

# Terraform

## Enablement Workshop

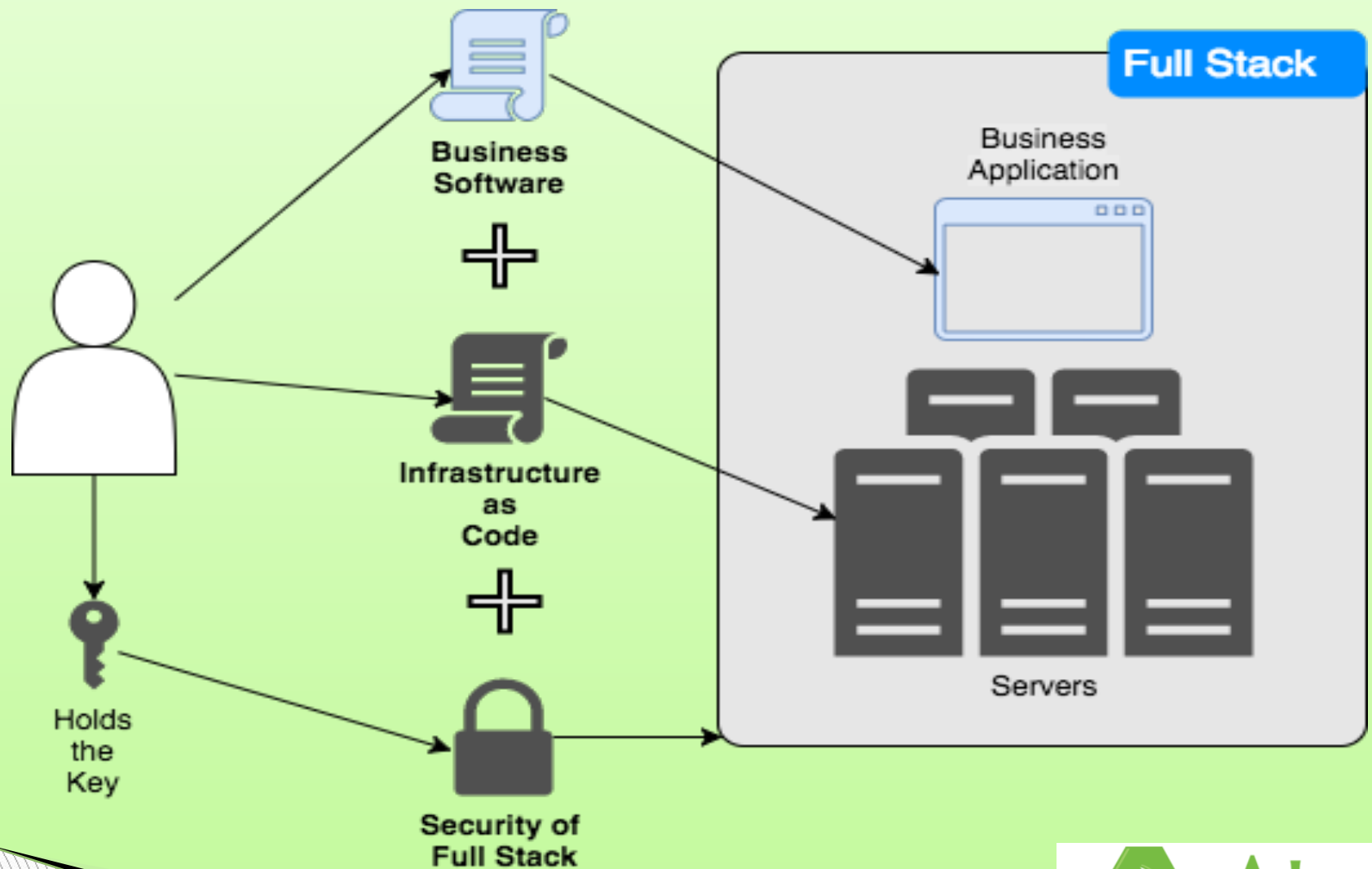
# Agenda

- ▶ What is IaC?
- ▶ DevOps – Terraform
- ▶ Terraform Setup
- ▶ Terraform Configurations
- ▶ Terraform Conditional Statements
- ▶ Terraform – Templates/Modules
- ▶ Terraform - Utility Resources
- ▶ CI/CD - Terraform

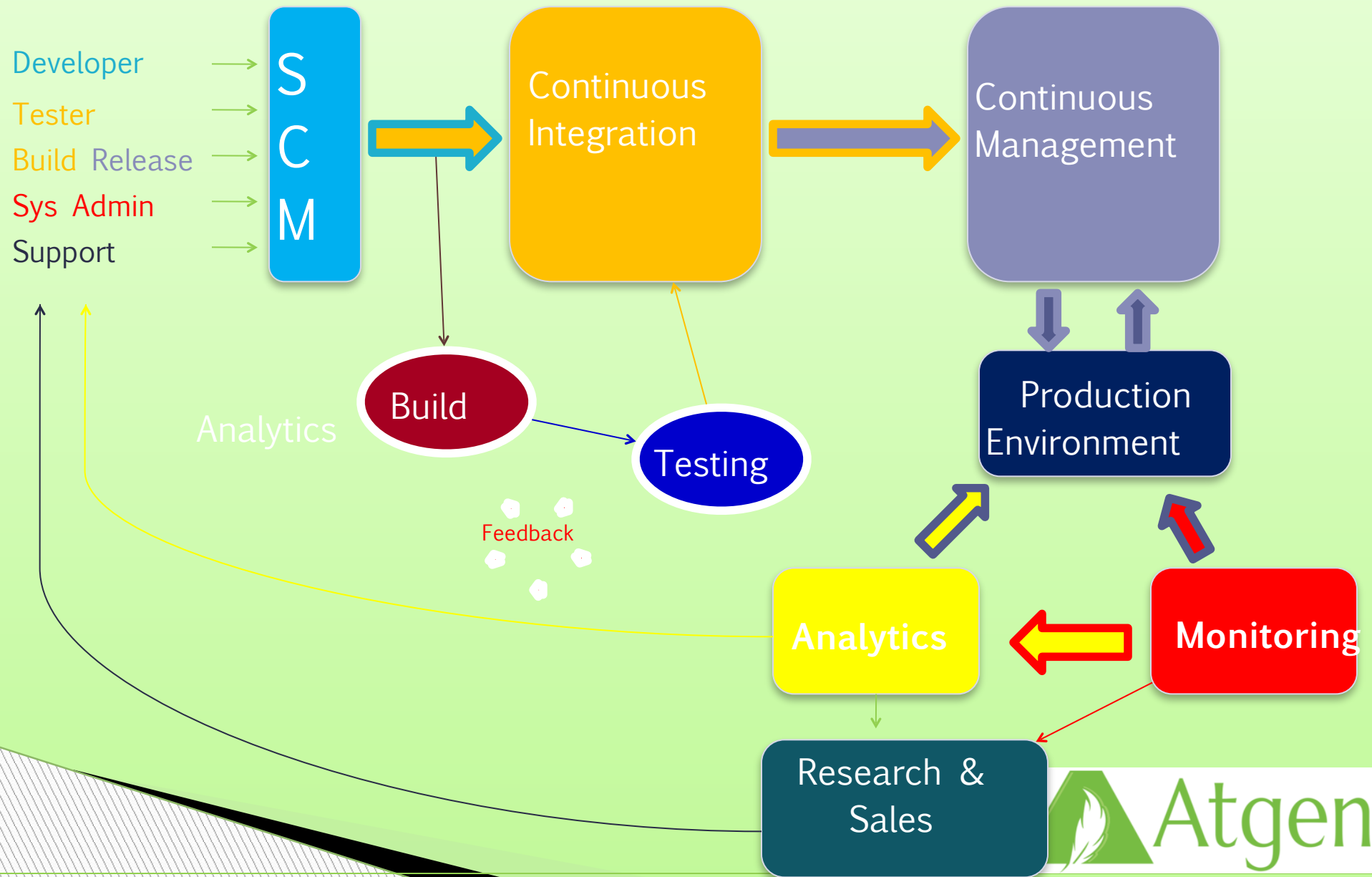
# Session: 1

## Infrastructure as Code

# What is Business Service?



# What is DevOps?



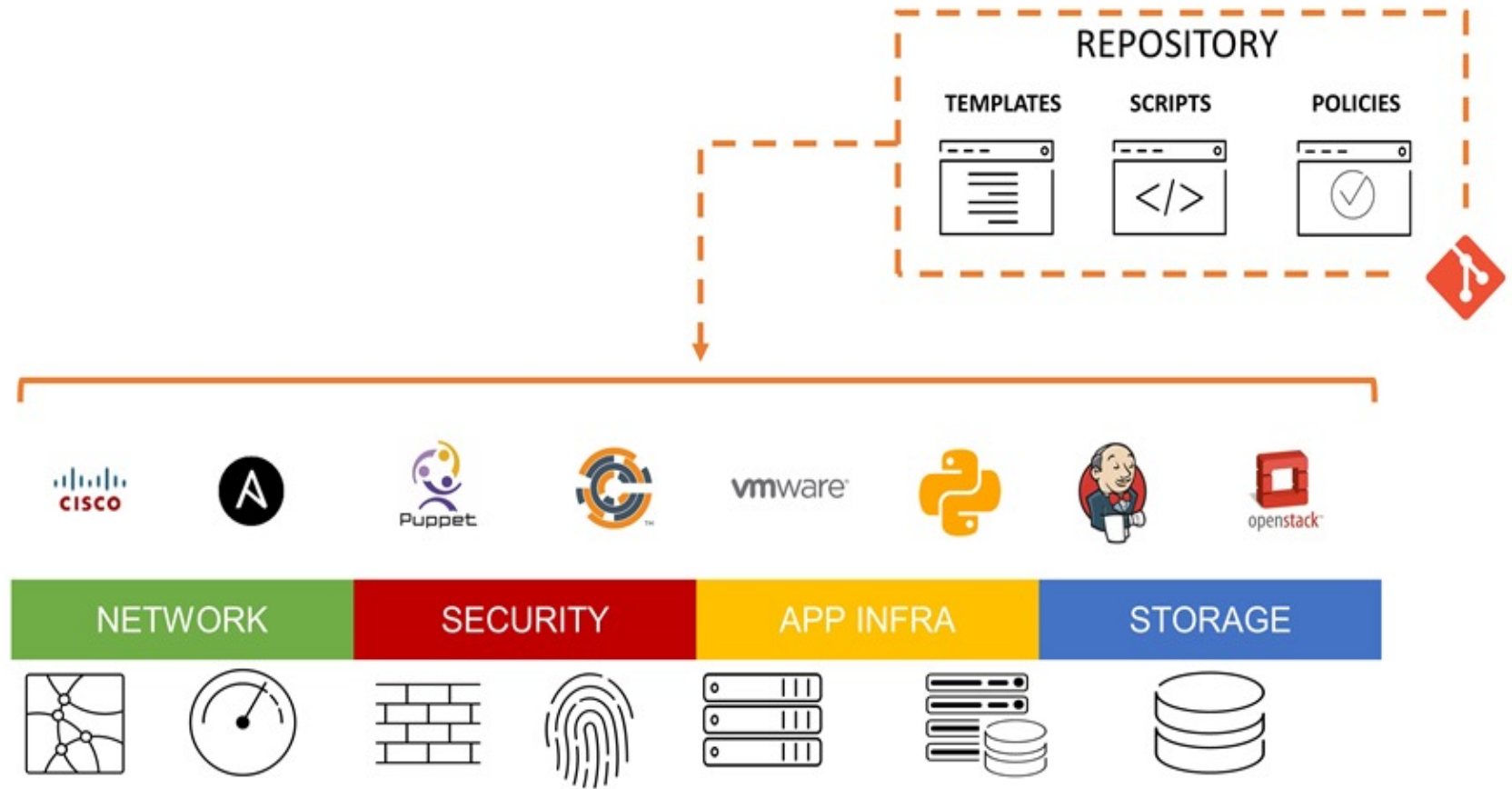
# What is IaC?

