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).

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:

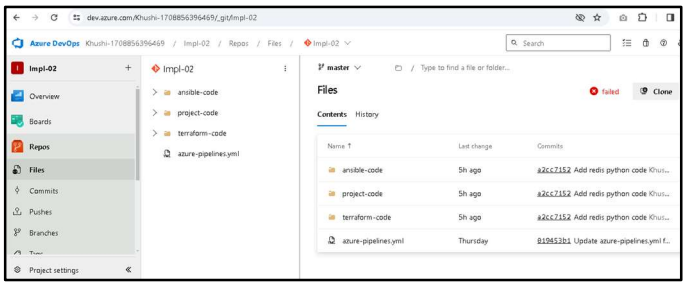
Infrastructure CI Pipeline to setup infrastructure for different environments (Dev/Prod)

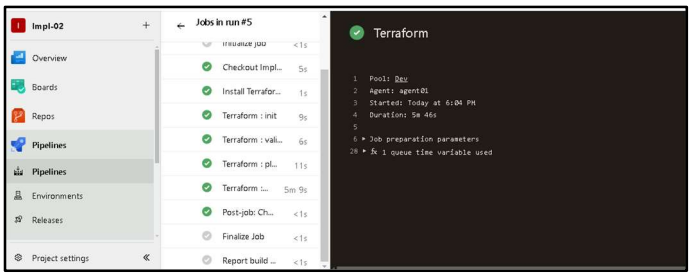
Steps:

- Terraform Init
- Terraform Plan
- Terraform Validate
- Terraform Apply
- Docker Build

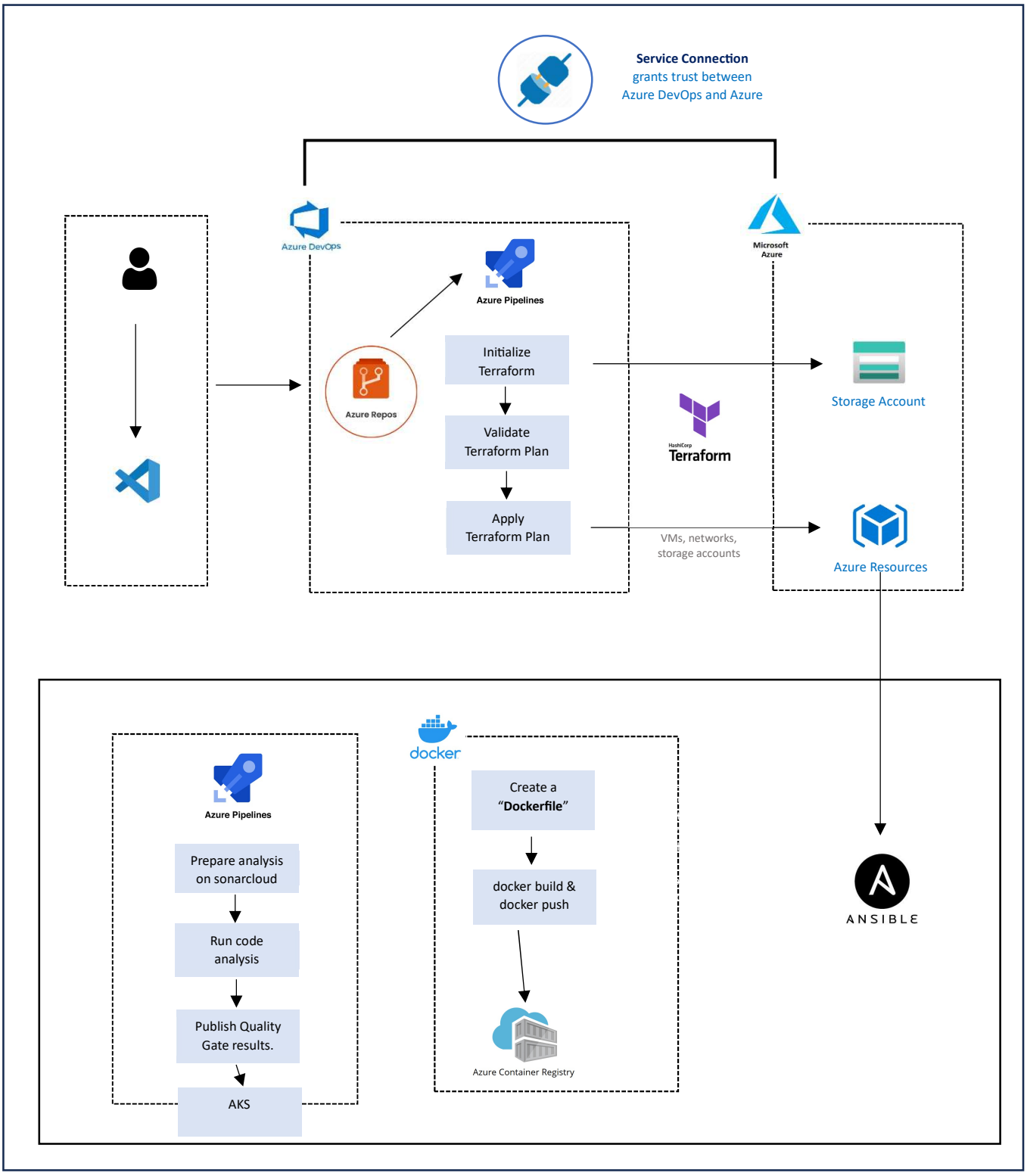
Azure Resources:

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





Architecture Diagram:



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.

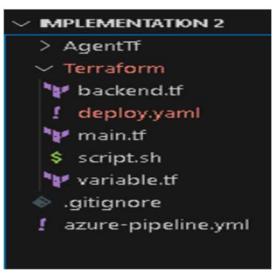
Final App Testing:

- 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 {} 0mes.\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
```

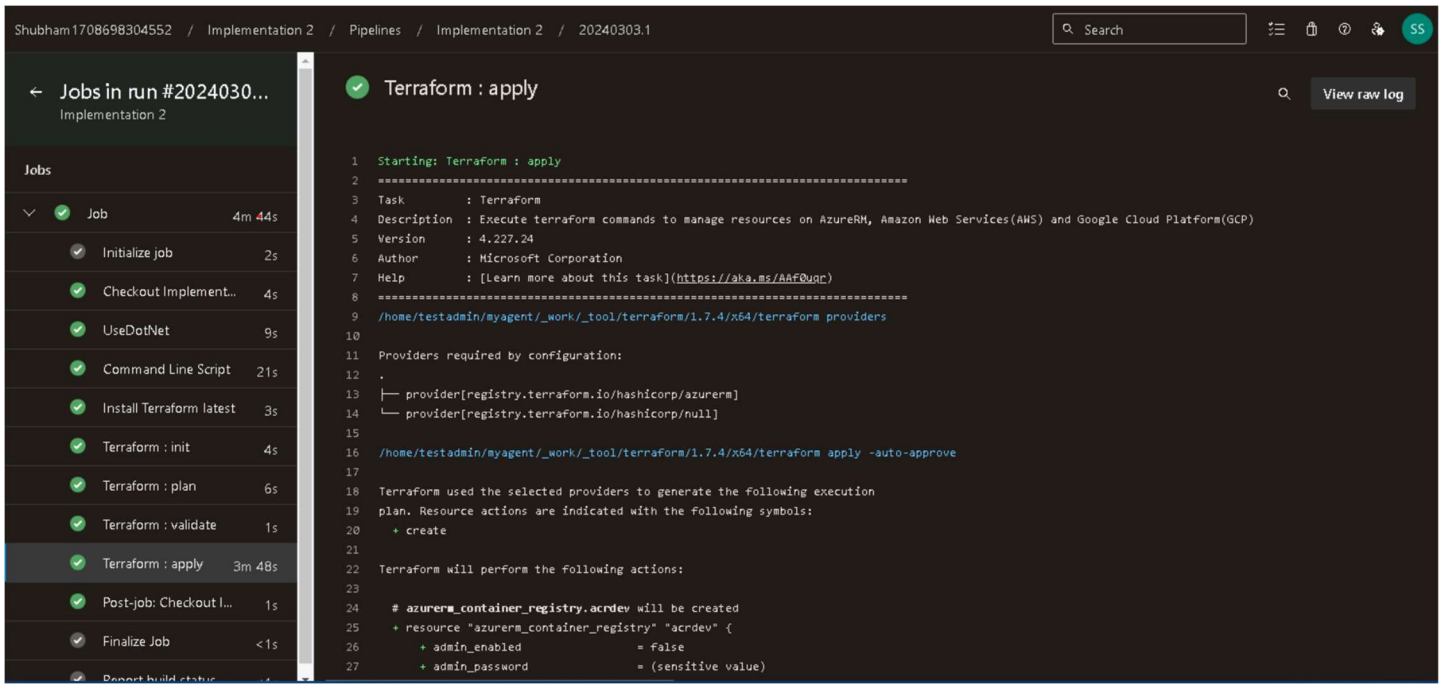
azure-pipeline.yaml

