Laboratory Record
of DEVOPS TOOLS LAB

Roll No.:
Experiment No.: 1
Sheet No.: 1
Date: 18/07/2025

WEEK-1

Aim: To understand DevOps: Principles, Practices, and DevOps Engineer Role and Responsibilities.

Description:

DevOps is a cultural and technical approach that integrates development (Dev) and operations (Ops) teams to deliver software quickly, efficiently, and reliably. It emphasizes automation, continuous integration (CI), continuous delivery (CD), and strong collaboration between teams. By breaking down silos and using tools to automate repetitive tasks, DevOps reduces manual effort, minimizes errors, and accelerates software delivery. It also relies on Agile principles, cloud computing, and infrastructure automation to streamline workflows. DevOps Engineers play a key role in designing, maintaining, and automating the entire software delivery pipeline, making it a critical approach for modern IT organizations.

1. DevOps Principles

- **Collaboration and Communication:** DevOps encourages developers, operations, QA, and security teams to work together, share information openly, and focus on a common goal of delivering quality software quickly.
- **Automation of Processes:** Manual steps in building, testing, deploying, and monitoring are automated to reduce errors, save time, and achieve consistency.
- **Continuous Integration and Continuous Delivery (CI/CD):** Frequent code integration, automated testing, and deployment pipelines enable faster and safer releases.
- **Monitoring and Continuous Feedback:** Systems are continuously monitored for performance, errors, and security issues, and feedback is used to improve processes and products.
- **Customer-Centric and Lean Approach:** DevOps focuses on delivering value to customers quickly by releasing small, frequent updates and adapting to changes rapidly.

2. DevOps Practices

- **Version Control and Source Management:** Teams use tools like Git to manage source code versions, enabling collaboration and easy rollback of changes.
- **Automated Build and Deployment Pipelines:** Tools such as Jenkins, GitLab CI, and Azure DevOps are used to automate compiling, testing, and deploying software.
- **Containerization and Orchestration:** Technologies like Docker and Kubernetes ensure applications run consistently across different environments and scale easily.
- **Infrastructure as Code (IaC):** Tools like Terraform and Ansible allow teams to manage servers, networks, and infrastructure through code rather than manual setups.
- **Automated Testing and Quality Checks:** Automated tests catch issues early in development, reducing bugs and improving quality.

- **Continuous Monitoring and Logging:** Systems are monitored in real-time with tools like Prometheus, Grafana, and ELK Stack to detect issues proactively.
- **Agile and Lean Workflows:** Practices like Scrum and Kanban boards are used to visualize tasks, limit work in progress, and improve productivity.

3. Role of a DevOps Engineer

- **Toolchain Management:** A DevOps Engineer selects, configures, and maintains the tools for CI/CD, infrastructure automation, and monitoring, ensuring smooth workflows.
- **Pipeline Automation:** They design and implement automated workflows for software building, testing, and deployment, minimizing manual intervention.
- **Infrastructure Provisioning and Maintenance:** DevOps Engineers create, manage, and scale cloud-based or on-premises infrastructure, ensuring reliability and performance.
- **Collaboration and Culture Building:** They promote collaboration between development, QA, and IT operations teams by integrating tools and processes that encourage teamwork.
- **Security and Reliability Focus:** DevOps Engineers incorporate security practices into pipelines (DevSecOps) and ensure that systems are highly available and resilient.

4. Responsibilities of a DevOps Engineer

- Build and maintain automated CI/CD pipelines for consistent and fast software delivery.
- Manage and provision cloud and server infrastructure, ensuring scalability and cost efficiency.
- Integrate monitoring, logging, and alerting systems to identify and resolve issues quickly.
- Automate repetitive operational tasks to save time and reduce human errors.
- Collaborate with developers, testers, and IT teams to align workflows and tools for seamless deployments.
- Ensure security, compliance, and reliability by applying DevSecOps principles.

Conclusion:

DevOps is a modern software development approach that unites development and operations teams to achieve faster, reliable, and high-quality software delivery. By focusing on principles such as collaboration, automation, CI/CD, monitoring, and a customer-first mindset, organizations can streamline workflows and respond quickly to business needs. DevOps practices like version control, automated pipelines, containerization, and infrastructure as code bring consistency, scalability, and efficiency. DevOps Engineers play a crucial role by managing tools, automating processes, and ensuring secure, resilient infrastructure. Overall, DevOps improves productivity, reduces downtime, and helps organizations deliver better software at a faster pace.

WEEK-2

Aim: To explore the Version Control System (VCS) tools for source code management.

Description:

Version Control Systems (VCS) are software tools that track and manage changes to source code and other project files over time. They allow teams to collaborate effectively, roll back to previous versions, and identify who made specific changes. VCS is essential in software development because it prevents accidental overwriting of code, makes teamwork easier, and maintains a full history of a project's evolution. Modern DevOps practices heavily rely on VCS for efficient collaboration, automation, and integration with CI/CD pipelines.

Version Control System (VCS)

A **Version Control System (VCS)** is a tool that records changes to files or projects over time, enabling developers to restore previous versions, review code changes, and work together seamlessly. It helps in tracking contributions, identifying issues, and ensuring software stability while supporting both individual and team-based development workflows.

