Jenkins for CI/CD Pipeline: A Step-by-Step Guide

Subtitle: Streamlining Continuous Integration and Continuous Deployment

Presented by: [Santhosh Kiran J]

Introduction to CI/CD

- Definition of CI/CD
  - CI/CD stands for continuous integration and continuous delivery or deployment, and it's a set of practices that speed up and streamline the software development process:
  - Continuous integration (CI): Automatically and frequently integrating code changes into a shared source code repository
  - Continuous delivery (CD): A two-part process that involves integrating, testing, and delivering code changes
  - Continuous deployment: Automatically releasing updates into the production environment

Importance of CI/CD in modern software development

- **Improve code quality**

CI/CD helps to detect and fix issues early in the development cycle, reducing the risk of bugs and regressions.

- **Reduce downtime**

CI/CD automates the manual intervention traditionally needed to get new code into production, which minimizes downtime and speeds up code releases.

- **Enhance collaboration**

CI/CD encourages communication and teamwork among developers, operations, and quality assurance teams.

- **Improve user feedback**

CI/CD allows for quicker feedback loops with stakeholders, ensuring that the final product aligns closely with user expectations.

- **Improve security**

Automatic testing in CI/CD allows developers to address vulnerabilities earlier in the development process.

- **Improve resource management**

CI/CD allows development teams to focus on creating new features and improvements rather than spending time on manual testing and deployment processes.

Benefits of implementing CI/CD:

Continuous integration (CI) and continuous delivery (CD) can provide many advantages for organizations, including:

- **Faster time to market**

CI/CD can help organizations get their products to market faster.

- **Improved customer satisfaction**

CI/CD can help organizations reduce bugs and errors, which can lead to improved customer satisfaction and confidence.

- **Reduced risk**

CI/CD can help organizations reduce risk by identifying and fixing issues early in the development cycle.

- **Faster bug fixes**

CI/CD can help organizations fix bugs faster by making it easier to isolate and fix issues.

- **Improved code quality**

CI/CD can help organizations improve code quality by integrating code seamlessly and automating the deployment process.

- **Reduced costs**

CI/CD can help organizations reduce development costs by reducing the time it takes to get a product to market.

- **Easier maintenance and updates**

CI/CD can make it easier to maintain and update software by automating the deployment process.

- **Increased team transparency and accountability**

CI/CD can help increase team transparency and accountability by providing real-time feedback on code.

- **Free up developers' time**

CI/CD can free up developers' time by automating the deployment process, allowing them to focus on more rewarding projects.

