Implementation 2

# Project Document: CI/CD Pipeline for Python Application

## Overview:

This document outlines the setup and configuration of a Continuous Integration (CI) and Continuous Deployment (CD) pipeline for a sample application using Azure DevOps, Terraform, Ansible, Docker, Azure Container Registry (ACR) and Azure Kubernetes Service (AKS). The application consists of two components: a Python frontend and a Redis cache serving as the database.

The CI/CD pipeline includes code quality checks, image/package builds, JFROG integration, testing, and deployment to Azure Kubernetes Service (AKS).

# PRE-REQUISITES

1. An Azure subscription
2. An Azure DevOps account and a project
3. There are a few extensions that are required in Azure DevOps pipelines to streamline and automate various aspects of the development process.
   * **Terraform** – It enables integration of Terraform into our pipelines, automating the provisioning and management of infrastructure resources on platforms like Azure, AWS and GCP.
   * **Docker build task** – This extension allows us to build Docker images within your Azure DevOps pipelines. We can use it to package our applications and their dependencies into lightweight, portable containers that can run consistently across different environments.
   * **JFrog** - JFrog Artifactory is a universal artifact repository manager that allows us to store and manage our software packages and dependencies. The JFrog extension in Azure DevOps enables seamless integration with JFrog Artifactory, allowing us to publish and retrieve artifacts from repositories as part of your CI/CD pipelines.
   * **SonarCloud** - SonarCloud is a cloud-based code quality and security analysis tool. The SonarCloud extension in Azure DevOps allows us to integrate static code analysis into our pipelines, providing insights into code quality, identifying bugs, vulnerabilities, and code smells early in the development process.

A screenshot of a computer

Description automatically generated

1. The VM must have few dependencies installed to run our application. So, we are using ansible-playbook in our project to install them. The dependencies required by our Virtual Machine are:
   * Docker

* Azure-cli
* Kubectl
* docker-compose.

1. A resource group, a storage account with a container – required by terraform stages.
2. A container registry – required to create service connection.
3. An Azure DevOps Docker Registry service connection with Azure Container Registry – For Image upload Registry
4. An Azure DevOps service connection with Azure Kubernetes Service – For deployment

### Repositories:

Application Code:

<https://dev.azure.com/Shubham1708698304552/Implementation%202/_git/Implementation%202%20-%20Code>

Terraform Infrastructure Code:

<https://dev.azure.com/Shubham1708698304552/Implementation%202/_git/Implementation%202>

## Infrastructure Provisioning with Terraform:

### Infra CI Pipeline:

Infrastructure CI Pipeline to setup infrastructure for different environments (Dev/Prod)

#### Steps:

* Terraform Init
* Terraform Plan
* Terraform Validate
* Terraform Apply
* Docker Build

#### Azure Resources:

1. Azure Kubernetes Cluster (AKS)
2. Azure Container Registry (ACR)
3. Virtual Machine with Ansible installed serves as a self-hosted Agent.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

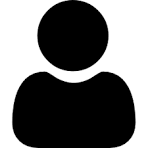
Description automatically generated

## Architecture Diagram:



**Service Connection** grants trust between Azure DevOps and Azure

A black circle with a white letter in it

Description automatically generated

Build and push docker images to docker hub or Azure Container Registry

docker build & docker push

Create a “**Dockerfile**”

Configure Infrastructure using   
**ansible-playbook**

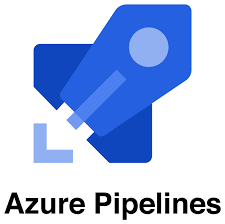
VMs, networks, storage accounts

Code Push

User

Code in   
local git repo

Code commit



Create a   
yaml pipeline.

Initialize Terraform

Validate Terraform Plan

Apply   
Terraform Plan



Storage Account

Azure Resources

Initialize backend.

