Set 1 Solutions

Q1: Ansible Playbook for Nginx

yaml

```
---
- name: Install and start Nginx web server
  hosts: webservers
  become: yes
  tasks:
    - name: Update package cache
      apt:
        update_cache: yes
      when: ansible_os_family == "Debian"

    - name: Install Nginx
      package:
        name: nginx
        state: present

    - name: Start and enable Nginx service
      systemd:
        name: nginx
        state: started
        enabled: yes

    - name: Verify Nginx is running
      uri:
        url: "http://localhost:80"
        status_code: 200
      register: nginx_status
      ignore_errors: yes

    - name: Display Nginx status
      debug:
        msg: "Nginx is installed and running successfully"
      when: nginx_status.status == 200
```

Q2: Run Docker Container from Jenkins Pipeline Image

bash

Run the container

docker run -d --name myapp-container -p 8080:8080 your-username/your-image:latest

Display logs

docker logs myapp-container

Or to follow logs in real-time

docker logs -f myapp-container

Q3: Manual Maven Build in Jenkins

1. **Create a new Jenkins job:**
    - Go to Jenkins → New Item → Freestyle project
    - Name: "Manual-Maven-Build"
2. **Configure the job:**
    - **Source Code Management:** Git
        - Repository URL: Your Git repository URL
    - **Build:** Invoke top-level Maven targets
        - Maven Version: Select your Maven installation
        - Goals: clean compile package
        - POM: pom.xml (default)
3. **Run the build:**
    - Click "Build Now"
    - Check console output for "BUILD SUCCESS" message

Expected console output snippet:

text

[INFO] Building jar: /path/to/your/project/target/your-app.jar

[INFO] --

[INFO] BUILD SUCCESS

[INFO] --

[INFO] Total time: 2.345 s

[INFO] Finished at: 2024-01-15T10:30:45Z

[INFO] --

Set 2 Solutions

Q1: Ansible Playbook for Apache2

yaml

```
---
- name: Install and start Apache2 web server
  hosts: webservers
  become: yes
  tasks:
    - name: Update package cache
      apt:
        update_cache: yes
      when: ansible_os_family == "Debian"

    - name: Install Apache2
      package:
        name: apache2
        state: present

    - name: Start and enable Apache2 service
      systemd:
        name: apache2
        state: started
        enabled: yes

    - name: Ensure Apache2 is accessible
      uri:
        url: "http://{{ ansible_host }}:80"
        status_code: 200
      register: apache_status

    - name: Display success message
      debug:
        msg: "Apache2 installed, service running, and accessible via browser"
      when: apache_status.status == 200
```

Q2: Dockerfile, Build and Push to Docker Hub

Dockerfile:

dockerfile

```
FROM nginx:alpine

COPY . /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

Build and push commands:

bash

```
# Build the image

docker build -t your-username/your-app:latest .


# Login to Docker Hub

docker login


# Push to Docker Hub

docker push your-username/your-app:latest
```

Expected output:

text

```
The push refers to repository [docker.io/your-username/your-app]

a1b2c3d4e5f6: Pushed

latest: digest: sha256:abc123... size: 1234
```

Q3: Git Repository with Multiple Versions

bash

```
# Create and initialize repository
mkdir my-project
cd my-project
git init

# Create first version of file
echo "<html><body><h1>Version 1</h1></body></html>" > index.html

# First commit
git add index.html
git commit -m "Initial commit - Version 1"

# Create second version
echo "<html><body><h1>Version 2 - Improved</h1><p>New features added</p></body></html>" > index.html

# Second commit
git add index.html
git commit -m "Update to Version 2 with new features"

# Push to GitHub (replace with your repo URL)
git remote add origin https://github.com/your-username/your-repo.git
git branch -M main
git push -u origin main
```

Set 3 Solutions

Q1: Ansible Playbook for HAProxy

yaml

```
---
- name: Install and configure HAProxy
  hosts: loadbalancers
  become: yes
  tasks:
    - name: Update system package repository
      apt:
        update_cache: yes
        cache_valid_time: 3600
      when: ansible_os_family == "Debian"

    - name: Install HAProxy package
      package:
        name: haproxy
        state: present

    - name: Enable HAProxy service
      systemd:
        name: haproxy
        enabled: yes