- **Easier rollback**

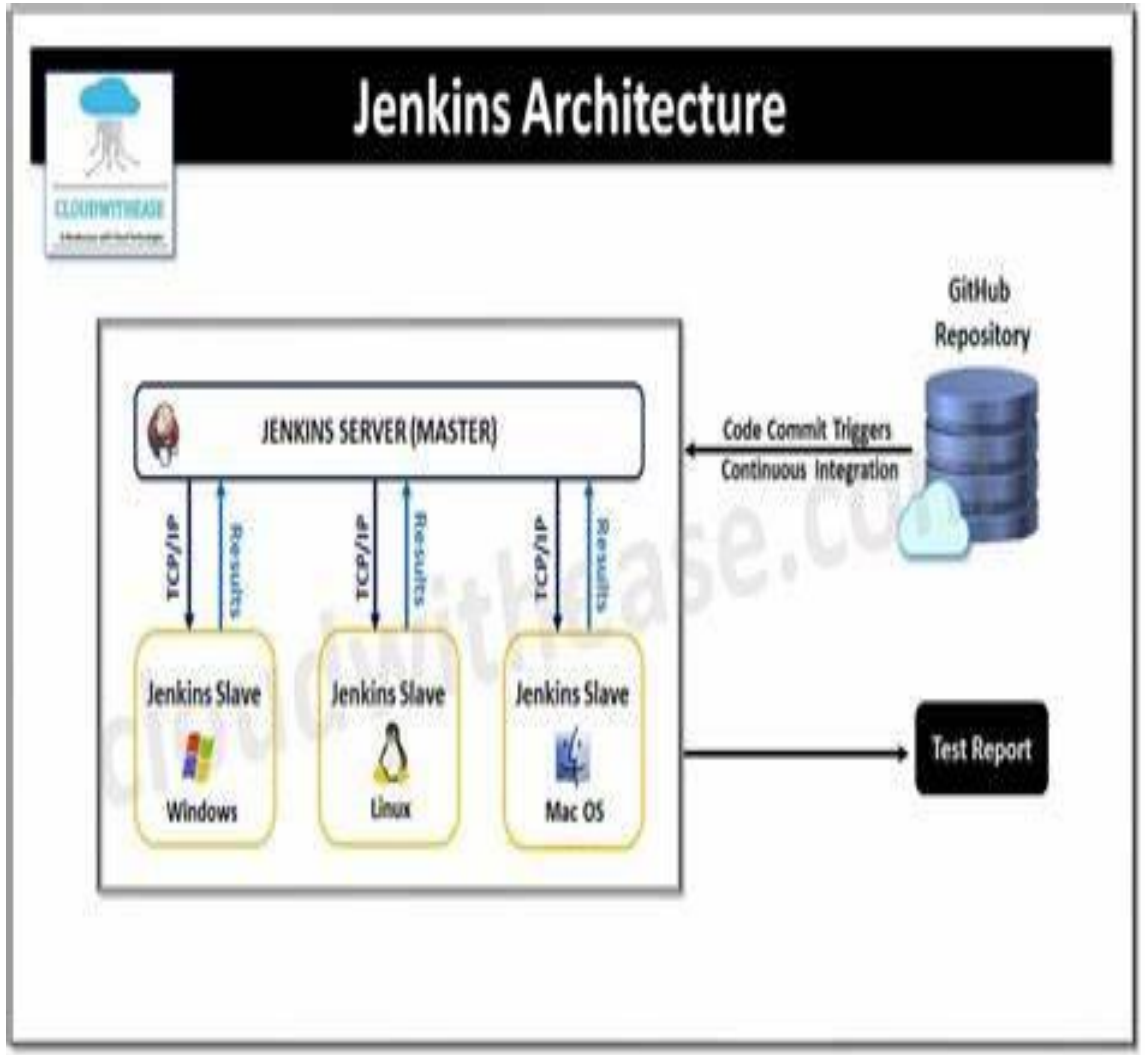
CI/CD can make it easier to roll back code if issues arise, which can save time and resources.

What is Jenkins?