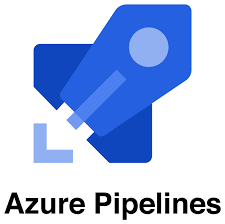
Create resources



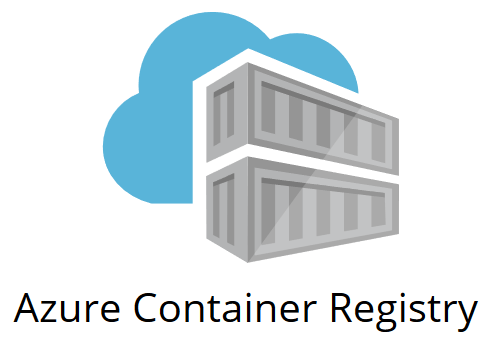
Prepare analysis on sonarcloud

Run code analysis

Publish Quality Gate results.



Include dependencies, configuration, and commands to run the application



AKS

### CI Pipeline:

#### Steps:

##### SonarQube Code Quality Check:

* Perform static code analysis to ensure code quality.

##### Build & Push Image & Package:

* Build Docker image for the application.
* Package the application with dependencies.
* Push both the Docker image with packaged application to Azure Container Registry (ACR).

##### JFROG Integration:

* Push images and packages to JFROG Artifactory.
* Scan images and packages for vulnerabilities.

##### Deployment to Agent using Docker-Compose:

* Deploy containers on a dedicated agent using Docker-Compose.
* Ensure successful deployment before proceeding to testing.

##### Testing:

* Conduct unit testing and integration testing.
* Proceed to the next step only if all tests passes.

### CD Pipeline:

#### Steps:

##### Download Image or Package:

* Download the Docker image or packaged application from JFROG Artifactory.

##### Deployment to AKS:

* Deploy the application to Azure Kubernetes Service (AKS).
* Configure environment-specific settings for dev and prod.

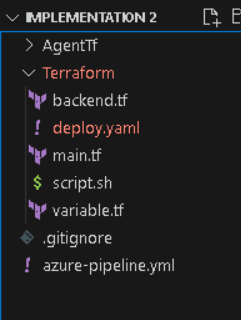
##### Final App Testing:

* Execute final testing on the deployed application in the AKS environment.

##### Infrastructure Provisioning with Terraform (Dev and Prod):

* Create separate infrastructure environments for development and production.
* Provision AKS, ACR, and a VM with Ansible installed using Terraform.

## Infrastructure Provision



Main.tf

provider "azurerm" {

features {}

}

# Resource Group

resource "azurerm\_resource\_group" "rgdev" {

name = var.rg.name

location = var.rg.location

}

# Azure Kubernetes Service (AKS)

resource "azurerm\_kubernetes\_cluster" "aksdev" {

name = "aksDemo01"

location = azurerm\_resource\_group.rgdev.location

resource\_group\_name = azurerm\_resource\_group.rgdev.name

dns\_prefix = "k8sdns"

kubernetes\_version = "1.27.7"

default\_node\_pool {

name = "default"

node\_count = 2

vm\_size = "Standard\_D2s\_v3"

os\_disk\_size\_gb = 30

}

service\_principal {

client\_id = "b3e7b1ea-6299-46b5-a755-b35935c2e50c"

client\_secret = "GouysOI~rf3DAiaBSC0foejZbfU0\_7RNNf"

}

role\_based\_access\_control\_enabled = true

tags = {

environment = "Demo"

}

}

# Azure Container Registry (ACR)

resource "azurerm\_container\_registry" "acrdev" {

name = "acrshubdemo01"

resource\_group\_name = azurerm\_resource\_group.rgdev.name

location = azurerm\_resource\_group.rgdev.location

sku = "Basic"

}

# Virtual Machine

# Create a virtual network within the resource group

resource "azurerm\_virtual\_network" "vnet01" {

name = var.vnet\_name

resource\_group\_name = azurerm\_resource\_group.rgdev.name

location = azurerm\_resource\_group.rgdev.location

address\_space = ["10.0.0.0/16"]

}

resource "azurerm\_public\_ip" "publicip01" {

name = "publiciptest01"

location = azurerm\_resource\_group.rgdev.location

resource\_group\_name = azurerm\_resource\_group.rgdev.name

allocation\_method = "Static"