Types of VCS

1. **Local Version Control Systems (LVCS):**
   Stores versions of files locally on a single developer's computer. It's simple but prone to mistakes and not suitable for collaboration.
2. **Centralized Version Control Systems (CVCS):**
   All versions are stored in a single central server. Developers check files in and out, which makes collaboration easier but introduces dependency on a central server. Examples include SVN and CVS.
3. **Distributed Version Control Systems (DVCS):**
   Each user has a full copy of the repository and its history, making it faster, more secure, and capable of offline work. Examples include Git and Mercurial.

Common VCS Tools

- **Git:**
  Git is a distributed VCS that is extremely popular due to its speed, flexibility, and branching features. It allows developers to work offline and merge changes efficiently. It is widely integrated with platforms like GitHub and GitLab, making it the industry standard.

- **Subversion (SVN):**
  SVN is a centralized VCS where all data is stored in one central repository. It's easier for beginners, offers fine-grained permissions, and is good for enterprises that need strict version control. However, it's slower and less flexible compared to Git.
- **Mercurial:**
  Similar to Git, Mercurial is a distributed VCS but focuses on simplicity and ease of learning. It's faster for large projects and easier to adopt but has less community support and fewer integrations than Git.
- **Perforce:**
  Perforce is a centralized, enterprise-grade VCS designed to manage massive codebases and binary files. It offers scalability and security but requires more setup and is often used in large organizations.
- **CVS (Concurrent Versions System):**
  CVS is one of the earliest centralized VCS tools, known for its simplicity and legacy use. While outdated compared to modern tools, it paved the way for systems like SVN. It lacks advanced branching and merging features, making it unsuitable for large modern projects.

Advantages of VCS

- Maintains a detailed history of changes for every file and project.
- Enables collaboration by allowing multiple developers to work simultaneously.
- Supports branching and merging for feature development without breaking main code.
- Provides rollback and recovery features to undo mistakes.
- Integrates with CI/CD tools for automation and DevOps workflows.

Disadvantages of VCS

- Distributed systems like Git can be complex for beginners to learn.
- Centralized systems depend heavily on server uptime; outages can disrupt work.
- Large repositories can require significant storage and careful management.
- Merge conflicts can be time-consuming to resolve.
- Mismanagement of permissions or configuration can lead to errors.

Conclusion:

Version Control Systems are essential tools in modern software development, enabling teams to collaborate, track changes, and ensure software stability. They can be categorized into local, centralized, and distributed systems, with Git being the most widely used distributed system today. Each VCS tool has unique strengths—Git and Mercurial offer flexibility and speed, while SVN, CVS, and Perforce provide structured control for enterprises. By using VCS effectively, developers can streamline workflows, reduce errors, and improve productivity, making VCS a core part of DevOps and software engineering practices.

WEEK-3

Aim: To install Git, create a GitHub account, and execute various Git operations.

Description:

Git is a distributed version control system that tracks changes to files and enables multiple developers to collaborate efficiently. GitHub is a cloud-based hosting platform where Git repositories can be stored, shared, and managed. In this experiment, Git was installed and configured, a GitHub account was used to create a remote repository, and several Git commands were practiced, such as initializing a repository, committing changes, creating branches, and pushing code to GitHub. This provides a strong foundation for version control and collaboration in software development.

Code:

Step 1: Configure Git
git config --global user.name "srisaisohan29"
git config --global user.email srisaisohan29@gmail.com

Step 2: Initialize Repository and Make First Commit
git init
git status
git add file1.txt
git commit -m "1st file added"

Step 3: Connect to GitHub and Push
git remote add origin https://github.com/srisaisohan29/git-practice.git
git push -u origin master
(Used Personal Access Token for authentication)

Step 4: Create and Work on a New Branch
git branch qa
git checkout qa
git status
git add file2.txt
git commit -m "added file2"

Step 5: Push the New Branch to GitHub
git push -u origin qa
git checkout master

Output:

```
student@CSELAB2-20:~/160122733061$ git config --global user.name "srisaisohan29"
student@CSELAB2-20:~/160122733061$ git config --global user.email "srisaisohan29@gmail.com"
```

```
student@CSELAB2-20:~/160122733061$ git init
Reinitialized existing Git repository in /home/student/160122733061/.git/
student@CSELAB2-20:~/160122733061$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    myfile1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt

no changes added to commit (use "git add" and/or "git commit -a")
student@CSELAB2-20:~/160122733061$ git add file1.txt
student@CSELAB2-20:~/160122733061$ git commit -m "1st file added"
[master c2096c8] 1st file added
 1 file changed, 1 insertion(+)
 create mode 100644 file1.txt
```