- ▶ Infrastructure as code, also referred to as IaC, is a type of IT setup wherein developers or operations teams automatically manage and provision the technology stack for an application through software, rather than using a manual process to configure discrete hardware devices and operating systems.
- ▶ Infrastructure as code is sometimes referred to as programmable or software-defined infrastructure.
- ▶ The concept of infrastructure as code is similar to programming scripts, which are used to automate IT processes. However, scripts are primarily used to automate a series of static steps that must be repeated numerous times across multiple servers.

# What is IaC?

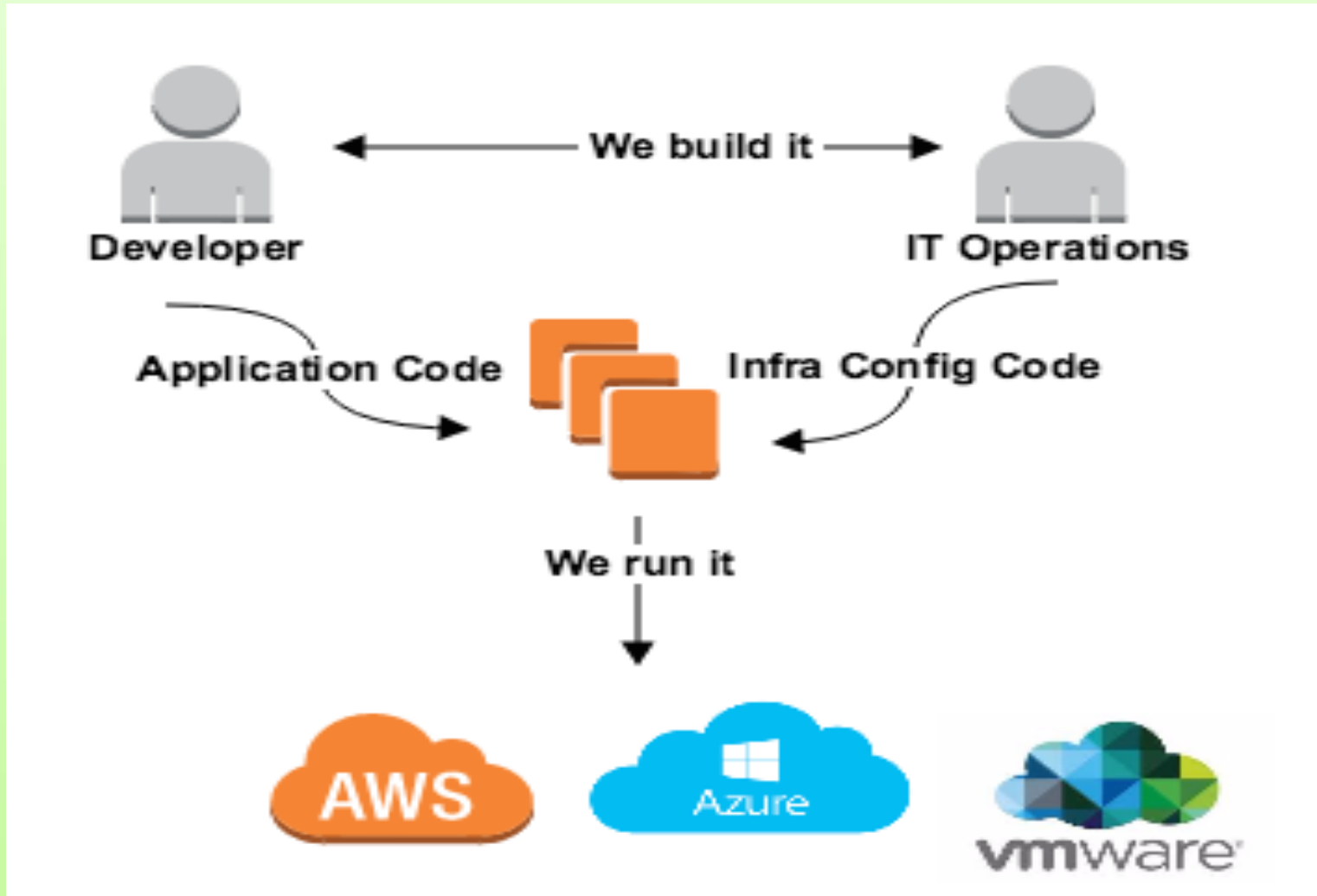
- ▶ Infrastructure as code uses higher-level or descriptive language to code more versatile and adaptive provisioning and deployment processes. For example, infrastructure-as-code capabilities included with Terraform, an IT management and configuration tool, can install WebServer, verify that WebServer is running properly, create a user account and password.
- ▶ The code-based infrastructure automation process closely resembles software design practices in which developers carefully control code versions, test iterations, and limit deployment until the software is proven and approved for production.

# What is IaC?

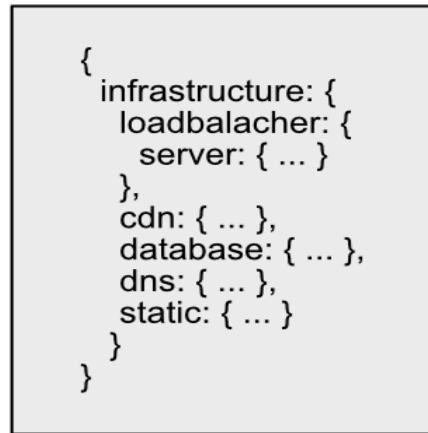




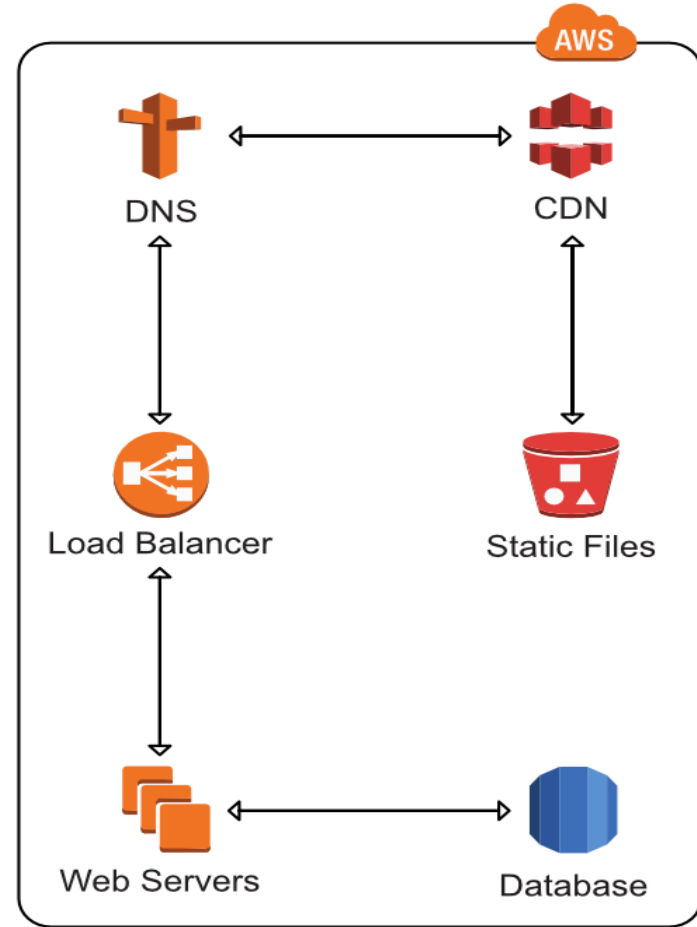
# Where we stand?



# IaC - Example



Tool →



# Benefits of IaC

- ▶ Software developers can use code to provision and deploy servers and applications, rather than rely on system administrators in a DevOps environment.
- ▶ With the infrastructure setup written as code, it can go through the same version control, automated testing, and other steps of a continuous integration and continuous delivery(CI/CD) pipeline that developers use for application code.
- ▶ Because the OS and hardware infrastructure is provisioned automatically and the application is encapsulated atop it, these technologies prove complementary for diverse deployment targets, such as test, staging and production.

# Benefits of IaC

- ▶ Despite its benefits, infrastructure as code poses potential disadvantages. It requires additional tools, such as a configuration management system, that could introduce learning curves and room for error.
- ▶ If administrators change server configurations outside of the set infrastructure-as-code template, there is potential for configuration drift. It's important to fully integrate infrastructure as code into systems administration, IT operations and DevOps practices with well-documented policies and procedures.

# Session: 2

## DevOps - Terraform

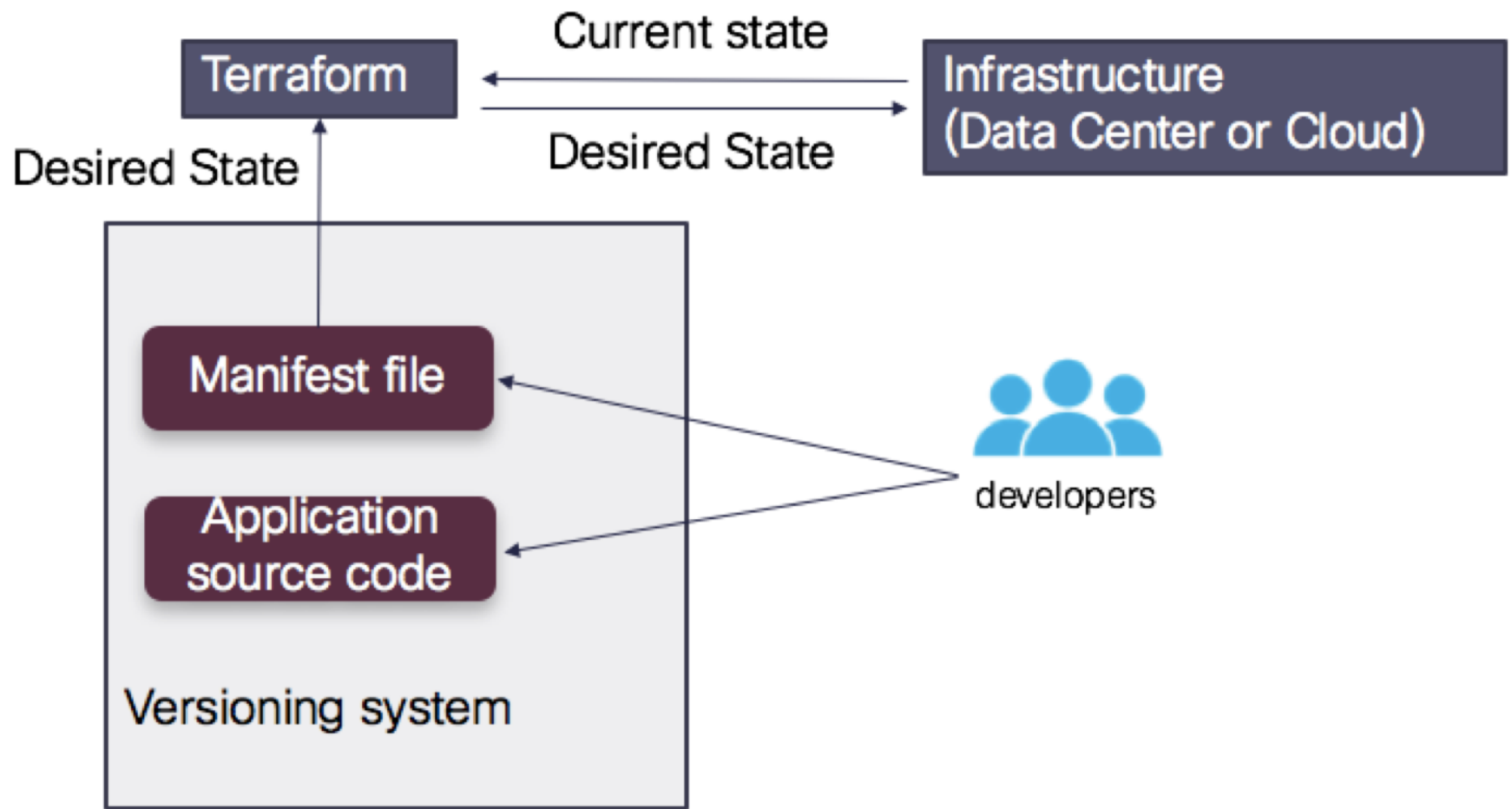
# What is Terraform?