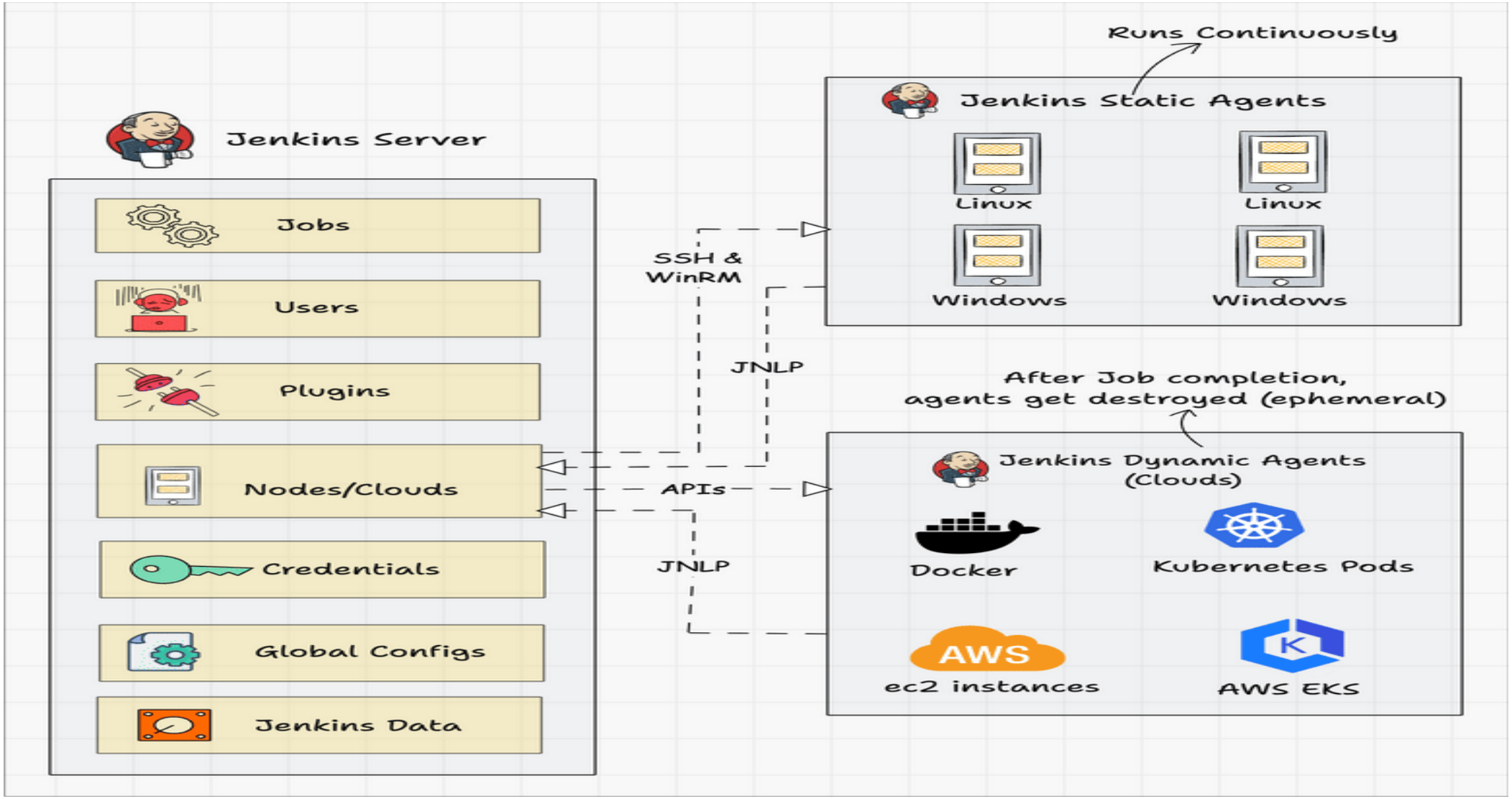
- Overview of Jenkins: An open-source automation server
- Key features: Extensible, cross-platform, active community support
- Jenkins in CI/CD: Automates build, test, and deployment processes

Jenkins Architecture

- Jenkins Master (Local): Central server managing jobs, plugins, and configurations.
- Local Node (Local): Machine connected to Jenkins Master for executing jobs.
- Remote Node (Offsite): External machine connected to Jenkins Master via SSH/HTTPS.
- GitHub Repository: Central repository storing code and triggering Jenkins builds.