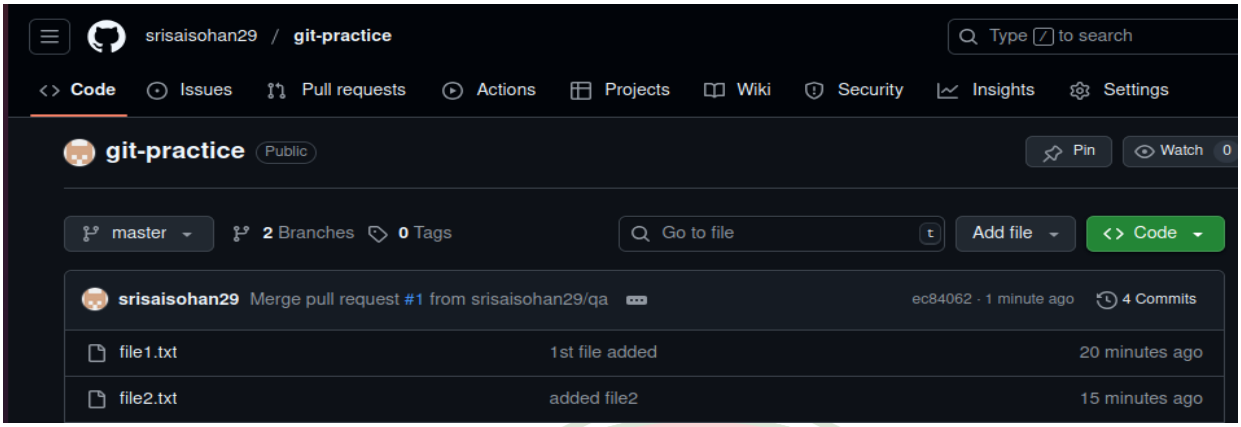
```
student@CSELAB2-20:~/160122733061$ git remote add origin https://github.com/srisaisohan29/git-practice.git
fatal: remote origin already exists.
student@CSELAB2-20:~/160122733061$ git push -u origin master
Username for 'https://github.com': srisaisohan29
Password for 'https://srisaisohan29@github.com':
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 482 bytes | 482.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To https://github.com/srisaisohan29/git-practice.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

```
student@CSELAB2-20:~/160122733061$ git branch qa
student@CSELAB2-20:~/160122733061$ git checkout qa
D       myfile1.txt
Switched to branch 'qa'
student@CSELAB2-20:~/160122733061$ git status
On branch qa
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    myfile1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file2.txt

no changes added to commit (use "git add" and/or "git commit -a")
student@CSELAB2-20:~/160122733061$ git add file2.txt
student@CSELAB2-20:~/160122733061$ git commit -m "added file2"
[qa 0ec260a] added file2
 1 file changed, 1 insertion(+)
 create mode 100644 file2.txt
```