    - name: Start HAProxy service
      systemd:
        name: haproxy
        state: started

    - name: Verify HAProxy is running
      systemd:
        name: haproxy
      register: haproxy_status

    - name: Display installation status
```

```
    debug:
      msg: "HAProxy successfully installed and running"
    when: haproxy_status.ActiveState == "active"
```

Q2: Dockerfile for Python Application

Dockerfile:

dockerfile

```
FROM python:3.9-alpine

WORKDIR /app

# Copy the Python script
COPY hello.py .

# Run the Python script
CMD ["python", "hello.py"]
```

hello.py:

python

```
#!/usr/bin/env python3
print("Hello from Docker and Python!")
```

Build and run commands:

bash

```
# Build the image
docker build -t python-hello-app .

# Run the container
docker run python-hello-app
```

Expected output:

text

```
Hello from Docker and Python!
```

Q3: Git Repository with index.html

bash

```
# Create and setup repository
mkdir website-project
cd website-project
git init

# Create index.html file
echo "<!DOCTYPE html>
<html>
<head>
  <title>My Website</title>
</head>
<body>
  <h1>Welcome to My Website</h1>
  <p>This is the initial version of my website.</p>
</body>
</html>" > index.html

# Initial commit and push
git add index.html
git commit -m "Initial commit"

# Add remote repository (replace with your GitHub repo URL)
git remote add origin https://github.com/your-username/website-project.git
git branch -M main
git push -u origin main
```

Verification:

- Go to your GitHub repository
- Check that:
    - Repository exists with index.html file
    - Commit history shows "Initial commit" message
    - File content matches what was committed

Set 4 Solutions

Q1: Git Repository with Java File

bash

```
# Create and initialize repository
mkdir java-project
cd java-project
git init

# Create HelloWorld.java file
cat > HelloWorld.java << 'EOF'
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
EOF

# Commit and push
git add HelloWorld.java
git commit -m "Add HelloWorld.java - initial Java program"
git branch -M main

# Add remote repository (replace with your GitHub URL)
git remote add origin https://github.com/your-username/java-project.git
git push -u origin main
```

Verification: Check GitHub repository to see:

- HelloWorld.java file present
- Commit message "Add HelloWorld.java - initial Java program" visible in history
- Code content matches the created file

Q2: Create and Build Maven Project

bash

```
# Create Maven project using archetype
mvn archetype:generate -DgroupId=com.example -DartifactId=my-app \
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

# Navigate to project directory
cd my-app

# Build the project
mvn clean package
```

Expected Output:

text

```
[INFO] Building jar: /path/to/my-app/target/my-app-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.456 s
[INFO] Finished at: 2024-01-15T10:30:45Z
[INFO] ------------------------------------------------------------------------
```

Verify .jar file:

bash

```
ls target/
# Should show my-app-1.0-SNAPSHOT.jar
```

Q3: Dockerfile for Maven Project

Dockerfile:

dockerfile

```
FROM maven:3.8.5-openjdk-11 AS build
WORKDIR /app
COPY . .
RUN mvn clean package -DskipTests

FROM openjdk:11-jre-slim
WORKDIR /app
COPY --from=build /app/target/*.jar app.jar
EXPOSE 8080
CMD ["java", "-jar", "app.jar"]
```

Build the Docker image:

bash

```
# Build image
docker build -t my-maven-app:latest .

