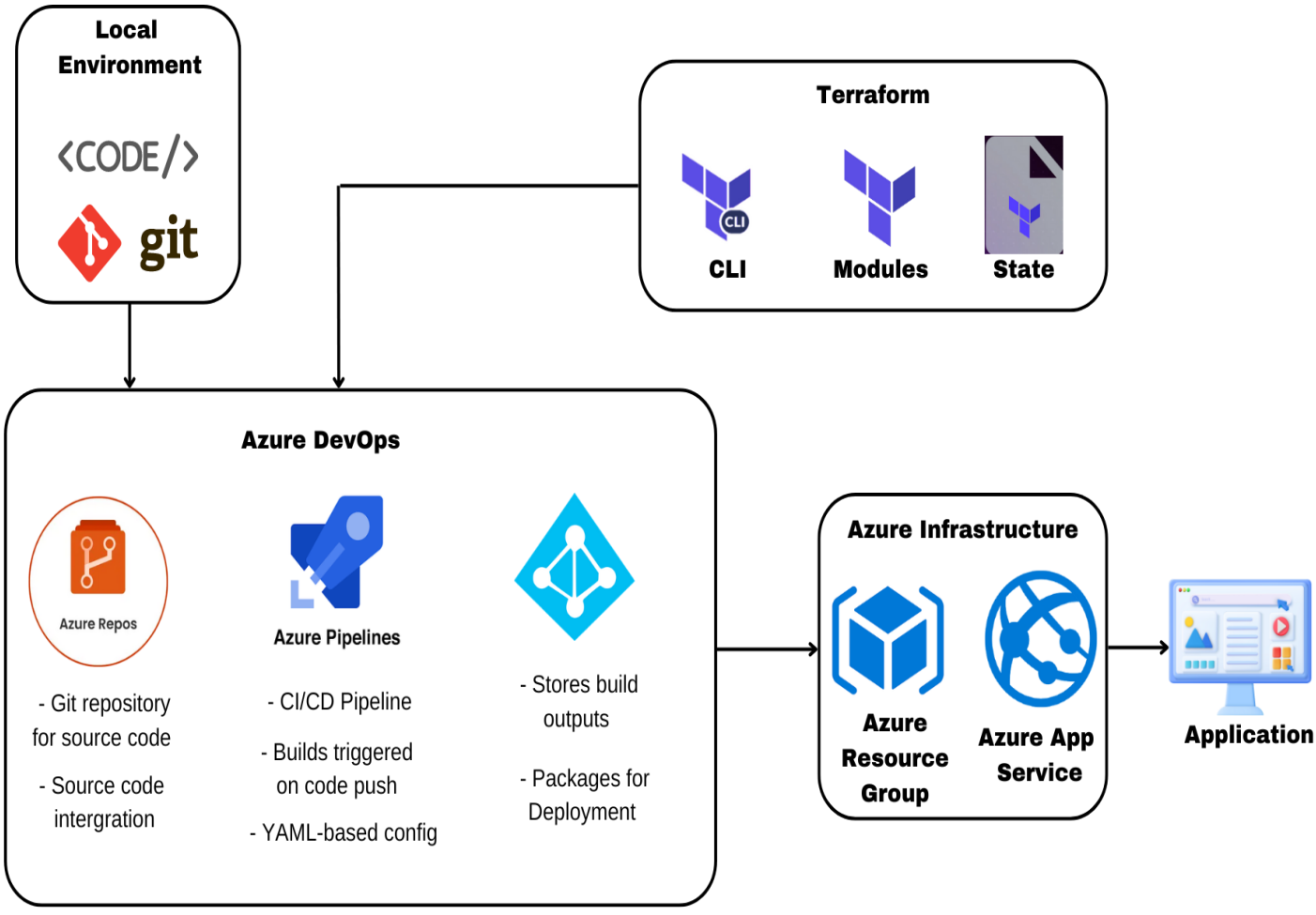
DevOps Shack

CI/CD Pipeline with Terraform on Azure

This project focuses on automating the setup of a complete CI/CD pipeline on Azure using Terraform, a powerful infrastructure as code (IaC) tool that allows for the streamlined management of cloud resources.

Flow Description:



Azure Repos:

- Stores the source code in a Git repository.
- Developer pushes code changes to Azure Repos.

Azure Pipelines:

- A CI/CD pipeline is configured to automatically trigger on code push events.
- Pipeline builds the application and performs testing.
- Artifacts from the build are stored in Azure Artifacts.

Azure Artifacts:

- Stores the build outputs, ready for deployment.

Terraform CLI:

- Manages the provisioning and configuration of Azure resources.
- Applies the Terraform configuration files to set up infrastructure like Resource Groups and App Service.

Terraform Modules:

- Contains reusable code templates for creating Azure resources, simplifying the Terraform configuration.

Terraform State:

- Maintains the current state of the infrastructure to manage updates and changes.

Azure Resource Group:

- Logical container that organizes the Azure resources such as the App Service.

