Azure Infrastructure Operations Project: Deploying a scalable IaaS web server in Azure

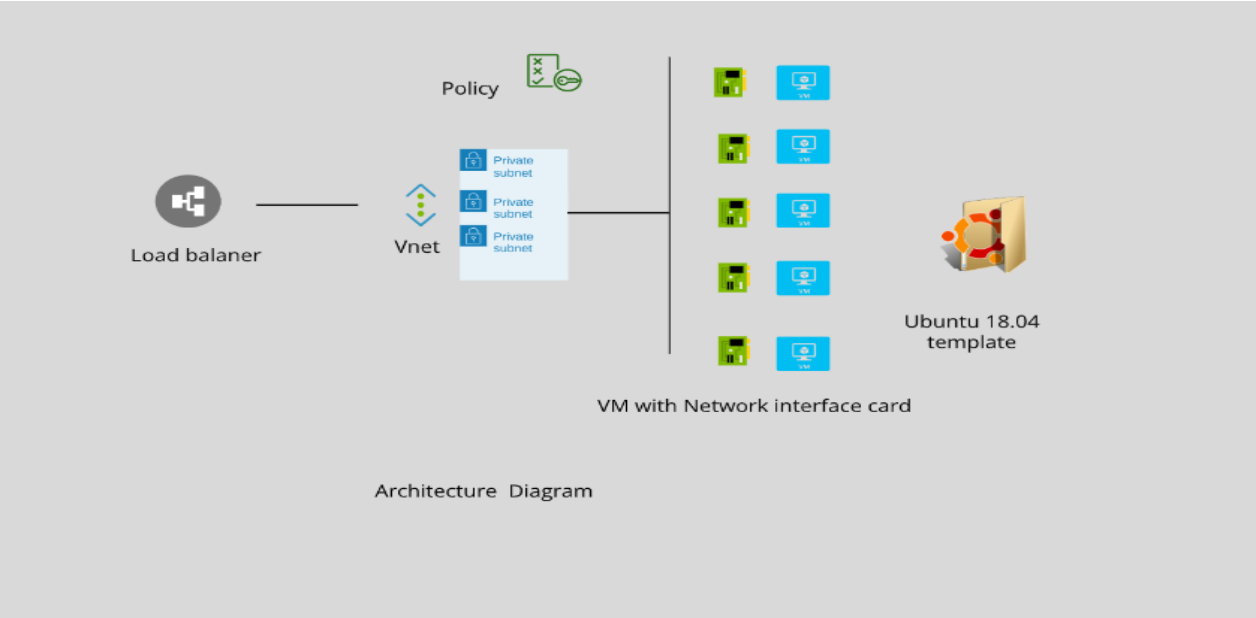
Introduction

Infrastructure as code is one of the five practices that we consider the heart of DevOps. Infrastructure as code provides a significant benefit in terms of creating, deploying, upgrading, and deleting infrastructure. Infrastructure as code makes the deployment procedure as simple as pressing a button.

Packer a server templating software written in JSON, will be used to create virtual machine images containing our application for repeatable deployments. Terraform is a provisioning tool that creates infrastructure according to a configuration file that you create. This file can be saved and reused which enables you to safely and predictably create, change, and improve infrastructure.

For this project, you will write a Packer template and a Terraform template to deploy a customizable, scalable web server in Azure.

Architecture diagram



Step 1

Deploy a Policy

Before we get started, we want to create a policy that ensures all indexed resources are tagged. This will help us with organization and tracking and make it easier to log when things go wrong.

Create and Apply a Tagging Policy

Create a policy that ensures all indexed resources in your subscription have tags and deny deployment if they do not.

Intro to Azure Policy

Azure Policies are vital for the DevOps expert. In addition to providing technical security, policies also give us ways to provide consistency in enforcing industry and business standards and compliance.

Some key points to remember about Azure Policies are:

- They give us a way to manage costs by restricting specific resources.
- Because of the policy-enabled secure architecture, we can determine the scope, source, and solution of problems more quickly.
- Through policy-enabled configuration management, unexpected or even unwanted configuration changes can be prevented

Azure policies are written in JSON and can be deployed in the portal or in the CLI. Policies have two modes: *indexed*, which evaluates only resource types that support tags and locations, and *all*, which evaluates resource groups, subscriptions, and all resource types.