```
student@CSELAB2-20:~/160122733061$ git push -u origin qa
Username for 'https://github.com': srisaisohan29
Password for 'https://srisaisohan29@github.com':
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 304 bytes | 304.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'qa' on GitHub by visiting:
remote:      https://github.com/srisaisohan29/git-practice/pull/new/qa
remote:
To https://github.com/srisaisohan29/git-practice.git
 * [new branch]      qa -> qa
Branch 'qa' set up to track remote branch 'qa' from 'origin'.
student@CSELAB2-20:~/160122733061$ git checkout master
D       myfile1.txt
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
```



Output Analysis:

- The command 'git config' successfully stored the username and email globally, ensuring every commit is associated with the correct author information.
- Running 'git init' created a hidden .git directory, converting the folder into a fully functional local Git repository.
- The 'git status' command correctly displayed the new file1.txt file as untracked, confirming Git was ready to track it.
- After running 'git add file1.txt', the file was moved to the staging area, and 'git commit' permanently saved it to the repository history with a descriptive message.
- The 'git remote add origin' command successfully linked the local repository to the remote GitHub repository, enabling synchronization.
- On using 'git push -u origin master', the committed changes were uploaded to GitHub, and Git prompted for a Personal Access Token, which authenticated successfully.
- Creating the qa branch with 'git branch qa' and switching to it using 'git checkout qa' worked smoothly, confirming Git's branching functionality.
- The second file 'file2.txt' was staged, committed, and pushed, and GitHub reflected the new branch and file immediately, proving proper remote sync.
- Finally, switching back to the master branch verified seamless branch navigation. Both master and qa branches with their files were visible on GitHub, confirming that all operations were successful and correctly executed.

Conclusion:

This experiment demonstrated the installation, configuration, and use of Git for basic version control tasks, along with GitHub integration for remote repository management. We learned how to initialize a local repository, stage and commit changes, create branches, and push changes to a remote repository. Using a Personal Access Token enhanced security and allowed successful authentication. These steps provide a strong foundation for efficient collaboration, code management, and DevOps workflows.

WEEK-4

Aim: To install and configure Jenkins, set up a build job, and understand the concept of Continuous Integration.

Description: Jenkins is an open-source automation server widely used for Continuous Integration (CI) and Continuous Delivery (CD). It automates building, testing, and deploying software, allowing teams to detect issues early and deliver updates faster. In this experiment, Jenkins was installed on a system with Java support, configured through its web interface, and used to execute a simple shell script. This hands-on setup demonstrates how Jenkins automates repetitive development tasks and simplifies integration workflows.

Code:

Step 1: Install Jenkins and Java

sudo apt install update

sudo apt install openjdk-21-jre-headless

sudo wget -O /etc/apt/keyrings/jenkins-keyring.asc https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

echo "deb [signed-by=/etc/apt/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/" |

sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update

sudo apt-get install jenkins

sudo systemctl start jenkins

sudo systemctl enable Jenkins

Step 2: Unlock Jenkins

Access Jenkins in browser

http://localhost:8080

Get the initial admin password

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

(Use the password to unlock Jenkins, install suggested plugins, and create an admin account.)

Step 3: Create and Prepare the Shell Script

touch samplefile.sh

nano samplefile.sh

Inside nano, add:

#!/bin/bash

echo "Hello from a Jenkins build!"

chmod +x samplefile.sh

mv /home/student/samplefile.sh /tmp/samplefile.sh

Step 4: Create and Configure Jenkins Job

1. Open Jenkins dashboard → New Item → Name it 'HelloJenkins' → Choose Freestyle Project → Click OK.
2. Scroll to Build → Click Add build step → Select Execute shell → Add:

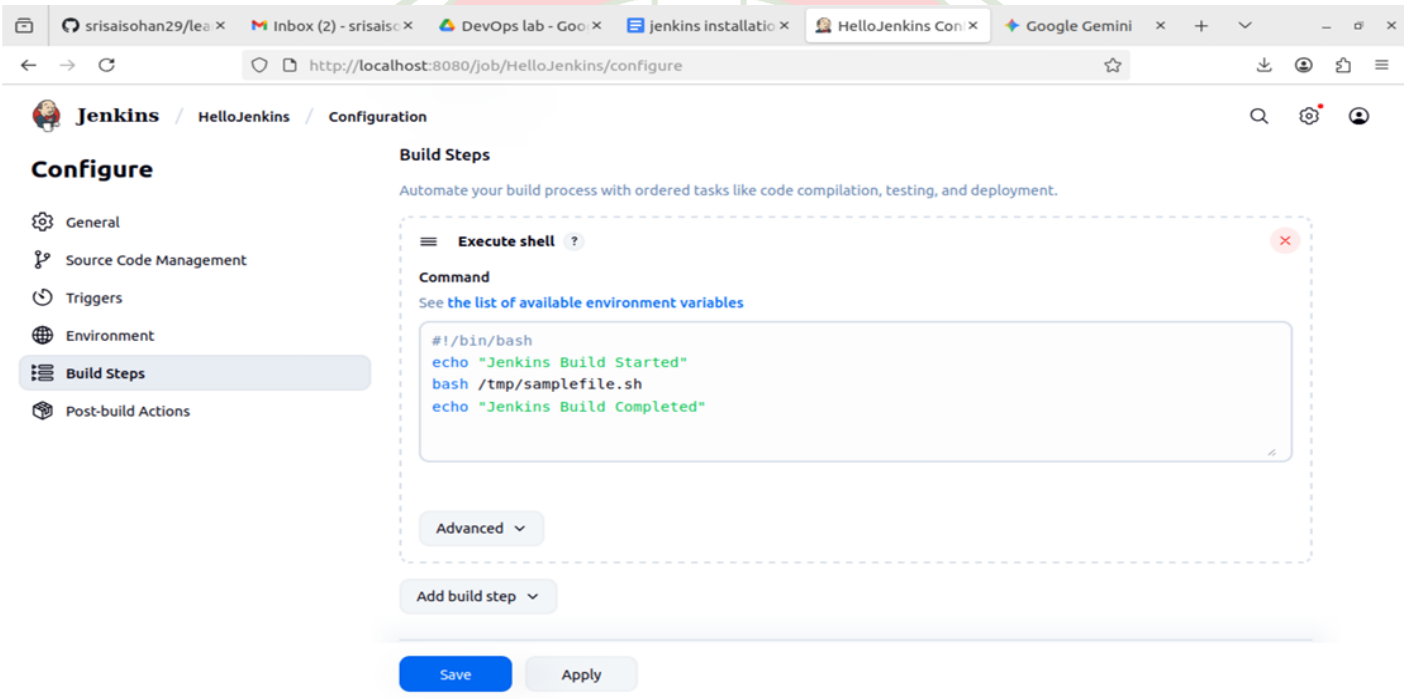
```
#!/bin/bash
echo "Jenkins Build Started"
bash /tmp/samplefile.sh
echo "Jenkins Build Completed"
```

3. Click Save and then Build Now.
4. Open Console Output to verify the results.

Output:

```
student@Student:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
6dfdbe1333e74412891cc4f6f2f14b37
```

