

Name: Vivek Saroj

Reg Email: viveksaroj098@gmail.com

Course Name: DevOps

Assignment Name: Module 32 Configuration Management.

Q1) Research and compare different configuration management tools (Ansible, Puppet, Chef) and write a report on their use cases and advantages.

Ans:

Sure! Here's a **detailed explanation** of **Ansible** with an **example tutorial** and a **sample playbook** to help you get started.

1) Ansible:

What is Ansible?

Ansible is an **open-source IT automation tool** developed by Red Hat. It allows you to:

- Provision infrastructure
 - Deploy applications
 - Automate system configuration
 - Manage servers without installing agents
-

How Ansible Works

- **Control Node:** The system where Ansible is installed.
- **Managed Nodes:** Target systems Ansible connects to via SSH.
- **Inventory:** A file (INI, YAML, or dynamic) that lists target hosts.
- **Playbook:** A YAML file describing the tasks Ansible should perform.

Basic Setup Requirements

- Python should be installed on the control node.
- Managed nodes must be accessible over SSH.
- No agent is needed on managed nodes.

Installation on Control Node (Ubuntu)

```
sudo apt update
sudo apt install ansible -y
```

Ansible Project Structure

```
my-ansible-project/
├── inventory.ini
└── playbook.yml
```

Step-by-Step Tutorial with Example

1. Create an Inventory File (**inventory.ini**)

```
[webservers]
192.168.1.10 ansible_user=ubuntu ansible_ssh_private_key_file=~/.ssh/id_rsa
```

This tells Ansible to connect to **192.168.1.10** using SSH and the private key.

2. Write a Basic Playbook (**playbook.yml**)

```
---
- name: Install Apache Web Server on Ubuntu
  hosts: webservers
  become: true # Runs as sudo
  tasks:
    - name: Update apt cache
      apt:
        update_cache: yes
```

```
- name: Install Apache2
  apt:
    name: apache2
    state: present

- name: Ensure Apache is running
  service:
    name: apache2
    state: started
    enabled: yes
```

What This Does:

- Updates the package cache
 - Installs Apache2
 - Ensures the Apache service is started and enabled on boot
-

3. Run the Playbook

```
ansible-playbook -i inventory.ini playbook.yml
```

Output will look like:

```
PLAY [Install Apache Web Server on Ubuntu] ***
TASK [Update apt cache] *****
ok: [192.168.1.10]
TASK [Install Apache2] *****
changed: [192.168.1.10]
TASK [Ensure Apache is running] *****
ok: [192.168.1.10]
```

Idempotency in Action

Re-running the playbook won't reinstall Apache if it's already present—this is called **idempotency**, a key feature of Ansible.

Common Ansible Modules

Module	Purpose	Example
<code>apt</code>	Manage packages on Debian/Ubuntu	Install software
<code>yum</code>	Manage packages on RHEL/CentOS	Install software
<code>copy</code>	Copy files to remote machines	Copy config files
<code>template</code>	Jinja2 templated config files	Generate config dynamically
<code>service</code>	Manage services (start/stop)	Start Apache or Nginx
<code>file</code>	Set file permissions/ownership	Create or modify file attributes

Advanced Use Case Example

Deploy a Web Page with Apache

```
---
- name: Deploy static website
  hosts: webservers
  become: yes
  tasks:
    - name: Install Apache
      apt:
        name: apache2
        state: present

    - name: Copy index.html
      copy:
        src: index.html
        dest: /var/www/html/index.html
```

And your `index.html` (local file):

```
<h1>Hello from Ansible!</h1>
```

Learning Resources

- [Official Ansible Docs](#)
 - [Ansible Galaxy](#) – community-contributed roles
 - Free courses on [YouTube](#)
-

Summary

Feature	Value
Language	YAML
Agentless	Yes (uses SSH)
Learning Curve	Easy
Best For	Quick automation, DevOps
Main Tool	<code>ansible-playbook</code>

2) Puppet:

Overview

- **Developed by:** Puppet, Inc.
 - **Language:** Puppet DSL (based on Ruby)
 - **Architecture:** Agent-based (Master-Agent model)
 - **Configuration Style:** Declarative
 - **Setup Complexity:** Moderate
-

How Puppet Works

Puppet uses a **client-server model**:

- **Puppet Master** (Server): Holds all configurations (manifests).
 - **Puppet Agent** (Client): Installed on each managed node, regularly checks in to apply configurations.
 - **Communication** is secure via SSL certificates.
-

Use Cases

- Managing large-scale infrastructures
 - Centralized configuration control
 - Enforcing compliance and security policies
 - Handling inter-resource dependencies
-

