DevOps

[Devops Intro 2](#_Toc194174646)

[Software Development Life Cycle (SDLC) & PhonePe Application Development 2](#_Toc194174647)

[Comparison of Waterfall and Agile Methodologies 4](#_Toc194174648)

[What is DevOps? 5](#_Toc194174649)

[DevOps Advantages 6](#_Toc194174650)

[DevOps Work Flow 8](#_Toc194174651)

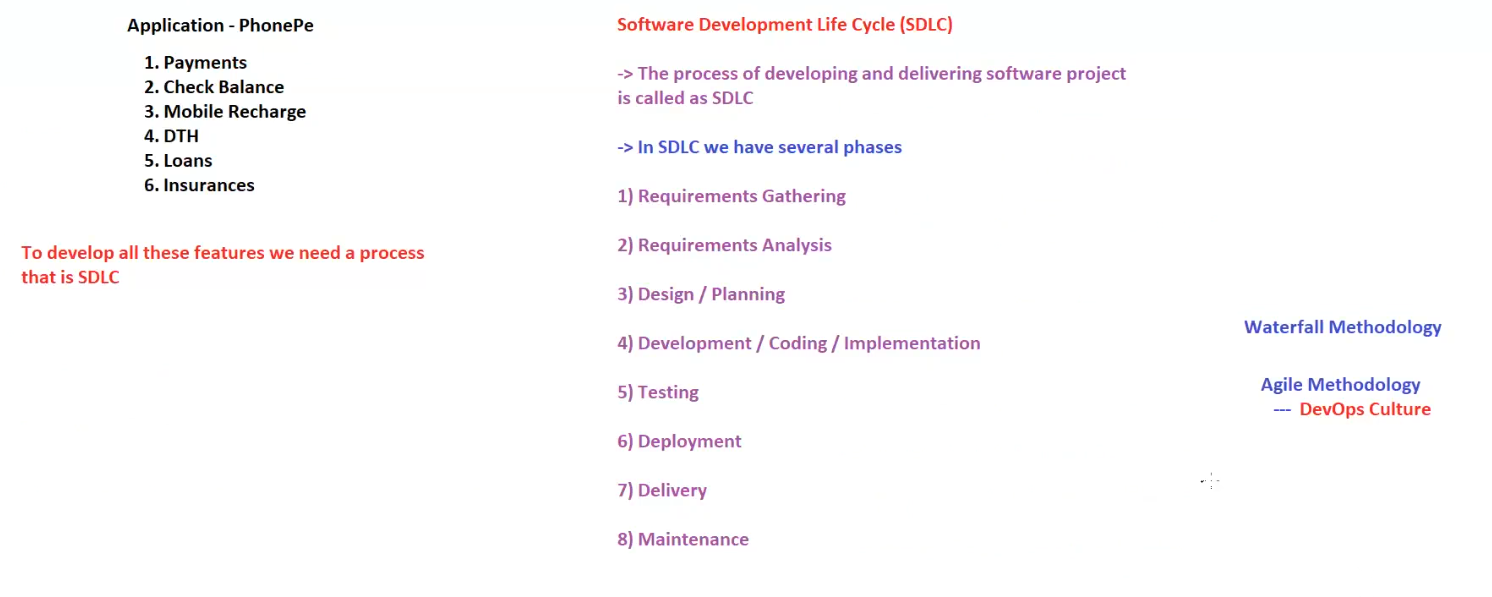
[Stages in DevOps Lifecycle 9](#_Toc194174652)

[CI/CD Pipeline 12](#_Toc194174653)

[CI/CD Pipeline and Deployment Architecture 14](#_Toc194174654)

# Devops Intro

## Software Development Life Cycle (SDLC) & PhonePe Application Development



**1. Application - PhonePe**

PhonePe is a digital payments and financial services application that provides various features, including:

1. **Payments** – Send and receive money securely.
2. **Check Balance** – View available account balance.
3. **Mobile Recharge** – Recharge prepaid mobile numbers.
4. **DTH** – Recharge Direct-to-Home (DTH) television services.
5. **Loans** – Avail different types of loans.
6. **Insurances** – Purchase and manage insurance policies.

**To develop all these features, a structured process is required, which is the SDLC (Software Development Life Cycle).**

**2. Software Development Life Cycle (SDLC)**

SDLC is a systematic process used to develop and deliver a software project efficiently.

**Phases of SDLC**

The SDLC consists of several phases:

1. **Requirements Gathering**
   * Identifying the business needs and objectives.
   * Collecting detailed requirements from stakeholders.
2. **Requirements Analysis**
   * Analyzing the feasibility of requirements.
   * Identifying potential challenges and solutions.
3. **Design / Planning**
   * Creating software architecture and system design.
   * Planning the technology stack and system components.
4. **Development / Coding / Implementation**
   * Writing the actual code based on the design.
   * Implementing the planned features and functionalities.
5. **Testing**
   * Performing unit testing, integration testing, and system testing.
   * Ensuring the software works as expected without defects.
6. **Deployment**
   * Releasing the software to production.
   * Configuring and setting up necessary infrastructure.
7. **Delivery**
   * Making the software available to end users.
   * Ensuring smooth transition and user adoption.
8. **Maintenance**
   * Regular updates, bug fixes, and performance enhancements.
   * Providing support for evolving user needs.

**3. Software Development Methodologies**

There are different methodologies used in SDLC, including:

**1. Waterfall Methodology**

* A sequential approach where each phase is completed before moving to the next.
* Suitable for projects with well-defined requirements.