}

resource "azurerm\_network\_interface" "nic01" {

name = "nic\_test\_01"

location = azurerm\_resource\_group.rgdev.location

resource\_group\_name = azurerm\_resource\_group.rgdev.name

ip\_configuration {

name = "internal"

subnet\_id = azurerm\_subnet.subnet01.id

private\_ip\_address\_allocation = "Dynamic"

public\_ip\_address\_id = azurerm\_public\_ip.publicip01.id

}

}

resource "azurerm\_subnet" "subnet01" {

name = var.subnet\_name

resource\_group\_name = azurerm\_resource\_group.rgdev.name

virtual\_network\_name = azurerm\_virtual\_network.vnet01.name

address\_prefixes = [var.subnet\_ip]

}

# ---------------------------------------------------------------------------

# This line is to follow company policy as boot diagnostics should be enabled

/\*Create a storage account to create blob storage for the boot diag output\*/

resource "azurerm\_storage\_account" "diagSA01" {

name = "bootdiagsa021220232"

resource\_group\_name = azurerm\_resource\_group.rgdev.name

location = azurerm\_resource\_group.rgdev.location

account\_tier = "${element(split("\_", var.boot\_diagnostics\_sa\_type),0)}"

account\_replication\_type = "${element(split("\_", var.boot\_diagnostics\_sa\_type),1)}"

}

# ---------------------------------------------------------------------------

resource "azurerm\_virtual\_machine" "vm01" {

name = "vm\_test\_01"

location = azurerm\_resource\_group.rgdev.location

resource\_group\_name = azurerm\_resource\_group.rgdev.name

network\_interface\_ids = [azurerm\_network\_interface.nic01.id]

vm\_size = "Standard\_DS1\_v2"

# ---------------------------------------------------------------------------

# This line is to follow company policy as boot diagnostics should be enabled

boot\_diagnostics {

enabled = "true"

storage\_uri = azurerm\_storage\_account.diagSA01.primary\_blob\_endpoint

}

# ---------------------------------------------------------------------------

# Uncomment this line to delete the OS disk automatically when deleting the VM

# delete\_os\_disk\_on\_termination = true

# Uncomment this line to delete the data disks automatically when deleting the VM

# delete\_data\_disks\_on\_termination = true

storage\_image\_reference {

publisher = "Canonical"

offer = "0001-com-ubuntu-server-jammy"

sku = "22\_04-lts"

version = "latest"

}

storage\_os\_disk {

name = "myosdisk1"

caching = "ReadWrite"

create\_option = "FromImage"

managed\_disk\_type = "Standard\_LRS"

}

os\_profile {

computer\_name = "hostname"

admin\_username = var.connection["username"]

admin\_password = var.connection["password"]

}

os\_profile\_linux\_config {

disable\_password\_authentication = false

}

}

resource "null\_resource" "copy\_ansible\_yaml" {

triggers = {

always\_run = timestamp()

}

provisioner "file" {

source = "deploy.yaml"

destination = "/tmp/deploy.yaml"

connection {

type = "ssh"

user = var.connection["username"]

password = var.connection["password"]

host = azurerm\_public\_ip.publicip01.ip\_address

}

}

depends\_on = [azurerm\_virtual\_machine.vm01]

}

resource "null\_resource" "copy\_script\_file" {

triggers = {

always\_run = timestamp()

}

provisioner "file" {

source = "script.sh"

destination = "/tmp/script.sh"

connection {

type = "ssh"

user = var.connection["username"]

password = var.connection["password"]

host = azurerm\_public\_ip.publicip01.ip\_address

}

}

depends\_on = [null\_resource.copy\_ansible\_yaml]

}

resource "null\_resource" "execute\_script" {

triggers = {

always\_run = timestamp()

}

provisioner "remote-exec" {

inline = [

"chmod +x /tmp/script.sh ",

"/tmp/script.sh"

]

connection {

type = "ssh"

user = var.connection["username"]

password = var.connection["password"]

host = azurerm\_public\_ip.publicip01.ip\_address

}

}

depends\_on = [null\_resource.copy\_script\_file]

