Microsoft

Infrastructure as Code

Shay Shahak – shays@microsoft.com
Azure CSA
Microsoft

Agenda :

- What is IaC
- Tooling Categories
- Mutable and Immutable Infrastructure
- Imperative Code vs. Declarative Code
- Azure ARM
- Terraform
- Azure Blueprint

What Is Infrastructure as (from) Code?

- Infrastructure as code (IaC) is an **approach** to infrastructure automation based on practices from software development.
- It emphasizes **consistent, repeatable** routines for **provisioning and changing** systems and their configuration.
- **Changes are made to definitions** and then rolled out to systems through **unattended processes** that include thorough **validation**.

Infrastructure as Code

- ☑ Reproducible Environments
- ☑ Automation – CI/ CD
- ☑ Trackable – GitHub
- ☑ Language – HCL \ ARM
- ☑ Workflow
- ☑ Providers

- ☒ Apply same config across clouds

Tooling Categories

Ad Hoc Scripts

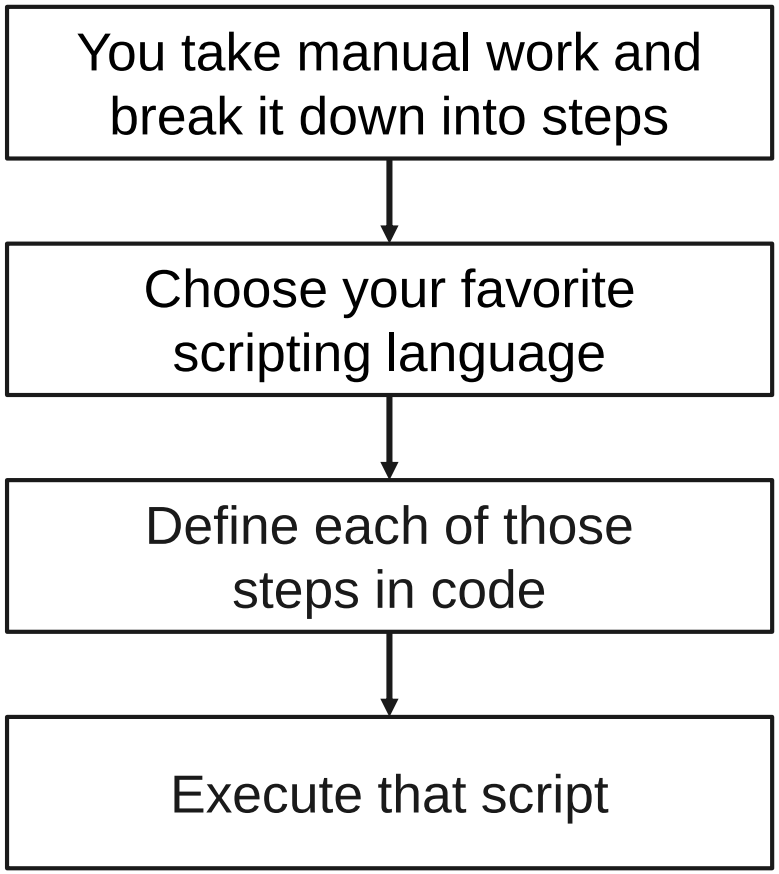
Configuration Management (CM) Tools

Server Templating Tools

Server Provisioning Tools

Ad-Hoc Scripts

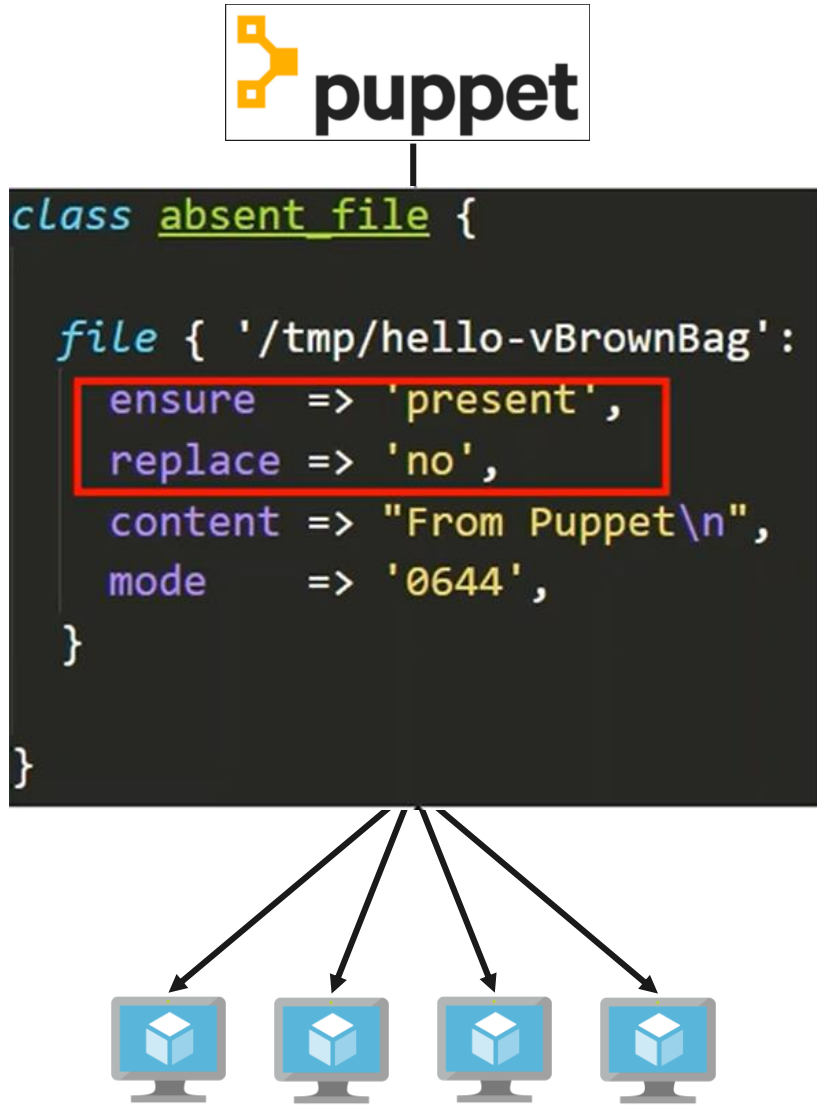
- The most straightforward approach to automating anything is to write an ad hoc script (procedural).




```
# Update the apt-get cache
sudo apt-get update

# Install PHP
sudo apt-get install -y php

# Install Apache
sudo apt-get install -y apache2

# Start Apache
sudo service apache2 start
```