# Verify image creation
docker images | grep my-maven-app
```

Expected Output:

text

```
REPOSITORY    TAG     IMAGE ID      CREATED       SIZE
my-maven-app   latest   abc123def456   2 minutes ago   250MB
```

Set 5 Solutions

Q1: Jenkins Freestyle Project with Maven

1. **Create New Jenkins Job:**
    - Go to Jenkins → New Item
    - Name: "Maven-GitHub-Build"
    - Type: Freestyle project
2. **Configure Source Code Management:**
    - Select "Git"
    - Repository URL: https://github.com/your-username/your-repo.git
    - Credentials: Add if repository is private
3. **Configure Build:**
    - Add build step: "Invoke top-level Maven targets"
    - Maven Version: Select your Maven installation
    - Goals: clean package
    - POM: pom.xml (default)
4. **Save and Build:**
    - Click "Save"
    - Click "Build Now"

Expected Output: Build shows blue ball (success) in build history with console output showing Maven build success.

Q2: Jenkins GitHub Webhook Auto-trigger

1. **Configure Jenkins Job:**
    - Go to your Jenkins job → Configure
    - Under "Build Triggers" section:
        - Check "GitHub hook trigger for GITScm polling"
2. **Configure GitHub Webhook:**
    - Go to your GitHub repository → Settings → Webhooks → Add webhook
    - Payload URL: http://your-jenkins-server/github-webhook/
    - Content type: application/json
    - Which events: "Just the push event"
    - Active: Checked
3. **Test the Setup:**

bash

Make a change to your code

echo "// New comment" >> src/main/java/App.java

git add .

git commit -m "Test webhook trigger"

git push

Expected Output: Jenkins job automatically starts building within seconds of pushing to GitHub.

Q3: Run Docker Container and Verify

bash

Run container from image (using nginx as example)

docker run -d --name my-container -p 8080:80 nginx:alpine

Verify container is running

docker ps

Check container logs

docker logs my-container

Expected docker ps **Output:**

text

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

a1b2c3d4e5f6 nginx:alpine "/docker-entrypoint...." 10 seconds ago Up 9 seconds 0.0.0.0:8080->80/tcp my-container

Expected Logs Output:

text

/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration

/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/

/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh

...

Set 6 Solutions

Q1: Clone, Modify, and Push Java File

bash

```
# Clone repository (replace with your repo URL)
git clone https://github.com/your-username/java-project.git
cd java-project

# Modify the Java file
cat >> HelloWorld.java << 'EOF'

// New method added
public static void greet(String name) {
  System.out.println("Hello, " + name + "!");
}
EOF

# Commit and push changes
git add HelloWorld.java
git commit -m "Add greet method to HelloWorld class"
git push origin main
```

Verification:

- Go to GitHub repository
- Check that:
    - HelloWorld.java shows the new method
    - New commit "Add greet method to HelloWorld class" appears in history
    - File diff shows the added lines

Q2: Jenkins Pipeline with Stages

Jenkinsfile:

groovy

```
pipeline {
  agent any

  tools {
    maven 'M3'
  }
```

```
  stages {
    stage('Checkout') {
      steps {
        git branch: 'main',
        url: 'https://github.com/your-username/your-repo.git'
        echo 'Code checkout completed'
      }
    }

    stage('Build') {
      steps {
        sh 'mvn clean compile'
        echo 'Build completed successfully'
      }
    }

    stage('Test') {
      steps {
        sh 'mvn test'
        echo 'Tests completed'
      }
    }
  }

  post {
    always {
      echo 'Pipeline execution completed'
    }
    success {
      echo 'Pipeline succeeded!'
    }
  }
}
```

Execution:

1. Create new Pipeline job in Jenkins
2. Paste the Jenkinsfile content
3. Run the pipeline

Expected Console Output:

text

[Pipeline] stage

[Pipeline] { (Checkout)

[Pipeline] git

[Pipeline] echo

Code checkout completed

[Pipeline] }

[Pipeline] // stage

[Pipeline] stage

[Pipeline] { (Build)

[Pipeline] sh

[INFO] BUILD SUCCESS

[Pipeline] echo

Build completed successfully

[Pipeline] }

[Pipeline] // stage

[Pipeline] stage

[Pipeline] { (Test)

[Pipeline] sh

[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[Pipeline] echo

Tests completed

[Pipeline] }

[Pipeline] // stage

[Pipeline] echo

Pipeline execution completed

[Pipeline] echo

Pipeline succeeded!

Q3: Build Docker Image for Java Project

Dockerfile:

dockerfile

```
FROM openjdk:11-jre-slim

WORKDIR /app

COPY target/*.jar app.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "app.jar"]
```

Build and Verify:

```
bash

# Ensure your Java project is built first

mvn clean package


# Build Docker image

docker build -t my-java-app:latest .


# Verify image creation

docker images
```

Expected Output:

```
text

REPOSITORY     TAG      IMAGE ID      CREATED         SIZE

my-java-app    latest   1a2b3c4d5e6f  30 seconds ago  250MB
```

Additional verification:

```
bash

# Check image details

docker image inspect my-java-app:latest


# Test running the container

docker run -d --name test-app -p 8080:8080 my-java-app:latest

docker ps | grep test-app
```