- ▶ Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently.
- ▶ Configuration files describe to Terraform the components needed to run a single application or your entire datacenter.
- ▶ Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure.
- ▶ As the configuration changes, Terraform is able to determine what changed and create incremental execution plans which can be applied.
- ▶ The infrastructure Terraform can manage includes low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc.

# Features of Terraform

- ▶ Infrastructure as Code(IaC)
- ▶ Execution Plans
- ▶ Resource Graph
- ▶ Change Automation
- ▶ Collaboration

# How Terraform works?





# How Terraform works?



# Terraform vs Puppet, Chef

- ▶ Configuration management tools install and manage software on a machine that already exists.
- ▶ Terraform is not a configuration management tool, and it allows existing tooling to focus on their strengths: bootstrapping and initializing resources.
- ▶ Using provisioners, Terraform enables any configuration management tool to be used to setup a resource once it has been created.
- ▶ Terraform focuses on the higher-level abstraction of the datacenter and associated services, without sacrificing the ability to use configuration management tools to do what they do best. It also embraces the same codification that is responsible for the success of those tools, making entire infrastructure deployments easy and reliable.

# Terraform vs Ansible,CF



CloudFormation



Ansible



Terraform

	CloudFormation	Ansible	Terraform
Syntax	JSON	Yaml	HCL
State Management	No	Yes	Yes
Execution Control	No	No	Yes
Manage Already Created Resources	No	Yes	Hard
Providers Support	AWS Only	+++	++



Atgen

# Session: 3

## Classroom Environment

# Terraform – Lab Setup

- ▶ Terraform will be installed for Lab Environment.

- ▶ Steps:

```
yum install -y yum-utils  
yum-config-manager --add-repo https://rpm.releases.hashicorp.com/RHEL/hashicorp.repo  
yum -y install terraform
```

# Session: 4

## Terraform - Configuration

# Terraform – Configuration

- ▶ Terraform uses text files to describe infrastructure and to set variables. These text files are called Terraform *configurations* and end in .tf.
- ▶ The format of the configuration files are able to be in two formats: Terraform format and JSON.
- ▶ The Terraform format is more human-readable, supports comments, and is the generally recommended format for most Terraform files.
- ▶ The JSON format is meant for machines to create, modify, and update.

# Load Order & Semantics

- ▶ When invoking any command that loads the Terraform configuration, Terraform loads all configuration files within the directory specified in alphabetical order.
- ▶ The files loaded must end in either .tf or .tf.json to specify the format that is in use. Otherwise, the files are ignored.
- ▶ Override files are the exception, as they're loaded after all non-override files, in alphabetical order.
- ▶ The configuration within the loaded files are appended to each other.
- ▶ The order of variables, resources, etc. defined within the configuration doesn't matter.



# Configuration Syntax

- ▶ The syntax of Terraform configurations is called HashiCorp Configuration Language (HCL).
- ▶ It is meant to strike a balance between human readable and editable as well as being machine-friendly.
- ▶ For machine-friendliness, Terraform can also read JSON configurations.
- ▶ For general Terraform configurations, however, we recommend using the HCL Terraform syntax.

# Providers

- ▶ Terraform is used to create, manage, and update infrastructure resources such as physical machines, VMs, network switches, containers, and more.
- ▶ Almost any infrastructure type can be represented as a resource in Terraform.
- ▶ A provider is responsible for understanding API interactions and exposing resources.
- ▶ Providers generally are an IaaS (e.g. AWS, GCP, Microsoft Azure, OpenStack), PaaS (e.g. Heroku), or SaaS services (e.g. Terraform Enterprise, DNSimple, CloudFlare).

# Providers

- ▶ All the terraform providers can be found at:

<https://registry.terraform.io/browse/providers>

# Provider Configurations

- ▶ Providers are responsible in Terraform for managing the lifecycle of a resource: create, read, update, delete.
- ▶ Most providers require some sort of configuration to provide authentication information, endpoint URLs, etc.
- ▶ By default, resources are matched with provider configurations by matching the start of the resource name. For example, a resource of type `vsphere_virtual_machine` is associated with a provider called `vsphere`.

# Provider Configurations - Example

- ▶ A provider configuration looks like the following:

```
provider "azurerm" {  
  features {}  
  client_certificate_path = "/etc/.azure/mycert.pfx"  
  subscription_id = "fd2abdcd-0718-4907-abcd-5a0ab7e1c05e"  
  client_id = "ab2abdcd-0718-4907-abcd-5a0ab7e1c05e"  
  tenant_id = "gh2abdcd-0718-4907-abcd-5a0ab7e1c05e"  
}
```

# Provider Initialisations

- ▶ Each time a new provider is added to configuration, it's necessary to initialise that provider before use.
- ▶ Initialisation downloads and installs the provider's plugin and prepares it to be used.
- ▶ Provider initialisation is one of the actions of terraform init. Running this command will download and initialise any providers that are not already initialised.
- ▶ Providers downloaded by terraform init are only installed for the current working directory.
- ▶ Note that terraform init cannot automatically download providers that are not distributed by HashiCorp.

# Multiple Provider Instances

- ▶ You can define multiple configurations for the same provider in order to support multiple regions, multiple hosts, etc.

# The default provider configuration

```
provider "azurerm" {  
    # ...  
}
```

# Additional provider configuration for west coast region

```
provider "azurerm" {  
    alias    = "west"  
    client_certificate_path = "/etc/.azure/mycert.pfx"  
}
```

# Multiple Provider Instances

- ▶ Using provider in resource:

```
resource "azurerm_linux_virtual_machine" "foo" {  
  provider = "azurerm.west"  
  # ...  
}
```



# Resource Configurations

- ▶ The most important thing you'll configure with Terraform are resources.
- ▶ Resources are a component of your infrastructure.
- ▶ It might be some low level component such as a physical server, virtual machine, or container. Or it can be a higher level component such as an email provider, DNS record, or database provider.
- ▶ The resource block creates a resource of the given TYPE (first parameter) and NAME (second parameter). The combination of the type and name must be unique.

# Resource Configurations - Example

- ▶ A resource configuration looks like the following:

```
resource "azurerm_linux_virtual_machine" "main" {  
  name                    = "myvm-vm"  
  size                   = "Standard_F2"  
  admin_username         = "adminuser"  
  admin_password         = "P@ssw0rd1234!"  
  disable_password_authentication = false  
  
  source_image_reference {  
    publisher = "Canonical"  
    offer     = "UbuntuServer"  
    sku       = "18.04-LTS"  
    version   = "latest"  
  }  
  
  os_disk {  
    storage_account_type = "Standard_LRS"  
    caching               = "ReadWrite"  
  }  
}
```

# Terraform Commands (CLI)

- ▶ Terraform is controlled via a very easy to use command-line interface (CLI).
- ▶ Terraform is only a single command-line application: terraform. This application then takes a subcommand such as "apply" or "plan".
- ▶ The terraform CLI is a well-behaved command line application. In erroneous cases, a non-zero exit status will be returned. It also responds to -h and --help as you'd most likely expect.

terraform --help



# Terraform Commands (CLI)

- ▶ Get Terraform plugins as per configuration:

`terraform init`

- ▶ Validate Terraform configurations:

`terraform validate`

- ▶ Validate configurations in Simulation mode:

`terraform plan`

- ▶ Apply Terraform configurations:

`terraform apply`

# Terraform Commands (CLI)

- ▶ Destroy Terraform configuration:

```
terraform destroy
```

- ▶ Save a plan:

```
terraform plan -out=./plan
```

- ▶ Apply a plan:

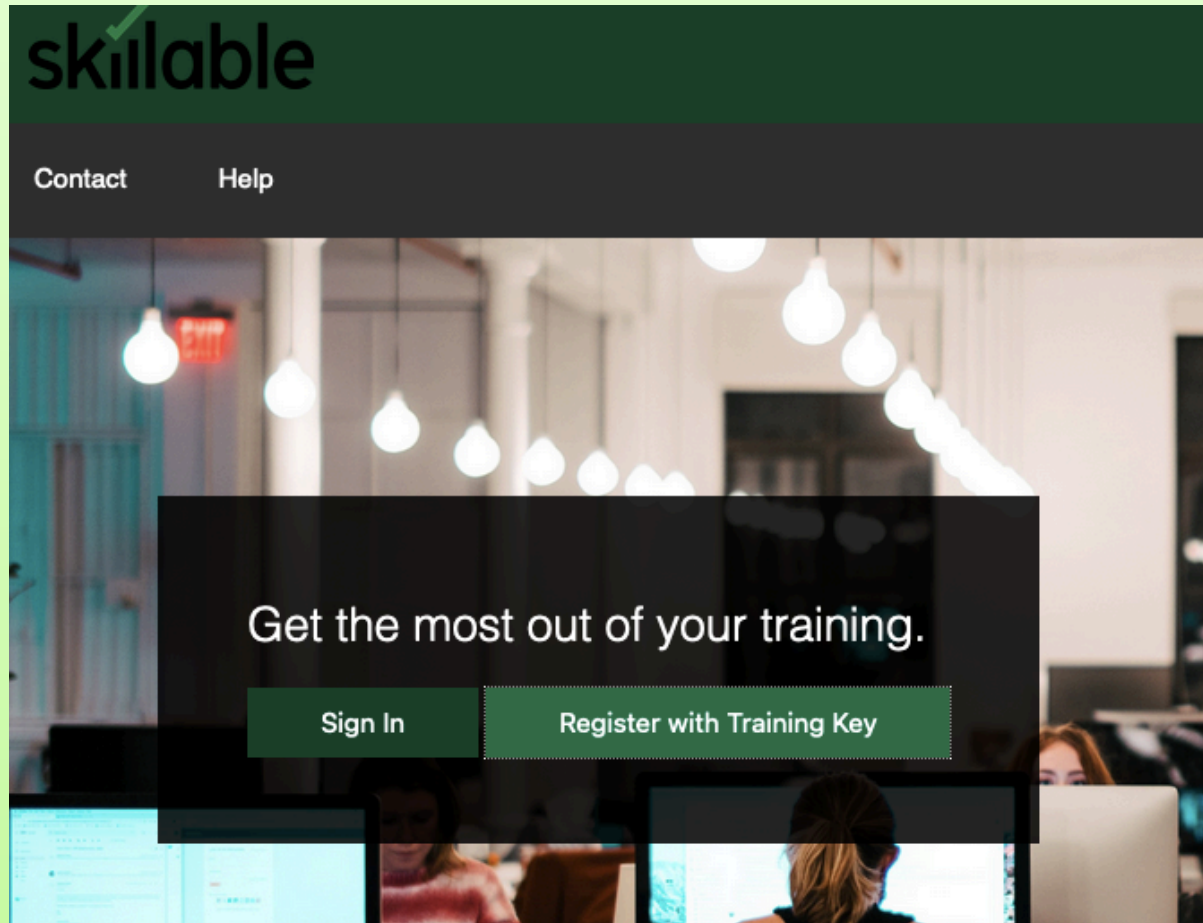
```
terraform apply ./plan
```

- ▶ Show plan or state:

```
terraform show
```

# Redeem Vouchers

- Go to <https://alh.learnondemand.net/>



# Redeem Vouchers

- ▶ Enter Code given and Register

Register with Training Key


×

**Register with a Training Key**

Register

# Redeem Vouchers

- ▶ Enter details and click on Save.

 Register

First Name:

Sagar

\*

Last Name:

Mehta

\*

Primary Email:

sagar.mehta@atgensoft.com

\*

Username:

sagar.mehta@atgensoft.com

Password:

.....

☐ Show Characters OK

Confirm Password:

.....

Password Policy

Country:

India

▼

\*

Time Zone:

(UTC+05:30) Chennai, Kolkata, Mumbai, New Delhi

▼



☒ Enable Notifications

Save




# Redeem Vouchers

- Click on My Training

Sagar ▾ 

[My Training](#) [My Transcript](#) [Contact](#) [Help](#)

 Current Training  
Sagar Mehta 🔍 Details ✎ Edit

[📄 Transcript](#) [🔑 Redeem Training Key](#)


All times shown in India Standard Time

Classes (1) ^

Class	Room	When ↑	Status
AZ-400T00-A Designing and Implementing Microsoft DevOps Solutions [Azure Pass Provided]	Alchemy Online Classroom 01	Monday, March 11, 2024 1:20 PM - Tuesday, April 30, 2024 11:55 PM (India Standard Time)	Enrolled

# Redeem Vouchers

- ▶ Click on AZ-400 Class and Accept Agreement

 Enrollment

Sagar Mehta

AZ-400T00-A Designing and Implementing Microsoft DevOps Solutions [Azure Pass Provided]

Basic Information

Student: Sagar Mehta Details ▾

Event: AZ-400T00-A Designing and Implementing Microsoft DevOps Solutions [Azure Pass Provided] Details ▾

Enrollment Status: Enrolled

Completion Status: Unknown

Classroom: Alchemy Solutions: Alchemy Online Classroom 01

Expires: Sunday, October 27, 2024

Is Retake: No

Enable Labs: Yes

Get Ready

Software Check

✓ OK

✓ 1. Microsoft End User License Agreement

View Agreement

You agreed on 3/11/2024 8:45 PM.

# Redeem Vouchers

## ► Launch Hands-on Lab

### Hands-on Labs



**Microsoft Azure DevOps Solutions** (Expected Duration 1 days, 16 hours, 0 minutes)  
AZ-400T00-A Designing and Implementing Microsoft DevOps Solutions [Azure Pass Provided], All Learning Paths



Required: Yes

Status: **Running**

Started: Monday, March 11, 2024 8:47 PM (India Standard Time)

Launch

Cancel

# Redeem Vouchers

- ▶ Get your Promo Code under Resources

Microsoft Azure DevOps Solutions


39 Hr 54 Min Remaining

100%

Instructions Resources Help

### Azure Promo Code

PromoCode

**SEA-DEV**

Username

Password

Ctrl+Alt+Delete

[Open in New Window](#)

# Redeem Vouchers

- ▶ Redeem using <https://www.microsoftazurepass.com/>

# Generate a Client Certificate

- ▶ Create CSR using openssl

```
openssl req -newkey rsa:4096 -nodes -keyout "mycert.key" -out  
"mycert.csr"
```

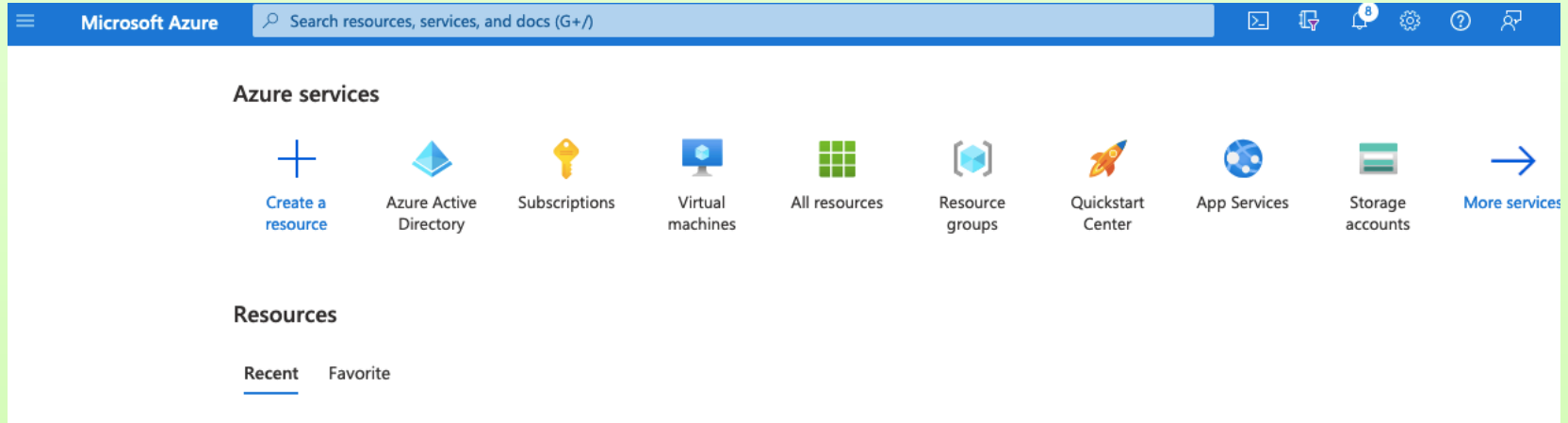
- ▶ Signing CSR

```
openssl x509 -signkey "mycert.key" -in "mycert.csr" -req -days 365  
-out "mycert.crt"
```

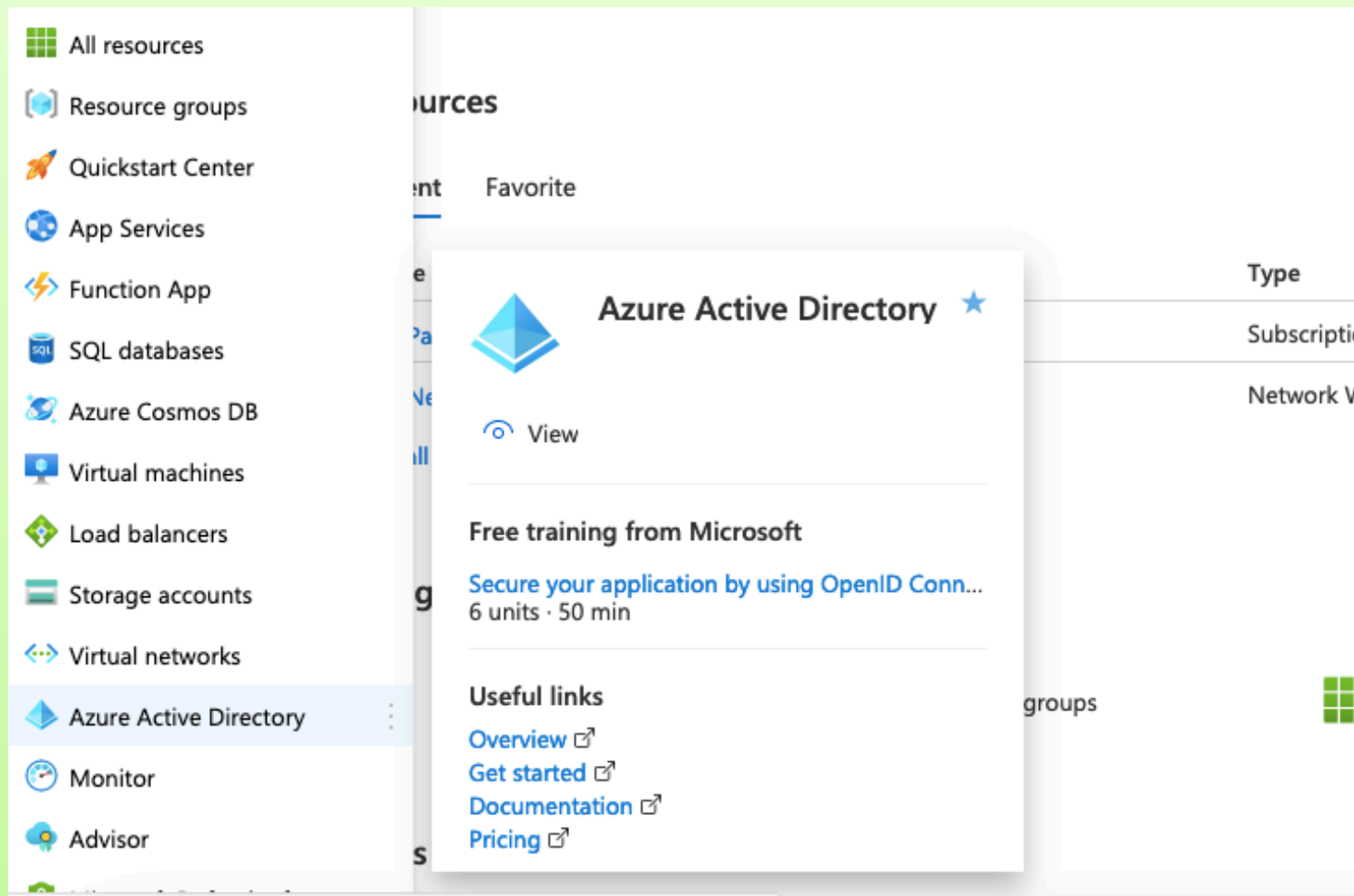
- ▶ Creating PFX

```
openssl pkcs12 -export -out "mycert.pfx" -inkey "mycert.key" -in  
"mycert.crt"
```

# Login to Azure Portal



# Go to Azure Active Directory









The screenshot shows the Azure portal interface. On the left, the navigation menu lists various resources, with 'Azure Active Directory' highlighted. A modal window for 'Azure Active Directory' is open, featuring the Azure logo, a 'View' button, and sections for training and useful links.