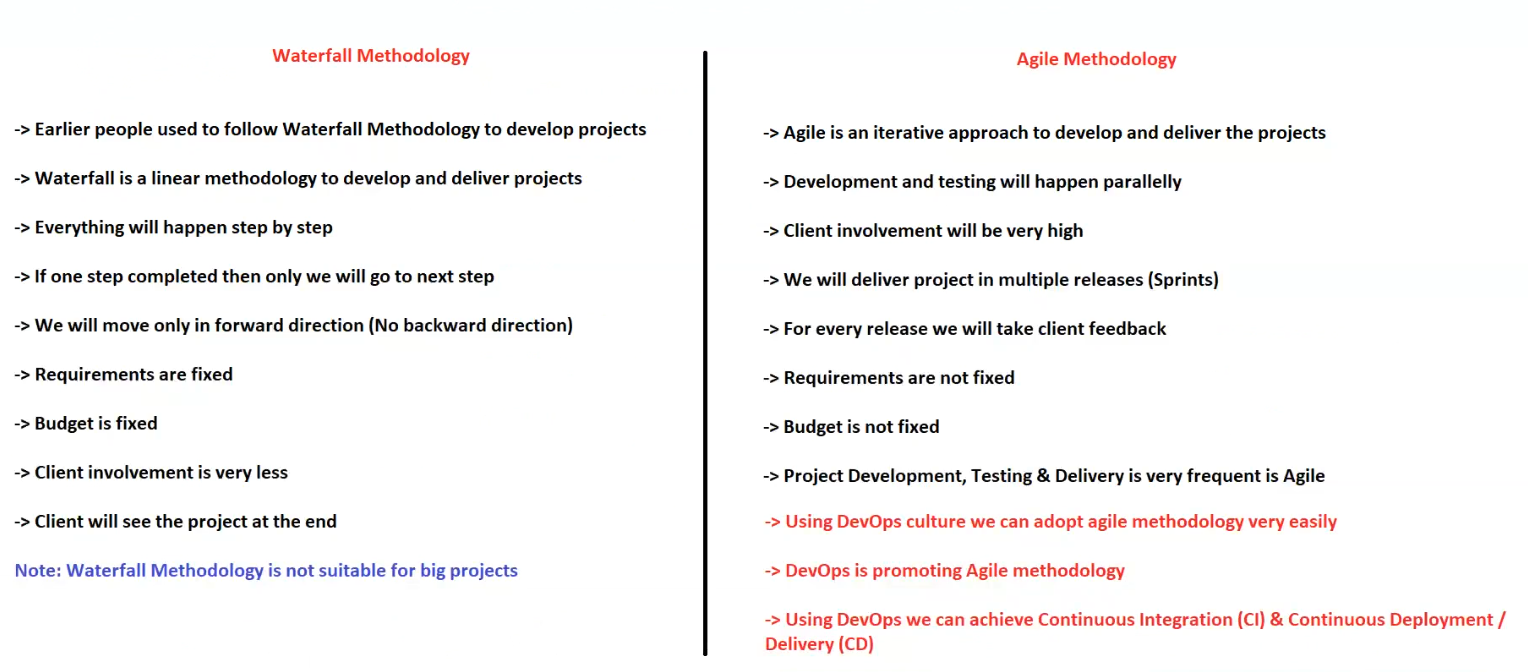
**2. Agile Methodology**

* An iterative and flexible approach to software development.
* Involves continuous feedback, collaboration, and incremental releases.

**→ DevOps Culture (Part of Agile)**

* Focuses on automation, continuous integration, and continuous deployment.
* Ensures faster delivery and improved collaboration between development and operations teams.

## Comparison of Waterfall and Agile Methodologies



**1. Waterfall Methodology**

The **Waterfall Model** is a traditional software development methodology that follows a sequential, step-by-step process.

**Key Characteristics:**

* Earlier, people commonly used the **Waterfall methodology** for software development.
* It is a **linear approach** to develop and deliver projects.
* Everything happens **step by step** in a structured manner.
* A **new step begins only after the previous step is completed** (no overlap).
* **No backward movement**—progress happens only in a forward direction.
* **Requirements are fixed** at the beginning of the project.
* **Budget is fixed** and predefined.
* **Client involvement is minimal**—they only see the project at the end.
* **Not suitable for large-scale projects** due to its rigid nature.

**Note:** Waterfall methodology is **not suitable for big projects** due to its inflexible and sequential nature.

**2. Agile Methodology**

The **Agile Model** is a modern, iterative approach to software development that emphasizes flexibility, client involvement, and continuous improvement.

**Key Characteristics:**

* Agile is an **iterative** approach that allows flexibility in development and delivery.
* **Development and testing happen in parallel** rather than sequentially.
* **Client involvement is high**, with frequent feedback loops.
* Projects are delivered in **multiple smaller releases (Sprints)** instead of a single final product.
* **Client feedback is taken at every release** to improve the product continuously.
* **Requirements are not fixed**—they evolve based on client needs.
* **Budget is flexible** and may change based on project scope adjustments.
* Agile ensures **frequent development, testing, and delivery cycles**.

