

**SOFTWARE DEVELOPER
EPAM (CLOUD & DEVOPS)**

A Training Report

Submitted in partial fulfilment of the requirements for the award of the degree of

**Bachelor of Technology
Computer Science and Engineering
(Data Science)**

**LOVELY PROFESSIONAL UNIVERSITY
PHAGWARA, PUNJAB**



FROM 10/1/2023 -Present

SUBMITTED BY

Name of the student - Sana Khan

Regd No. -11902857

Signature - Sanakhan

SUBMITTED TO

Name of the supervisor- Amandeep Kaur

Designation:

Signature:

Student Declaration

To whom so ever it may concern

I, Sana Khan, 11902857, hereby declare that the work done by me on

"EPAM CLOUD AND DEVOPS " from 11TH JAN-2023to Present, under the supervision of **Amandeep Kaur**, and Name of Internal supervisor - Amandeep Kaur
Lovely Professional University,

Phagwara, Punjab, is a record of original work for the partial fulfilment of the requirements for the award of the degree Computer Science and Engineering.

Name of the Student (Registration Number)

Sana Khan(11902857)

Sana Khan

Signature of the student Dated: 01/05/2023

Declaration by the supervisors

To whom so ever it may concern

This is to certify that Divyansh 11912910 from Lovely Professional University, Phagwara, Punjab, has worked as a trainee in AETHEREUS on "SALESFORCE" under my supervision from 11th July,2022to Present. It is further stated that the work carried out by the student is a record of original work to the best of my knowledge for the partial fulfilment of the requirements for the B-TECH, Computer Science and Engineering award.

Name of Internal Supervisor: Amandeep Kaur

Signature of the Internal Supervisor

Dated :

Table of Contents

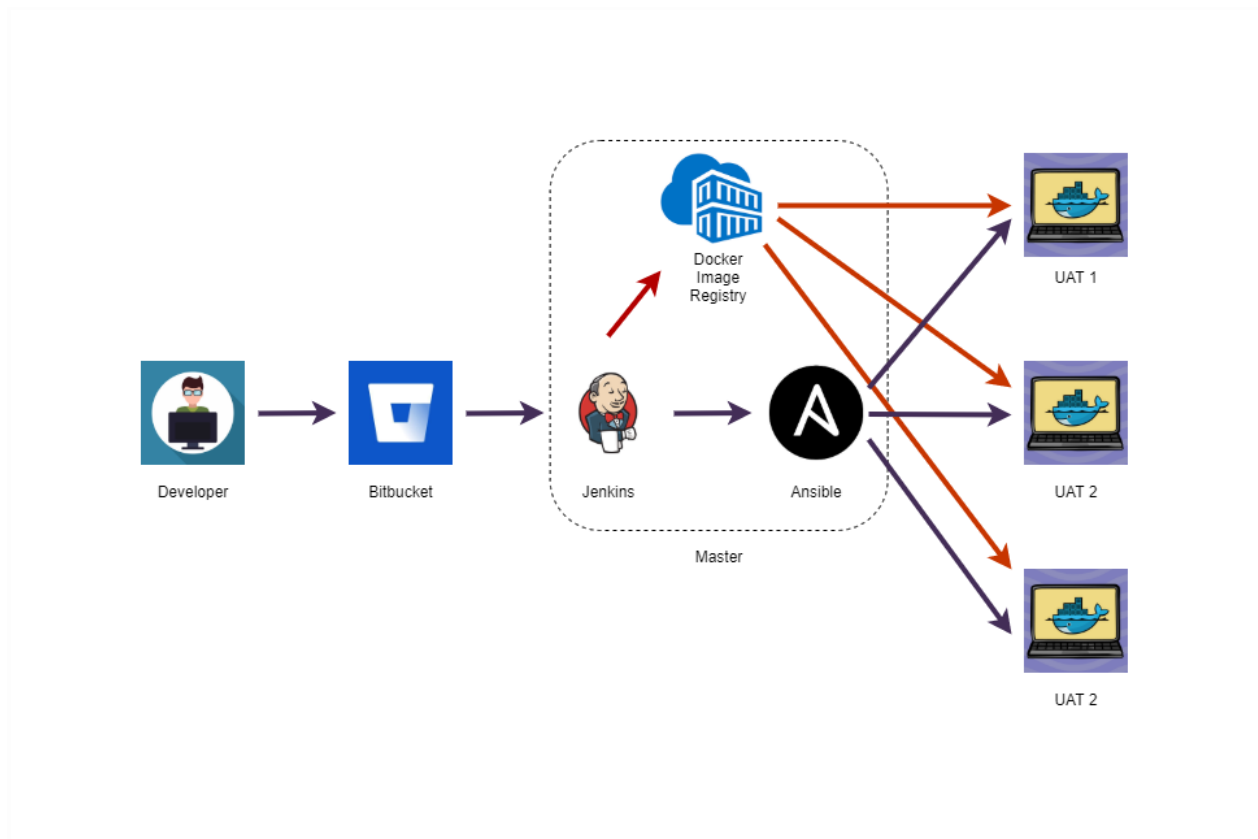
S.No.	Title
1.	Declaration by Student
2.	Declaration by Supervisors
3.	Undertaking form
4.	Acknowledgement
5.	List of Content
6.	CHAPTER1: PROJECT INTRODUCTION & OVERVIEW
7.	CHAPTER2: THE PROCEDURE STEPS AND SCREENSHOTS
8.	Chapter-3 TECHNOLOGIES LEARNT DURING Training
9.	Chapter-4 CONCLUSION