**Navigation Menu:**

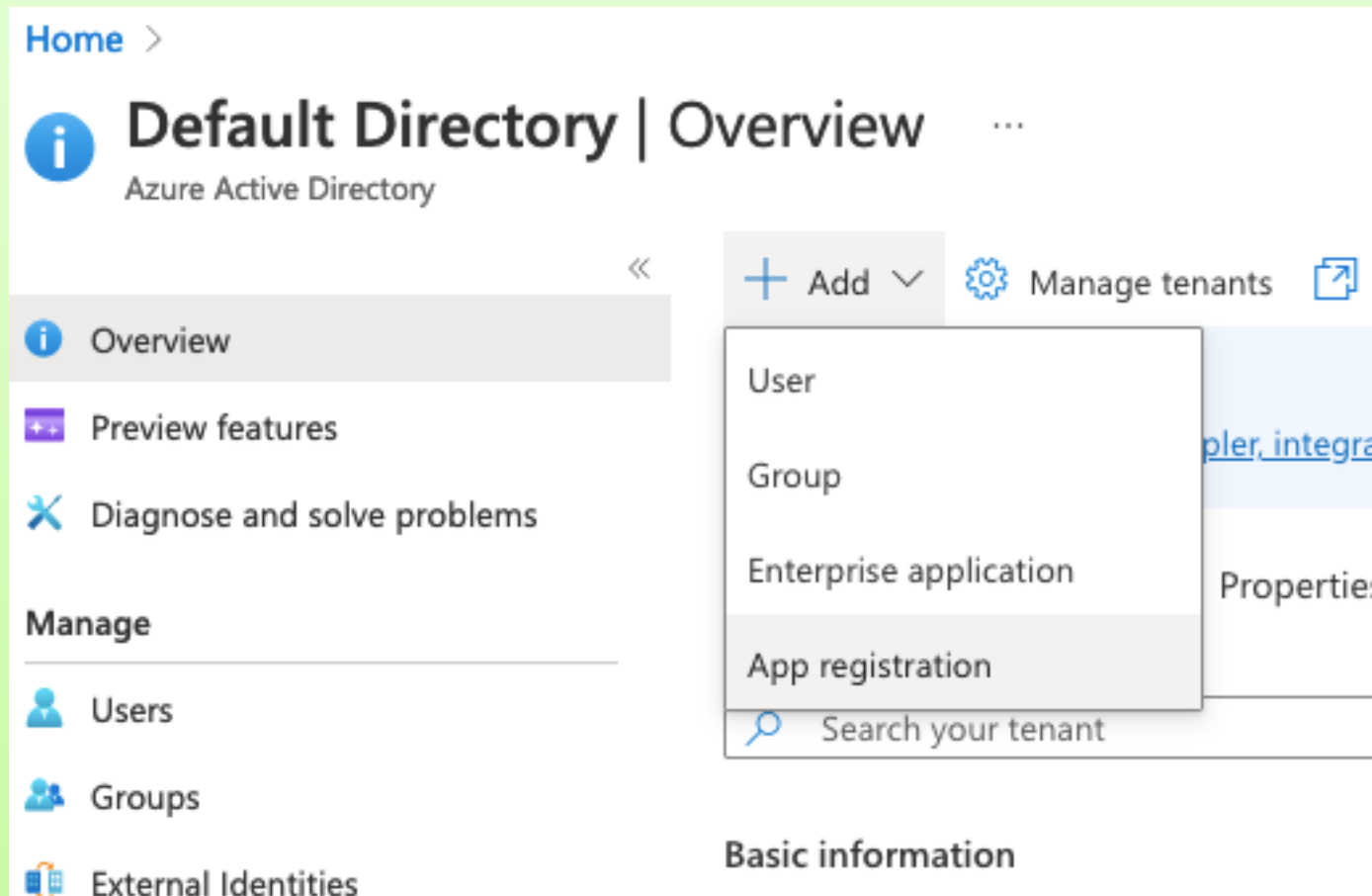
- All resources
- Resource groups
- Quickstart Center
- App Services
- Function App
- SQL databases
- Azure Cosmos DB
- Virtual machines
- Load balancers
- Storage accounts
- Virtual networks
- Azure Active Directory**
- Monitor
- Advisor

**Azure Active Directory Modal:**

-  **Azure Active Directory** ★
-  View
- Free training from Microsoft**
- [Secure your application by using OpenID Conn...](#)  
6 units · 50 min
- Useful links**
- [Overview](#) 
- [Get started](#) 
- [Documentation](#) 
- [Pricing](#) 



# Add - App registration



# Enter App Name

[Home](#) > [Default Directory](#) | [Overview](#) >

## Register an application ...

### \* Name

The user-facing display name for this application (this can be changed later).



### Supported account types

Who can use this application or access this API?

- ☒ Accounts in this organizational directory only (Default Directory only - Single tenant)
- ☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- ☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
- ☐ Personal Microsoft accounts only

[Help me choose...](#)

# Enter 'who can use' and 'Redirect URI':

## Supported account types

Who can use this application or access this API?

- ☒ Accounts in this organizational directory only (Default Directory only - Single tenant)
- ☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- ☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
- ☐ Personal Microsoft accounts only

[Help me choose...](#)

## Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.





Web


Register an app you're working on here. Integrate gallery apps and other apps from outside your organization by adding from [Enterprise applications](#).



# Add Certificate to App

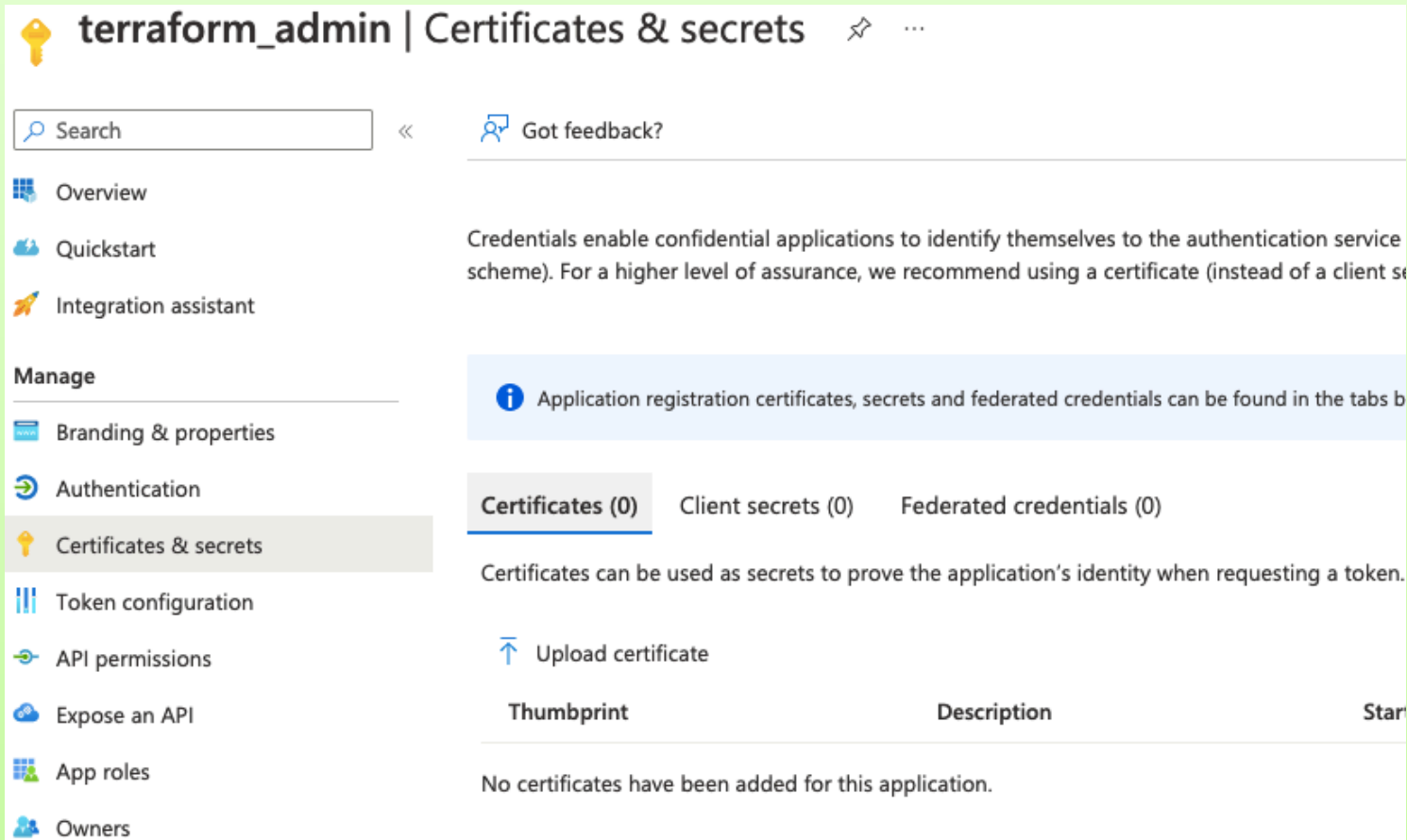
^ Essentials

Display name	: <a href="#">terraform_admin</a>	Client credentials	: <a href="#">Add a certificate or secret</a> 
Application (client) ID	: bb057971-d3f1-4055-bdd7-e456db2d807b	Redirect URIs	: <a href="#">Add a Redirect URI</a> 
Object ID	: 7c12e4d7-415e-45b2-92cb-6217 	Application ID URI	: <a href="#">Add an Application ID URI</a>
Directory (tenant) ID	: df8af829-df8a-41fe-8679-3e0a616165c5 	Managed application in l...	: <a href="#">terraform_admin</a>
Supported account types	: <a href="#">My organization only</a>		

 Welcome to the new and improved App registrations. Looking to learn how it's changed from App registrations (Legacy)? [Learn more](#)

## Do Note Client ID & Tenant ID

# Upload Certificate “\*.crt”



The screenshot shows the 'terraform\_admin | Certificates & secrets' page. The left sidebar contains a navigation menu with the following items: Overview, Quickstart, Integration assistant, Manage (a section header), Branding & properties, Authentication, Certificates & secrets (the active page), Token configuration, API permissions, Expose an API, App roles, and Owners. The main content area has a search bar and a 'Got feedback?' link. Below this, there is a paragraph explaining that credentials enable confidential applications to identify themselves to the authentication service scheme, and a recommendation to use a certificate instead of a client secret. A blue information banner states: 'Application registration certificates, secrets and federated credentials can be found in the tabs below'. There are three tabs: 'Certificates (0)' (selected), 'Client secrets (0)', and 'Federated credentials (0)'. Below the tabs, a paragraph states: 'Certificates can be used as secrets to prove the application's identity when requesting a token.' There is an 'Upload certificate' button with an upward arrow icon. Below this is a table with columns 'Thumbprint', 'Description', and 'Start'. The table is currently empty, with a message below it stating: 'No certificates have been added for this application.'

terraform\_admin | Certificates & secrets

Search << Got feedback?