**3. Role of DevOps in Agile**

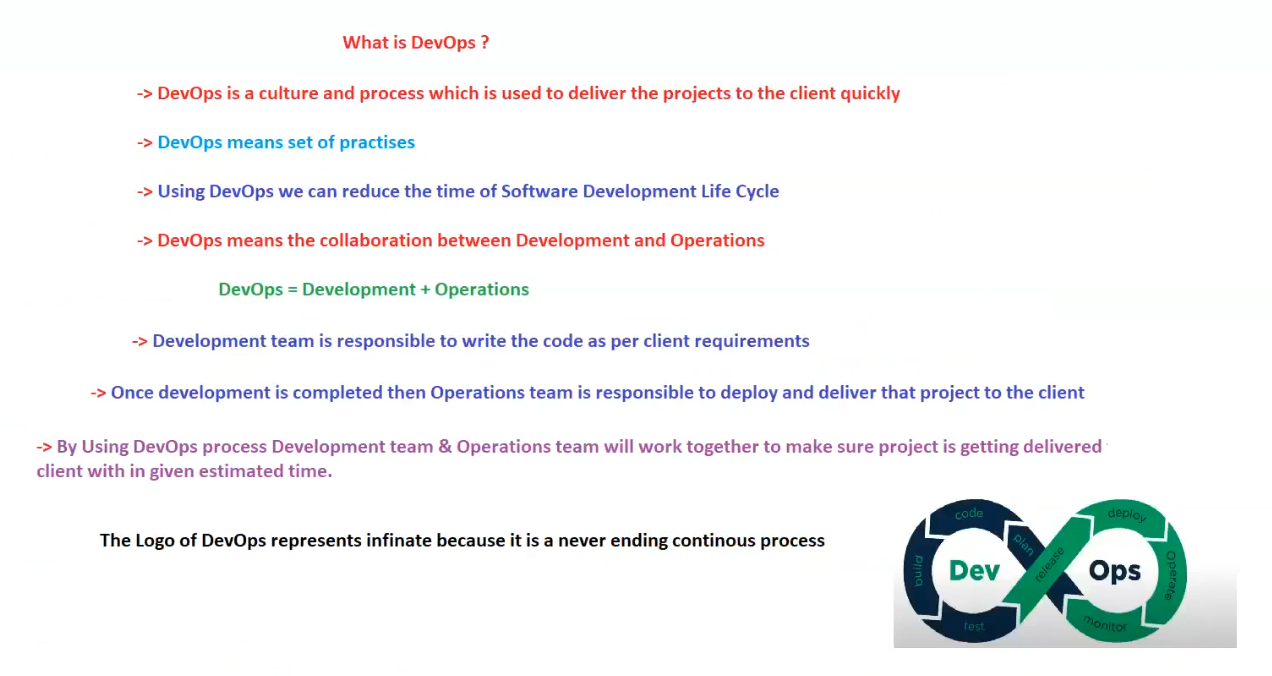
* **DevOps culture** helps in adopting Agile methodology more efficiently.
* **DevOps promotes Agile methodology** by enabling automation and continuous integration.
* **Using DevOps, we can achieve Continuous Integration (CI) & Continuous Deployment (CD).**

**Conclusion:**

* **Waterfall is suitable for small, well-defined projects with fixed requirements.**
* **Agile is more flexible, making it ideal for large-scale and evolving projects.**
* **DevOps further enhances Agile by streamlining development, integration, and deployment.**

🚀 **Agile + DevOps** = Faster Development, Higher Efficiency, and Continuous Improvement!

## What is DevOps?



**Definition:**

* DevOps is a **culture and process** that helps deliver projects to clients **quickly and efficiently**.
* It is a **set of practices** aimed at improving collaboration between development and operations teams.
* **Using DevOps, we can reduce the time required in the Software Development Life Cycle (SDLC).**
* DevOps ensures **seamless collaboration between Development and Operations teams** to improve efficiency.

**DevOps Formula:**

📌 **DevOps = Development + Operations**

**Roles & Responsibilities in DevOps**

**1. Development Team Responsibilities**

* Writing code as per client requirements.
* Implementing features and functionalities.
* Ensuring smooth integration of new changes.

**2. Operations Team Responsibilities**

* Deploying and delivering the project to the client.
* Managing infrastructure and performance.
* Ensuring system reliability and continuous monitoring.

**3. How DevOps Helps?**

* **Brings development and operations together** for faster project delivery.
* Ensures projects are delivered **on time** within the given estimated schedule.
* Reduces development lifecycle delays and **improves software quality**.