```
student@Student:~$ touch samplefile.sh
student@Student:~$ nano samplefile.sh
student@Student:~$ chmod +x samplefile.sh
student@Student:~$ ^C
student@Student:~$ mv /home/student/samplefile.sh /tmp/samplefile.sh
```



Output Analysis:

• Jenkins installation completed successfully, and the service started running on port 8080, confirming the server was active.

• The browser interface opened Jenkins, and using the initialAdminPassword, the setup wizard was completed with plugins installed and an admin account created.

• The custom shell script (samplefile.sh) was created, made executable, and moved to /tmp to ensure Jenkins had permission to execute it.

• The freestyle project HelloJenkins was successfully created and configured with a simple shell execution step.

• Running Build Now triggered the script, and the console output displayed:

Jenkins Build Started

Hello from a Jenkins build!

Jenkins Build Completed

This confirmed that Jenkins executed the script without errors, demonstrating automated builds.

Conclusion:

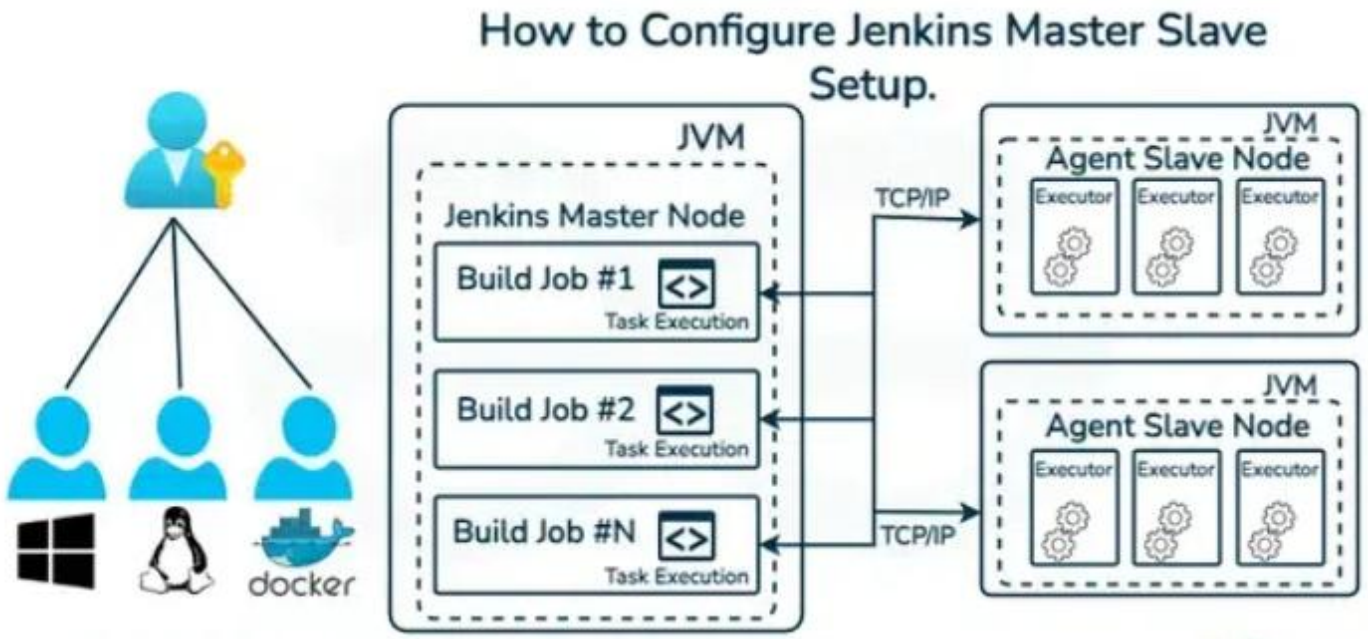
This experiment demonstrated the process of installing Jenkins, configuring it, and setting up a build job to execute a simple script, illustrating the concept of Continuous Integration. By integrating with scripts and automating tasks, Jenkins helps developers detect errors early, reduce manual steps, and streamline software delivery. Understanding Jenkins setup and job configuration forms the foundation for building advanced CI/CD pipelines in DevOps workflows.

WEEK-5

Aim: To understand Jenkins Master-Slave architecture and scale a standalone Jenkins setup by implementing slave nodes.

Description:

In Jenkins, the master node controls the scheduling and distribution of jobs, while agent nodes (slaves) execute the jobs assigned by the master. This architecture enables scalability, load distribution, and parallel execution of builds. The master is responsible for job configuration and orchestration, while agents handle the actual workload such as building code, running tests, or executing scripts. By adding and connecting agents, Jenkins can efficiently manage multiple jobs across different environments, improving performance and resource utilization.



The diagram shows how Jenkins Master-Slave architecture works. The Jenkins Master Node runs jobs inside a JVM and sends them over TCP/IP to connected Agent Nodes. Each agent also runs in a JVM and contains multiple executors that perform the tasks. Different environments (Windows, Linux, Docker, etc.) can be used as agents, allowing Jenkins to scale across platforms and execute jobs in parallel.

Code:

Part 1: Setting up the Master Server

```
# Update and install Java
sudo apt update
sudo apt install openjdk-21-jre-headless

# Add Jenkins repository key
sudo wget -O /etc/apt/keyrings/jenkins-keyring.asc \
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

# Add Jenkins repository
echo "deb [signed-by=/etc/apt/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/" | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null

# Update and install Jenkins
sudo apt-get update
sudo apt-get install jenkins

# Start and enable Jenkins
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

- Open browser → http://localhost:8080
- Unlock Jenkins using initial admin password:

  ```
  sudo cat /var/lib/jenkins/secrets/initialAdminPassword
  ```
- Install suggested plugins and create admin user.