Steps to take to develop a policy for this project

- Make a folder called policy that will contain our policy.
- cd Azure-Devops-Deploy-Web-Server/
- mkdir policy
- Cd policy and mkdir Tags
- Inside Tag folder create policy definition, rule and parameters

```
.
├── azurepolicy.json
├── azurepolicy.parameters.json
└── azurepolicy.rules.json

0 directories, 3 files
```

Tag policy

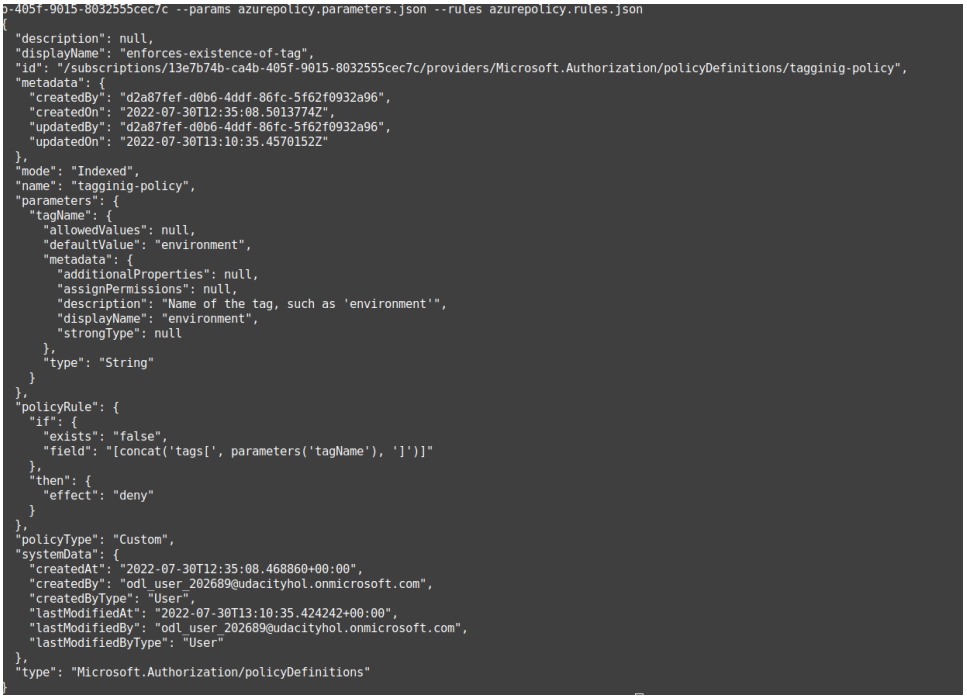

```
{
  "properties": {
    "displayName": "Deny the creation of resources that do not have tags",
    "policyType": "Custom",
    "mode": "Indexed",
    "description": "Enforces existence of a tag. Does not apply to resource groups.",
    "metadata": {
      "version": "1.0.1",
      "category": "Tags"
    },
    "parameters": {
      "tagName": {
        "type": "String",
        "metadata": {
          "displayName": "Tag Name",
          "description": "Name of the tag, such as 'environment'"
        }
      }
    },
    "policyRule": {
      "if": {
        "field": "[concat('tags[', parameters('tagName'), ']')]",
        "exists": "false"
      },
      "then": {
        "effect": "deny"
      }
    }
  },
  "id": "/providers/Microsoft.Authorization/policyDefinitions/871b6d14-10aa-478d-b590-94f262ecfa99",
  "type": "Microsoft.Authorization/policyDefinitions",
  "name": "871b6d14-10aa-478d-b590-94f262ecfa99"
}
```


