Implementation2

CICD Pipeline for Python Application

- BY SHUBHAM SHARMA

Outline

Introduction

Comprehensive end-to-end configuration and setup of a Continuous Integration (CI) and Continuous Deployment (CD) pipeline.

Sample application which consists of two components: a Python frontend and a Redis cache serving as the database.

Tech Stack: Utilizing Azure DevOps, Terraform, Ansible, Docker, ACR, and AKS

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

Automated deployment to Azure Kubernetes Service (AKS)

Why CICD?

Consistency Across Environments

Automated testing

Infrastructure as Code (IaC) with Terraform

Centralized repository with Azure repo

Efficient Docker Containerization with ACR

Artifact Management with Azure Artifacts

Scalability with AKS

Pre-Requisites

An Azure subscription

An Azure DevOps account and a project

Devops Extensions

- Terraform
- Docker
- Jfrog
- SonarCloud

Pre-Requisites

Agent Dependencies

- Docker
- Azure-cli
- Kubectl
- docker-compose.

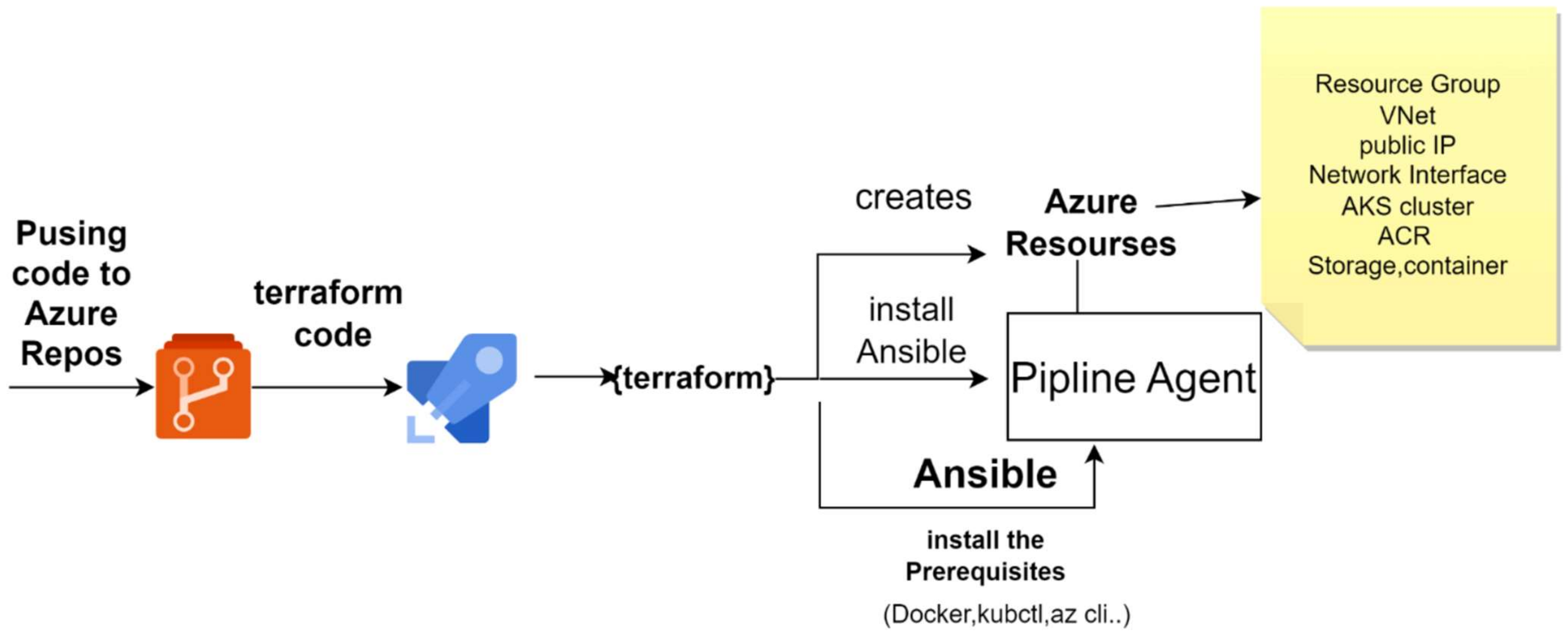
A container registry – required to create service connection.