Part 2: Preparing the Agent Machine

```
# Update and install Java on the agent
sudo apt update
sudo apt install openjdk-21-jre-headless

# Create workspace directory
mkdir ~/jenkins-agent
```

Part 3: Connecting the Agent to the Master

1. On Master Dashboard → Manage Jenkins > Nodes > New Node.
   - Node name: lab-agent-01
   - Remote root dir: /home/student/jenkins-agent
   - Label: linux-agent
   - Launch method: *Launch agent by connecting it to the master*
2. On Master Terminal, allow agent communication:

   hostname -I

   sudo ufw allow 50000/tcp
3. On Agent Machine, run commands provided by Master:

   cd ~/jenkins-agent

   curl -s http://<master-ip>:8080/jnlpJars/agent.jar -o agent.jar

   java -jar agent.jar -url http://<master-ip>:8080 -secret <secret-key> \

   -name "lab-agent-01" -workDir /home/student/jenkins-agent

Part 4: Running a Job on the Agent

1. On Master Dashboard → New Item → Freestyle Project → Agent-Test-Job.
2. Restrict project to run on: linux-agent.
3. Add Build Step → Execute Shell:

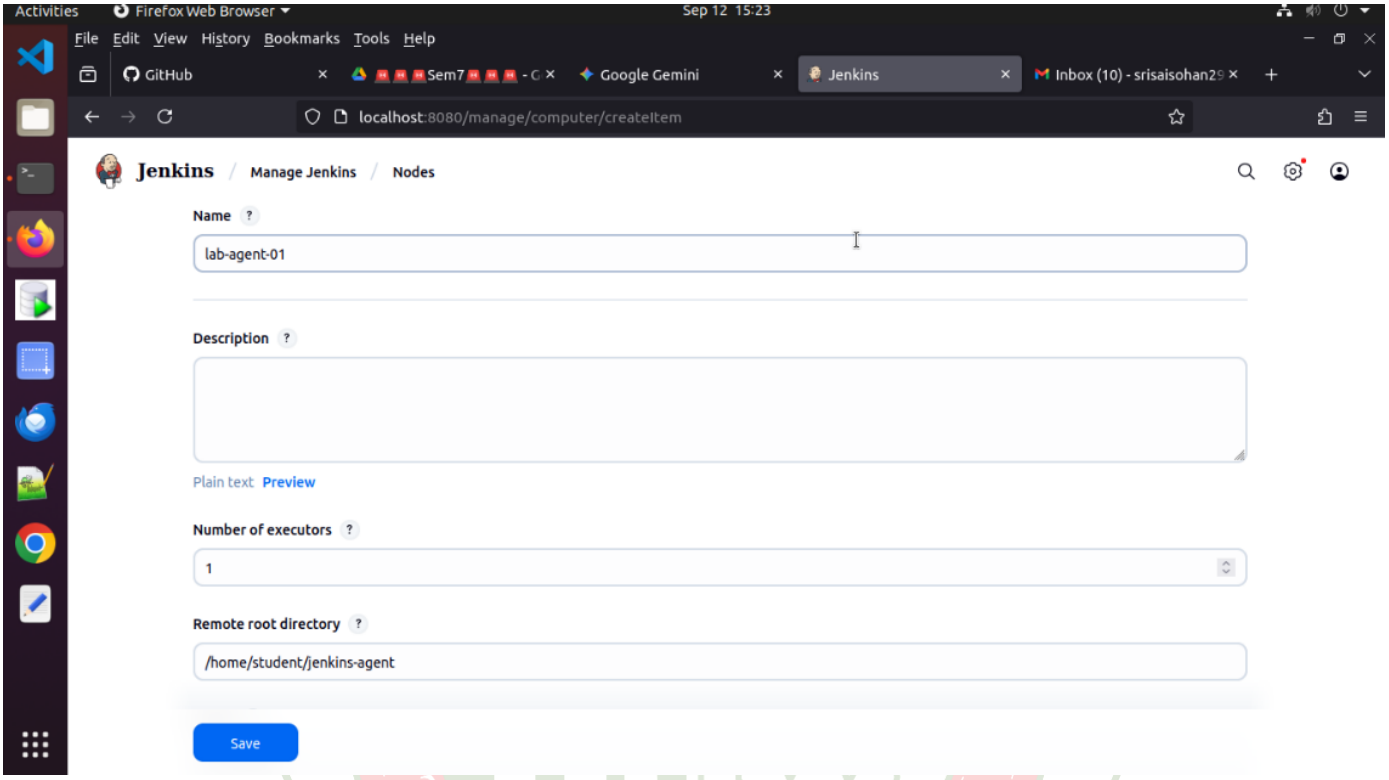
   echo "This job is running on the following machine:"

