**1.GIT Commands any 10**

**2. Jenkins javacode program**

Login in

New item -> prog1 -> freestyle project

Configure -> command -> cd C:\Program Files\Java\jre-17.0.9\bin

javac helloworld.java

java helloworld -> save

then click on build now (Console output Mai dekho output)

**3. Jenkins code pipeline**

Open Jenkins:

Go to http://localhost:8080 and log in.

Create a New Pipeline Job:

From the Jenkins dashboard, click New Item.

Enter a name (e.g., Java-Pipeline-Job), select Pipeline, and click OK.

Configure the Pipeline:

Scroll down to the Pipeline section.

Code:

pipeline {

agent any

stages {

stage('Addition') {

steps {

script {

int num1 = 10

int num2 = 5

int sum = num1 + num2

echo "The sum of ${num1} and ${num2} is: ${sum}"

int sub = num1 - num2

echo "The sub of ${num1} and ${num2} is: ${sub}"

int mul = num1 \* num2

echo "The mul of ${num1} and ${num2} is: ${mul}"

int div = num1 / num2

echo "The div of ${num1} and ${num2} is: ${div}"

}

}

}

}

post {

always {

echo 'Pipeline execution completed.'

}

}

}

Then do okay and run by clicking build up then see result.

**4. Docker - deploy container**

Need to download Docker

Docker images

Docker pull hello-world

Docker images

Docker run hello-world

Docker ps

Docker ps -a

**5. Build an image for sample Web application using Docker**

**npm install express**

**create public folder**

**inde.html**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>Express App with Button</title>**

**<style>**

**body {**

**font-family: Arial, sans-serif;**

**text-align: center;**

**margin-top: 50px;**

**}**

**button {**

**padding: 10px 20px;**

**font-size: 16px;**

**cursor: pointer;**

**}**

**</style>**

**</head>**

**<body>**

**<h1>Welcome to My Express App</h1>**

**<button onclick="handleClick()">Click Me</button>**

**<script>**

**function handleClick() {**

**alert("Button clicked!");**

**}**

**</script**

**</body>**

**</html>**

**Next.js**

**const express = require('express');**

**const app = express();**

**const port = 3000;**

**// Serve static files from the "public" directory**

**app.use(express.static('public'));**

**// Start the server**

**app.listen(port, () => {**

**console.log(`App is running at http://localhost:${port}`);**

**});**

Step 2: Create a Dockerfile

# Step 1: Use a Node.js base image

FROM node:18

# Step 2: Set the working directory inside the container

WORKDIR /app

# Step 3: Copy package.json and package-lock.json to the working directory

COPY package\*.json ./

# Step 4: Install dependencies

RUN npm install

# Step 5: Copy the rest of the application code to the working directory

COPY . .

# Step 6: Expose the port the app runs on

EXPOSE 3000

# Step 7: Start the application

CMD ["node", "index.js"]

**Step 3: Create .dockerignore File**

node\_modules

npm-debug.log

Dockerfile

.dockerignore

**Step 4: Build the Docker Image**

docker build -t my-express-app .

Step 5: Run the Docker Container

docker run -p 3000:3000 my-express-app

**6. Study expts- puppet n NAGIOS**

**### 6. Study Experiments: Puppet and Nagios**

**Both \*\*Puppet\*\* and \*\*Nagios\*\* are widely used tools in IT for \*\*automation\*\* and \*\*monitoring\*\* respectively. Let’s go through an overview of how you can study and experiment with \*\*Puppet\*\* for configuration management and \*\*Nagios\*\* for monitoring.**

**---**

**### Part 1: Puppet (Configuration Management Tool)**

**\*\*Puppet\*\* is a configuration management tool that automates the deployment, management, and configuration of servers. It uses a declarative language to describe the state of your systems.**

**#### Setting Up a Puppet Experiment**

**1. \*\*Install Puppet\*\***

**- \*\*Master-Agent Architecture\*\*: Puppet works with a centralized \*\*Puppet Master\*\* that manages configurations and multiple \*\*Puppet Agents\*\* that run on the nodes (servers).**

**- For experiments, you can either use \*\*Puppet Open Source\*\* or \*\*Puppet Enterprise\*\*.**

**\*\*Installation Example (Puppet Master on Ubuntu):\*\***

**```bash**

**sudo apt update**

**sudo apt install puppetmaster**

**```**

**2. \*\*Write a Puppet Manifest\*\***

**Puppet uses manifest files (`.pp` files) to define configurations. A manifest describes the desired state of the system using Puppet's declarative language.**

**\*\*Example of a simple manifest:\*\***

**```puppet**

**# install apache2 package and ensure it is running**

**package { 'apache2':**

**ensure => installed,**

**}**

**service { 'apache2':**

**ensure => running,**

**enable => true,**

**require => Package['apache2'],**

**}**

**file { '/var/www/html/index.html':**

**ensure => present,**

**content => "Hello, Puppet managed Apache server!",**

**}**

**```**

**This manifest will:**

**- Install Apache**

**- Ensure the Apache service is running**

**- Create an `index.html` file with custom content.**

**3. \*\*Apply the Manifest\*\***

**- For standalone setups, you can run Puppet in \*\*apply mode\*\* where you manually apply the manifest.**

**\*\*Run Puppet on the node:\*\***

**```bash**

**sudo puppet apply /path/to/manifest.pp**

**```**

**4. \*\*Master-Agent Setup (Optional)\*\***

**If you're studying the master-agent architecture, you need to set up Puppet Master and configure nodes to act as \*\*Puppet Agents\*\*. Agents will request configurations from the Puppet Master.**