Overview

Quickstart

Integration assistant

Manage

Branding & properties

Authentication

**Certificates & secrets**

Token configuration

API permissions

Expose an API

App roles

Owners

Credentials enable confidential applications to identify themselves to the authentication service scheme). For a higher level of assurance, we recommend using a certificate (instead of a client secret).

Application registration certificates, secrets and federated credentials can be found in the tabs below

**Certificates (0)** Client secrets (0) Federated credentials (0)

Certificates can be used as secrets to prove the application's identity when requesting a token.

Upload certificate


Thumbprint	Description	Start
No certificates have been added for this application.		

# Upload Certificate “\*.crt”

Upload certificate

×

Upload a certificate (public key) with one of the following file types: .cer, .pem, .crt \*





Description

Add


Cancel



# Upload Certificate “\*.crt”

 Application registration certificates, secrets and federated credentials can be found in the tabs below. 

**Certificates (1)**   Client secrets (0)   Federated credentials (0)


Certificates can be used as secrets to prove the application’s identity when requesting a token. Also can be referred to as public keys.


 Upload certificate


Thumbprint	Description	Start date	Expires	Certificate ID
EC6DB0E3D692FFFC6851FA7013B10C...	O=Default Company Ltd, L=Default C...	12/17/2022	12/17/2023	d9ca9d1a-e00f-4498- ...  


# Go to Subscriptions

## Azure services

  
[Create a resource](#)



  
Azure Active Directory

  
Subscriptions

  
Quickstart Center


## Resources

Recent Favorite



Name
 <a href="#">Pay-As-You-Go</a>
 <a href="#">NetworkWatcher_eastus</a>

[See all](#)


## Navigate






### Subscriptions

  
 [View](#)

#### Free training from Microsoft


[Create an Azure account](#)   
7 units · 39 min

#### Useful links

[Overview](#)   
[Get started](#)   
[Pricing](#) 








# Click on Subscription

**Subscriptions**  ...

Default Directory (sagarmehtaatgensoft.onmicrosoft.com)

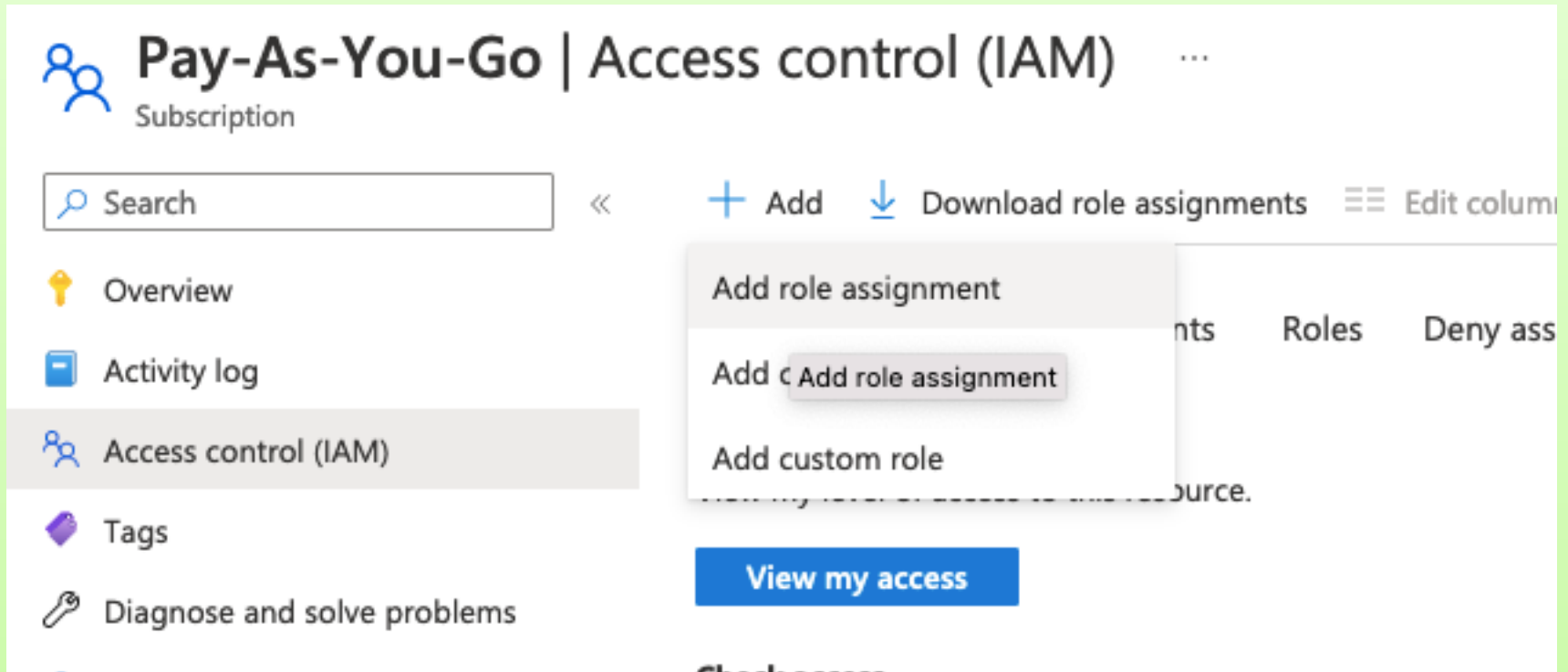
[+ Add](#) [Manage Policies](#) [View Requests](#) [View eligible subscriptions](#)

[Subscriptions == global filter](#) [My role == all](#) [Status == all](#) [Add filter](#)

Subscription name 	Subscription ID 	My role 	Current cost	Secure Score 
<a href="#">Pay-As-You-Go</a>	e7f340c5-89ac-403d-82ed-0571261dea04	Account admin		-

## Do Note Subscription ID

# Click IAM - Add role assignment




The screenshot displays the 'Pay-As-You-Go | Access control (IAM)' interface. On the left, a sidebar contains navigation links: Overview, Activity log, Access control (IAM) (which is highlighted), Tags, and Diagnose and solve problems. The main content area features a search bar, a 'Add' button with a plus icon, and a 'Download role assignments' button with a download icon. A dropdown menu is open from the 'Add' button, showing three options: 'Add role assignment' (highlighted), 'Add custom role', and 'Add role assignment' (repeated). Below the dropdown is a blue button labeled 'View my access'.

# Select appropriate role

## Add role assignment ...

 Got feedback?

Role   **Members**   [Review + assign](#)

A role definition is a collection of permissions. You can use the built-in roles or you can create your own custom roles. [Learn more](#) 

 Search by role name, description, or ID

Type : **All**

Category : **All**

Name ↑↓

Description ↑↓

Owner

Grants full access to manage all resources, including the ability to assign roles in Azure RBAC

Contributor

Grants full access to manage all resources, but does not allow you to assign roles in Azure

Reader

View all resources, but does not allow you to make any changes.

Access Review Operator Service Role


Lets you grant Access Review System app permissions to discover and revoke access as ne

AcrDelete

acr delete

# Select members

**Add role assignment** ...

 Got feedback?

Role

**Members**

Review + assign

**Selected role**

Owner

**Assign access to**

☒ User, group, or service principal

☐ Managed identity

**Members**

[+ Select members](#)


Name	Object ID
------	-----------

# Select members as App Name

Select members

Select ⓘ

terraform\_admin

 terraform\_admin

# Review + assign

[Role](#) [Members](#) [Review + assign](#)

**Selected role** Owner

**Assign access to** ☒ User, group, or service principal ☐ Managed identity

**Members** [+ Select members](#)

Name	Object ID	Type
terraform_admin	ddef2660-b611-482d-992c-dce7d9b059...	App


**Description**

Optional

[Review + assign](#) [Previous](#) [Next](#)

# Review + assign

Add role assignment

 Got feedback?

Role

Members

Review + assign

Role

Scope

Members

Description

Owner

/subscriptions/e7f340c5-89ac-403d-82ed-0571261dea04


Name	Object ID
terraform_admin	ddef2660-b611-482d-992c-dce7d9b05979

No description


Review + assign


Previous


# Generate a Client Certificate


**Pay-As-You-Go** ☆ ...  
Subscription


<<


 Overview


 Activity log

 Access control (IAM)


 Tags


 Diagnose and solve problems


 Security


 Events


**Cost Management**


 Cost analysis


 Cost alerts


 Budgets


 Cancel subscription


 Rename

 Change directory

 Switch Offer

 Transfer billing owner

 Tokenization is required to store card details due to a Reserve Bank of India directive. To tokenize your card detail add and verify your card. [Learn more](#)

 Essentials

Subscription ID

e7f340c5-89ac-403d-82ed-0571261dea04

Directory

Default Directory (sagarmehtaatgensoft.onmicrosoft.com)

My role

Account admin

Offer

Pay-As-You-Go

Offer ID

MS-AZR-0003P

Parent management group

---

Subscription name

[Pay-As-You-Go](#)

Current billing period

12/11/2022-1/10/2023

Currency

INR

Status

Active

Secure Score

[Not available](#)



# Installing Azure Client

- ▶ `sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc`
- ▶ `cat /etc/yum.repos.d/azure-cli.repo`

```
[azure-cli]
name=Azure CLI
baseurl=https://packages.microsoft.com/yumrepos/azure-cli
enabled=1
gpgcheck=1
gpgkey=https://packages.microsoft.com/keys/microsoft.asc
```
- ▶ `sudo yum install azure-cli`

# Installing Azure Client

- ▶ Get Locations

```
az account list-locations
```

- ▶ Get Image List

```
az vm image list
```

- ▶ Get SKU's

```
az vm list-skus --location eastus --zone --all --output table
```

# Exercise

- ▶ Create an Azure Linux(Ubuntu 22.04) Machine instance using terraform.
- ▶ Region: Central India

# State

- ▶ Terraform must store state about your managed infrastructure and configuration.
- ▶ This state is used by Terraform to map real world resources to your configuration, keep track of metadata, and to improve performance for large infrastructures.
- ▶ This state is stored by default in a local file named "terraform.tfstate".
- ▶ Terraform uses this local state to create plans and make changes to your infrastructure.
- ▶ Prior to any operation, Terraform does a refresh to update the state with the real infrastructure.