1. az login
2. az ad signed-in-user show


```
(base) anelay@anelay:~/Desktop/Devops Engineer/Azure-Devops-Deploy-Web-Server$ az login -u odl_user_202448@udacityhol.onmicrosoft.com -p jtbh31GEU*uF
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "f958e84a-92b8-439f-a62d-4f45996b6d07",
    "id": "90a83934-1b3e-4b32-8146-c2760bbda23b",
    "isDefault": true,
    "managedByTenants": [],
    "name": "UdacityDS - 29",
    "state": "Enabled",
    "tenantId": "f958e84a-92b8-439f-a62d-4f45996b6d07",
    "user": {
      "name": "odl_user_202448@udacityhol.onmicrosoft.com",
      "type": "user"
    }
  }
]
```

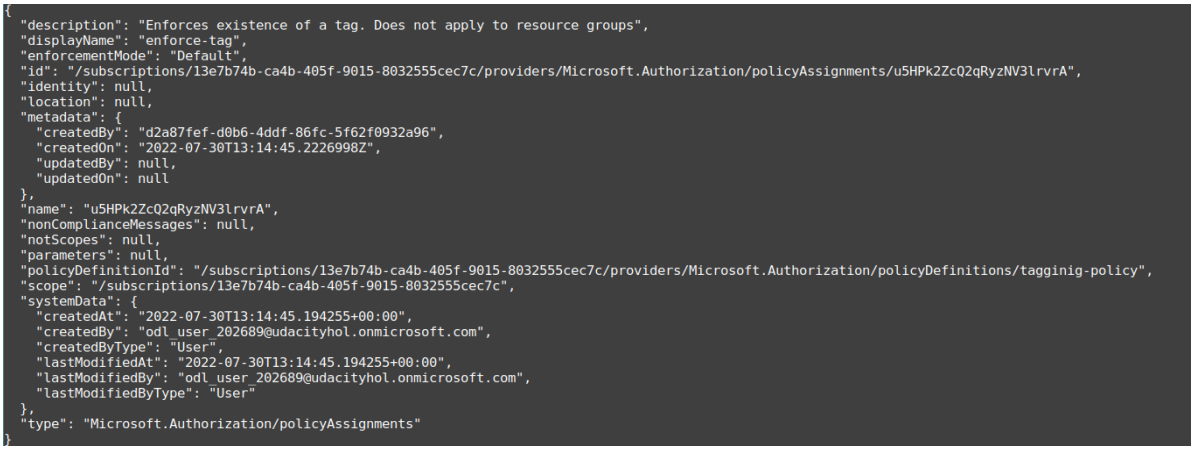
1. **Deploy policy definition**

az policy definition create --name tagginig-policy --display-name enforces-existence-of-tag --subscription 13e7b74b-ca4b-405f-9015-8032555cec7c --params azurepolicy.parameters.json --rules azurepolicy.rules.json



2. **Set up policy assignment**

az policy assignment create --display-name enforce-tag --description "Enforces existence of a tag. Does not apply to resource groups" --policy tagginig-policy



3 List policy assignments

az policy assignment list


```
{
  "description": "Enforces existence of a tag. Does not apply to resource groups",
  "displayName": "enforce-tag",
  "enforcementMode": "Default",
  "id": "/subscriptions/13e7b74b-ca4b-405f-9015-8032555cec7c/providers/Microsoft.Authorization/policyAssignments/u5HPk2ZcQ2qRyzNV3lrvrA",
  "identity": null,
  "location": null,
  "metadata": {
    "createdBy": "d2a87fef-d0b6-4ddf-86fc-5f62f0932a96",
    "createdOn": "2022-07-30T13:14:45.2226998Z",
    "updatedBy": null,
    "updatedOn": null
  },
  "name": "u5HPk2ZcQ2qRyzNV3lrvrA",
  "nonComplianceMessages": null,
  "notScopes": null,
  "parameters": null,
  "policyDefinitionId": "/subscriptions/13e7b74b-ca4b-405f-9015-8032555cec7c/providers/Microsoft.Authorization/policyDefinitions/tagginig-policy",
  "scope": "/subscriptions/13e7b74b-ca4b-405f-9015-8032555cec7c",
  "systemData": {
    "createdAt": "2022-07-30T13:14:45.194255+00:00",
    "createdBy": "odl_user_202689@udacityhol.onmicrosoft.com",
    "createdByType": "User",
    "lastModifiedAt": "2022-07-30T13:14:45.194255+00:00",
    "lastModifiedBy": "odl_user_202689@udacityhol.onmicrosoft.com",
    "lastModifiedByType": "User"
  },
  "type": "Microsoft.Authorization/policyAssignments"
},
```


2 Step 2

Packer template

In order to support application deployment, we'll need to create an image that different organizations can take advantage of to deploy their own apps! To do this, we'll create a packer image that anyone can use, and we'll leverage in our own Terraform template.