Configuration Management Tools

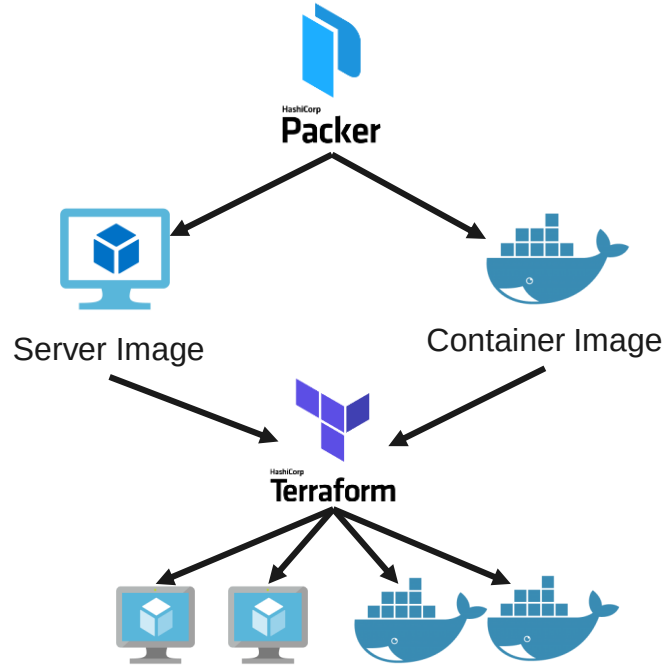
- Chef, Puppet, Ansible, and SaltStack are all configuration management tools, designed to install and manage software on existing servers.
- **Coding conventions** – Consistent & predictable structure, file layout, clearly named parameters, secrets management, etc.
- **Idempotent Code** – Execute the same code repeatedly while producing the same result.
- **Distribution** – Unlike ad hoc scripts, CM tools are designed specifically for managing large numbers of remote servers.



Server Templating Tools

- Growing in it's popularity, are server templating tools such as Docker, Packer, and Vagrant.
- Create an image of a server that captures a fully self-contained "snapshot" of the operating system, the software, the files, and all other relevant details.
- Server templating is a key component of the shift to immutable infrastructure.

```
{
  "builders": [{
    "type": "azure-arm",

    "client_id": "d4db5ab8-ca6b-451e-b0a8-2905523cf168",
    "client_secret": "Passw0rd",
    "tenant_id": "72f988bf-86f1-41af-91ab-2d7cd011db47",
    "subscription_id": "e73c1dbe-2574-4f38-9e8f-c813757b1786",

    "managed_image_resource_group_name": "PackerDemo-RG",
    "managed_image_name": "myPackerImage",

    "os_type": "Linux",
    "image_publisher": "Canonical",
    "image_offer": "UbuntuServer",
    "image_sku": "16.04-LTS",
```