# Variables

- ▶ Variables serve as parameters for a Terraform module.

```
variable "key" {  
    type = "string"  
}
```

```
variable "zones" {  
    default = ["us-east-1a", "us-east-1b"]  
}
```

# Variables

```
variable "list" {  
  default = [ "1", "2", "3" ]  
}
```

```
output "a" {  
  value = var.list[0]  
}
```

```
variable "string" {  
  type = string  
  default = "Hello, this is sample string"  
}
```

```
output "b" {  
  value = var.string  
}
```

# Variables

```
variable "number" {  
  type = number  
  default = 100  
}
```

```
output "c" {  
  value = var.number  
}
```

```
variable "bool" {  
  type = bool  
  default = true  
}
```

```
output "d" {  
  value = var.bool  
}
```

# Variables

```
variable "map" {  
  type = map  
  default = {name = "Mabel", age = 52}  
}
```

```
output "e" {  
  value = var.map.name  
}
```



## Terraform Variable Files

### Production

```
env_id = "production"  
azure_location = "East Us"  
...
```

### Staging

```
env_id = "staging"  
azure_location = "East Us"  
...
```

### Testing

```
env_id = "testing"  
azure_location = "East Us"  
...
```



## Terraform Configuration Files

### Resource Group

```
name = "${var.env_id}-rg"  
location = "${var.azure_location}"  
...
```

### Storage Account

```
name = "${var.env_id}"  
location = "${var.azure_location}"  
...
```

### Storage Container

```
name = "${var.env_id}-sc"  
location = "${var.azure_location}"  
...
```



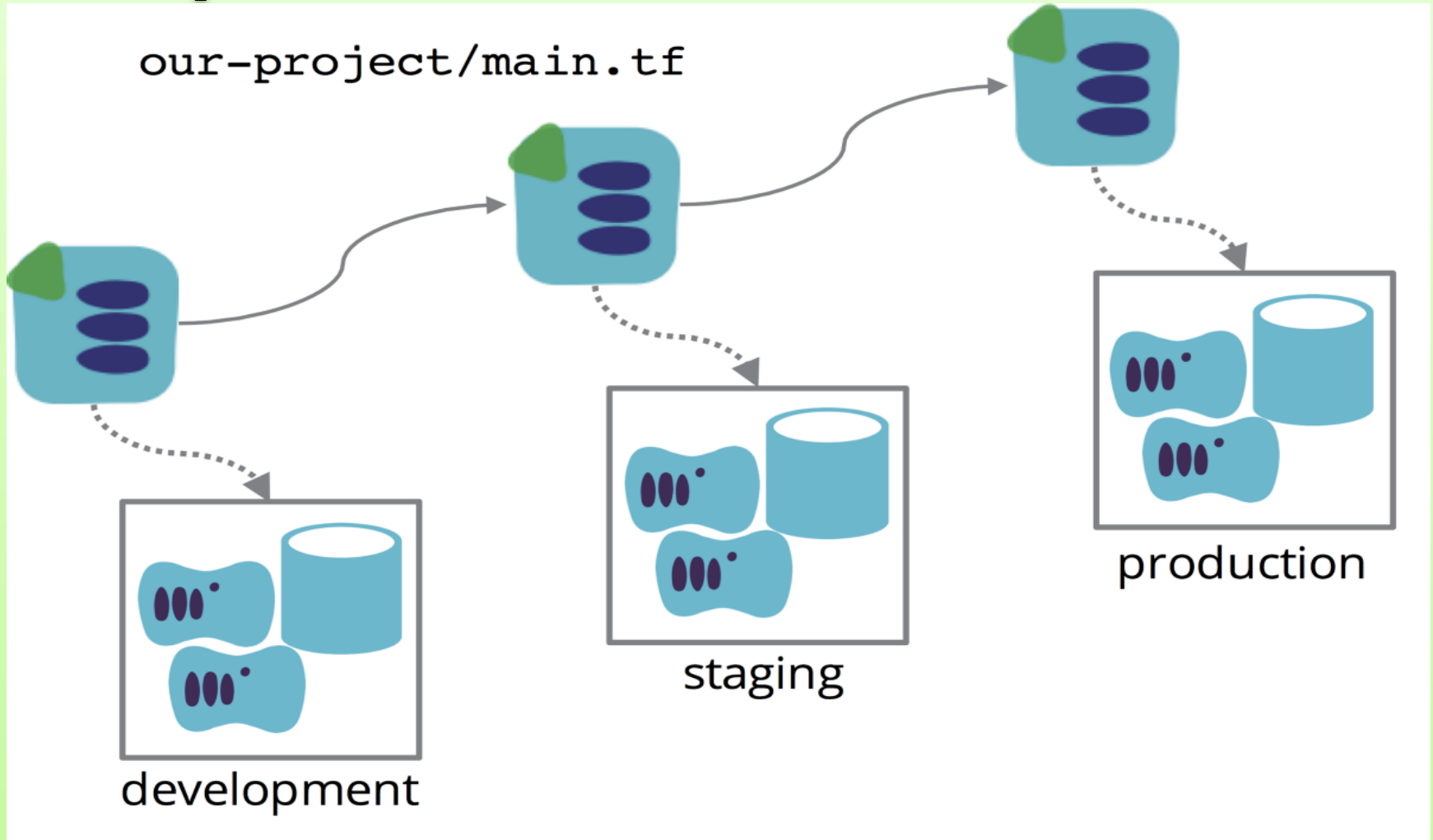
## Azure Resources and Terraform State

**Production  
Resources**

**Staging  
Resources**

**Testing  
Resources**

# Multiple Environments



# Exercise

- ▶ Create an Azure Linux(Ubuntu 22.04) Machine instance using terraform with variables defined in variable.tf.

# Output Configuration

- ▶ Outputs define values that will be highlighted to the user when Terraform applies, and can be queried easily using the output command.
- ▶ Terraform knows a lot about the infrastructure it manages. Most resources have attributes associated with them, and outputs are a way to easily extract and query that information.
- ▶ For Example,

```
output "Public_IP" {  
    value = azurerm_linux_virtual_machine.main.public_ip_address  
}
```

- ▶ This will output a string value corresponding to the public IP address of the Terraform-defined AWS instance named "main".

# Exercise

- ▶ Create an Azure Linux(Ubuntu 22.04) Machine instance using terraform with variables defined in vars.tf and should display Public IP/Private IP as output item.

# Provisioners

- ▶ Provisioners are used to execute scripts on a local or remote machine as part of resource creation or destruction.
- ▶ Provisioners can be used to bootstrap a resource, cleanup before destroy, run configuration management, etc.
- ▶ Provisioners are added directly to any resource:

```
resource "azurerm_linux_virtual_machine" "main" {  
    # ...  
    provisioner "local-exec" {  
        command = "echo ${self.private_ip_address} > file.txt"  
    }  
}
```

# Provisioners

- ▶ Below are type:
  - Connection
  - File
  - Local-exec
  - Remote-exec

# Provisioners

```
resource "azurerm_linux_virtual_machine" "main" {  
  ###  
  
  provisioner "file" {  
    source = "test.sh"  
    destination = "/tmp/test.sh"  
    connection {  
      type = "ssh"  
      user = "USERNAME"  
      private_key = "${file("~/.ssh/id_rsa")}"  
      host = self.public_ip  
    }  
  }  
}
```



# Provisioners

```
provisioner "remote-exec" {  
  inline = [  
    "chmod +x /tmp/test.sh",  
    "/tmp/test.sh",  
  ]  
  connection {  
    type = "ssh"  
    user = "USERNAME"  
    private_key = "${file("~/.ssh/id_rsa")}"  
    host = self.public_ip  
  }  
}
```

# Exercise

- ▶ Create an Azure Linux(Ubuntu 22.04) Machine instance using terraform with variables defined in vars.tf and should display Public IP/Private IP as output item and do ssh login(using keys) to instance.

# Exercise

- ▶ Create an Azure Linux(Ubuntu 22.04) Machine instance using terraform with variables defined in vars.tf and should display Public IP/Private IP as output item and should display Welcome Message as “**Welcome to Terraform World!**” on ssh login to instance.

# Session: 5

## Conditional Statements

# Loops

- ▶ To accomplish for-loop in terraform, “count” was introduced.
- ▶ For Example,

```
resource "azurerm_linux_virtual_machine" "main" {  
  count = 3  
  name           = "myvm-vm-${count.index}"  
  size           = "Standard_F2"  
  admin_username = "adminuser"  
  admin_password = "P@ssw0rd1234!"  
  disable_password_authentication = false  
  
  source_image_reference {  
    publisher = "Canonical"  
    offer      = "UbuntuServer"  
    sku        = "18.04-LTS"  
    version    = "latest"  
  }  
  
  os_disk {  
    storage_account_type = "Standard_LRS"  
    caching               = "ReadWrite"  
  }  
}
```

# Exercise

- ▶ Create a Three Azure Linux(Ubuntu 22.04) Machine instance using terraform with variables defined in vars.tf using loops.

# For Each

## ► For Example,

```
resource "azurerm_linux_virtual_machine" "main" {
  for_each = toset( ["blr", "mum", "hyd"] )
  name      = "myvm-vm-${each.key}"
  size      = "Standard_F2"
  admin_username = "adminuser"
  admin_password = "P@ssw0rd1234!"
  disable_password_authentication = false

  source_image_reference {
    publisher = "Canonical"
    offer     = "UbuntuServer"
    sku       = "18.04-LTS"
    version   = "latest"
  }

  os_disk {
    storage_account_type = "Standard_LRS"
    caching               = "ReadWrite"
  }
}
```

# Exercise

- ▶ Create a Two Azure Linux(Ubuntu 22.04) Machine instance using terraform with variables defined in vars.tf using for\_each with sizes “Standard\_F2” and “Standard\_F1”.



# If-else

- ▶ If-else can be accomplished in terraform, using “count”.
- ▶ For Example,  
variable "create\_instance" {  
 description = "Create an instance if set to True"  
 default = true  
}

# If-else

```
resource "azurerm_linux_virtual_machine" "main" {  
  count = "${var.create_instance == true ? 1 : 0}"  
  name   = "myvm-vm-${count.index}"  
  size   = "Standard_F2"  
  admin_username = "adminuser"  
  admin_password = "P@ssw0rd1234!"  
  disable_password_authentication = false  
  
  source_image_reference {  
    publisher = "Canonical"  
    offer      = "UbuntuServer"  
    sku        = "22.04-LTS"  
    version    = "latest"  
  }  
  
  os_disk {  
    storage_account_type = "Standard_LRS"  
    caching               = "ReadWrite"  
  }  
}
```

# Exercise

- ▶ Create an Azure Linux(Ubuntu 22.04) Machine instance using variable values.

# Session: 6

## Terraform Templates

# Templates

- ▶ The template provider exposes data sources to use templates to generate strings.
- ▶ Templates are helpful in scenario of managing multiple environments.

# Templates

- For Example,

```
data "template_file" "user_data" {  
  template = "${file("${path.module}/user_data")}"  
}
```

# Templates

## ► For Example,

```
resource "azurerm_linux_virtual_machine" "main" {
  name = "myvm-vm-${count.index}"
  size = "${trimspace(data.template_file.user_data.rendered)}"
  admin_username      = "adminuser"
  admin_password      = "P@ssw0rd1234!"
  disable_password_authentication = false

  source_image_reference {
    publisher = "Canonical"
    offer     = "UbuntuServer"
    sku       = "18.04-LTS"
    version   = "latest"
  }

  os_disk {
    storage_account_type = "Standard_LRS"
    caching               = "ReadWrite"
  }
}
```

# Exercise

- ▶ Create Two Azure Machines(Ubuntu 22.04) instance using terraform variables defined in variable.tf and having different sizes based on environment prod and dev.