CHAPTER 1

INTRODUCTION TO THE PROJECT

Docker container deployment with Jenkins and Ansible (Project Assigned)

When we work as developers, we write code. Surely (if you are in a small scale company) we have to set up the applications and release them for Dev, QA and UAT environments. That is lots of work, out of scope. So how about having all those automated and we don't need to worry about CI/CD at all



Why we need this setup?

- Single point of deployment

- Less time to deployments and focus more on development
- No need to SSH to different servers time to time (loosely couple slave nodes from master)
- Easy replication of environments i.e. identical nodes
- Binding multiple phases i.e. testing phase
- Flexible configuration of pipelines

What we are going to do ?

1. Create a simple pipeline to deploy `Hello Service`
2. Create a parameterized pipeline to deploy `Hello Service`
3. Create a generic service deployment pipeline
4. Create a generic service build pipeline
5. Create a parameterized pipeline to clean environments

Description of the Project

The project involves the deployment of a sample web application in a Docker container using Jenkins and Ansible. The web application is a simple "Hello World" application that is written in Python and uses the Flask web framework.

The project involves the following steps:

1. Building a Docker image for the web application
2. Pushing the Docker image to a Docker registry
3. Deploying the Docker container using Ansible
4. Testing the deployed application

The purpose of this report is to provide a detailed overview of the Docker container deployment process using Jenkins and Ansible. This report will cover the following topics:

- Overview of Docker container deployment
- Introduction to Jenkins and Ansible
- Description of the project
- Implementation of the project using Jenkins and Ansible
- Conclusion and future work

Overview of Docker Container Deployment

Docker container deployment is the process of deploying applications in a containerized environment using Docker technology. Docker containers provide a lightweight, portable, and isolated environment for running applications.

Docker containers are created using Docker images, which are essentially templates for creating containers. Docker images can be stored in a central repository called a Docker registry, which can be accessed by multiple users.

Docker provides a range of features that make it easy to create and manage containers. Some of these features include:

- **Dockerfile:** A Dockerfile is a script that is used to build a Docker image. It contains a set of instructions for creating a container image.
- **Docker Compose:** Docker Compose is a tool that is used to define and run multi-container Docker applications. It provides a simple way to define and run complex applications.
- **Docker Swarm:** Docker Swarm is a native clustering tool for Docker. It allows you to create and manage a cluster of Docker nodes.

Introduction to Jenkins and Ansible

Jenkins is an open-source automation server that is used to automate various tasks such as building, testing, and deploying applications. Jenkins provides a user-friendly web interface for configuring and running automated tasks.

Jenkins has a large number of plugins that can be used to extend its functionality. Some of the popular plugins for Docker container deployment include:

- **Docker Pipeline Plugin:** This plugin provides a set of Jenkins pipeline steps for building, testing, and deploying Docker images.
- **Docker Build and Publish Plugin:** This plugin allows you to build and publish Docker images from within Jenkins.
- **Docker Hub Notification Plugin:** This plugin notifies Docker Hub about the status of your Jenkins builds.

Ansible is an open-source configuration management tool that is used to automate the deployment of applications and services. Ansible provides a simple, declarative language for describing the desired state of a system.

Ansible is an open-source configuration management tool that is used to automate the deployment of applications and services. Ansible provides a simple, declarative language for describing the desired state of a system.

Ansible uses SSH to communicate with remote servers and can manage servers running a wide range of operating systems, including Linux, macOS, and Windows.