Advantages

- Mature and enterprise-proven
 - Excellent for compliance-heavy environments
 - Rich reporting and auditing tools
 - Scalable and modular with reusable modules
 - Strong community and Puppet Forge
-

Puppet Components

Component	Description
-----------	-------------

Manifest	File written in Puppet DSL (.pp) that contains configuration code
Module	A collection of manifests and files for a specific purpose
Resource	The basic unit (e.g., a file, package, or service)
Node	Represents a managed machine
Catalog	Compiled version of the manifest sent to an agent

Example: Basic Puppet Manifest

This example installs and manages Apache on an Ubuntu system.

apache.pp

```
class apache {  
  package { 'apache2':  
    ensure => installed,  
  }  
  
  service { 'apache2':  
    ensure => running,  
    enable => true,  
    require => Package['apache2'],  
  }  
  
  file { ['/var/www/html/index.html':  
    ensure => file,  
    content => '<h1>Hello from Puppet!</h1>',  
    require => Package['apache2'],  
  }  
}  
  
include apache
```

Step-by-Step Guide

1. Install Puppet Master and Agent

On the **Puppet Master** (e.g., Ubuntu):

```
sudo apt update
```

```
sudo apt install puppetserver
```

On a **Puppet Agent** node:

```
sudo apt update  
sudo apt install puppet-agent
```

Start the services:

```
sudo systemctl start puppet  
sudo systemctl enable puppet
```

2. Configure Manifest Directory

Default manifest location (on Master):

```
/etc/puppetlabs/code/environments/production/manifests/site.pp
```

Put the content of `apache.pp` into this file.

3. Sign Certificates

On the Master, sign the agent's certificate:

```
sudo puppetserver ca list  
sudo puppetserver ca sign --certname agent-hostname
```

4. Run Puppet on Agent

On the Agent node:

```
sudo puppet agent --test
```

This will pull the catalog from the Master and apply it.

Puppet Forge (<https://forge.puppet.com/>) hosts thousands of ready-to-use modules created by the community and vendors. You can install them using:

```
puppet module install puppetlabs-apache
```

✓ Summary

Feature	Value
Language	Puppet DSL (Ruby-based)
Architecture	Agent-Master (SSL-secured)
Best For	Large, compliance-heavy setups
Learning Curve	Moderate
Main Tool	<code>puppet agent --test</code>

3) Chef:

✓ Overview

- **Developed by:** Progress Software (formerly Chef Software Inc.)
 - **Language:** Ruby (Chef DSL for writing recipes)
 - **Architecture:** Agent-based (typically Chef Client-Server)
 - **Configuration Style:** Primarily Imperative (with some declarative aspects)
 - **Setup Complexity:** High
-

⚙️ How Chef Works

Chef uses a **client-server architecture**:

- **Chef Server:** Central repository for cookbooks, recipes, policies.
- **Chef Workstation:** Developer environment to write and upload code.
- **Chef Client:** Installed on each managed node, pulls configurations from the server and applies them.

Communication happens over HTTPS, using signed authentication keys.



Use Cases

- Managing complex and scalable infrastructure
 - Full application lifecycle management
 - Infrastructure as Code (IaC) with high customization
 - Automated testing of infrastructure code
-



Advantages

- Flexible and programmable using full Ruby
 - Great for complex logic and conditions
 - Excellent integration with AWS, Azure, and GCP
 - Supports **Test-Driven Infrastructure** (ChefSpec, InSpec)
 - Rich community ecosystem (Supermarket with cookbooks)
-



Core Concepts

Concept	Description
---------	-------------

Recipe	Collection of resources to configure a system
Cookbook	Folder structure that organizes recipes and other files
Resource	Represents a piece of configuration (e.g., file, package)
Node	A system managed by Chef
Runlist	Ordered list of recipes/roles applied to a node

Folder Structure (Cookbook Example)

my_apache_cookbook/

```
|— recipes/
|   |— default.rb
|— templates/
|   |— default/
|       |— index.html.erb
|— metadata.rb
|— README.md
```

Sample Recipe to Install Apache

recipes/default.rb

```
# Install Apache

package 'apache2' do

  action :install

end


# Enable and start the Apache service

service 'apache2' do

  action [:enable, :start]

end


# Create a custom homepage

file '/var/www/html/index.html' do

  content '<h1>Hello from Chef!</h1>'

  mode '0644'

  owner 'root'

  group 'root'

end
```



Test Infrastructure (ChefSpec + InSpec)

- **ChefSpec** allows you to write unit tests for your recipes.
- **InSpec** lets you test infrastructure compliance and security.