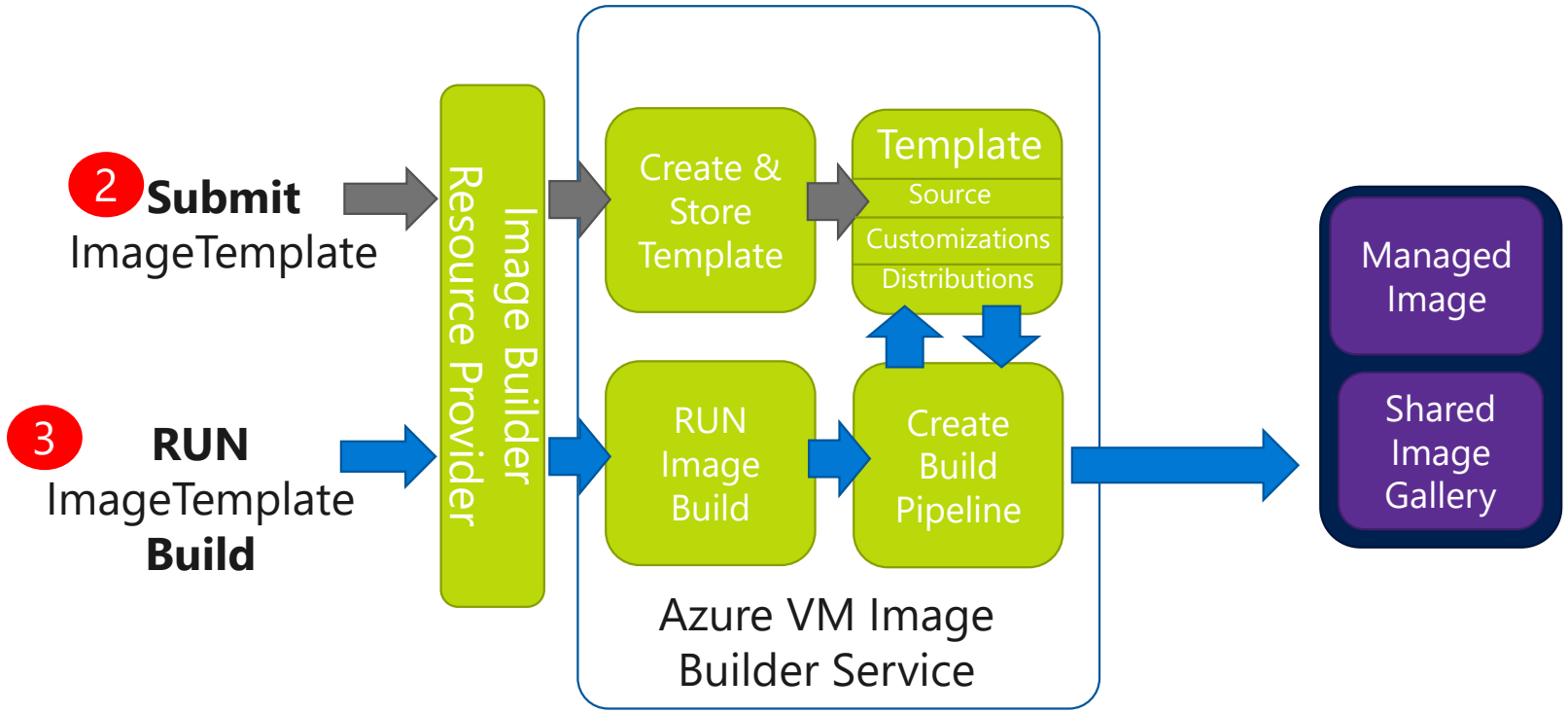
Demo

Packer on Azure

New: Azure Image Builder Service

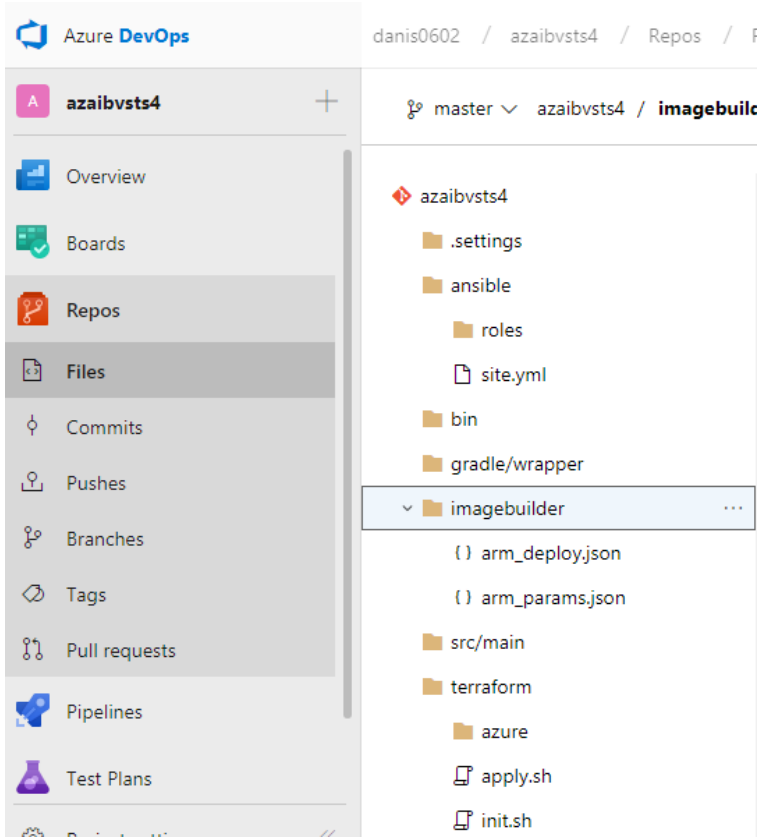
1 Create Image Template

```
"type": "Microsoft.VirtualMachineImages",
"apiVersion": "2018-02-01-preview",
"location": "westcentralus",
"dependsOn": [],
"properties": {
    "source": {
        "type": "PlatformImage",
            "publisher": "Canonical",
            "offer": "UbuntuServer",
            "sku": "18.04-LTS",
            "version": "18.04.201808140"

    },
    "customize": [
        {
            "type": "shell",
            "name": "ProdShellScript",
            "script": "https://raw.githubusercontent.com/../testscript.sh"
        }
    ],
    "distribute":
        [
            {   "type":"managedImage",
                "imageId": "/subscriptions/../../ubuntu091203",
                "location": "westcentralus",
                "runOutputName": "ubuntu091203",
                "tags": {
                    "source": "goimagebuilderarm",
                    "baseosimg": "ubuntu1804"
                }
            }
        ]
    }
```