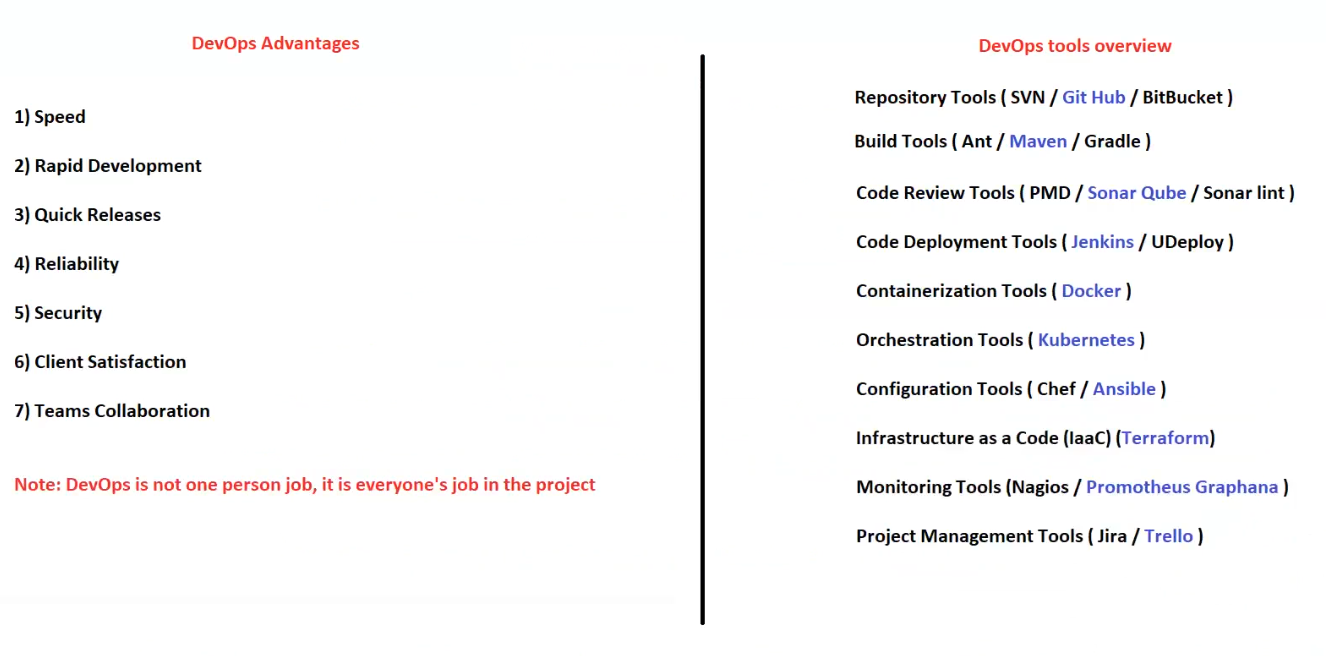
**DevOps Logo & Its Meaning**

* The **DevOps logo represents infinity (∞)** because it is a **continuous, never-ending process**.
* It signifies continuous **integration, delivery, deployment, monitoring, and improvement**.

**Key Takeaways:**

✅ DevOps **enhances collaboration** between teams.  
✅ **Speeds up project delivery** and reduces deployment failures.  
✅ Enables **continuous integration & deployment (CI/CD)**.  
✅ Supports **automation, monitoring, and infrastructure as code (IaC)**.

## DevOps Advantages



1️ **Speed** – Faster development and deployment.  
2️ **Rapid Development** – Continuous integration and delivery (CI/CD).  
3️ **Quick Releases** – Frequent software updates with minimal downtime.  
4️ **Reliability** – Ensures stability, monitoring, and rollback mechanisms.  
5️ **Security** – Automated security checks and compliance integration.  
6️ **Client Satisfaction** – Faster response to client needs and feedback.  
7️ **Team Collaboration** – Enhanced cooperation between Development & Operations teams.

📌 **Note:** DevOps is not a one-person job; it requires **collaboration from everyone** in the project.

**DevOps Tools Overview**

**1. Repository Tools**

* **SVN**, **GitHub**, **BitBucket** – Version control and source code management.

**2. Build Tools**

* **Ant**, **Maven**, **Gradle** – Automate the build process.

**3. Code Review Tools**

* **PMD**, **SonarQube**, **SonarLint** – Code analysis for quality and security.

**4. Code Deployment Tools**

* **Jenkins**, **UDeploy** – Automate software deployment and CI/CD pipelines.

**5. Containerization Tools**

* **Docker** – Package and run applications in isolated environments.

**6. Orchestration Tools**

* **Kubernetes** – Automate container deployment, scaling, and management.

**7. Configuration Management Tools**

* **Chef**, **Ansible** – Automate server setup, configuration, and updates.

**8. Infrastructure as Code (IaC) Tools**

* **Terraform** – Manage infrastructure through code.

