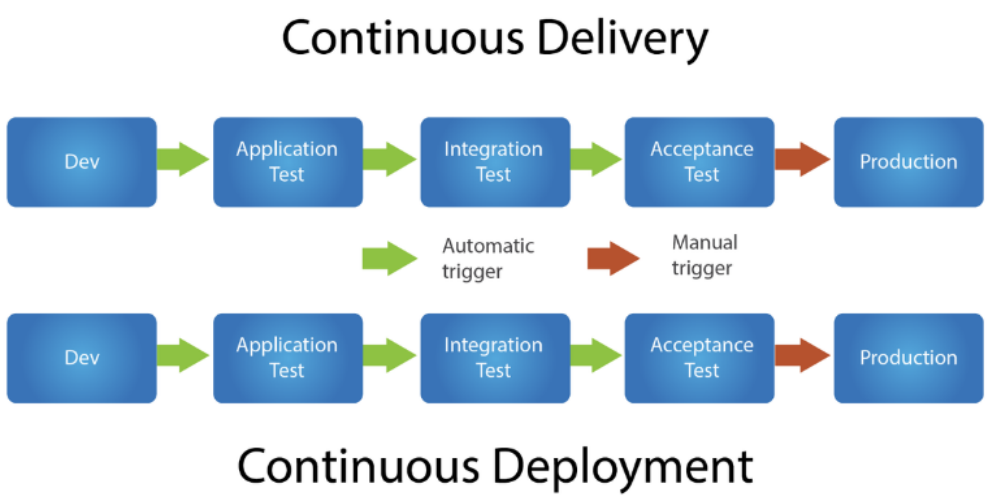
Differences Between Continuous Delivery and Continuous Deployment

Lesson Plan

PW SKILLS



Continuous Delivery (CD)

Definition: Continuous Delivery is a software development practice where code changes are automatically built, tested, and prepared for a release to production. However, the deployment to production is done manually.

Key Characteristics:

1. Automated Testing: Continuous Delivery involves extensive automated testing (unit, integration, system, and acceptance tests) to ensure that the code is always in a deployable state.

2. Manual Approval: Even though the code is ready for deployment at any time, the actual release to production requires manual approval or triggering. This step allows teams to review and decide when to release the new code.

3. Release on Demand: With Continuous Delivery, the software can be released to production at any time. This allows for flexibility in deciding when to push new features or updates live.

4. Pipeline Integration: The Continuous Delivery pipeline integrates various stages of the development process, from code commit to production readiness, ensuring a consistent and reliable release process.

Benefits:

- **Flexibility:** Teams can choose the most appropriate time to deploy, aligning releases with business needs.
- **Control:** Manual approval adds an extra layer of control, which can be important for critical or large-scale deployments.

- **Reduced Risk:** Because the code is always in a deployable state, the risk of introducing bugs in production is minimized.

Example:

- A team working on a financial application uses Continuous Delivery. The code passes all automated tests and is ready for production, but the team waits until a scheduled maintenance window to manually deploy the changes.

Continuous Deployment (CD)

Definition: Continuous Deployment is an extension of Continuous Delivery where every code change that passes automated tests is automatically deployed to production, without requiring manual approval.

Key Characteristics:

1. Fully Automated Pipeline: Continuous Deployment takes automation further by removing the manual approval step. Once the code passes all tests, it is automatically deployed to production.

2. High Frequency of Releases: Continuous Deployment allows for rapid, frequent releases to production, sometimes multiple times a day. This is particularly useful in environments where continuous updates are beneficial, such as web services or SaaS platforms.

3. Immediate Feedback: Changes are immediately reflected in production, providing real-time feedback and allowing for quick iterations based on user input or monitoring data.

4. Robust Monitoring: Continuous Deployment requires robust monitoring and alerting systems in place to detect and respond to any issues that arise after deployment.

Benefits:

- **Speed:** Continuous Deployment maximizes the speed of delivering new features, bug fixes, and improvements to users.
- **Efficiency:** It eliminates the bottleneck of manual approval, making the release process more efficient.
- **Continuous Innovation:** Teams can continuously innovate and release new features, staying ahead of competitors.

Example:

- A team developing a social media platform uses Continuous Deployment. Each new code commit that passes automated testing is deployed directly to production, allowing users to immediately experience new features and updates.

Key Differences

Aspect	Continuous Delivery	Continuous Deployment
Deployment Trigger	Manual approval required before production deployment	Automatic deployment to production without manual intervention
Deployment Frequency	Deployment is flexible, controlled by the team	Deployment is continuous and frequent, often multiple times a day
Release Control	Teams have control over when to deploy	Deployment is automatic, with less manual control
Use Case	Suitable for environments requiring manual checks or compliance requirements	Ideal for environments that benefit from rapid and frequent updates, like SaaS or web platforms
Risk Management	Allows for additional manual checks and risk assessments before deploying	Requires strong automated testing and monitoring to mitigate risks in real-time
Complexity	Generally simpler to implement due to manual control points	Requires a more advanced, fully automated pipeline with robust testing and monitoring