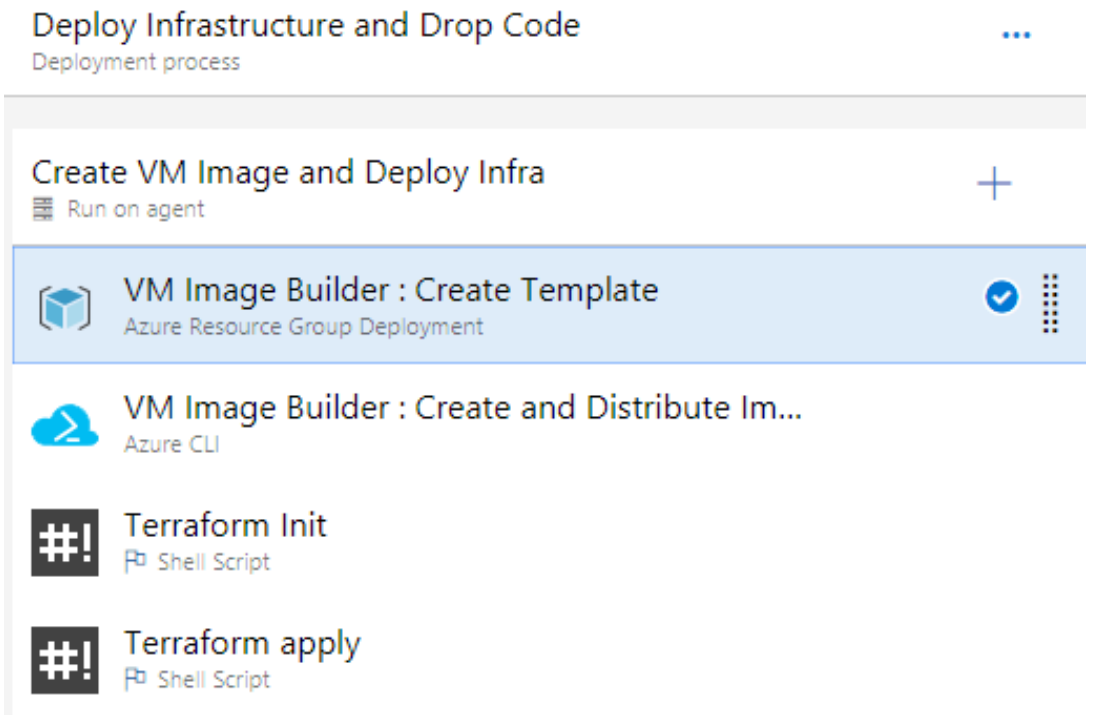


Integration to existing CI/CD Pipeline : Nuts'n'Bolts

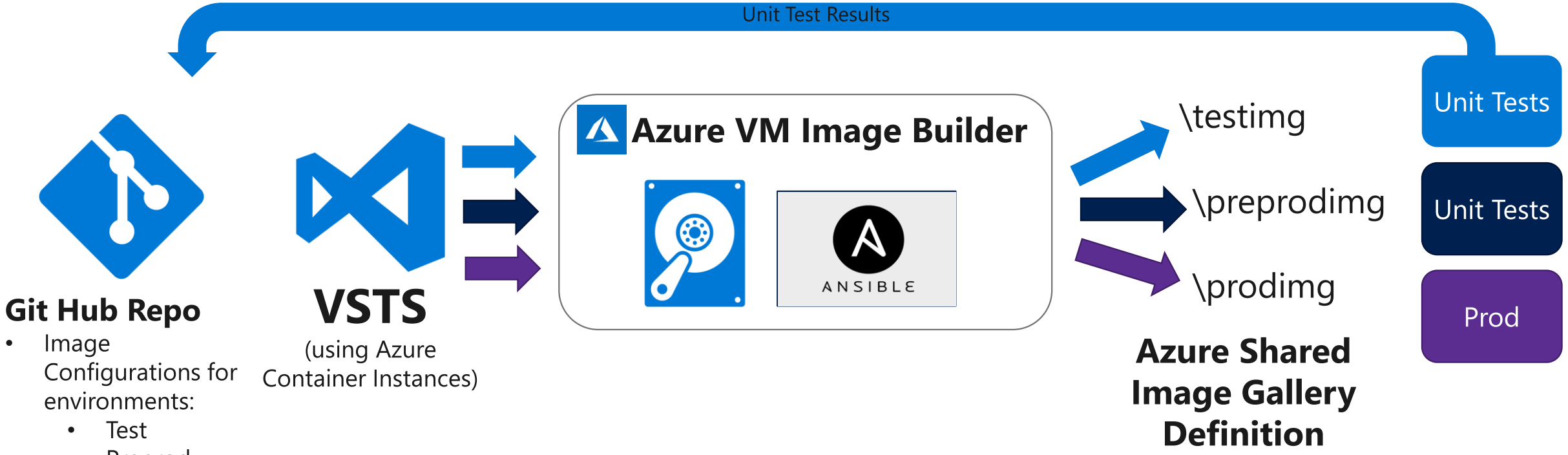
- Image & Infrastructure as Code



- Integrate into Release Pipeline



Integration in Release Processes



- Test image configuration using Image Builder to distribute newly updated images to different Shared Image Definitions.
- When Unit tests complete, send signal of test result, then update the next environment image config