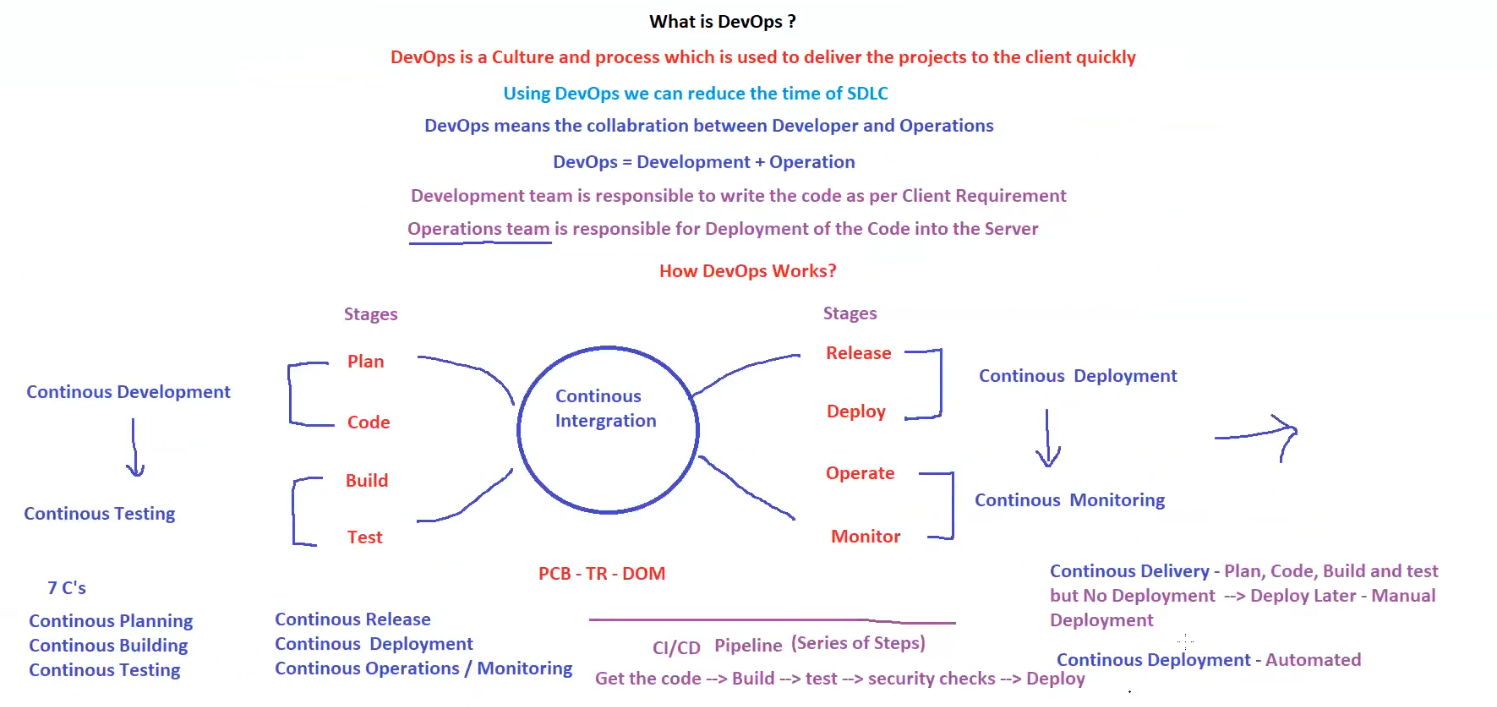
**9. Monitoring Tools**

* **Nagios**, **Prometheus**, **Grafana** – Track system performance and detect issues.

**10. Project Management Tools**

* **Jira**, **Trello** – Plan, track, and manage software development projects.

## DevOps Work Flow



**What is DevOps?**

🔹 **DevOps is a culture and process** that helps deliver projects quickly.  
🔹 **Reduces SDLC (Software Development Life Cycle) time.**  
🔹 **Collaboration between Development and Operations teams.**  
🔹 **DevOps = Development + Operations.**  
🔹 The **Development team** writes the code based on client requirements.  
🔹 The **Operations team** handles deployment and server management.

**How DevOps Works?**

**1️⃣ Continuous Integration (CI)**

🔹 Involves merging code changes frequently.

**2️⃣ Continuous Development Stages**

🔹 **Plan** → **Code** → **Build** → **Test**

**3️⃣ Continuous Deployment (CD) Stages**

🔹 **Release** → **Deploy** → **Operate** → **Monitor**

📌 **Key DevOps Practices:**

* **Continuous Development** – Iterative coding and planning.
* **Continuous Testing** – Automated test execution.
* **Continuous Integration** – Frequent code merges and builds.
* **Continuous Deployment** – Automated code deployment.
* **Continuous Monitoring** – Performance tracking and issue detection.

📌 **7 C’s of DevOps:**

1. Continuous Planning
2. Continuous Building
3. Continuous Testing
4. Continuous Release
5. Continuous Deployment
6. Continuous Operations
7. Continuous Monitoring

**CI/CD Pipeline – A Series of Steps**

🔹 **Get the code** → **Build** → **Test** → **Security checks** → **Deploy**

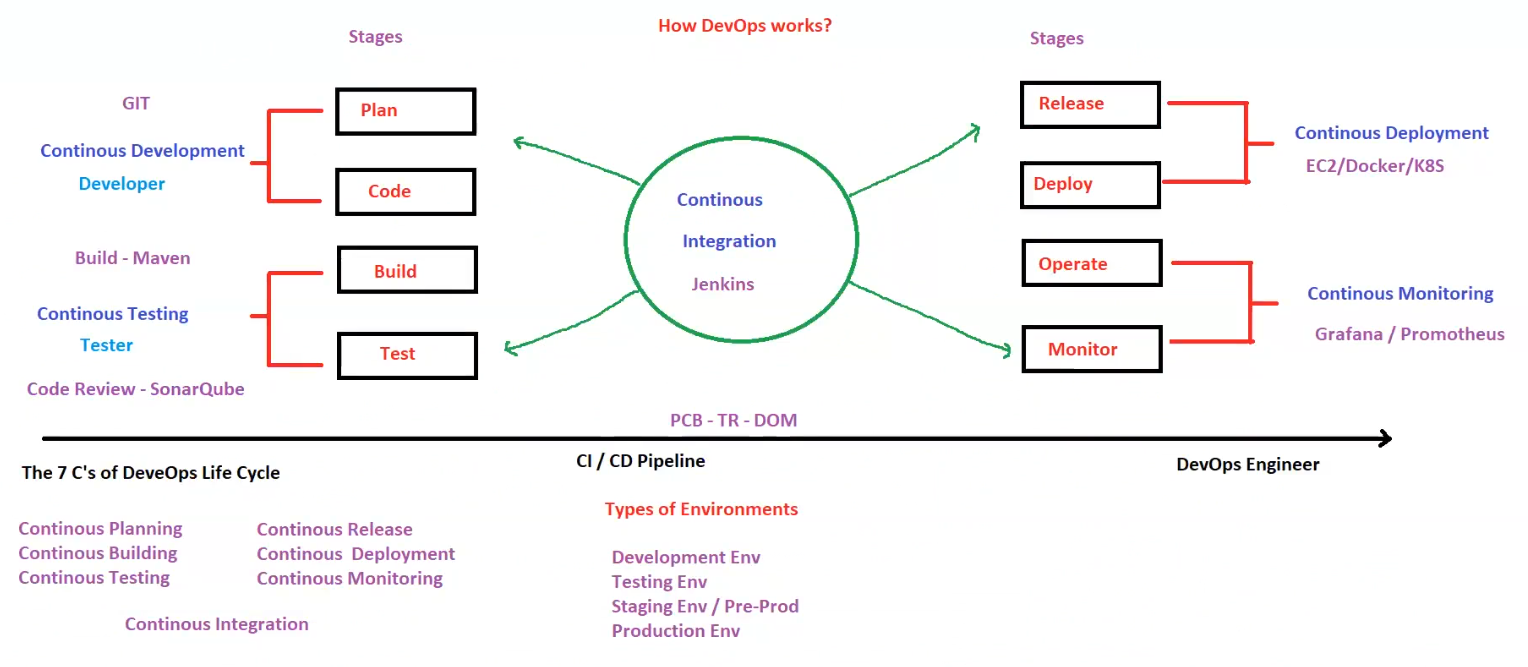
📌 **Continuous Delivery vs. Continuous Deployment:**