Ansible provides a range of modules that can be used to perform tasks such as installing packages, configuring services, and managing users.

The project was implemented using the following steps:

1. **Setting up a Jenkins server:** A Jenkins server was set up on a Linux server running Ubuntu. The server was configured with the necessary plugins for Docker container deployment.
2. **Creating a Jenkins job:** A Jenkins job was created to build the Docker image for the web application. The job was configured to use a Dockerfile to build the image and then push it to a Docker registry. The Dockerfile included the necessary instructions for installing the required packages and dependencies for the web application.
3. **Creating an Ansible playbook:** An Ansible playbook was created to deploy the Docker container on a remote server. The playbook was configured to use the Docker Compose

Evaluation on tools

1. **Jenkins - Powerful CI/CD tool used to automate build pipelines**

2. Ansible - Powerful IT automation tool which is heavily used to configuration management, automating deployments, consuming pull configurations
 - Similar products - chef, puppet, salt stack (mostly push configurations)
3. Jenkins vs Ansible: <https://www.javatpoint.com/jenkins-vs-ansible>

Jenkins VS Ansible

Ansible is a powerful tool for automation to the provision of the target environment and to then deploy the application. It helps you to do the configuration management, application deployment, task automation, and also IT orchestration. It can run tasks in a sequence and create a chain of events happening on different servers or devices.

And Jenkins is a popular tool for IT automation and used for CI/CD to provision the target environment.

Jenkins

Jenkins is the most popular open-source automation server that was written in a Java programming language. It facilitates the automation process of continuous integration and continuous delivery (CI/CD) in the software development process.

Jenkins supports over 1,400 plugins for other software tools. These plugins expand Jenkins into five years; platforms, UI, administration, source code management, and build management.

Jenkins is easy to install and use. It provides an impressive browser-hosted project management dashboard.

Some of the common reasons to evaluate and choose Jenkins include:

- Open-source and free
- Widely used and well documented
- Vibrant user community
- Integration with a large variety of tools and technologies.
- Plugin support
- Easy to install, configure and upgrade

- Distributed builds
- Monitoring external jobs
- Support for various authentication methods, notification, version control system, etc.

Ansible

Ansible is an IT automation tool. It can deploy software, configure systems, and orchestrate more advanced IT tasks such as CD (Continuous Deployment) or zero downtime rolling updates.

Automation simplifies complex tasks, not just making developers' jobs more manageable but allowing them to focus attention on other tasks that value to an organization.

In other words, it frees up time and increases efficiency. Ansible is rapidly rising to the top in the world of automation tools.

Ansible uses the simple YAML syntax. One of the other features of Ansible is its Agentless architecture. For automating configuration management, a lightweight and secure solution is Ansible. There are several modules in Ansible. Within Jenkins pipeline, applications could be deployed, and the environment could be provisioned using the Ansible tool.

Let's see some advantages and features of Ansible:

- Ansible is an open-source tool.
- No special coding skills are required to use Ansible's playbooks.
- Ansible allows you to model even highly complex IT workflows.
- You can orchestrate the entire application environment no matter where it is deployed. You can also customize it based on your needs.
- You do not need to install any other software or firewall ports on the client systems you want to automate.

- You do not need to set up a separate management structure.
- Because you don't have to install any extra software, there is more room for application resources on your server.
- Ansible is designed to be very simple, reliable, and consistent for configuration management.

What is a container?

Simply put, a container is a sandboxed process on your machine that is isolated from all other processes on the host machine. That isolation leverages [kernel namespaces and cgroups](#), features that have been in Linux for a long time. Docker has worked to make these capabilities approachable and easy to use. To summarize, a container:

- Is a runnable instance of an image. You can create, start, stop, move, or delete a container using the DockerAPI or CLI.
- Can be run on local machines, virtual machines or deployed to the cloud.
- Is portable (can be run on any OS).
- Is isolated from other containers and runs its own software, binaries, and configurations.

What is a container image?

When running a container, it uses an isolated filesystem. This custom filesystem is provided by a container image. Since the image contains the container's filesystem, it must contain everything needed to run an application - all dependencies, configurations, scripts, binaries, etc. The image also contains other configuration for the container, such as environment variables, a default command to run, and other metadata.

You'll dive deeper into images later on in this guide, covering topics such as layering, best practices, and more.

Step 0 - Configuring SSH connection b/w master and slaves

There are several methods to create a secure connection between master and the remote slave nodes. There are no person to enter username/password therefore best way is to create SSH connection with username + private key.