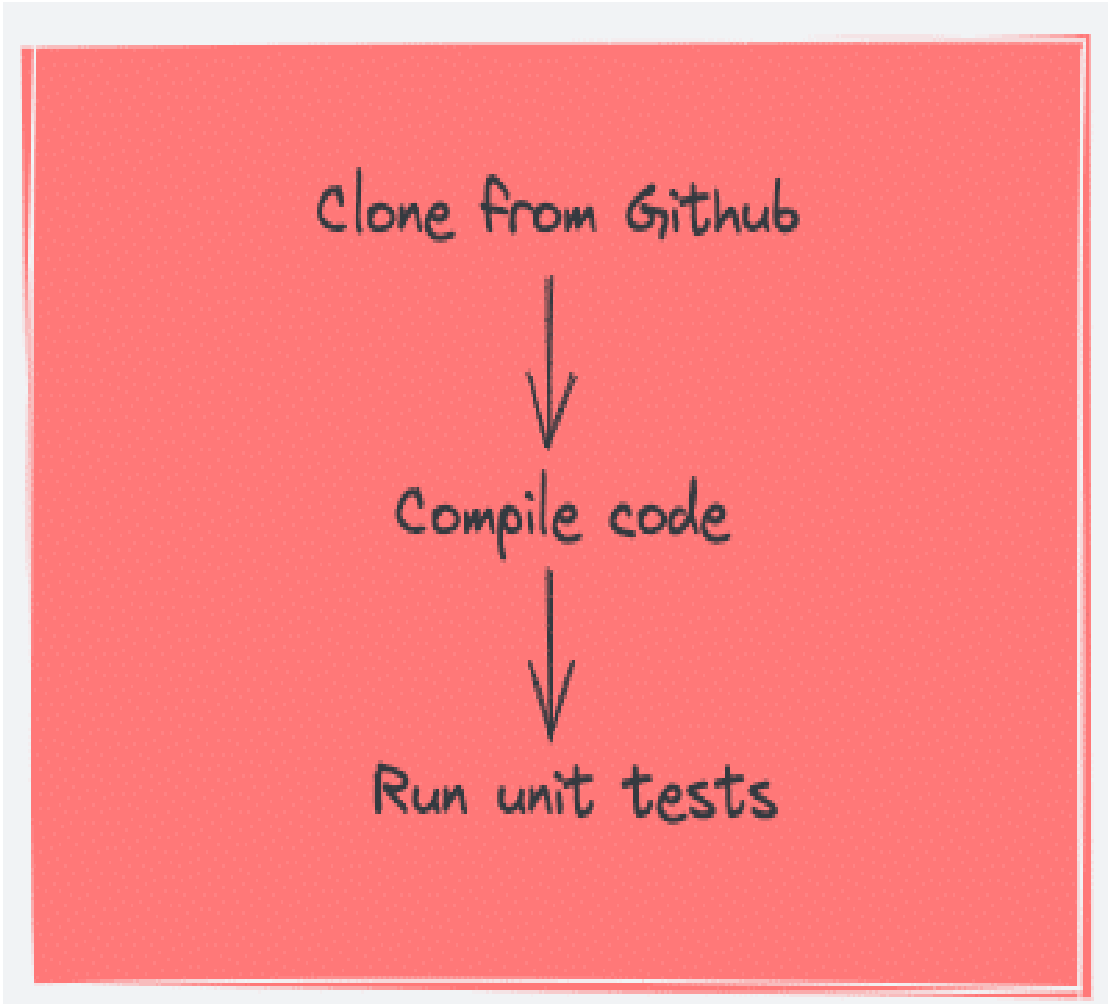
Jenkins In Action



Jenkins In Action

```
        stage('Code Checkout') {
            steps {
                checkout([
                    $class: 'GitSCM',
                    branches: [[name: '*/master']],
                    userRemoteConfigs: [[url: 'https://github.com/spring-
projects/spring-petclinic.git']]
                ])
            }
        }

        stage('Code Build') {
            steps {
                 sh 'mvn install -Dmaven.test.skip=true'
            }
        }
```



Jenkins Workflow

- Developer Push: Code changes pushed to GitHub Repository.
- GitHub Webhook: Triggers Jenkins build on push event.
- Jenkins Trigger: Jenkins Master receives webhook, triggering specified job.
- Job Execution: Job runs on Local Node or Remote Node.
- Artifact Deployment: Deployed to production environment.

Jenkins Triggers

- GitHub Push Trigger: Builds triggered on code push.
- Schedule Trigger: Builds triggered at specified intervals.
- Manual Trigger: Builds triggered manually.

Node Configuration:

- Local Node: Connected to Jenkins Master via LAN
- Remote Node: Connected to Jenkins Master via SSH/HTTPS.