* **Continuous Delivery** – Automates build and test, but deployment is manual.
* **Continuous Deployment** – Fully automated deployment.

🚀 **DevOps enables faster, reliable, and more efficient software development!**

**This image provides a structured overview of the DevOps Workflow and CI/CD Pipeline. Here’s a breakdown:**

## Stages in DevOps Lifecycle



**1️⃣ Continuous Development**

* **Tools Used: GIT**
* **Roles: Developer**
* **Stages:**
  + **Plan**
  + **Code**

**2️⃣ Continuous Testing**

* **Tools Used: Maven (Build), SonarQube (Code Review)**
* **Roles: Tester**
* **Stages:**
  + **Build**
  + **Test**

**3️⃣ Continuous Integration**

* **Tool Used: Jenkins**
* **Purpose: Automates builds, tests, and integration of code changes.**

**4️⃣ Continuous Deployment**

* **Tools Used: EC2, Docker, Kubernetes**
* **Stages:**
  + **Release**
  + **Deploy**

**5️⃣ Continuous Monitoring**

* **Tools Used: Grafana, Prometheus**
* **Stages:**
  + **Operate**
  + **Monitor**

**CI/CD Pipeline & DevOps Engineer Role**

* **Pipeline Flow:**
  + **Get Code → Build → Test → Security Checks → Deploy**
* **DevOps Engineer Responsibilities:**
  + **Manages CI/CD pipeline**
  + **Ensures automation of build, test, and deployment**

**7 C’s of DevOps Lifecycle**

1. **Continuous Planning**
2. **Continuous Building**
3. **Continuous Testing**
4. **Continuous Integration**
5. **Continuous Release**
6. **Continuous Deployment**
7. **Continuous Monitoring**

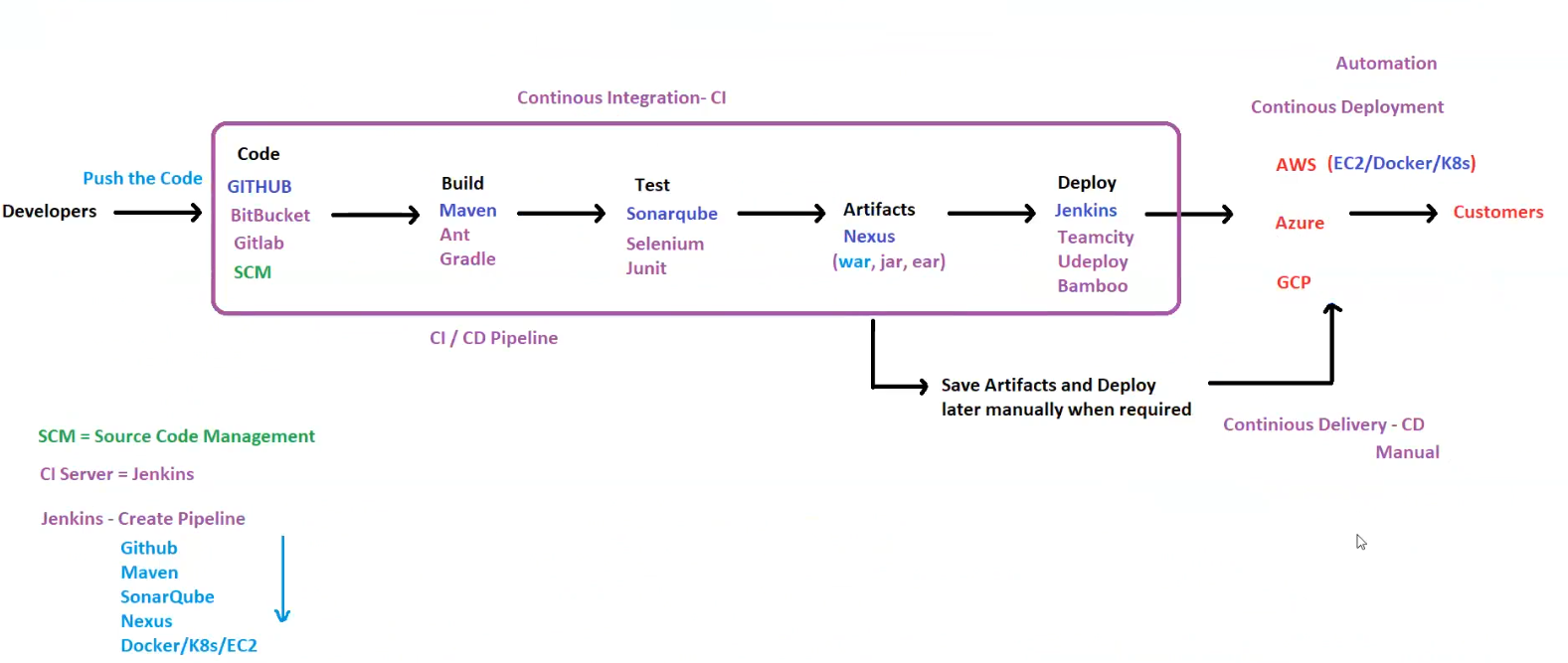
**Types of Environments**

1. **Development Environment**
2. **Testing Environment**
3. **Staging/Pre-Production Environment**
4. **Production Environment**

