SKILLS | DevOps and Cloud Computing

Continuous Deployment

Objective

- Understand the importance of Continuous Deployment in DevOps workflows.
- Differentiate between Continuous Delivery and Continuous Deployment.
- Explore popular CD tools like Jenkins, GitLab CI, and Spinnaker.
- Design and implement a deployment pipeline for automating software releases.

PW SKILLS

Why CD is essential for modern software development

Let’s see

Continuous Delivery (CD) is essential for modern software development because it automates the deployment process, enabling teams to deliver high-quality software quickly, reliably, and consistently. Here are the key reasons why CD is crucial:

- Faster Delivery of Features
- Improved Code Quality
- Reduced Development Costs
- Minimized Downtime
- Enhanced Collaboration
- Adaptability to Market Needs

Pop Quiz

Q. How does Continuous Deployment contribute to software quality?

A

By allowing manual approval of all changes

B

By automating testing at each stage of the piline

Pop Quiz

Q. How does Continuous Deployment contribute to software quality?

A

By allowing manual approval of all changes

B

By automating testing at each stage of the piline

PW SKILLS

Highlight key benefits:

Key benefits

Here are the key benefits of Continuous Delivery (CD) highlighted:

1. Faster Release Cycles
2. Reduced Manual Intervention
3. Improved Software Stability
    - Automated Testing
    - Automated Deployment

Define Continuous Delivery:

Continuous Delivery

Continuous Delivery (CD) is a software development practice that automates the processes of building, testing, and packaging code to ensure it is always in a deployable state.

- However, it requires manual approval for deployment to production environments, allowing teams to maintain control over the final release while benefiting from automation in earlier stages

Continuous Deployment

Continuous Deployment is a fully automated software release process where code changes are directly deployed to production after passing all automated tests and inspections.

- It eliminates manual approval, ensuring rapid, consistent, and reliable delivery of updates to end users

PW SKILLS

Discuss pros and cons of both approaches.

Let’s discuss

Continuous Delivery

Pros:

- Control over releases
- Improved code quality
- Flexibility

Cons:

- Slower deployment speed
- Higher operational effort

Let’s discuss

Continuous Deployment

Pros:

- Faster release cycles
- Reduced manual effort
- Quick feedback loops

Cons:

- Higher risk of errors in production
- Complex setup requirements

PW SKILLS

Take A 5-Minute Break!

- Stretch and relax
- Hydrate
- Clear your mind
- Be back in 5 minutes

Introduce popular CD tools

Popular CD tools

Here's a brief introduction to three popular CD tools:

1. **Jenkins** is the most customizable and widely used automation server for CI/CD. It offers over 1,000 plugins, supports various version control systems, and provides highly flexible pipeline scripts. However, it requires significant setup and maintenance.
2. **GitLab CI/CD** is tightly integrated with GitLab repositories and offers robust continuous delivery features. It provides built-in CI/CD pipelines, Auto DevOps for automated setups, and Kubernetes integration. GitLab CI/CD is particularly strong in security and compliance features.
3. **Spinnaker** is an open-source, multi-cloud continuous delivery platform originally developed by Netflix and now backed by major tech companies. It excels in deploying across multiple cloud platforms, offering advanced deployment strategies and easy rollbacks.

PW SKILLS

Compare strengths and weaknesses of each tool

Let’s compare

Tool	Strengths	Weakness
Jenkins	Highly customizable with 1,800+ plugins.	Complex setup and maintenance.
	Widely adopted, extensive community support.	Older interface and steep learning curve.
	Scales well for large teams.	Requires external scripts for modern cloud deployments.
GitLab CI	All-in-one platform with native CI/CD, version control, and issue tracking.	Limited customization compared to Jenkins.
	Modern, user-friendly interface.	Primarily focused on GitLab ecosystem.
	Built-in security features and seamless integration.	
Spinnaker	Multi-cloud deployment capabilities.	Focused mainly on CD; requires integration with other CI tools.
	Advanced deployment strategies (e.g., blue/green, canary).	Steeper learning curve for multi-cloud setups.
	Easy rollbacks and strong monitoring tool integrations.	Smaller plugin ecosystem compared to Jenkins.

PW SKILLS

Explaining the structure of a deployment pipeline

Structure of a deployment pipeline

Build → Test → Deploy to Staging → Automated Tests → Deploy to Production.

1. **Build:**

The process begins with compiling the source code into executable artifacts. This stage includes dependency management and static code analysis to ensure the code adheres to standards and is free of basic errors.