We need to create public/private keys in master.

Run `ssh-keygen` on master and `cat /root/.ssh/id_rsa` to get the private key which we need to add to ansible-playbook script as mentioned in tutorial[1]

Next we need to push public keys to slave nodes, run `ssh-copy-id <slave-ip>`. Do this for all the node group you need to access with this certificate. You can have multiple certificates for group-wise ; app-servers, db-servers etc.

In tutorial[2] , other few SSH configuration methods have been mentioned but here we are going with `SSH username with private key method`

If successful `ansible -i hosts -m ping` should give success response. Here `hosts` file is the ansible inventory.

Step 0.1 - Add following plugins to jenkins server

- Parameterized build trigger plugin
- Bitbucket plugin
- Ansible plugin

Step 0.2 - Create a simple webservice to deploy

I have created simple hello world springboot project

(<https://github.com/isurunuwanthilaka/hello-world>), Once we deploy to slave node we should be able to get the response at `http://<ip>:8082/hello`

Create a simple pipeline to deploy `Hello Service`

Step 1.1 - Creating pipeline

Login into Jenkins server and start a `pipeline` type project and write the following declarative pipeline.

Basic Pipeline

Code Blame 58 lines (45 loc) · 1.46 KB Raw Copy Download Edit Search

```
1 pipeline {
2   agent any
3
4   tools{
5     maven 'maven'
6   }
7
8   environment{
9     BRANCH = 'main'
10    TAG = 'dev'
11    REGISTRY = '<ip>:5000'
12    DEPLOY_T0 = 'dev2'
13  }
14
15  stages {
16    stage('Clone') {
17      steps {
18        git branch: "${BRANCH}" , credentialsId: 'bitbucket', url: 'https://isurunuwanthilaka@bitbucket.org/isurunuwanthilaka/hello
19
20      }
21    }
22
23    stage('Maven Build') {
24      steps {
25        sh "mvn clean package"
26
27      }
28    }
29
30    stage('Docker Build') {
31      steps {
32        sh "docker build -t ${REGISTRY}/hello:${TAG} ./"
33
34    }
```

jenkins-ansible-docker-deployments / basic-pipeline ↑ Top

Code Blame 58 lines (45 loc) · 1.46 KB Raw Copy Download Edit Search

Docker-clean-env.yml

Step 1.2 Create ansible playbook

Create directory at `/home/src/ansible-scripts` where we store ansible-playbooks

(In future we will store everything in a git repo so it is easy to version)

Now create `docker-deployment.yml`, this is the ansible playbook for first pipeline

Step 1.3 Lets create ansible inventory

Create another file `inventory.inv` with the IPs to slave nodes.

```
1  [dev1]
2  <dev1-ip> ansible_user=root
3
4  [dev2]
5  <dev2-ip> ansible_user=root
```

CodeBlame29 lines (27 loc) · 744 BytesRawCopyDownloadEditDropdownCode

```
1  ---
2  - hosts: "{{ENV}}"
3    gather_facts: false
4    tasks:
5      - name: Get running containers
6        docker_host_info:
7          containers: yes
8          register: docker_info
9
10     - name: Stop running containers
11       docker_container:
12         name: "{{ item }}"
13         state: stopped
14       loop: "{{ docker_info.containers | map(attribute='Id') | list }}"
15     - name: Remove Stopped docker containers
16       shell: |
17         docker rm $(docker ps -a -q);
18       when: docker_info.containers != 0
19
20     - name: Get details of all images
21       docker_host_info:
22         images: yes
23         verbose_output: yes
24       register: image_info
25     - name: Remove all images
26       docker_image:
27         name: "{{ item }}"
28         state: absent
29       loop: "{{ image_info.images | map(attribute='Id') | list }}"
```

UW PICO 5.09File: docker-clean.ymlModified

```
---
- hosts: "{{ENV}}"
  gather_facts: false
  tasks:
    - name: Get running containers
      docker_host_info:
        containers: yes
        register: docker_info

    - name: Stop running containers
      docker_container:
        name: "{{ item }}"
        state: stopped
      loop: "{{ docker_info.containers | map(attribute='Id') | list }}"
    - name: Remove Stopped docker containers
      shell: |
        docker rm $(docker ps -a -q);
      when: docker_info.containers != 0

    - name: Get details of all images
      docker_host_info:
        images: yes
        verbose_output: yes
      register: image_info
    - name: Remove all images
      docker_image:
        name: "{{ item }}"
        state: absent
      loop: "{{ image_info.images | map(attribute='Id') | list }}"
```