}

Backend.tf

terraform {

backend "azurerm" {

resource\_group\_name = "rg\_agent"

storage\_account\_name = "storageaccountagentshub"

container\_name = "agent-container"

key = "terraform.tfstate"

}

}

Variable.tf

variable "env" {

type = string

default = "Production"

}

variable "rg" {

type = map

default = {

"name" = "rg\_prod"

"location" = "East US"

}

}

variable "vnet\_name" {

type = string

default = "vnet\_dev"

}

variable "subnet\_ip" {

type = string

default = "10.0.0.0/24"

}

variable "subnet\_name" {

type = string

default = "subnet\_dev"

}

variable "boot\_diagnostics\_sa\_type" {

default = "Standard\_LRS"

}

variable "connection" {

type = map

default = {

"username" = "testadmin"

"password" = "Password1234!"

}

}

Deploy.yaml

---

- name: Install Prerequisites and Configure Azure DevOps Agent

hosts: localhost

tasks:

- name: Update apt packages

become: true

apt:

update\_cache: yes

- name: Install Python and Python dependencies

become: true

apt:

name:

- python3

- python3-pip

- virtualenv

- python3-setuptools

state: present

- name: Upgrade pip

become: true

pip:

name: pip

executable: /usr/bin/python3

state: latest

- name: Install required packages

become: true

apt:

name:

- docker.io

- docker-compose

- apt-transport-https

- ca-certificates

- curl

- gnupg-agent

- software-properties-common

state: present

- name: Install Azure CLI

become: true

shell: |

curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash

args:

executable: /bin/bash

- name: Install kubectl

become: true

shell: |

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

args:

executable: /bin/bash

- name: Install Docker Compose

become: true

shell: |

sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

args:

executable: /bin/bash

Script.sh

#!/bin/bash

# Update package lists

sudo apt update

# Install necessary dependencies

sudo apt install -y software-properties-common

# Add Ansible repository

sudo apt-add-repository --yes --update ppa:ansible/ansible

# Install Ansible

sudo apt install -y ansible

# Display Ansible version

ansible --version

echo "Ansible has been successfully installed."

cd /tmp

ls

ansible-playbook -i localhost deploy.yaml --ssh-extra-args='-o StrictHostKeyChecking=no'

echo "Ansible Playbook Executed Successully !!!"

echo "Azure self hosted agent installation started."

mkdir myagent && cd myagent

wget -O vsts-agent-linux-x64-3.234.0.tar.gz https://vstsagentpackage.azureedge.net/agent/3.234.0/vsts-agent-linux-x64-3.234.0.tar.gz

tar zxvf vsts-agent-linux-x64-3.234.0.tar.gz

./config.sh --unattended --url https://dev.azure.com/Shubham1708698304552/ --auth pat --token x4x4wzwy7w53ikj54ipzhbnmjjnfohcewotpag7zx27ugkgusqmq --pool TestAgentPool --agent LinuxAgent02 --acceptTeeEula --replace

echo "Azure self hosted agent installation successful."

Azure-pipeline.yaml

trigger:

branches:

include:

- main

pool:

name: 'Default'

steps:

- task: UseDotNet@2

inputs:

packageType: 'sdk'

version: '3.x'

installationPath: $(Agent.ToolsDirectory)/dotnet

- script: |

sudo apt update -y

sudo apt install unzip -y

displayName: 'Command Line Script'

- task: ms-devlabs.custom-terraform-tasks.custom-terraform-installer-task.TerraformInstaller@1

displayName: 'Install Terraform latest'