Servers (or “resources”) Provisioning Tools

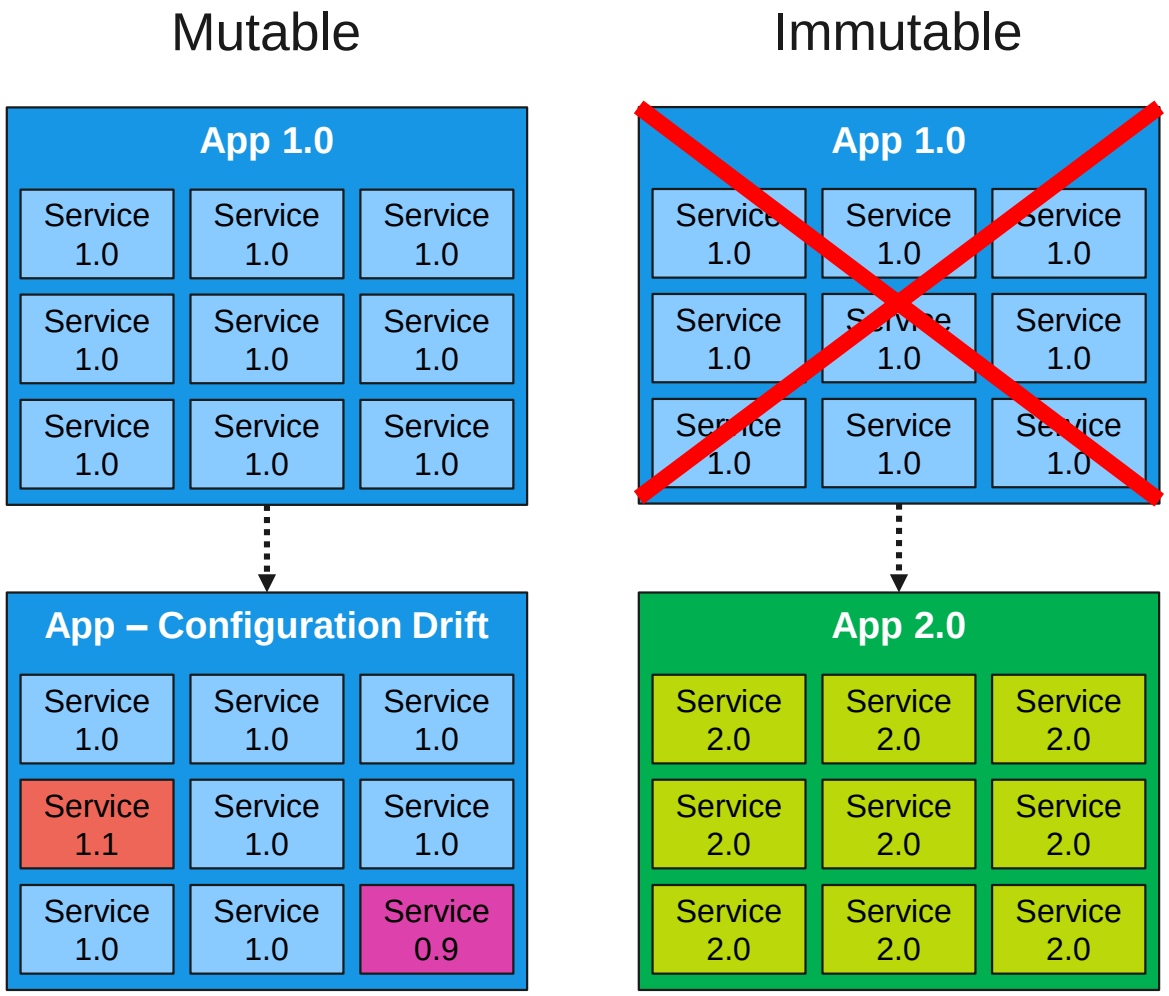
- Provisioning tools such as Terraform, Azure ARM Templates, AWS CloudFormation and OpenStack Heat are responsible for creating the servers themselves.

- You can use this tools not only create servers, but also other **resources** such as databases, load balancers, firewall settings, storage, etc.

```
# Create an Azure resource group
resource "azurerm_resource_group" "terraform" {
  name     = "Terraform-RG"
  location = "${var.location}"
}

# Create a virtual network in the Terraform resource group
resource "azurerm_virtual_network" "terraform" {
  name                = "Terraform-VNet"
  address_space       = ["172.16.0.0/16"]
  resource_group_name = "${azurerm_resource_group.terraform.name}"
  location = "${var.location}"
}

# Create a subnet in Terraform VNet
resource "azurerm_subnet" "terraform" {
  name                 = "Subnet-01"
  resource_group_name  = "${azurerm_resource_group.terraform.name}"
  virtual_network_name = "${azurerm_virtual_network.terraform.name}"
  address_prefix       = "172.16.1.0/24"
}
```

Mutable & Immutable Infrastructure

- The Pets and Cattle debate.
- One approach is not necessary better then the other, it depends on your use-case.
- With the mutable approach, the team needs to be aware of the infrastructure “history”.
- Generally speaking, the immutable approach is better for stateless applications.
- Immutable drives no deviation and no changes. It is what it is.



Imperative Code vs. Declarative Code

Imperative (procedural):
Defines specific commands that need to be executed in the appropriate order to end with the desired conclusion.

AKA “The How”

