**Pratham Chavan - 1MS23CS409**

1. What is DevOps? How is it different from Waterfall and Agile methodology?

2. Explain the 3 Axis of DevOps Culture with a neat diagram.

3. List and explain benefits of establishing a DevOps culture.

4. With a neat diagram explain Continuous Integration (CI) in detail.

5. With a neat diagram explain Continuous Delivery (CD) in detail.

6. What is Continuous Deployment? How is it different from Continuous Delivery?

7. With a neat diagram explain the different techniques of implementing Continuous Deployment.

8. What are IaC Practices? List the benefits of IaC Practices.

9. Describe different IaC Languages used in DevOps. Explain the same with a simple code snippet.

10. List and explain the IaC Topologies.

11. Describe the IaC best practices followed in the DevOps pipeline.

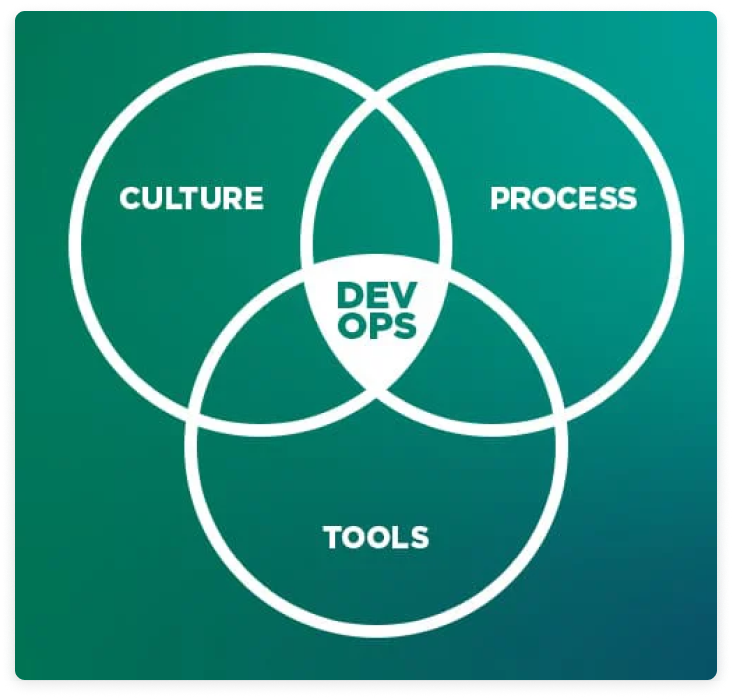
1. DevOps is a culture and set of practices that reduce the barriers between developers, who want to innovate and deliver faster, and operations, who want to guarantee the stability of production systems and the quality of the system changes they make.

**Comparison of Waterfall, Agile, and DevOps**

1. **Development Approach:**
   * **Waterfall:** Follows a sequential and linear process where each phase must be completed before moving to the next.
   * **Agile:** Uses an iterative and incremental methodology, allowing for flexibility and frequent updates.
   * **DevOps:** Focuses on continuous integration and continuous delivery (CI/CD), enabling rapid and automated deployments.
2. **Focus:**
   * **Waterfall:** Emphasizes detailed planning at the beginning of the project to ensure a structured workflow.
   * **Agile:** Prioritizes collaboration and adaptability to changes, allowing teams to adjust plans as necessary.
   * **DevOps:** Centers around collaboration between development and operations teams, with an emphasis on automation.
3. **Process Flexibility:**
   * **Waterfall:** Has a rigid structure with minimal flexibility for changes once the process begins.
   * **Agile:** Offers high flexibility, making it easier to incorporate changes based on feedback.
   * **DevOps:** Provides very high flexibility with frequent feedback and releases, ensuring quick iterations and improvements.
4. **Team Involvement:**
   * **Waterfall:** Developers and testers work in silos, meaning communication and collaboration between them are limited.
   * **Agile:** Encourages cross-functional teams where developers, testers, and other stakeholders collaborate closely.
   * **DevOps:** Similar to Agile but extends collaboration to include both development and operations teams (Dev + Ops).
5. **Delivery Time:**
   * **Waterfall:** Requires a long development cycle, as each phase is completed sequentially before moving to the next.
   * **Agile:** Uses short development cycles known as sprints, ensuring quicker releases.
   * **DevOps:** Focuses on continuous delivery and faster deployments, often deploying multiple times a day or week.
6. **Communication:**
   * **Waterfall:** Uses a formal communication structure at each phase, often requiring extensive documentation.
   * **Agile:** Encourages continuous communication throughout the sprint, enabling quick decision-making.
   * **DevOps:** Integrates continuous and seamless communication between teams, fostering a collaborative environment.
7. **Key Tools:**
   * **Waterfall:** Relies heavily on documentation as the primary means of tracking progress and decisions.
   * **Agile:** Uses agile tools like Jira and Scrum boards to manage workflows and track progress.
   * **DevOps:** Leverages CI/CD pipelines and automation tools such as Jenkins and Docker to streamline development and deployment