```
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)'
```

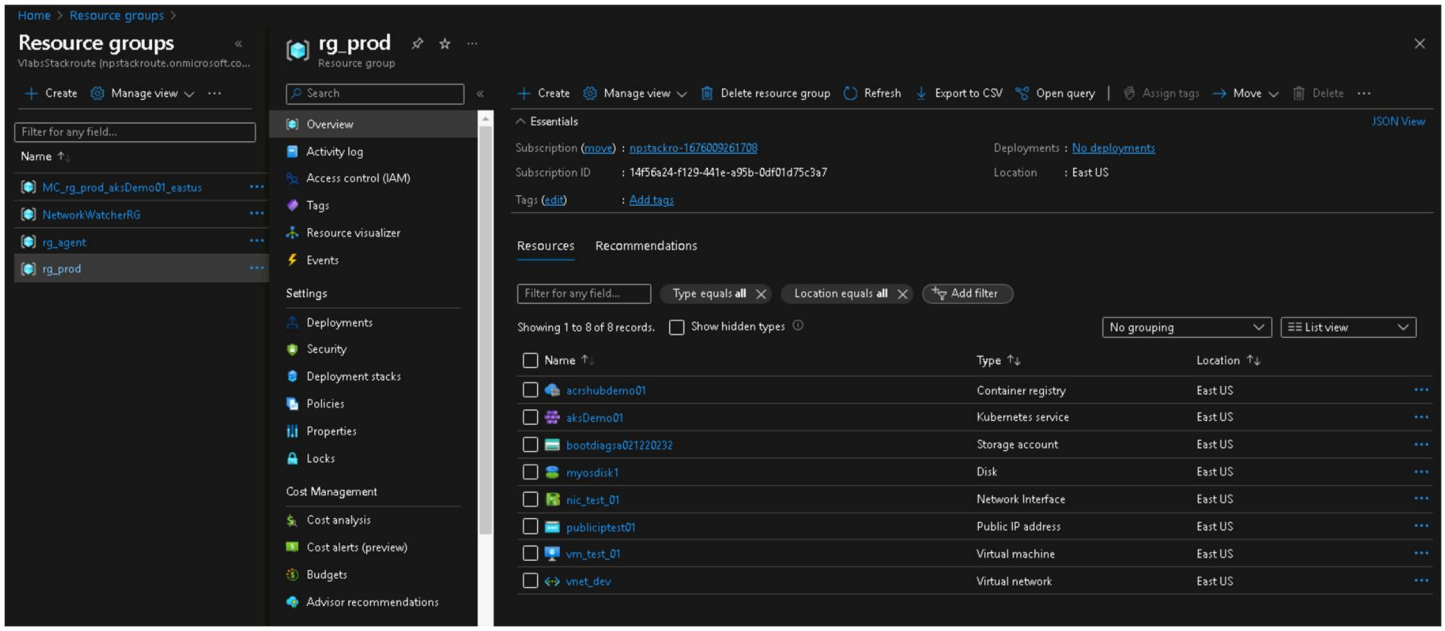
```
        command: 'buildAndPush'
        Dockerfile: '$(Build.SourcesDirectory)/code/Dockerfile'
        tags: |
          $(Build.BuildId)
          $(tag)
    - task: PublishBuildArtifacts@1
      displayName: 'Publish Artifact: drop'
      inputs:
        PathtoPublish: code

- 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.