Create a Server Image

The first thing we're going to want to do is use Packer to create a server image, ensuring that the provided application is included in the template. You can feel free to write your own from scratch or use the server.json starter code from the Github repository.

```
{
  "variables": {
    "client_id": "{{env `ARM_CLIENT_ID`}}",
    "client_secret": "{{env `ARM_CLIENT_SECRET`}}",
    "subscription_id": "{{env `ARM_SUBSCRIPTION_ID`}}"
  },
  "builders": [{
    "type": "azure-arm",

    "client_id": "{{user `client_id`}}",
    "client_secret": "{{user `client_secret`}}",
    "subscription_id": "{{user `subscription_id`}}",

    "os_type": "Linux",
    "image_publisher": "Canonical",
    "image_offer": "UbuntuServer",
    "image_sku": "18.04-LTS",

    "managed_image_resource_group_name":"Azuredevops",
    "build_resource_group_name": "Azuredevops",
    "managed_image_name": "PackerImage",
    "azure_tags": {
        "environment": "testing"
      },
    "vm_size": "Standard_D2s_v3"
  }],
  "provisioners": [{
    "inline": [
      "echo 'Hello, World!' > index.html",
      "echo 'Hello, from Dawit!' > index.html",
      "nohup busybox httpd -f -p 80 &"
    ],
    "inline_shebang": "/bin/sh -x",
    "type": "shell"
  }]
}
```

Step 3

Terraform Template

Our Terraform template will allow us to reliably create, update, and destroy our infrastructure. In this example, we want to use the skills we've built with variables and loops, along with our knowledge of Azure infrastructure, to deploy a web app that has been loaded into our Packer template already.

Before we get started, we'll need to verify that the policy we deployed in an earlier lesson (that one that requires tags) is still available using the Azure CLI, and include a screenshot of that policy output in our repository.

