# **Introduction to DevOps & CI/CD**

## **What is DevOps?**

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

### **Key Principles of DevOps:**

* Collaboration between development and operation teams
* Automation of processes wherever possible
* Continuous Improvement through feedback loops
* Customer Centric approach to development
* End-to-End responsibility from development to deployment

### **The DevOps Lifecycle**

1. Plan – Define requirements and track progress
2. Code – Software development and review
3. Build – Compile code into executable form
4. Test – Automated testing to verify functionality
5. Release – Prepare for deployment
6. Deploy – Move to Production environment
7. Operate – Monitor and manage the production environment
8. Monitor – Gather performance data and user feedback

### **What is CI/CD?**

CI/CD stands for Continuous Integration and Continuous Delivery/Deployment, which are key practices in DevOps.

### **Continuous Integration (CI)**

* Developers frequently merge code changes into a central repository
* Each merge triggers automated builds and tests
* Early detection of Integration issues
* Requires Comprehensive test suites

### **Continuous Delivery (CD)**

* Ensures code is always in a deployable state
* Automated deployment to staging environments
* Manual approval for production deployment

### **Benefits of DevOps and CI/CD**

1. **Faster time to market –** More frequent releases
2. **Improved quality –** Early bug detection
3. **Reduced Risk –** Smaller changes are easier to troubleshoot
4. **Greater efficiency –** Automation reduces manual work
5. **Better collaboration –** Shared responsibility across teams
6. **Increased reliability –** Consistent deployment processes

### **Common DevOps Tools**

* **Version Control:** Git, GitHub, GitLab, Bitbucket
* **CI/CD Servers:** Jenkins, CicleCI, Travis CI, GitHub Actions
* **Configuration Management:** Ansible, Chef, Puppet
* **Containerization:** Docker, Kubernetes
* **Monitoring:** Prometheus, Grafana, ELK Stack
* **Cloud Platforms:** AWS, Azure, Google Cloud

## **Traditional vs. DevOps Workflow: Key Differences**

The main difference between **Traditional (Waterfall/Siloed) workflows** and **DevOps workflows** lies in their approach to software development, collaboration, and deployment. Below is a detailed comparison:

**1. Development Approach**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Traditional Workflow** | **DevOps Workflow** |
| **Methodology** | Follows a **linear (Waterfall)** approach (Requirements -> Design -> Development -> Testing -> Deployment) | Follows an Agile & Iterative approach with continuous integration and delivery. |
| **Team Structure** | **Siloed teams** (Dev, QA, Ops work separately) | Cross-functional teams (Dev + Ops Collaborate closely) |
| **Feedback Loop** | Long feedback cycles (weeks/months) | Short feedback loops (minutes/hours via CI/CD) |

**2. Release & Deployment**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Traditional Workflow** | **DevOps Workflow** |
| **Release Frequency** | Infrequent (months/Years) | **Frequent releases** (daily/hourly) |
| **Deployment** | Manual, error-prone and slow | **Automated** (CI/CD pipeline handle builds, tests, and deployments) |
| **Risk of Failures** | High (big releases = big failures) | **Low** (small, incremental changes) |

**3. Testing & Quality Assurance**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Traditional Workflow** | **DevOps Workflow** |
| **Testing Phase** | Separate phase after development (slow feedback) | Continuous Testing (automated tests |
| **Bug Detection** | Bugs found late (costly to fix) | Bugs caught early (shift-left testing) |

**4. Infrastructure & Operations**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Traditional Workflow** | **DevOps Workflow** |
| Infrastructure | Manual setup, static servers | Infrastructure as Code (IaC) (Terraform, Ansible) |
| Scaling | Difficult (requires manual intervention) | Auto-scaling (Cloud, Kubernetes) |
| Monitoring | Reactive (issues detected after failure) | Proactive monitoring (logs, metrics, alerts |

**5. Culture & Collaboration**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Traditional Workflow** | **DevOps Workflow** |
| Team mindset | “Throw it over the wall” (Dev -> QA -> Ops) | Shared responsibility (Dev + Ops work together) |
| Blame Game | Common (Ops blames Dev for bugs, Dev blames Ops for environment issues) | No blame culture (focus on automation & collaboration) |

# **Version Control with Git & GitHub**