Git: A Distributed Version Control System

Git is the most widely used version control system in DevOps. It provides features such as:

* Branching and Merging: Developers can work on separate branches and merge changes seamlessly.
* Commit History: Each change is recorded with timestamps and commit messages.
* Distributed Workflow: Every developer has a full copy of the project repository.

Basic Git Workflow

# Initialize a new repository

git init

# Clone an existing repository

git clone <repository\_url>

# Check repository status

git status

# Add changes to staging

git add .

# Commit changes

git commit -m "Commit message"

# Push changes to remote repository

git push origin main

# Pull latest changes from remote

git pull origin main

GitHub for Collaboration

GitHub is a cloud-based Git repository hosting service that enables teams to:

* Work on projects collaboratively.
* Use pull requests for code review.
* Automate workflows with GitHub Actions.

Hands-on Activities:

* Create a GitHub repository and initialize it.
* Make commits and push changes.
* Create a new branch, modify files, and merge back.
* Use GitHub pull requests for collaboration.

Day 3: CI/CD & Jenkins A black text on a transparent background

Description automatically generated with medium confidence

What is CI/CD?

* Continuous Integration (CI): The practice of automatically integrating code changes from multiple contributors into a shared repository several times a day. CI tools automate the building and testing of applications to detect issues early.
* Continuous Deployment (CD): The automated release of validated software to production environments. It minimizes manual intervention and speeds up delivery.

Benefits of CI/CD

* Faster time to market
* Reduced risks and errors
* Improved developer productivity
* Higher software quality

Jenkins Overview

Jenkins is an open-source automation server that facilitates CI/CD by automating build, test, and deployment processes.

Setting Up Jenkins

1. Install Jenkins and configure it.
2. Create and configure a new Jenkins job.
3. Integrate with Git for source code versioning.
4. Implement an automated CI/CD pipeline.

Key Features of Jenkins:

* Automates build and deployment workflows.
* Integrates with Git, Docker, Kubernetes, and more.
* Supports pipeline as code with Jenkinsfile.

Hands-on Activity

* Install Jenkins and configure a simple CI pipeline.
* Automate a code build process using Jenkins.

Day 4: Containerization with Docker

What is Docker?

Docker is a containerization platform that packages applications along with their dependencies in isolated environments (containers).

Benefits of Docker:

* Ensures consistency across different environments.
* Enables faster application deployment.
* Reduces infrastructure costs.

Docker Basics

* Images: Read-only templates used to create containers.
* Containers: Running instances of Docker images.
* Dockerfile: Script to build Docker images.

Docker Commands

# Pull an image from Docker Hub

docker pull nginx

# List running containers

docker ps

# Build a Docker image

docker build -t myapp .

# Run a Docker container

docker run -d -p 8080:80 myapp

Hands-on Activity:

* Build and run a Docker container.
* Deploy a sample application inside a container.

Day 5: Kubernetes for Container Orchestration

Why Kubernetes?

Kubernetes automates container deployment, scaling, and management.

Kubernetes Components

* Pods: Smallest deployable units.
* Deployments: Manage pod replicas.
* Services: Expose applications within a cluster.

Hands-on Activity:

* Deploy an application on a Kubernetes cluster.
* Scale a Kubernetes deployment.

Day 6: Infrastructure as Code (IaC) with Terraform

What is IaC?

IaC automates infrastructure provisioning using code, eliminating manual processes.

Terraform Basics

* Providers: AWS, Azure, GCP.
* Modules: Reusable configurations.

Hands-on Activity:

* Deploy cloud resources using Terraform.

Day 7: Monitoring & Logging with Prometheus and Grafana

Importance of Monitoring

Monitoring ensures system health, while logging captures application events.

Prometheus Basics

* Time-series monitoring database.
* Uses exporters to collect metrics.

Grafana for Visualization

* Builds dashboards for system insights.

Hands-on Activity:

* Set up monitoring for a sample application.