5

My house is #9 and will be on the right ☺

1

Leave the house

2

Get in the the car

3

Drive straight on Morty Blvd. for 3km

4

Turn right on Rick street and drive for 5 blocks

Imperative Code vs. Declarative Code

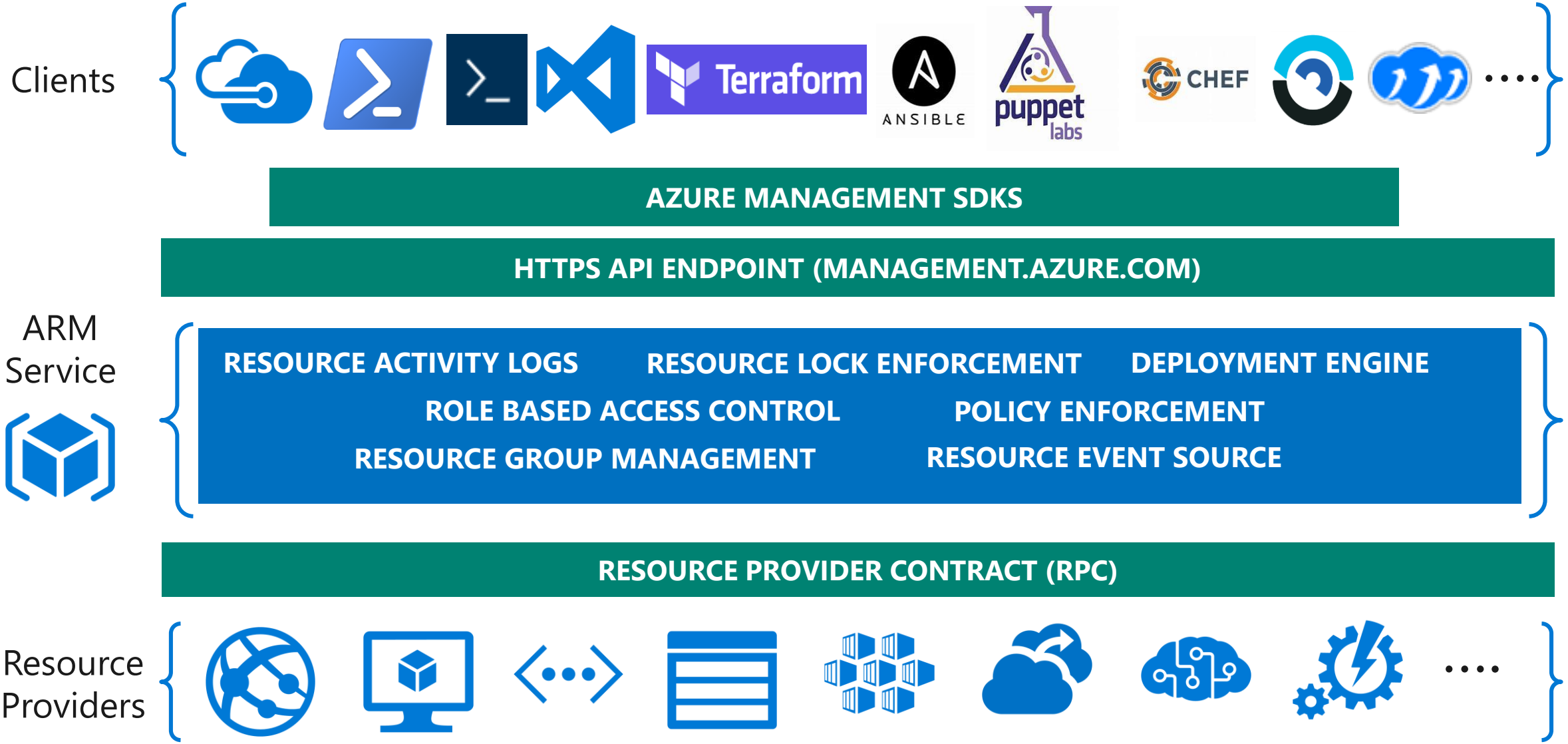
Declarative (functional):
Defines the desired state and the system executes what needs to happen to achieve that desired state.

AKA “The What”

My address is 9 Rick Street, Tel-Aviv Israel 4580800

I’m assuming you have GPS app yes?!

What is Azure Resource Manager?



Demo

Azure ARM Templates using Azure Devops

Azure Provider

Authentication

- Azure CLI
- Service Principal
- MSI

Arguments

```
provider "azurerm" {
    subscription_id     = "{My Subscription ID}"
    client_id           = "{My Service Principle ID}"
    client_secret       = "{My Service Principle Password}"
    tenant_id           = "{My Tenant ID}"
}
```

Environment Variables

Azure Resources & Datasources

```
# Configure the Azure Provider
provider "azurerm" { }

# Create a resource group
resource "azurerm_resource_group" "network" {
  name     = "production"
  location = "West US"
}

# Create a virtual network within the resource group
resource "azurerm_virtual_network" "network" {
  name                = "production-network"
  address_space       = ["10.0.0.0/16"]
  location            = "${azurerm_resource_group.network.location}"
  resource_group_name = "${azurerm_resource_group.network.name}"

  subnet {
    name           = "subnet1"
    address_prefix = "10.0.1.0/24"
  }

  subnet {
    name           = "subnet2"
    address_prefix = "10.0.2.0/24"
  }

  subnet {
    name           = "subnet3"
    address_prefix = "10.0.3.0/24"
  }
```