Example InSpec test:

```
describe package('apache2') do

  it { should be_installed }
```

end

```
describe service('apache2') do
```

```
  it { should be_running }
```

```
  it { should be_enabled }
```

```
end
```

Setup Process (High-Level)

Install Chef Workstation on your local machine

```
curl https://omnitruck.chef.io/install.sh | sudo bash -s -- -P chef-workstation
```

1.

Generate a cookbook

```
chef generate cookbook my_apache_cookbook
```

2.

3. **Write your recipe** in `recipes/default.rb`

Upload the cookbook to the Chef Server

```
knife cookbook upload my_apache_cookbook
```

4.

Bootstrap a node (agent)

```
knife bootstrap 192.168.1.10 -U user -P password --node-name webserver01 --run-list  
'recipe[my_apache_cookbook]'
```

5.

Learning Resources

- [Chef Official Docs](#)
 - [Chef Supermarket](#) – Community cookbooks
 - Tools: `knife`, `chef-client`, `chef generate`
-

✓ Summary

Feature	Value
Language	Ruby (Chef DSL)
Architecture	Client-Server (with optional Solo mode)
Best For	Complex, customizable deployments
Learning Curve	High
Notable Tools	ChefSpec, InSpec, Knife

2) Extend the CI pipeline to include continuous delivery using Jenkins or GitLab CI. Implement a blue-green deployment strategy?

Ans:

Blue-Green Deployment:

Blue-Green Deployment is a release strategy that minimizes downtime and risk:

- **Blue environment:** Current live production.

- **Green environment:** New version, tested and ready to go live.
 - **Switch traffic:** After validation, traffic is routed from Blue to Green.
-

Assumptions

- The application is containerized (Docker-based).
 - Using NGINX or a Load Balancer to switch environments.
 - Deploying to AWS EC2, EKS, or any VM with two environments (**blue** and **green**).
-

Jenkins Implementation

Prerequisites

- Jenkins installed with:
 - Docker
 - SSH or AWS CLI access
 - Blue and Green environments prepared
- Load balancer (e.g., NGINX or AWS ALB)

Jenkinsfile with Blue-Green Deployment

```
pipeline {
```

```
    agent any
```

```
environment {  
    APP_NAME = 'myapp'  
    BLUE_PORT = '8081'  
    GREEN_PORT = '8082'  
    ACTIVE_ENV_FILE = '/var/www/env/active' // e.g., file storing 'blue' or  
'green'  
}
```

```
stages {  
    stage('Checkout') {  
        steps {  
            git 'https://github.com/your-org/your-repo.git'  
        }  
    }  
}
```

```
stage('Build Docker Image') {  
    steps {  
        sh 'docker build -t myapp:latest .'    }  
}
```

```
stage('Determine Target Environment') {  
    steps {  
        script {
```



```
    def currentEnv = sh(script: "cat ${ACTIVE_ENV_FILE}", returnStdout:
true).trim()
```

```
    def targetEnv = (currentEnv == 'blue') ? 'green' : 'blue'
```

```
    env.TARGET_ENV = targetEnv
```

```
    env.TARGET_PORT = (targetEnv == 'blue') ? BLUE_PORT :
GREEN_PORT
```

```
    }
```

```
  }
```

```
}
```

```
stage('Deploy to Target Environment') {
```

```
  steps {
```

```
    script {
```

```
      sh """
```

```
        docker stop ${APP_NAME}_${TARGET_ENV} || true
```

```
        docker rm ${APP_NAME}_${TARGET_ENV} || true
```

```
        docker run -d -p ${TARGET_PORT}:80 --name
${APP_NAME}_${TARGET_ENV} myapp:latest
```

```
      """
```

```
    }
```

```
  }
```

```
}
```

```
stage('Health Check') {
```

```
  steps {
```

```
    sh "curl --fail http://localhost:${TARGET_PORT}/health"
  }
}
```

```
stage('Switch Traffic') {
  steps {
    script {
      // Assuming the load balancer reads env from a file or dynamic config
      sh "echo ${TARGET_ENV} > ${ACTIVE_ENV_FILE}"
    }
  }
}
```

```
stage('Cleanup Old Environment') {
  steps {
    script {
      def oldEnv = (env.TARGET_ENV == 'blue') ? 'green' : 'blue'
      sh "docker stop ${APP_NAME}_${oldEnv} || true && docker rm
${APP_NAME}_${oldEnv} || true"
    }
  }
}
}
```

GitLab CI/CD Implementation

`.gitlab-ci.yml` for Blue-Green Deployment

stages:

- build
- deploy
- switch
- cleanup

variables:

BLUE_PORT: "8081"

GREEN_PORT: "8082"