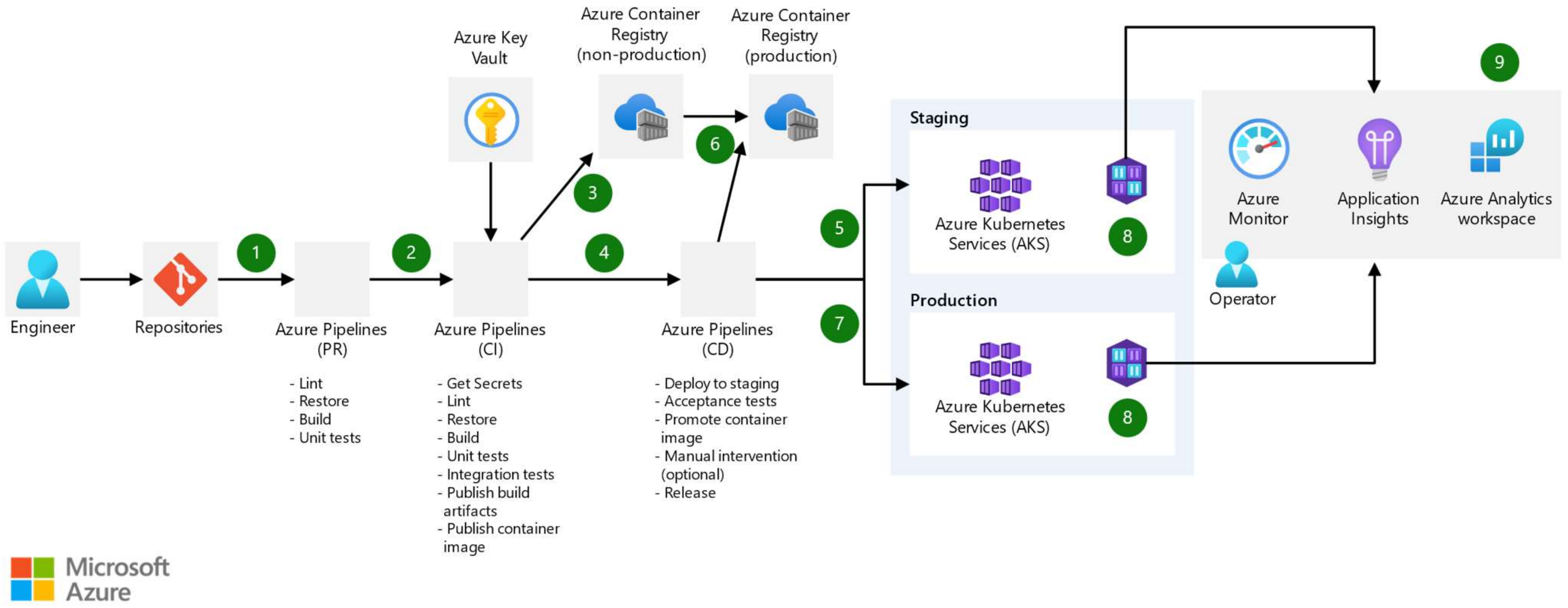
An Azure DevOps Docker Registry service connection with Azure Container Registry – For Image upload Registry

An Azure DevOps service connection with Azure Kubernetes Service – For deployment

Project Architecture

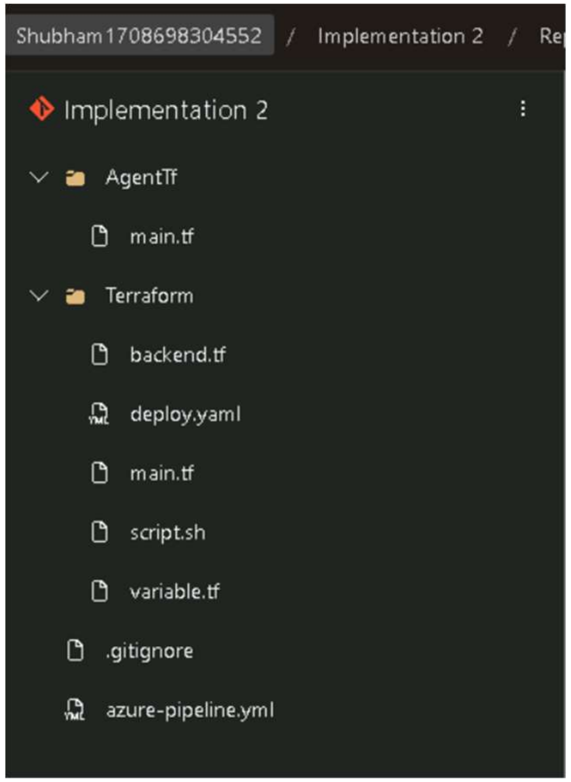
Terraform Pipeline - Agent Setup





Azure CICD Pipeline

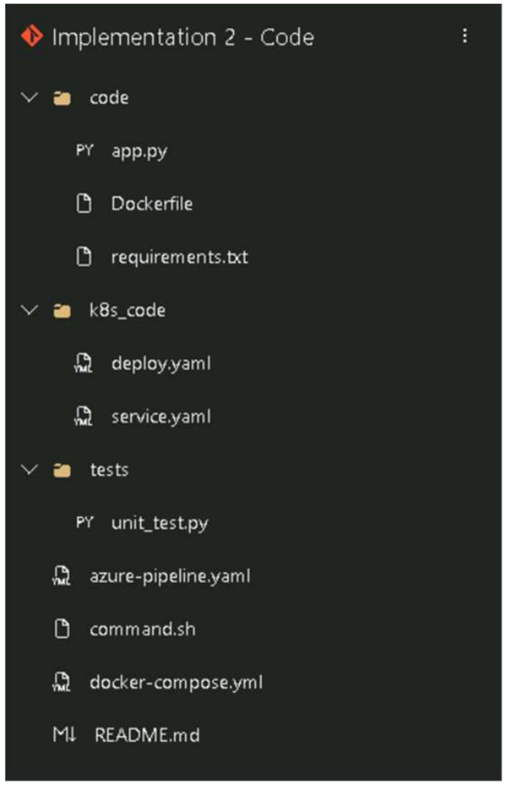
Project Code



Shubham1708698304552 / Implementation 2 / Re

Implementation 2

```
AgentTf
    main.tf
Terraform
    backend.tf
    deploy.yaml
    main.tf
    script.sh
    variable.tf
.gitignore
azure-pipeline.yml
```



Implementation 2 - Code

```
code
    app.py
    Dockerfile
    requirements.txt
k8s_code
    deploy.yaml
    service.yaml
tests
    unit_test.py
azure-pipeline.yaml
command.sh
docker-compose.yml
README.md
```

Thank you