2. **Test:**

Automated tests, such as unit tests and integration tests, are executed to validate the functionality, reliability, and correctness of the code. This ensures that only high-quality builds proceed further.

Structure of a deployment pipeline

3. Deploy to Staging:

The application is deployed to a staging environment that closely resembles production. This allows for thorough testing under realistic conditions, including performance and user acceptance testing.

4. Automated Tests:

Additional automated tests are run in the staging environment to verify the application's behavior under near-production conditions. These tests may include end-to-end, regression, and load testing.

5. Deploy to Production:

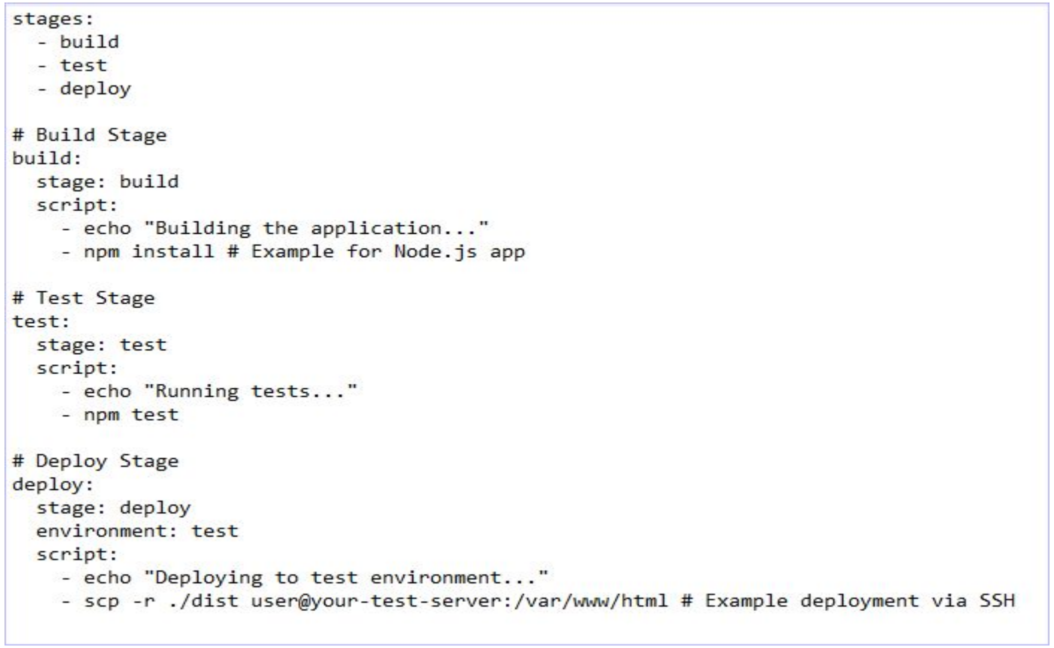
Once all tests pass and approvals (if required) are obtained, the application is deployed to the production environment. This stage often involves monitoring tools to ensure stability and detect any issues after deployment.

Demonstrating setting up a basic deployment pipeline

Let’s do it

1. Create ‘.gitlab-ci.yml’ File

The pipeline is defined in the ‘.gitlab-ci.yml’ file, which specifies the stages and jobs.



```
stages:
  - build
  - test
  - deploy

# Build Stage
build:
  stage: build
  script:
    - echo "Building the application..."
    - npm install # Example for Node.js app

# Test Stage
test:
  stage: test
  script:
    - echo "Running tests..."
    - npm test

# Deploy Stage
deploy:
  stage: deploy
  environment: test
  script:
    - echo "Deploying to test environment..."
    - scp -r ./dist user@your-test-server:/var/www/html # Example deployment via SSH
```

Let’s do it

2. Explanation of Pipeline Stages

- Build: Installs dependencies and prepares the application for deployment.
- Test: Runs automated tests to ensure code quality.
- Deploy: Transfers the built application to the test server using tools like scp or Docker.

3. Automating Deployment

- Ensure your server is configured to accept deployments (e.g., SSH keys are set up).
- Replace ‘./dist’ with your build directory and ‘/var/www/html’ with your server's web root.

4. Triggering the Pipeline

- Push changes to the GitLab repository. The pipeline will automatically execute for each commit, running through build, test, and deploy stages sequentially.

Time for case study!

Important

- Complete the post-class assessment
- Complete assignments (if any)
- Practice the concepts and techniques taught in this session
- Review your lecture notes
- Note down questions and queries regarding this session and consult the teaching assistants

Thanks !