Get HelpExitWriteOutJustifyRead FileWhere isPrev PgNext PgCut TextUnCut TextCur PosTo Spell

Step 1.4 Create environment files

These files include properties associated with environments. In my case I have `dev1` and `dev2` in inventory. So I will have two files

```
1    hello1:
2      port: 8082
3    hello2:
4      port: 8083
```

```
1    hello1:
2      port: 8082
3    hello2:
4      port: 8083
```

Create a parameterized pipeline to deploy `Hello Service`

Previously we loaded everything from environment block, but we cant always come to pipeline script and change, so we use this parameterized build trigger approach which gives us more controllability over pipe.

```
Code Blame 58 lines (45 loc) · 1.46 KB
1 pipeline {
2   agent any
3
4   tools{
5     maven 'maven'
6   }
7
8   environment{
9     BRANCH = 'main'
10    TAG = 'dev'
11    REGISTRY = '<ip>:5000'
12    DEPLOY_TO = 'dev2'
13  }
14
15  stages {
16    stage('Clone') {
17      steps {
18        git branch: "${BRANCH}" , credentialsId: 'bitbucket', url: 'https://isurunuwanthilaka@bitbucket.org/isurunuwanthilaka/hello
19
20      }
21    }
22
23    stage('Maven Build') {
24      steps {
25        sh "mvn clean package"
26
27      }
28    }
29
30    stage('Docker Build') {
31      steps {
32        sh "docker build -t ${REGISTRY}/hello:${TAG} ./"
33
34    }
```

Create a generic service deployment pipeline

In previous scenarios everything bound with one service **Hello service**, now we want

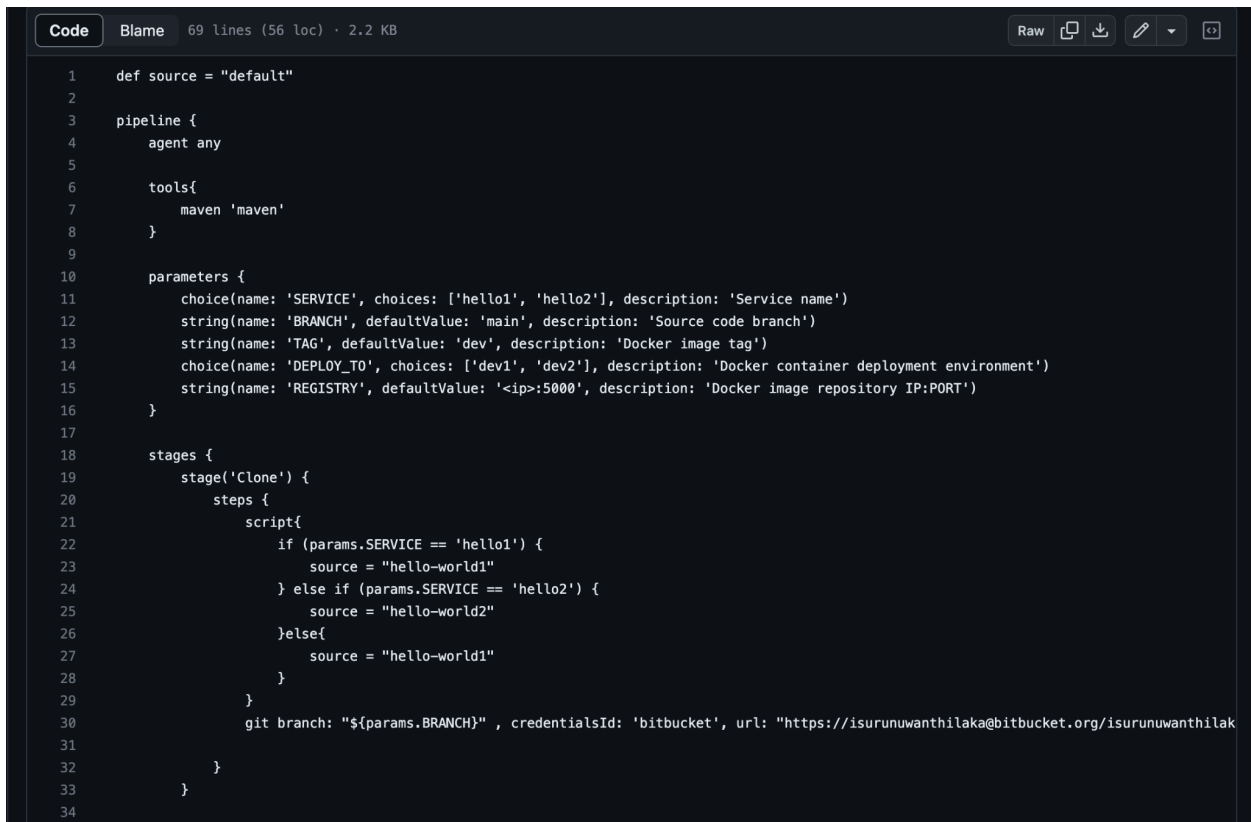
```
Code Blame 16 lines (13 loc) · 512 Bytes
1 pipeline {
2   agent any
3
4   parameters {
5     choice(name: 'ENV', choices: ['dev1', 'dev2'], description: 'Environment to be cleaned')
6   }
7
8   stages {
9
10    stage('Clean') {
11      steps {
12        ansiblePlaybook credentialsId: 'dev-server', disableHostKeyChecking: true, extras: "-e ENV=${params.ENV}", installation: 'a
13      }
14    }
15  }
16 }
```