**2. Three Axes of the DevOps Movement:**

1. **Culture of Collaboration**:
   * Breaks down silos between teams (Dev, Ops, Testers, etc.).
   * Multidisciplinary teams work together towards a common goal: rapid value delivery.
2. **Processes**:
   * Agile-based iterative processes for continuous improvement.
   * **Key Phases**:
     + **Planning & Prioritizing** functionalities
     + **Development** of features
     + **Continuous Integration & Delivery** (CI/CD)
     + **Continuous Deployment** to production
     + **Continuous Monitoring** of systems and performance
   * These phases are cyclical and iterative throughout the project's life.
3. **Tools**:
   * Tools must support collaboration and integration across Dev and Ops.
   * Avoid tool silos by using shared tools that streamline development and deployment processes.



**3. The benefits of establishing a DevOps culture within an enterprise are as follows:**

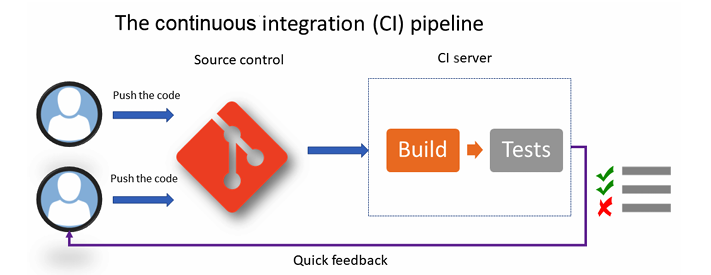
 • Better collaboration and communication in teams, which has a human and social impact within the company

• Shorter lead times to production, resulting in better performance and end user satisfaction

 • Reduced infrastructure costs with IaC

• Significant time saved with iterative cycles that reduce application errors and automation tools that reduce manual tasks, so teams focus more on developing new functionalities with added business value.

1. **Collaboration and Integration**:

****

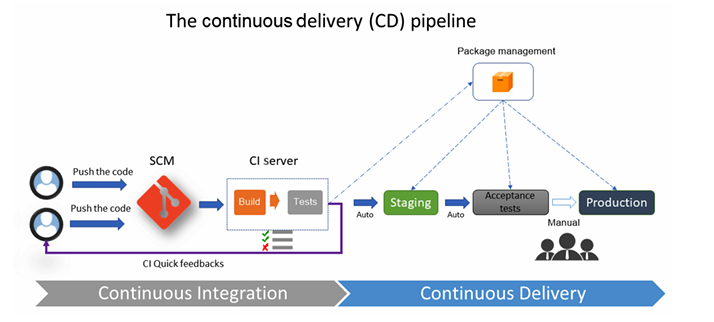
* CI fosters collaboration by ensuring that all team members integrate their code frequently. This process helps detect integration issues quickly, ensuring smoother teamwork and faster feedback.

"Continuous integration is a software development practice where members of a team integrate their work frequently... Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible."

Implementing CI Therefore, to set up CI, it is necessary to have a Source Code Manager (SCM) that will centralize the code of all members. This code manager can be of any type: Git, SVN, or Team Foundation Version Control (TFVC). It's also important to have an automatic build manager (CI server) that supports continuous integration, such as Jenkins, GitLab CI, TeamCity, Azure Pipelines, GitHub Actions, Travis CI, and Circle CI.