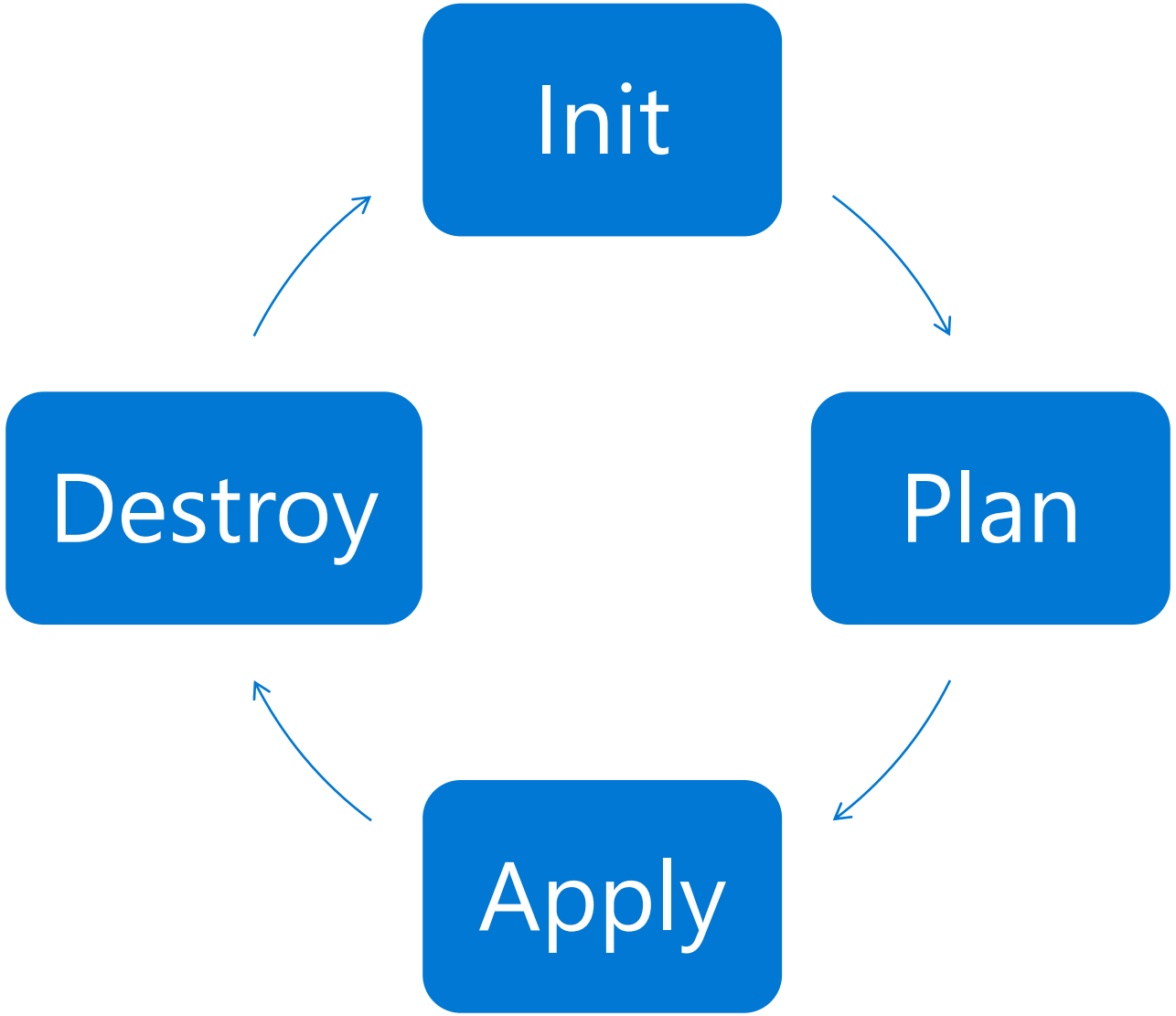
Provisioning for Azure IaaS

Compute (VMSS, Disk, Image, Snapshot, ...)
Networking (Vnet, LB, DNS, ...)
Azure Active Directory
Database (MySQL, PostgreSQL, SQL)
Monitoring
Storage (Storage Account, Blob, Share, ...)
...

Provisioning for Azure PaaS

Containers (AKS, ACI)
Web Apps
CosmosDB
Data Lake
Logic Apps
KeyVault
...

Workflow



A word about the desired state

Example: Ansible Playbook

```
tasks:
  - name: Create VM
    azure_rm_virtualmachine:
      name: "MyServer"
      count: "5"
      resource_group: "My_Resource_Group"
      vm_size: "Standard_DS2_v2"
```

```
tasks:
  - name: Create VM
    azure_rm_virtualmachine:
      name: "MyServer"
      count: "10"
      resource_group: "My_Resource_Group"
      vm_size: "Standard_DS2_v2"
```

15 Servers

Example: Terraform Plan

```
resource "azurerm_virtual_machine" "terraform" {
  name = "MyServer"
  count = "5"
  resource_group_name = "My_Resource_Group"
  vm_size = "Standard_DS2_v2"
```

```
resource "azurerm_virtual_machine" "terraform" {
  name = "MyServer"
  count = "10"
  resource_group_name = "My_Resource_Group"
  vm_size = "Standard_DS2_v2"
```