ACTIVE_ENV_FILE: "/var/www/env/active"

build:

stage: build

script:

- docker build -t myapp:latest .

deploy:

stage: deploy

script:

- export CURRENT_ENV=\$(cat \$ACTIVE_ENV_FILE)
- export TARGET_ENV=\$(["\$CURRENT_ENV" = "blue"] && echo green || echo blue)
- export TARGET_PORT=\$(["\$TARGET_ENV" = "blue"] && echo \$BLUE_PORT || echo \$GREEN_PORT)
- docker stop myapp_\$TARGET_ENV || true
- docker rm myapp_\$TARGET_ENV || true
- docker run -d -p \$TARGET_PORT:80 --name myapp_\$TARGET_ENV myapp:latest

health_check:

stage: switch

script:

- curl --fail http://localhost:\$TARGET_PORT/health

switch_traffic:

stage: switch

script:

- echo \$TARGET_ENV > \$ACTIVE_ENV_FILE

cleanup:

stage: cleanup

script:

- export OLD_ENV=\$(["\$TARGET_ENV" = "blue"] && echo green || echo blue)
- docker stop myapp_\$OLD_ENV || true

- docker rm myapp_\$OLD_ENV || true

Load Balancer Configuration (Example: NGINX)

Configure NGINX to route to the active environment:

```
upstream myapp {  
    include /etc/nginx/conf.d/active_env.conf;  
}
```

```
server {  
    listen 80;  
    location / {  
        proxy_pass http://myapp;  
    }  
}
```

active_env.conf (dynamically written)

server 127.0.0.1:8081; # or 8082 depending on active environment


This file is updated by the CI job based on the **\$ACTIVE_ENV_FILE**.

Tools That Help

- **Terraform:** For infrastructure provisioning
 - **Consul** or **Etcd:** To track active environment in a more scalable way
 - **Kubernetes:** If you want to manage blue-green deployments at cluster scale
 - **Argo Rollouts:** For advanced deployment strategies in K8s
-

Summary

Feature	Jenkins	GitLab CI
Setup Effort	Medium to High	Medium
Built-in Git integration	Needs plugins	Native Git integration
CD Support	Native through pipelines & plugins	Native pipelines
Blue-Green Strategy	Manual script + load balancer or plugin	Inline scripting with YAML

**Jenkins**

🔍

🔔 1

👤 vivek ramsaran saroj

🚪 log out

Dashboard >

+ New Item

📁 Build History

⚙️ Manage Jenkins

📅 My Views

Build Queue

No builds in the queue.

Build Executor Status

0/2

All

+


S	W	Name ↓	Last Success	Last Failure	Last Duration
🟢	☁️	clone	5 days 8 hr #22	5 days 8 hr #20	9 sec
🟢	☀️	sample-1	5 days 10 hr #4	N/A	1 sec

Icon: S M L

✎ Add description

REST APIJenkins 2.504.1

☰

vivek-rob-mec / React-CD-Setup

🔍 Type ↵ to search

🗑️

+

🔄

🔧

📧

👤

<> Code

🔗 Issues

🔄 Pull requests

🔄 Actions

📁 Projects

📖 Wiki

🛡️ Security

📊 Insights

⚙️ Settings

📁 Files

📁 master

🔍 Go to file

> .github

> public

> src

📄 .gitignore

📄 Jenkinsfile

📄 README.md

📄 eslint.config.js

📄 index.html

📄 package-lock.json

📄 package.json

📄 vite.config.js

React-CD-Setup / Jenkinsfile

👤 vivek-rob-mec Added jenkins file ✓ 21465f7 · 3 days ago History

Code

Blame

10 lines (10 loc) · 156 Bytes

🚀 Code 55% faster with GitHub Copilot

Raw

📄

📄

✎

🔍

```
1 pipeline {
2   agent any
3   stages{
4     stage ("Build"){
5       steps{
6         echo "Hello world"
7       }
8     }
9   }
10 }
```

← → ↺ 🏠 🌐 localhost:8080/job/clone/configure ☆ 📄 ↺ ☆ 🌐 ⋮

Dashboard > clone > Configuration

Configure

⚙️ General

🕒 Triggers

🔧 Pipeline

🔗 Advanced

Script ?

```
9      }
10    }
11  }
12  stage("checkout code") {
13    steps {
14      git branch: 'main', url: 'https://github.com/vivek-rob-mec/Python_server_FastAPI.git'
15    }
16  }
17 }
18 post{
19   success{
20     archiveArtifacts artifacts: "**/*"
21   }
22 }
23 }
```

☒ Use Groovy Sandbox ?

[Pipeline Syntax](#)

Save

Apply