Azure App Service:

- Hosts the deployed web application, configured and managed by Terraform.

Application Deployment:

- The web application is deployed to Azure App Service as part of the CI/CD pipeline.

Steps to Setup the Project.

Step 1: Set Up Terraform

- **Install Terraform**: Ensure that Terraform is installed on your local machine.
- **Azure CLI**: Install the Azure CLI and log in to your Azure account.

Step 2: Create a Service Principal

Create a Service Principal in Azure to enable Terraform to manage resources.

```
az ad sp create-for-rbac --name terraform-sp --role Contributor \
  --scopes /subscriptions/YOUR_SUBSCRIPTION_ID \
  --sdk-auth
```

Save the output JSON for later use.

Step 3: Set Up the Terraform Configuration

Create a directory for your Terraform configuration and add the following files:

3.1 Provider Configuration (**provider.tf**)

```
provider "azurerm" {
 features = {}
}

provider "azuredevops" {
```

```
  org_service_url = "https://dev.azure.com/YOUR_ORG"
  personal_access_token = "YOUR_PAT"
}
```

3.2 Variables Configuration (**variables.tf**)

```
variable "resource_group_name" {
  type    = string
  default = "terraform-rg"
}

variable "location" {
  type    = string
  default = "East US"
}

variable "app_service_name" {
  type    = string
  default = "terraform-appservice"
}
```

3.3 Resource Group Configuration (**resource_group.tf**)

```
resource "azurerm_resource_group" "rg" {
  name     = var.resource_group_name
  location = var.location
}
```

3.4 App Service Configuration (**app_service.tf**)

```
resource "azurerm_app_service_plan" "app_service_plan" {
 name                = "app-service-plan"
 location            = azurerm_resource_group.rg.location
 resource_group_name = azurerm_resource_group.rg.name
 sku {
  tier = "Free"
  size = "F1"
 }
}

resource "azurerm_app_service" "app_service" {
 name                = var.app_service_name
 location            = azurerm_resource_group.rg.location
 resource_group_name = azurerm_resource_group.rg.name
 app_service_plan_id = azurerm_app_service_plan.app_service_plan.id
}
```

Step 4: Azure DevOps Resources

Create the Azure DevOps resources like Repos, Pipelines, and Artifacts.

4.1 Azure Repos Configuration (**azure_repos.tf**)

```
resource "azuredevops_project" "project" {
 name               = "Terraform-CICD"
```

```
 visibility       = "private"
 version_control   = "Git"
}

resource "azuredevops_git_repository" "repo" {
 project_id = azuredevops_project.project.id
 name      = "terraform-cicd-repo"
 initialization {
  init_type = "Clean"
 }
}
```

4.2 Azure Pipelines Configuration (**azure_pipelines.tf**)

```
resource "azuredevops_build_definition" "pipeline" {
 project_id = azuredevops_project.project.id
 name      = "terraform-cicd-pipeline"
 repository {
  repo_id   = azuredevops_git_repository.repo.id
  repo_type = "TfsGit"
  branch_name = "main"
 }

 ci_trigger {
  use_yaml = true
 }
```

```
  yaml {
    path = "azure-pipelines.yml"
  }
}
```

Step 5: CI/CD Pipeline Definition

Create the azure-pipelines.yml file for your CI/CD pipeline.

```
trigger:
- main

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: UseDotNet@2
  inputs:
    packageType: 'sdk'
    version: '5.x'
    installationPath: $(Agent.ToolsDirectory)/dotnet

- script: |
    dotnet build
  displayName: 'Build'

- script: |
```

```
  dotnet publish -c Release -o $(Build.ArtifactStagingDirectory)
 displayName: 'Publish'

- task: PublishBuildArtifacts@1
 inputs:
  PathtoPublish: '$(Build.ArtifactStagingDirectory)'
  ArtifactName: 'drop'
  publishLocation: 'Container'
```

Step 6: Deploying the Application

Deploy the application to Azure App Service using the following command in the pipeline.

```
- task: AzureWebApp@1
 inputs:
  azureSubscription: 'AzureSPN'
  appName: $(appServiceName)
  package: '$(System.DefaultWorkingDirectory)/drop'
```

Step 7: Initialize and Apply Terraform Configuration

- Initialize Terraform: Run the following command to initialize Terraform.

```
terraform init
```

- Apply the Terraform Configuration: Apply the configuration to provision resources.

```
terraform apply
```

Step 9: Test the Pipeline

Push changes to your Git repository and observe the automated deployment process in Azure Pipelines. The application should be deployed to Azure App Service.

Learning Outcome

By completing this project, you'll understand how to automate the setup of a CI/CD pipeline using Terraform on Azure, including the provisioning of Azure DevOps resources and deploying an application to Azure App Service.

The implementation of this CI/CD pipeline showcases the effectiveness of automation in reducing manual errors, accelerating deployment cycles, and ensuring consistent environments across different stages of development. Terraform's declarative approach to infrastructure management allowed for a scalable and repeatable setup, making it easier to manage and update resources as project requirements evolve.