**\*\*To install Puppet Agent (on another node/server):\*\***

**```bash**

**sudo apt update**

**sudo apt install puppet**

**```**

**\*\*Configure the Puppet Agent to talk to the Master:\*\***

**Edit the `/etc/puppet/puppet.conf` file and add the Puppet Master’s details:**

**```ini**

**[agent]**

**server = puppetmaster.localdomain**

**```**

**\*\*Run the agent to get the configuration from the Puppet Master:\*\***

**```bash**

**sudo puppet agent --test**

**```**

**5. \*\*Experiment Ideas\*\***

**- Automate installation of multiple packages (e.g., install Apache, MySQL, PHP on a web server).**

**- Manage file permissions or configure SSH settings across multiple servers.**

**- Use Puppet modules to configure more complex setups (e.g., database servers).**

**---**

**### Part 2: Nagios (Monitoring Tool)**

**\*\*Nagios\*\* is a powerful monitoring tool that helps you monitor servers, network devices, and applications. It alerts administrators about issues such as service downtimes, resource usage, or network disruptions.**

**#### Setting Up a Nagios Experiment**

**1. \*\*Install Nagios Core\*\***

**Nagios Core is the free version of Nagios and can be used to monitor different nodes.**

**\*\*Example Installation on Ubuntu:\*\***

**```bash**

**sudo apt update**

**sudo apt install nagios4 nagios-plugins nagios-nrpe-plugin**

**```**

**2. \*\*Configure Hosts and Services\*\***

**Nagios uses \*\*configuration files\*\* (`.cfg` files) to define which hosts and services to monitor.**

**\*\*Monitor a Host:\*\***

**- Add a new host to Nagios by editing the `hosts.cfg` file:**

**```cfg**

**define host {**

**use linux-server ; Inherit default settings**

**host\_name webserver**

**alias My Web Server**

**address 192.168.1.10 ; IP address of the server to monitor**

**}**

**```**

**\*\*Monitor a Service (e.g., HTTP service on a web server):\*\***

**- Add the following configuration to `services.cfg`:**

**```cfg**

**define service {**

**use generic-service**

**host\_name webserver**

**service\_description HTTP**

**check\_command check\_http**

**}**

**```**

**3. \*\*Start the Nagios Service\*\***

**After configuring your hosts and services, start or restart Nagios:**

**```bash**

**sudo systemctl restart nagios**

**```**

**4. \*\*Access the Nagios Web Interface\*\***

**Nagios provides a web interface where you can view the current status of hosts and services, set up alerts, and view performance data.**

**- Open a browser and go to `http://<your-server-ip>/nagios`.**

**- Login using the default credentials (you can set the admin password during installation).**

**5. \*\*Monitor Remote Hosts with NRPE\*\***

**Nagios uses the \*\*Nagios Remote Plugin Executor (NRPE)\*\* to monitor remote Linux hosts.**

**\*\*Steps to monitor a remote host:\*\***

**- Install NRPE and Nagios plugins on the remote host:**

**```bash**

**sudo apt install nagios-nrpe-server nagios-plugins**

**```**

**- Edit the NRPE configuration on the remote host (`/etc/nagios/nrpe.cfg`) to allow the Nagios server’s IP:**

**```ini**

**allowed\_hosts=127.0.0.1,192.168.1.5**

**```**

**- Restart NRPE:**

**```bash**

**sudo systemctl restart nagios-nrpe-server**

**```**

**- Add the remote host to Nagios configuration on the monitoring server as a new host, and configure the services to monitor (CPU, disk usage, etc.).**

**6. \*\*Experiment Ideas\*\***

**- Set up monitoring for different services (e.g., HTTP, MySQL, SSH) across multiple hosts.**

**- Use \*\*Nagios plugins\*\* to monitor specific application metrics (e.g., memory usage, CPU load, disk space).**

**- Configure \*\*email or SMS alerts\*\* when certain thresholds are crossed (e.g., high CPU usage).**

**---**

**### Conclusion**

**- \*\*Puppet\*\* is excellent for automating server configurations. With Puppet, you can easily experiment by creating manifests and applying them to various nodes, both in standalone or master-agent setups.**

**- \*\*Nagios\*\* is great for monitoring the state of your infrastructure. It helps you identify issues such as downtimes and abnormal resource usage. You can monitor multiple hosts and services, and set up alerts to get notified about issues.**

**Both tools are essential for DevOps practices, and experimenting with them will provide you with a strong foundation in \*\*automation\*\* and \*\*monitoring\*\*.**

**Command**

**Docker pull nginx**

**Docker -d –name my-container -p 8080:80 nginx**

**Docker run -d –name my-container -p 8080:80 nginx**