**🚀 Summary**

**This diagram outlines how DevOps automates development, testing, integration, deployment, and monitoring, ensuring a smooth CI/CD pipeline with tools like Jenkins, Docker, Kubernetes, SonarQube, and Prometheus.**

## CI/CD Pipeline



**1. Overview of CI/CD Pipeline**

* **CI/CD (Continuous Integration/Continuous Deployment) pipeline** is used for automating the software development lifecycle.
* The process includes:
  + **Continuous Integration (CI)**: Developers push code, build, test, and store artifacts.
  + **Continuous Delivery (CD)**: Artifacts are saved and deployed manually when required.
  + **Continuous Deployment (CD)**: Fully automated deployment to cloud platforms.

**2. CI/CD Pipeline Flow**

**Step 1: Developers Push Code**

* Developers commit and push code to **Source Code Management (SCM)** repositories.
* Supported SCM tools:
  + **GitHub**
  + **BitBucket**
  + **GitLab**
  + **SCM (General source code management systems)**

**Step 2: Build Process**

* Once the code is pushed, it goes through a **build process** using:
  + **Maven**
  + **Ant**
  + **Gradle**
* This step compiles the code and prepares it for further testing.

**Step 3: Testing**

* The built code undergoes testing using:
  + **SonarQube** (Static Code Analysis)
  + **Selenium** (Automated UI Testing)
  + **JUnit** (Unit Testing)

**Step 4: Storing Artifacts**

* After testing, the **artifacts** (built files) are stored in:
  + **Nexus** (Artifact repository)
  + Artifacts types:
    - .war (Web Application Archive)
    - .jar (Java Archive)
    - .ear (Enterprise Archive)

**Step 5: Deployment**

* Deployment can be automated or manual using:
  + **Jenkins** (Most commonly used CI/CD tool)
  + **TeamCity**
  + **UDeploy**
  + **Bamboo**
* Artifacts can be stored and deployed later when required.

**Step 6: Deployment to Cloud Platforms**

* The final step is deploying applications to cloud environments:
  + **AWS** (EC2, Docker, Kubernetes)
  + **Azure**
  + **GCP (Google Cloud Platform)**

**3. Automation and Deployment Strategies**

* **Continuous Deployment**: Fully automated deployment to cloud.
* **Continuous Delivery**: Artifacts saved and deployed **manually** when needed.
* **Automation tools**:
  + **Jenkins Pipeline** integrates:
    - **GitHub**
    - **Maven**
    - **SonarQube**
    - **Nexus**
    - **Docker/K8s/EC2**