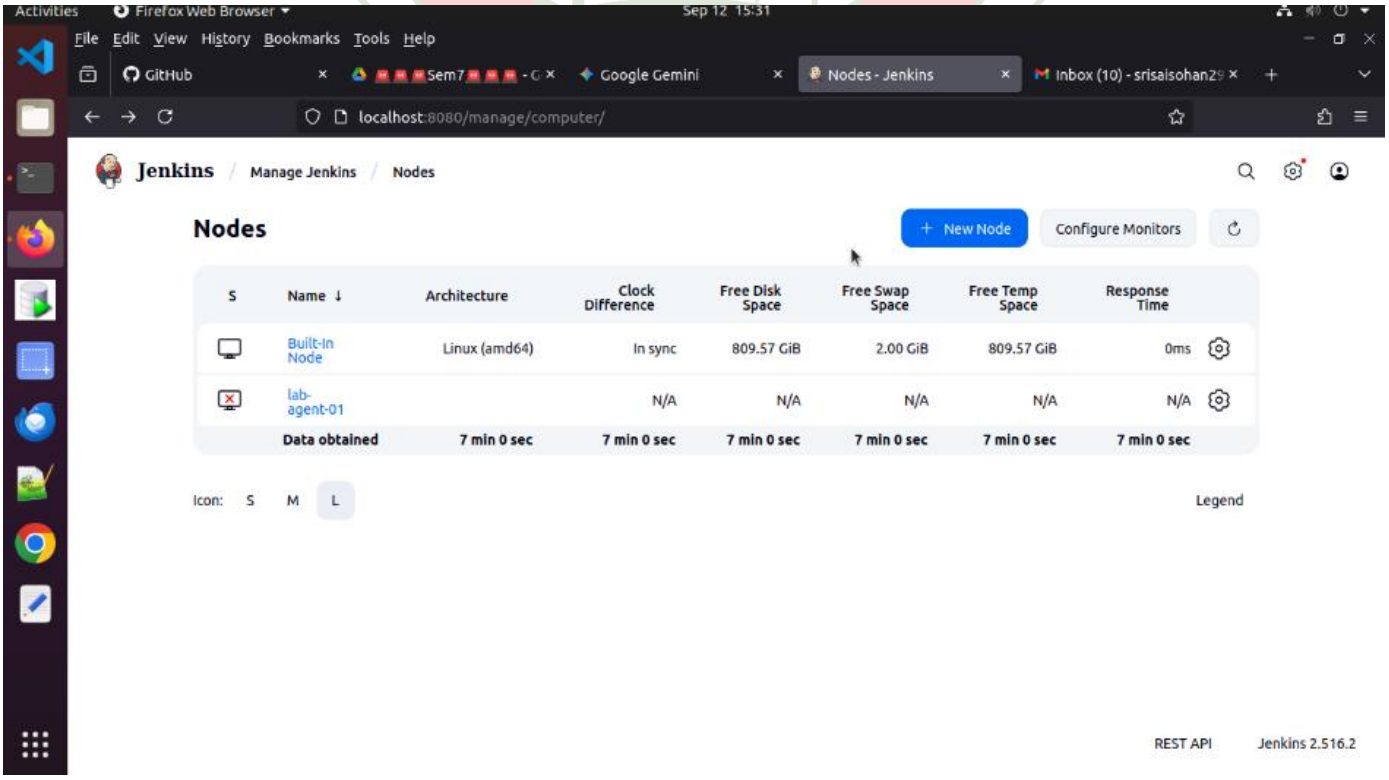
   hostname

   echo "The workspace is:"

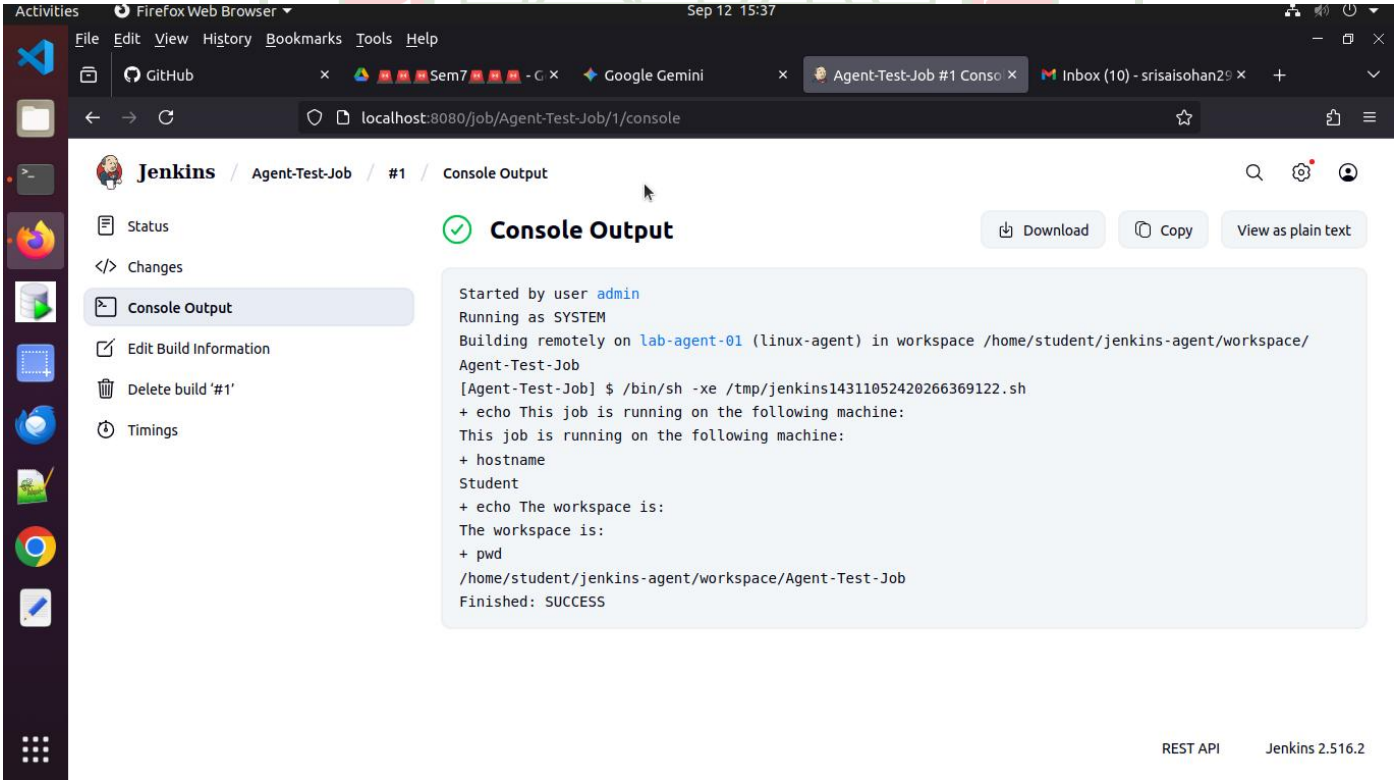
   pwd
4. Save → Build Now → Console Output.