This CI process must be optimized as soon as possible so that it can run fast, and so that developers can gather quick feedback on the integration of their code. For example, code that has been archived and does not compile or whose test execution fails can impact and block the entire team. Sometimes, bad practices can cause tests to fail during CI. To deactivate this test's execution, you must take it is not serious, it is necessary to deliver quickly, or the code that compiles it is essential as an argument. On the contrary, this practice can have serious consequences when the errors that are detected by the tests are revealed in production. The time that's saved during CI will be lost on fixing errors with hotfixes and redeploying them quickly, which can cause stress. This is the opposite of DevOps culture as there's poor application quality for end users and no real feedback; instead of developing new features, we spend time correcting errors.

**5.**



**Automated Deployment to Staging**:

* Continuous Delivery (CD) automates the deployment of an application to non-production environments, such as staging. It uses automated tasks (e.g., unzipping files, restarting services, copying files) and may include functional and acceptance testing.

**Testing the Entire Application**:

* Unlike Continuous Integration (CI), which tests individual components, CD tests the entire application, including dependencies, microservices, and APIs, to ensure everything works together as a whole in the staging environment.

**Consistency Across Environments**:

* The application package generated by CI must remain unchanged through all environments, ensuring consistency. While configuration files may differ by environment, the core code (binaries, Docker images, JAR files) must remain the same.

**Automated or Manual Trigger for Deployments**:

* CD deployments can be triggered automatically after successful testing in prior environments (e.g., from integration to pre-production), or manually for more sensitive environments like production, where approval is needed before deployment.

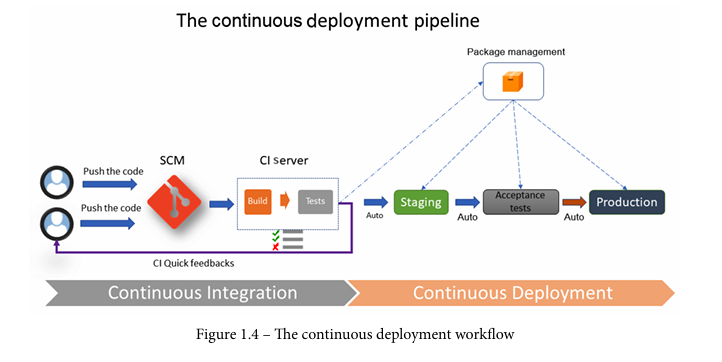
**Package Management and Configuration Management**:

* CD tools often work with package managers (e.g., Nexus, Artifactory) to store and retrieve generated packages. Configuration management tools handle environment-specific configuration changes during the deployment process.

**6.** Continuous deployment Continuous deployment is an extension of CD, but this time, with a process that automates the entire CI/CD pipeline from the moment the developer commits their code to deployment in production through all of the verification steps. This practice is rarely implemented in enterprises because it requires a variety of tests (unit, functional, integration, performance, and so on) to be covered for the application. Successfully executing these tests is sufficient to validate the proper functionality of the application regarding all of these dependencies. However, it also allows you to automatically deploy to a production environment without any approval action required.

**Difference between Continuous Deployment and Continuous Delivery:**

* **Continuous Delivery** automates the deployment process up to staging environments, and requires **manual approval for production deployment**.
* **Continuous Deployment** extends this automation to **production** as well, automatically deploying code to production without human approval after all tests pass.



**7.**

Different ways of implementing continuous deployment

**1. Feature Toggles (Feature Flags):**

* **What it is**: Feature toggles allow new features or changes to be deployed to production without immediately making them available to users. These features can be turned on or off dynamically.
* **How it works**: Developers deploy new code with certain features hidden behind feature flags. Once the code is deployed, the feature can be activated in the production environment, allowing testing or gradual rollout without redeploying.
* **Use Case**: Useful for testing new features in production without impacting all users, and allows for safer rollbacks if issues arise.

**2. Blue-Green Deployment:**

* **What it is**: Blue-green deployment is a strategy where two identical production environments are maintained: one active (blue) and one inactive (green).
* **How it works**: New versions of the application are deployed to the inactive environment (green). Once validated, traffic is switched from the active environment (blue) to the green environment. The blue environment can then be updated for the next deployment.
* **Use Case**: Ensures zero downtime during deployment by switching between environments without affecting users.