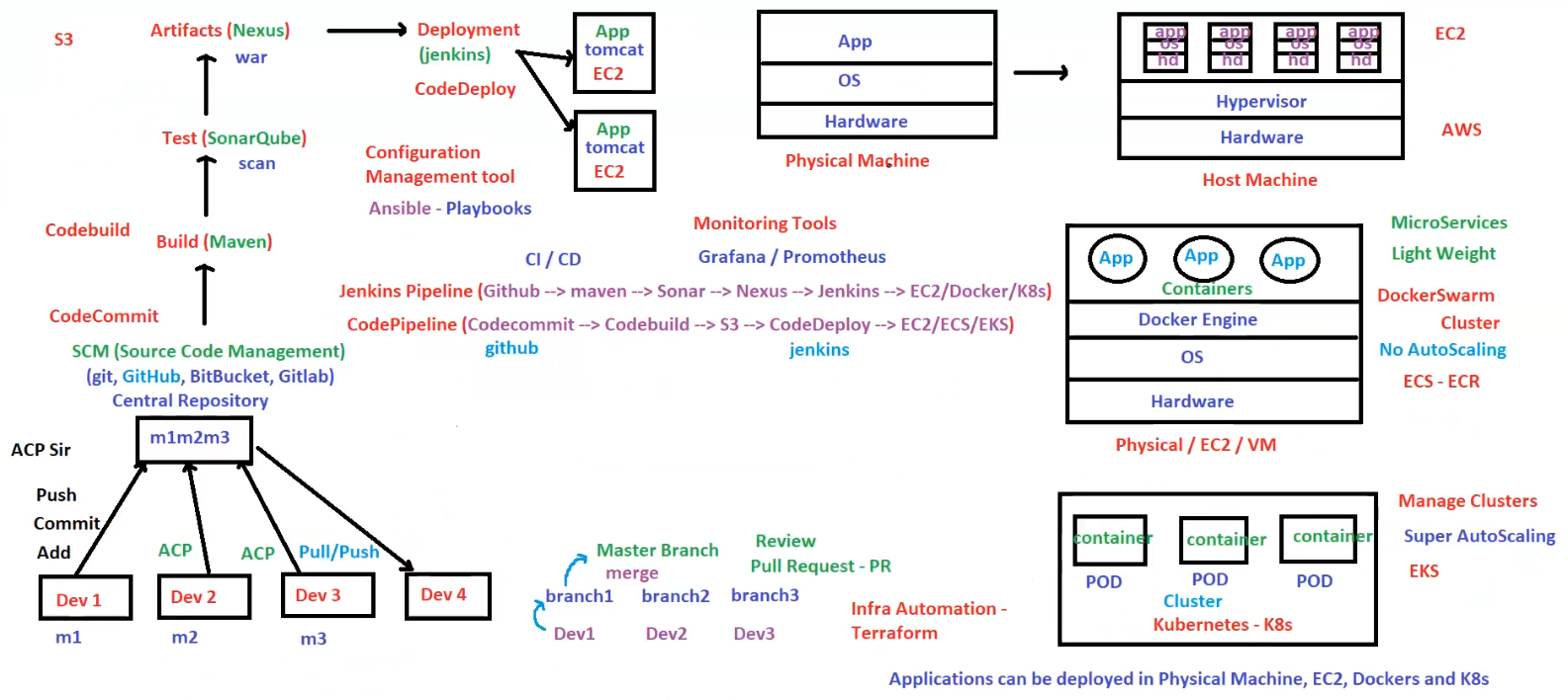
**4. Key Terms & Definitions**

* **SCM (Source Code Management)**: Version control for source code.
* **CI (Continuous Integration)**: Automating build and testing.
* **CD (Continuous Delivery/Deployment)**:
  + **Continuous Delivery**: Manual deployment after verification.
  + **Continuous Deployment**: Fully automated deployment.
* **Artifacts**: Build outputs stored in Nexus.

**5. Summary**

* **CI/CD pipeline automates software development** by integrating source code, building, testing, storing artifacts, and deploying to production.
* **Tools used**:
  + **SCM**: GitHub, GitLab, BitBucket
  + **Build**: Maven, Ant, Gradle
  + **Testing**: SonarQube, Selenium, JUnit
  + **Artifact Repository**: Nexus
  + **Deployment**: Jenkins, TeamCity, UDeploy, Bamboo
  + **Cloud Providers**: AWS, Azure, GCP

## CI/CD Pipeline and Deployment Architecture



**1. CI/CD Pipeline Overview**

The CI/CD pipeline automates software development, testing, and deployment across multiple environments, ensuring reliability and efficiency.

**2. Development Workflow**

* **Developers (Dev 1, Dev 2, Dev 3, Dev 4)** work on code in separate branches (branch1, branch2, branch3).
* Developers **Push, Commit, and Add** changes to a **Central Repository (SCM)**.
* **SCM (Source Code Management)** tools:
  + **Git**
  + **GitHub**
  + **BitBucket**
  + **GitLab**

**3. Build and Testing**

* **CodeCommit** → **CodeBuild** → **Maven (Build Process)**
* **Testing using SonarQube (Static Code Analysis)**:
  + SonarQube scans the code for security vulnerabilities and code quality.

**4. Storing and Managing Artifacts**

* **Nexus Repository** stores build artifacts (.war files).
* Artifacts are uploaded to **Amazon S3** for later deployment.

**5. Deployment Process**

* **Jenkins Deployment Pipeline**:
  + Jenkins pulls code from **GitHub** and processes it through:
    - **Maven** (Build)
    - **SonarQube** (Scan)
    - **Nexus** (Artifact Repository)
    - **Jenkins Deployment** to:
      * **EC2 Instances**
      * **Docker Containers**
      * **Kubernetes (K8s) Clusters**
* **AWS CodePipeline**:
  + **CodeCommit** → **CodeBuild** → **S3** → **CodeDeploy** → **EC2, ECS, or EKS**

**6. Deployment Environments**

**A. Physical Machines**

* Traditional hardware with OS and applications running directly on physical servers.

**B. Virtual Machines (VMs) on Host Machines**

* **EC2 Instances on AWS**
  + Multiple VMs running on a **Hypervisor**.
  + Each VM has:
    - **App**
    - **OS**
    - **Hardware Resources**
  + Used for **scalability** and **isolation**.

**C. Docker Containers**

* **Docker Engine** runs **lightweight containers** instead of full VMs.
* Benefits:
  + **Microservices-based architecture**
  + **Faster startup times**
  + **Resource efficiency**
* **DockerSwarm Cluster**:
  + Containers are grouped into clusters.
  + **No AutoScaling** feature in Docker Swarm.

**D. Kubernetes (K8s) Clusters**

* Applications are **containerized** and managed using Kubernetes.
* Key features:
  + **PODs**: Smallest deployable units in K8s.
  + **Cluster**: Group of PODs managing application instances.
  + **Super AutoScaling**: Kubernetes automatically scales applications based on demand.
  + **EKS (Elastic Kubernetes Service)**: Managed Kubernetes service in AWS.

**7. Infrastructure Automation**

* **Terraform** is used for automating infrastructure provisioning.

**8. Monitoring & Configuration Management**

* **Configuration Management Tool**: **Ansible (Playbooks)**
* **Monitoring Tools**:
  + **Grafana**
  + **Prometheus**

**9. Summary**

* **SCM**: GitHub, GitLab, BitBucket
* **Build**: Maven, CodeBuild
* **Testing**: SonarQube
* **Artifact Repository**: Nexus, S3
* **Deployment Tools**: Jenkins, CodeDeploy, CodePipeline
* **Deployment Targets**:
  + Physical Machines
  + EC2 Instances (VMs)
  + Docker Containers
  + Kubernetes (K8s) Clusters
* **Infrastructure Automation**: Terraform
* **Monitoring**: Grafana, Prometheus

**Final Thought**

This pipeline integrates multiple DevOps tools for a complete CI/CD workflow, ensuring automation, scalability, and efficient application delivery. 🚀