- task: ms-devlabs.custom-terraform-tasks.custom-terraform-release-task.TerraformTaskV4@4

displayName: 'Terraform : init'

inputs:

workingDirectory: Terraform

backendServiceArm: 'npstackro-1676009261708 (14f56a24-f129-441e-a95b-0df01d75c3a7)'

backendAzureRmResourceGroupName: 'rg\_agent'

backendAzureRmStorageAccountName: storageaccountagentshub

backendAzureRmContainerName: 'agent-container'

backendAzureRmKey: terraform.tfstate

- task: ms-devlabs.custom-terraform-tasks.custom-terraform-release-task.TerraformTaskV4@4

displayName: 'Terraform : plan'

inputs:

command: plan

workingDirectory: Terraform

environmentServiceNameAzureRM: 'npstackro-1676009261708 (14f56a24-f129-441e-a95b-0df01d75c3a7)'

backendServiceArm: 'npstackro-1676009261708 (14f56a24-f129-441e-a95b-0df01d75c3a7)'

backendAzureRmResourceGroupName: 'rg\_agent'

backendAzureRmStorageAccountName: storageaccountagentshub

backendAzureRmContainerName: 'agent-container'

backendAzureRmKey: terraform.tfstate

- task: ms-devlabs.custom-terraform-tasks.custom-terraform-release-task.TerraformTaskV4@4

displayName: 'Terraform : validate'

inputs:

command: validate

workingDirectory: Terraform

backendServiceArm: 'npstackro-1676009261708 (14f56a24-f129-441e-a95b-0df01d75c3a7)'

backendAzureRmResourceGroupName: 'rg\_agent'

backendAzureRmStorageAccountName: storageaccountagentshub

backendAzureRmContainerName: 'agent-container'

backendAzureRmKey: terraform.tfstate

- task: ms-devlabs.custom-terraform-tasks.custom-terraform-release-task.TerraformTaskV4@4

displayName: 'Terraform : apply'

inputs:

command: apply

workingDirectory: Terraform

environmentServiceNameAzureRM: 'npstackro-1676009261708 (14f56a24-f129-441e-a95b-0df01d75c3a7)'

backendServiceArm: 'npstackro-1676009261708 (14f56a24-f129-441e-a95b-0df01d75c3a7)'

backendAzureRmResourceGroupName: 'rg\_agent'

backendAzureRmStorageAccountName: storageaccountagentshub

backendAzureRmContainerName: 'agent-container'

backendAzureRmKey: terraform.tfstate

## Python Code Project

App.py

from flask import Flask

from redis import Redis

app = Flask(\_\_name\_\_)

redis = Redis(host='redis', port=6379)

@app.route('/')

def hello():

count = redis.incr('hits')

return 'Hello World! I have been seen {} Ɵmes.\n'.format(count)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host="0.0.0.0", port=8000, debug=True)

Dockerfile

FROM python:3.12-alpine

ADD requirements.txt /code/requirements.txt

ADD app.py /code/app.py

WORKDIR /code

EXPOSE 80

RUN pip install -r requirements.txt

CMD ["python", "app.py"]

Requirement.txt

flask

redis

unit\_test.py

from ..code import app

from flask.testing import FlaskClient

def test\_hello():

client = app.test\_client()

response = client.get('/')

assert response.status\_code == 200

assert b'Test Case Passed !!' in response.data

azure-pipeline.yaml

name: Build and Deploy Python App

trigger:

- main

parameters:

- name: poolname

type: string

default: 'TestAgentPool'

resources:

- repo: self

variables:

imageRepo: 'pythonApp'

tag: 'v1'

stages:

- stage: Build\_And\_Test\_App

displayName: 'BuilD and Test Python App'

jobs:

- job: RunUnitTests

displayName: 'BuilD and Test'

pool:

name: ${{ parameters.poolname }}

steps:

- script: 'python3 -m pip install --upgrade pip && pip install -r requirements.txt'

displayName: 'Install dependencies'

- script: 'pip install pytest && pytest tests --doctest-modules --junitxml=junit/test-results.xml'

displayName: 'Unit Test'

- task: PublishTestResults@2

displayName: 'Publish Test Results \*\*/test-results.xml'

inputs:

testResultsFiles: '\*\*/test-results.xml'

testRunTitle: 'Python App Results'

- stage: Build\_And\_Push\_Image

jobs:

- job: Build\_Image

displayName: Build\_Image

pool:

name: ${{ parameters.poolname }}

steps:

- task: Docker@2

inputs:

containerRegistry: 'svc\_acr\_cred'

repository: '$(imageRepo)'

command: 'buildAndPush'