**8.**

**Infrastructure as Code (IaC)** is the practice of managing and provisioning computing infrastructure through machine-readable configuration files rather than through physical hardware configuration or interactive configuration tools.

The benefits of IaC are as follows:

 • The standardization of infrastructure configuration reduces the risk of errors.

• The code that describes the infrastructure is versioned and controlled in a source code manager.

• The code is integrated into CI/CD pipelines.

 • Deployments that make infrastructure changes are faster and more efficient.

• There's better management, control, and a reduction in infrastructure costs.

9.

**Scripting IaC Languages**: Use scripts (e.g., Bash, PowerShell) to directly manage infrastructure through cloud provider CLIs (e.g., Azure CLI, AWS CLI). Requires explicit instructions for every action.

* *Example*: az group create in Azure CLI.

**Declarative IaC Languages**: Define the desired end state of infrastructure, and the tool handles the necessary steps. Common tools include Terraform, Ansible, and Puppet.

* *Example*: Terraform configuration to create a resource group.

**Programmatic IaC Languages**: Leverage general-purpose programming languages (e.g., TypeScript, Python) for infrastructure definition, offering more flexibility and logic control.

* *Example*: Using TypeScript with Terraform CDK to create resources.

**Tools & Use Cases**:

* Scripting tools are best for granular control.
* Declarative tools are easier for straightforward infrastructure setup.
* Programmatic tools offer flexibility and are suitable for complex environments.

**10**. The IaC topology in a cloud infrastructure, IaC is divided into several typologies:

• Deploying and provisioning the infrastructure

• Server configuration and templating

• Containerization

• Configuration and deployment in Kubernetes

**Deploying and Provisioning the Infrastructure**: This involves defining and provisioning cloud resources (e.g., VMs, storage, networks) automatically using IaC tools like Terraform or AWS CloudFormation. It focuses on creating and managing the basic infrastructure.

**Server Configuration and Templating**: This involves automating the configuration of servers after they are provisioned, ensuring consistency in setup and deployment. Tools like Ansible, Puppet, and Chef are used to define configuration templates for servers and services.

**Containerization**: This focuses on defining and managing the deployment of containers (e.g., Docker) on infrastructure. IaC tools like Docker Compose or Kubernetes YAML files help automate container orchestration and management.

**Configuration and Deployment in Kubernetes**: This refers to managing and automating the deployment, scaling, and management of containerized applications in Kubernetes clusters. Tools like Helm, Kubernetes manifests, and Terraform are used to manage the infrastructure and application lifecycle within Kubernetes.

## 11. Best Practices for Infrastructure as Code (IaC) in a DevOps Pipeline

1. **Version Control:** Manage IaC scripts and configurations like application code by storing them in version control systems (e.g., Git). This facilitates collaboration, tracks changes, and maintains an infrastructure history.
   * **Why it matters:** Enables rollbacks, auditing, and ensures infrastructure consistency.
2. **Modular Design:** Structure IaC into reusable modules and components to improve maintainability, scalability, and efficiency.
   * **Key benefit:** Minimizes code duplication and simplifies management of complex environments.
3. **Automated Testing:** Use tools like Test Kitchen, Inspec, or Terraform validate to test IaC scripts for syntax errors, security vulnerabilities, and misconfigurations.
   * **Impact:** Detects issues early, ensuring infrastructure stability before deployment.
4. **Immutable Infrastructure:** Adopt an immutable infrastructure approach where resources are never modified post-deployment. Instead, replace them when updates are required (e.g., using containerization or re-provisioning).
   * **Advantage:** Increases stability, consistency, and minimizes human errors.
5. **CI/CD Integration for IaC Validation:** Embed infrastructure validation within CI/CD pipelines to automate testing and deployment of configurations before they reach production.
   * **Why it’s important:** Guarantees that every infrastructure change is validated before going live.
6. **State Management:** Properly manage infrastructure states, particularly when using tools like Terraform. Store state remotely with locking mechanisms (e.g., Terraform Cloud, S3 with DynamoDB) to enable collaboration and prevent conflicts.
   * **Benefit:** Ensures that the latest state is accessible to all team members, reducing the risk of inconsistencies.