10 Servers

Demo

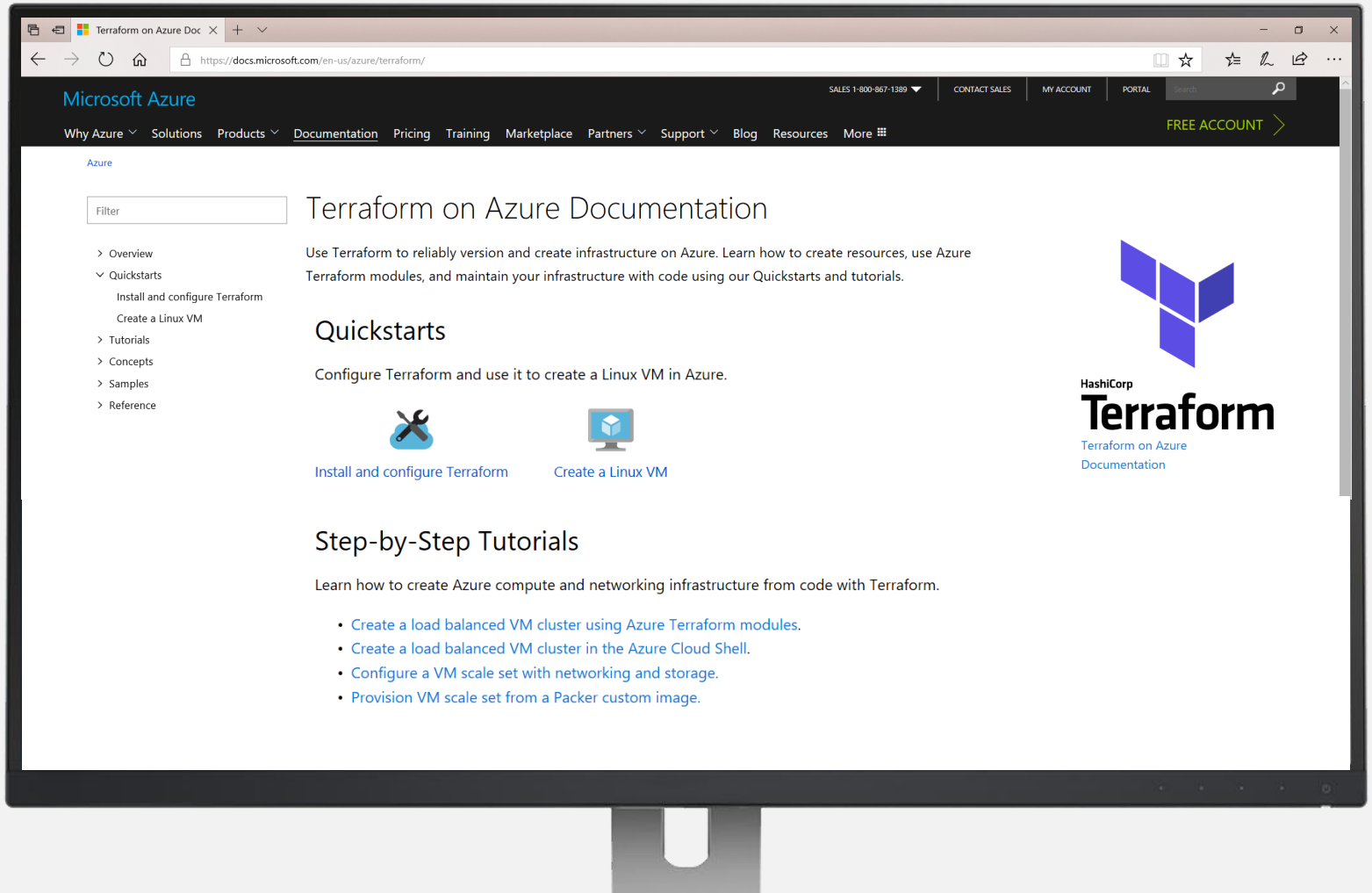
Terraform On Azure

Developer Hub for Terraform

http://aka.ms/tfhub

- https://docs.microsoft.com/azure/Terraform/
- The best place to find technical guidance for Terraform on Azure
- Jenkins and Ansible Developer Hubs also available

ANSIBLE

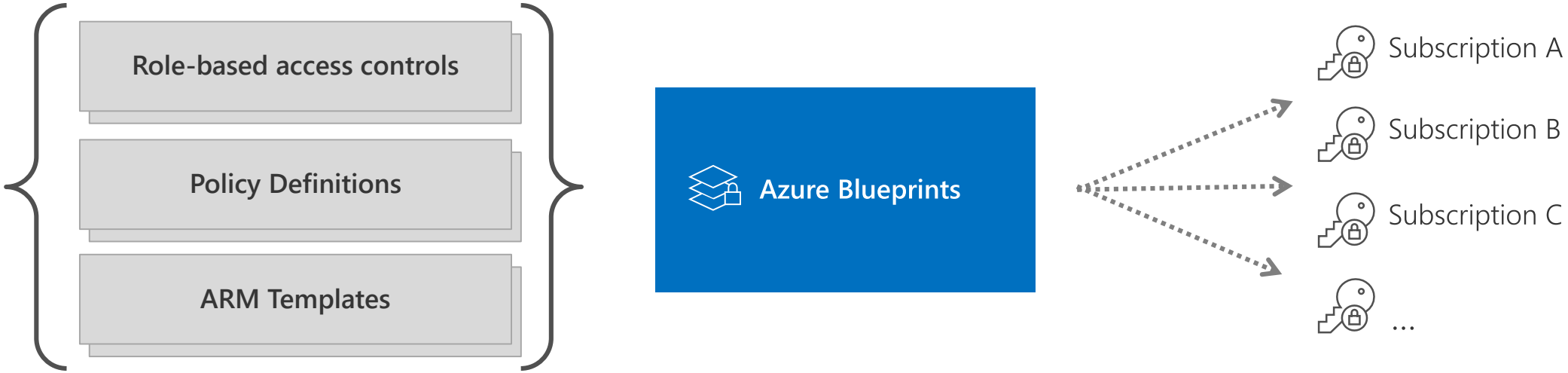


The Major (cross-platform) Players

Tool	Tool Type	Infrastructure	Architecture	Approach	Manifest Written Language
puppet	Configuration Management	Mutable	Pull	Declarative	Domain Specific Language (DSL) & Embedded Ruby (ERB)
CHEF	Configuration Management	Mutable	Pull	Declarative & Imperative	Ruby
ANSIBLE	Configuration Management	Mutable	Push	Declarative & Imperative	YAML
SALTSTACK	Configuration Management	Mutable	Push & Pull	Declarative & Imperative	YAML
HashiCorp Terraform	Provisioning	Immutable	Push	Declarative	HashiCorp Configuration Language (HCL)

Azure Blueprints

deploy and update cloud environments in a repeatable manner using composable artifacts



compose | manage | scale

Demo

Azure Blueprint

Microsoft

Thanks!

Q&A

If you have questions please proceed to the
Q&A MICROPHONE located in your session room.