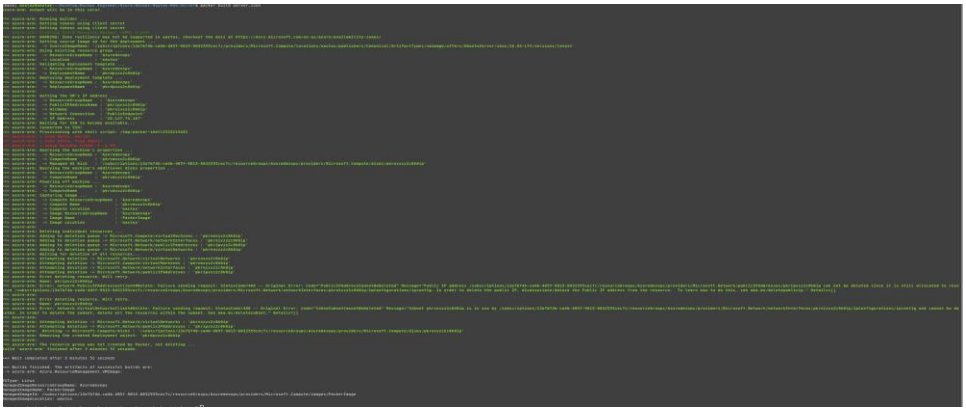
Create directory structures

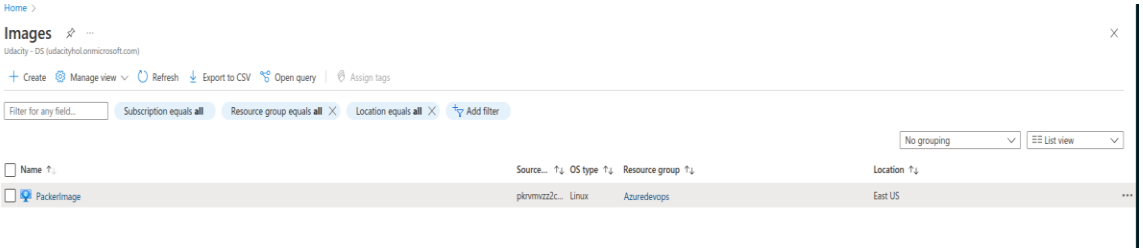
```
.
├── diagram
│   ├── Architectur-Diagram.png
│   └── Architecture  Diagram.pdf
├── images
│   └── tagginigPolicy.png
├── policy
│   └── Tags
│       ├── azurepolicy.json
│       ├── azurepolicy.parameters.json
│       └── azurepolicy.rules.json
├── README.md
├── server.json
└── terraform-provider-azurerm
    └── web-server
        ├── main.tf
        ├── outputs.tf
        └── variables.tf

6 directories, 11 files
```

Setp 4

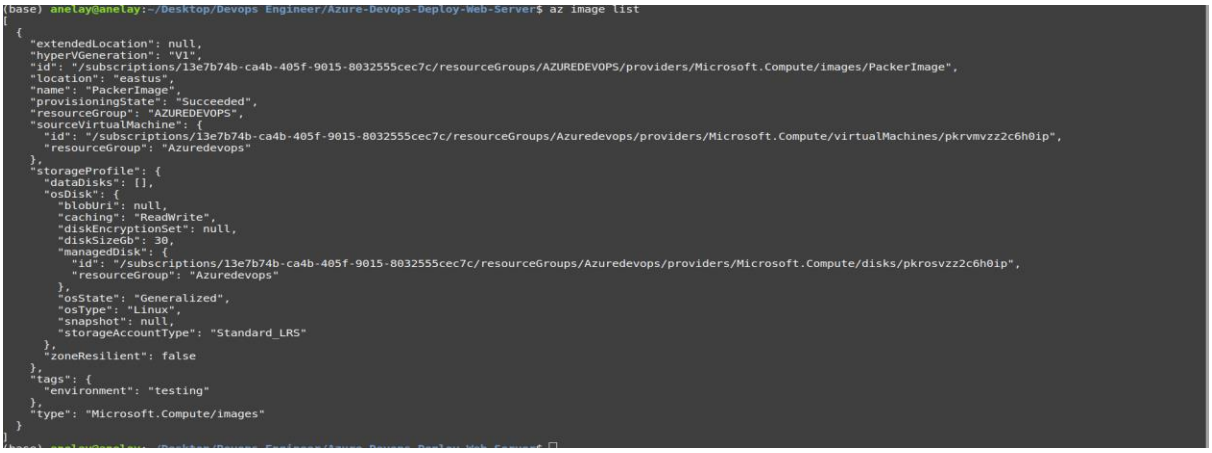
Rum commands

Run pakcer build server.json





Az image list



Import resource group

We've previously established the resources group for our PackerImage, hence we can't deploy the same resource group. We must import the existing resource group before it can determine which resource group to install.

Run the following command

- az group show --name legacy-resource-group --query id --output tsv

This will output the *resourceid* in the following format:

/subscriptions/<SUBSCRIPTIONID>/resourceGroups/legacy-resource-group

- terraform import azurerm_resource_group.legacy-resource-group /subscriptions/<SUBSCRIPTONID>/resourceGroups/legacy-resource-group

- terraform import azurerm_resource_group.main /subscriptions/13e7b74b-ca4b-405f-9015-8032555cec7c/resourceGroups/Azuredevops

  <| Error: Inconsistent dependency lock file

  | The following dependency selections recorded in the lock file are inconsistent with the current configuration:

  | - provider registry.terraform.io/hashicorp/azurerm: required by this configuration but no version is selected

| To make the initial dependency selections that will initialize the dependency lock file, run:

| terraform init >

Solved with

terraform init

Successful result