Setting Up Jenkins

- 1. Install Jenkins:
  - Download from jenkins.io
  - Run installer (Windows) / Use package manager (Linux) / Homebrew (macOS)
- 2. Initial Configuration:
  - Access Jenkins via localhost:8080
  - Complete setup wizard
- 3. Install Essential Plugins:
  - Git plugin
  - Pipeline plugin
  - GitHub Integration plugin

PR from GitBash

```
gh pr create --base main --head feature-branch --title "Add feature X" --body "This PR adds feature X which includes..."
```

```
santh@spiderman MINGW64 /e/LocalGit/UserRegistration (mdev)
$ gh pr create --base main --head mdev --title "Add feature X" --body "This PR is to test the github web hook"

Creating pull request for mdev into main in jsanthoshkiran/gnitcsedevops

https://github.com/jsanthoshkiran/gnitcsedevops/pull/5

santh@spiderman MINGW64 /e/LocalGit/UserRegistration (mdev)
$
```

Running Jenkins from command line:

java -jar jenkins.war

Creating Your First Pipeline Job

1. Dashboard Navigation:
   - Access Jenkins dashboard
2. Create New Item:
   - Click on 'New Item'
   - Enter job name, select 'Pipeline', and click 'OK'
3. Configure Pipeline:
   - Choose 'Pipeline script from SCM'
   - Enter repository URL and branch

Writing a Jenkinsfile

1. What is a Jenkinsfile?

   - A text file containing the pipeline script

2. Basic Structure:

```
pipeline {
  agent any
  stages
   {
     stage('Build') {
         steps { ... }
      }
  }
}
```

Writing a Jenkinsfile

--- **Jenkinsfile for a Simple Project:**

\`\`\`groovy

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git branch: 'main', url: 'https://github.com/your-username/your-repo.git'
            }
        }
        stage('Build') {
            steps {
                echo 'Building the project...'
            }
        }
        stage('Deploy') {
            steps {
                sh 'scp -o StrictHostKeyChecking=no index.html user@your_server_ip:/var/www/html'
            }
        }
    }
}
```

\`\`\`

Integrating with GitHub

1. Creating a Webhook:
    - Go to repository Settings > Webhooks
    - Add webhook: Enter Jenkins URL (http://your-jenkins-server/github-webhook/)
    - Content type: Select "application/json“
    - Which events to send: Select "Pull request“
    - Active: Enable the webhook
2. Configuring Jenkins Job:
    - Create a new Jenkins job: Configure the job to build your project when triggered by a specific event
    - Ensure 'GitHub hook trigger for GITScm polling' is enabled

Running the Pipeline

1. Manual Trigger:
   - Click 'Build Now' on Jenkins job page
2. Automatic Trigger:
   - Push changes to GitHub repository
   - Jenkins webhook triggers the pipeline

Monitoring Builds

1. Console Output:
   - View build logs in real-time
2. Build History:
   - Check status of previous builds
3. Notifications:
   - Set up email or Slack notifications for build results

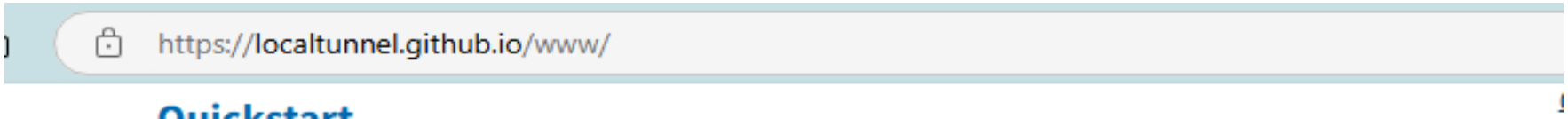
Scaling with Jenkins

1. Distributed Builds:
   - Configure Jenkins agents to run builds on multiple nodes
2. Master-Slave Configuration:
   - Set up master and slave nodes for distributed builds
3. Pipeline as Code:
   - Store Jenkinsfiles in version control for reproducible builds

Conclusion

- Recap of Jenkins CI/CD pipeline setup
- Benefits of automating the build, test, and deployment processes
- Encouragement to explore advanced Jenkins features and plugins

Local Tunnel Setup



Quickstart

Install Localtunnel globally (requires NodeJS) to make it accessible anywhere:

```
$ npm install -g localtunnel
```

Start a webserver on some local port (eg http://localhost:8000) and use the command line interface to request a tunnel to your local server:

```
$ lt --port 8000
```

You will receive a url, for example https://gqgh.localtunnel.me, that you can share with anyone for as long as your local instance of lt remains active. Any requests will be routed to your local service at the specified port.