Output:


Activities
Firefox Web Browser
Sep 12 15:23
File Edit View History Bookmarks Tools Help
GitHub
Google Gemini
Jenkins
Inbox (10) - srisaisohan2
localhost:8080/manage/computer/createItem
Jenkins / Manage Jenkins / Nodes
Name
lab-agent-01
Description
Plain text Preview
Number of executors
1
Remote root directory
/home/student/jenkins-agent
Save


Activities
Firefox Web Browser
Sep 12 15:31
File Edit View History Bookmarks Tools Help
GitHub
Google Gemini
Nodes - Jenkins
Inbox (10) - srisaisohan2
localhost:8080/manage/computer/
Jenkins / Manage Jenkins / Nodes
Nodes
New Node
Configure Monitors
S
Name
Architecture
Clock Difference
Free Disk Space
Free Swap Space
Free Temp Space
Response Time
Built-In Node
Linux (amd64)
In sync
809.57 GiB
2.00 GiB
809.57 GiB
0ms
lab-agent-01
N/A
Data obtained
7 min 0 sec
Icon: S M L
Legend
REST API
Jenkins 2.516.2

```
student@Student:~/jenkins-agent$ curl -sO http://172.20.11.190:8080/jnlpJars/agent.jar
student@Student:~/jenkins-agent$ java -jar agent.jar -url http://172.20.11.190:8080/ -secret ca4df0ac7272966242fdf8bc0904c73a8079f246e95a92659
a0b9456de9cce23 -name "lab-agent-01" -webSocket -workDir "/home/student/jenkins-agent"
Sept 12, 2025 3:34:10 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /home/student/jenkins-agent/remoting as a remoting work directory
Sept 12, 2025 3:34:10 PM org.jenkinsci.remoting.engine.WorkDirManager setupLogging
INFO: Both error and output logs will be printed to /home/student/jenkins-agent/remoting
Sept 12, 2025 3:34:10 PM hudson.remoting.Launcher createEngine
INFO: Setting up agent: lab-agent-01
Sept 12, 2025 3:34:10 PM hudson.remoting.Engine startEngine
INFO: Using Remoting version: 3309.v27b_9314fd1a_4
Sept 12, 2025 3:34:10 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /home/student/jenkins-agent/remoting as a remoting work directory
Sept 12, 2025 3:34:11 PM hudson.remoting.Launcher$CuiListener status
INFO: WebSocket connection open
Sept 12, 2025 3:34:11 PM hudson.remoting.Launcher$CuiListener status
INFO: Connected
```



Output Analysis:

- Jenkins Master was installed and accessed successfully through http://localhost:8080, confirming that the service was active.
- The Agent machine was prepared with Java and a dedicated workspace directory, ensuring compatibility with Jenkins.
- The master node created a permanent agent configuration with label linux-agent. After running the java -jar agent.jar command on the agent, the node connected successfully, and its status changed from offline to online on the Jenkins dashboard.
- A Freestyle job was created and restricted to run only on the agent machine.
- The Console Output of the build displayed the agent's hostname and workspace path, proving that the job was executed on the agent node and not on the master.
- This confirmed that Jenkins successfully delegated tasks from the master to the agent, demonstrating the scalability of the architecture.

Conclusion:

This experiment successfully demonstrated Jenkins Master-Slave architecture, where the master handled job scheduling and configuration while the agent executed the assigned tasks. The setup showed how Jenkins can scale beyond a standalone installation by distributing workloads across multiple machines. Running a test job confirmed that the agent was properly connected and functioning. This architecture improves efficiency, supports parallel builds, and allows Jenkins to manage larger and more complex software projects effectively.