to create a generic docker container deployment. We only need to create a new pipeline , other files are same.

Step 3.1 generic deployment pipeline

Create a generic service build pipeline

Step 4.1 generic build pipeline



```
Code Blame 69 lines (56 loc) · 2.2 KB
1 def source = "default"
2
3 pipeline {
4     agent any
5
6     tools{
7         maven 'maven'
8     }
9
10    parameters {
11        choice(name: 'SERVICE', choices: ['hello1', 'hello2'], description: 'Service name')
12        string(name: 'BRANCH', defaultValue: 'main', description: 'Source code branch')
13        string(name: 'TAG', defaultValue: 'dev', description: 'Docker image tag')
14        choice(name: 'DEPLOY_TO', choices: ['dev1', 'dev2'], description: 'Docker container deployment environment')
15        string(name: 'REGISTRY', defaultValue: '<ip>:5000', description: 'Docker image repository IP:PORT')
16    }
17
18    stages {
19        stage('Clone') {
20            steps {
21                script{
22                    if (params.SERVICE == 'hello1') {
23                        source = "hello-world1"
24                    } else if (params.SERVICE == 'hello2') {
25                        source = "hello-world2"
26                    }else{
27                        source = "hello-world1"
28                    }
29                }
30                git branch: "${params.BRANCH}" , credentialsId: 'bitbucket', url: "https://isurunuwanthilaka@bitbucket.org/isurunuwanthilak
31            }
32        }
33    }
34}
```

Create a parameterized pipeline to clean environments

Finally we need to clean environment, so I have created another parameterized pipeline as follows.

Step 5.1 Environment cleaning pipeline

```
pipeline {
    agent any
```

```

    parameters {
        choice(name: 'ENV', choices: ['dev1', 'dev2'], description:
'Environment to be cleaned')
    }

    stages {

        stage('Clean') {
            steps {
                ansiblePlaybook credentialsId: 'dev-server',
disableHostKeyChecking: true, extras: "-e ENV=${params.ENV}",
installation: 'ansible', inventory:
'/home/src/ansible-scripts/inventory.inv', playbook:
'/home/src/ansible-scripts/docker-clean-env.yml'
            }
        }
    }
}

```

Step 5.2 Environment cleaning ansible-playbook

`docker-clean-env.yml`

- hosts: "{{ENV}}"

gather_facts: false

tasks:

- name: Get running containers

docker_host_info:

containers: yes

register: docker_info

```
- name: Stop running containers
  docker_container:
    name: "{{ item }}"
    state: stopped
  loop: "{{ docker_info.containers | map(attribute='Id') | list }}"

- name: Remove Stopped docker containers
  shell: |
    docker rm $(docker ps -a -q);
  when: docker_info.containers != 0

- name: Get details of all images
  docker_host_info:
    images: yes
    verbose_output: yes
  register: image_info

- name: Remove all images
  docker_image:
    name: "{{ item }}"
    state: absent
  loop: "{{ image_info.images | map(attribute='Id') | list }}"
```

CONCLUSION

In conclusion, this report provided an overview of the Docker container deployment process using Jenkins and Ansible. The project demonstrated the ease of deployment of applications in a containerized environment using these tools.

In future work, we could explore the use of Kubernetes for container orchestration and scaling. Kubernetes is an open-source container orchestration platform that is widely used in production environments. It provides features such as automatic scaling, rolling updates, and self-healing, which can improve the reliability and availability of applications.