Dockerfile: '$(Build.SourcesDirectory)/code/Dockerfile'

tags: |

$(Build.BuildId)

$(tag)

- task: PublishBuildArtifacts@1

displayName: 'Publish Artifact: drop'

inputs:

PathtoPublish: code

- stage: Upload\_Artifacts

dependsOn: Build\_And\_Push\_Image

jobs:

- job: UploadArtifacts

displayName: Upload Artifacts to JFrog

pool:

name: ${{ parameters.poolname }}

steps:

- script: |

# Install JFrog CLI

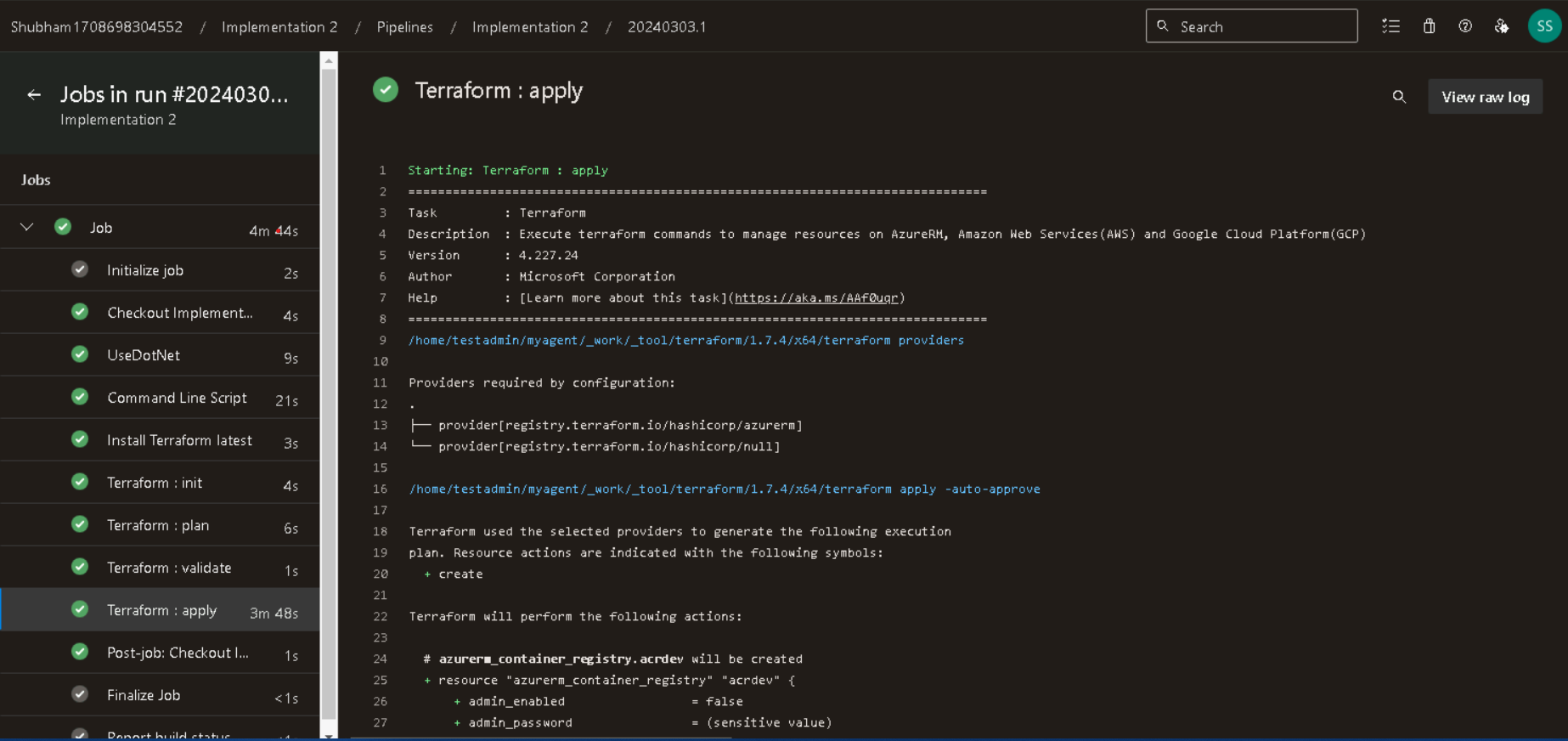
curl -fL https://getcli.jfrog.io | sh

sudo cp jfrog /usr/local//bin

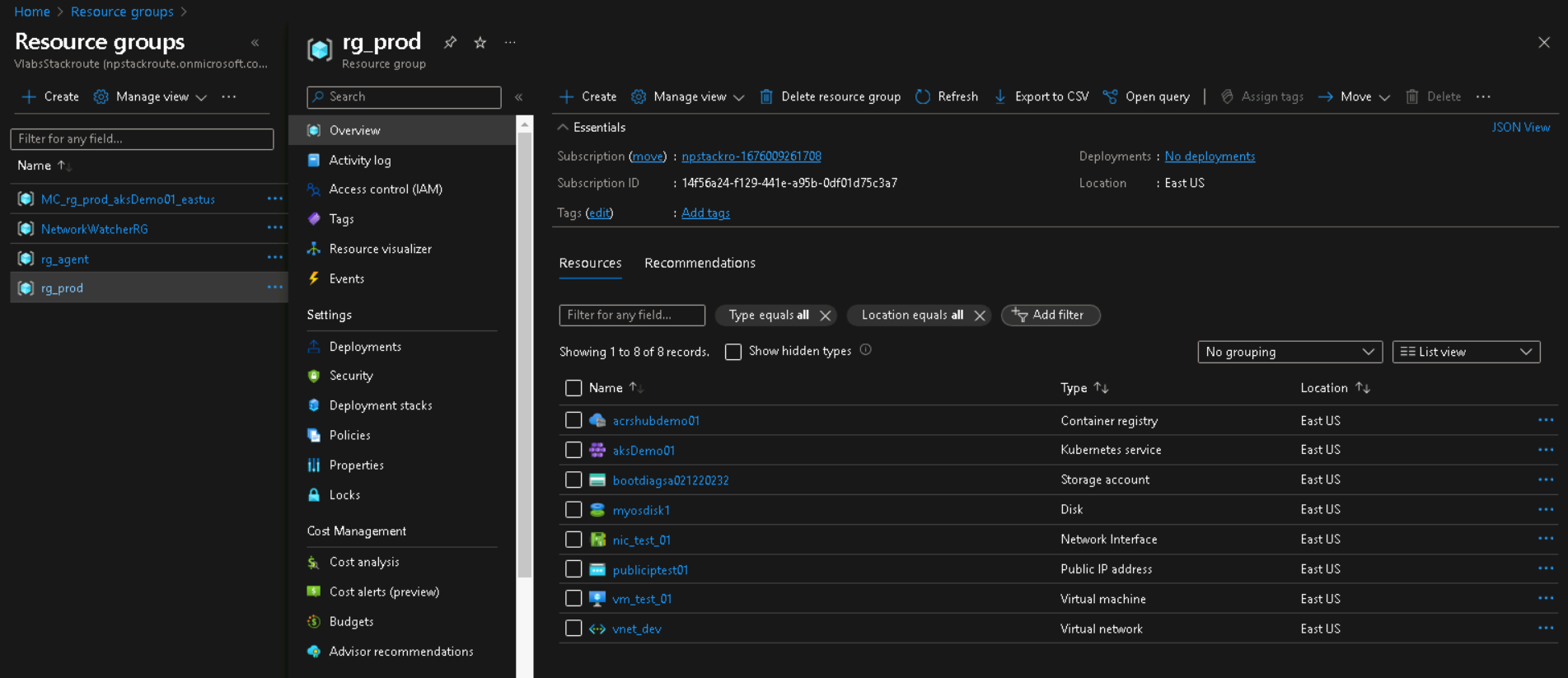
displayName: 'Install JFrog CLI'

## Devops

Terraform Azure Pipeline



Results



### Conclusion:

This document provides an overview of the CI/CD pipeline architecture, Git repository details, infrastructure setup, and the branching strategy adopted for the sample application.