# Terraform JSONDECODE

- Create a JSON file with **mydata.json**

```
{
  "project": [
    {
      "user_name": "devuser",
      "env": "dev"
    },
    {
      "user_name": "prduser",
      "env": "prd"
    }
  ]
}
```

# Terraform JSONDECODE

## ► Reading JSON

```
locals {  
    # get json  
    data = jsondecode(file("${path.module}/mydata.json"))  
    anothervar = "abc"  
}  
  
output "printenv" {  
    value = local.data  
}  
  
output "printdevuser" {  
    value = local.data.project[0].user_name  
}  
  
output "printprdenv" {  
    value = local.data.project[1].env  
}  
  
output "anothervarout" {  
    value = local.anothervar  
}  
  
output "names" {  
    value = [  
        for i in local.data.project:  
            "${i.env} ${i.user_name}"  
    ]  
}
```

# Modules

- ▶ A Terraform module is very simple: any set of Terraform configuration files in a folder is a module.
- ▶ Here are some of the ways that modules help solve the problems:
  - ▶ Organise configuration
  - ▶ Encapsulate configuration
  - ▶ Re-use configuration
  - ▶ Provide consistency

# Modules

- ▶ Create directory say “terraform\_modules”.
- ▶ Create Sub-directory inside it “modules/services/webserver-cluster”.
- ▶ Create below structure:

```
# tree modules/services/webserver-cluster/  
modules/services/webserver-cluster/
```

```
|— main.tf  
|— outputs.tf  
|— vars.tf
```

```
0 directories, 3 files
```

# Modules - Calling

- ▶ Calling Terraform modules:

```
[terraform_modules]# cat main.tf
module "webserver_cluster" {
  source = "../modules/services/webserver-cluster"
  size = "Standrad_F1"
}
```

# Exercise

- ▶ Create Two Azure Machines using terraform variables defined in variable.tf and having different instance\_types based on environment prod(Standard\_F1) and dev(Standard\_F2) using modules.

# Remote State/Backend

- ▶ A backend defines where Terraform stores its state data files.
- ▶ Below are different types of Backends:
  - ▶ local
  - ▶ artifactory
  - ▶ Azurerm
  - ▶ S3
  - ▶ Kubernetes
  - ▶ http
  - ▶ Swift

# Remote State/Backend

```
terraform {  
  backend "local" {  
    path = "relative/path/to/terraform.tfstate"  
  }  
}
```

```
data "terraform_remote_state" "foo" {  
  backend = "local"
```

```
  config = {  
    path = "${path.module}/../terraform.tfstate"  
  }  
}
```



# Terraform Import

- ▶ Terraform 'import' feature lets you manage already created resource through Terraform.
- ▶ Below are steps:
  - ▶ Create "main.tf" with provider and import

```
[root@minion aws_import]# cat main.tf
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "3.97.1"
    }
  }
}

provider "azurerm" {
  features {}
  client_id                = "37a0369c-41ef-43bb-bb6c-27ea9f246ce0"
  client_certificate_path  = "/root/mycert.pfx"
  tenant_id                = "df8af829-df8a-41fe-8679-3e0a616165c5"
  subscription_id          = "bbbd26d1-25ec-4756-8dd7-085194738e44"
}

import {
  to = azurerm_linux_virtual_machine.sample
  id = "/subscriptions/bbbd26d1-25ec-4756-8dd7-085194738e44/resourceGroups/mehta-rg/providers/Microsoft.Compute/virtualMachines/mehta-vm"
}
```



# Terraform Import

- ▶ Initialise the plugins

```
terraform init
```

- ▶ Generate new “TF” file

```
terraform plan -generate-config-out=generated.tf
```

- ▶ Edit generated.tf as per Conflicting Arguments

- ▶ Generate new “TFSTATE” file

```
terraform apply
```

# Terraform Get Data - http

- Terraform can download data from web API as shown below:

```
data "http" "example" {  
  url = "https://www.atgensoft.com"  
  request_headers = {  
    Accept = "application/json"  
  }  
}
```

# Terraform Random

- ▶ The "random" provider allows the use of randomness within Terraform configurations.

- ▶ Below is example:

```
resource "random_integer" "num" {  
  min = 1  
  max = 50000  
}
```

- ▶ Task: Create a random string of length 16.

# Terraform Local

- ▶ The "local" provider allows to manage local files.
- ▶ Below is example:

```
resource "local_file" "foo" {  
  content = "foo"  
  filename = "${path.module}/foo.bar"  
}
```

# Resource time\_sleep & dependency

- ▶ The "time" resource allows sleep and depends\_on for dependency between resources.
- ▶ Below is example:

```
resource "null_resource" "previous" {}  
resource "time_sleep" "wait_30_seconds" {  
    depends_on = [null_resource.previous]  
    create_duration = "30s"  
}  
resource "null_resource" "next" {  
    depends_on = [time_sleep.wait_30_seconds]  
}
```

# Terraform Workspaces

- ▶ Terraform starts with a single workspace named "default". This workspace is special both because it is the default and also because it cannot ever be deleted.

```
terraform workspace new dev  
terraform workspace select dev  
terraform workspace show
```

# Exercise

- ▶ Create an Azure Machine(Ubuntu 22.04) instance using terraform variables defined in variable.tf, having output Url/IP and Port and running Web Server.
- ▶ Steps Apache Web Server Install:
  - ▶ `sudo apt-get update -y`
  - ▶ `sudo apt-get install apache2 -y`
  - ▶ `sudo systemctl restart apache2`
  - ▶ `sudo systemctl enable apache2`



# Exercise

- ▶ Create an AKS Cluster and save config file using Terraform.

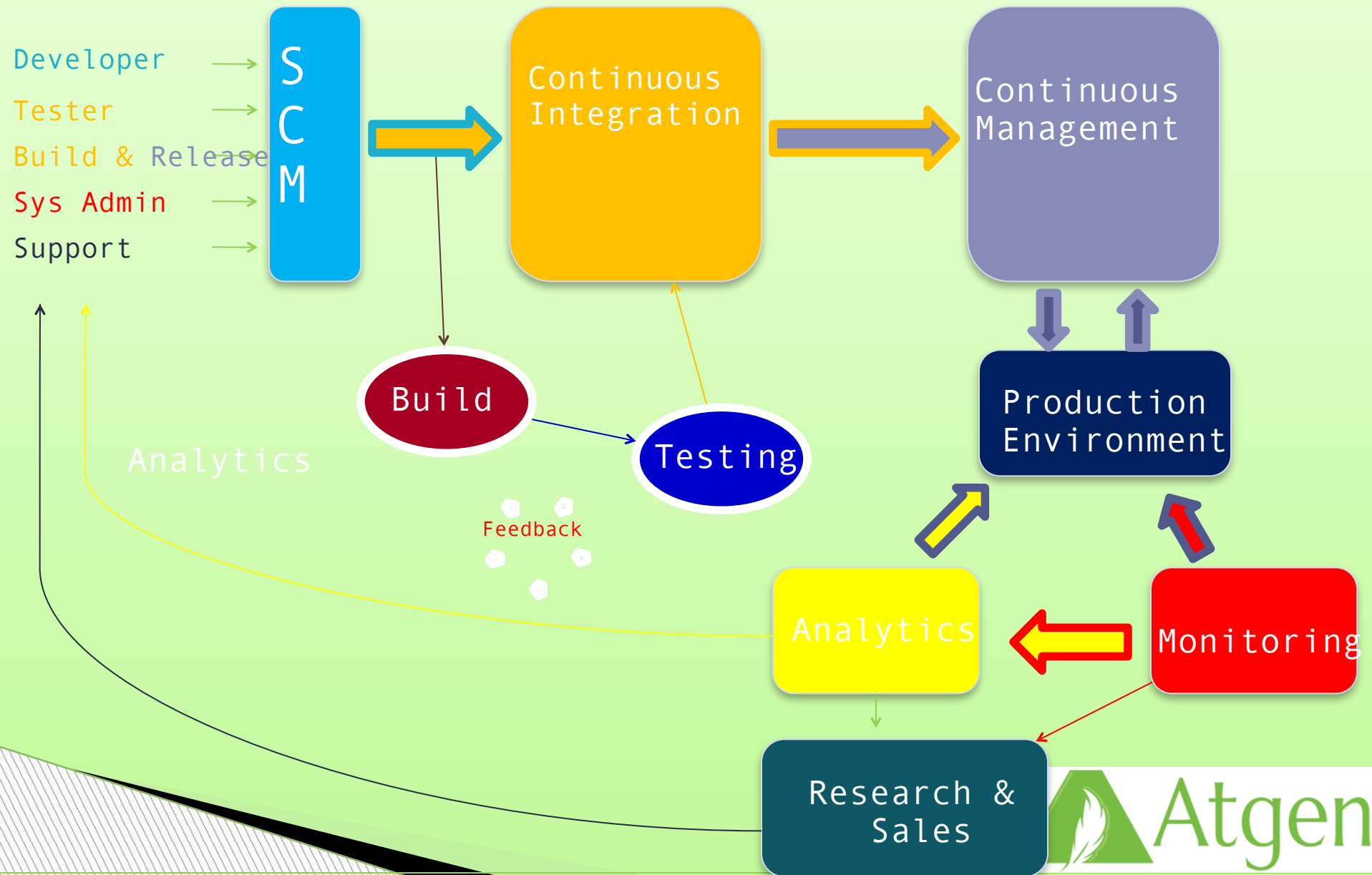
# Exercise

- ▶ Deploy an application Python Web App(smehta26/pyweb:1.0, port:5000) and Redis DB(redis:alpine) using Deployment/Service on AKS cluster.

# Session: 7

CI/CD - Terraform

# What is DevOps?



# Jenkins for CI

- ▶ Jenkins – open source continuous integration server
- ▶ Jenkins (<http://jenkins-ci.org/>) is
  - ▶ Easy to install
  - ▶ Easy to use
  - ▶ Multi-technology
  - ▶ Multi-platform
  - ▶ Widely used
  - ▶ Extensible
  - ▶ Free

# Jenkins User Interface

Actions

Nodes

Jobs

The screenshot shows the Jenkins dashboard in a web browser. The browser address bar displays `http://fc-demo1:8080/`. The Jenkins logo is at the top left, and a search bar is at the top right. Below the logo, there are links for [New Job](#), [Manage Jenkins](#), [People](#), and [Build History](#). The **Build Queue** section shows "No builds in the queue." The **Build Executor Status** section shows two executors, both in an "Idle" state.

The main section displays a table of jobs. The table has columns for **S** (Success), **W** (Warning), **Job**, **Last Success**, **Last Failure**, and **Last Duration**. The jobs listed are **C-Demo**, **Java-Demo**, and **Python-Demo**.

S	W	Job	Last Success	Last Failure	Last Duration
		<a href="#">C-Demo</a>	6 min 34 sec (#1)	N/A	2.6 sec
		<a href="#">Java-Demo</a>	6 min 40 sec (#6)	N/A	1 sec
		<a href="#">Python-Demo</a>	2 min 43 sec (#2)	N/A	1.5 sec

Below the table, there is a legend for RSS feeds: [Legend](#), [RSS for all](#), [RSS for failures](#), and [RSS for just latest builds](#). The footer of the page indicates it was generated on April 6, 2011, at 10:08:52 PM, and the Jenkins version is 1.405.

# More Power - Jenkins Plugins

- ▶ Jenkins has plugins for various operations like
  - ▶ Software configuration management
  - ▶ Builders
  - ▶ Test Frameworks
  - ▶ Virtual Machine Controllers
  - ▶ Notifiers
  - ▶ Static Analyzers

# Jenkins - Integration

- ▶ Jenkins help your release to be
  - ▶ Faster
  - ▶ Safer
  - ▶ Easier
  - ▶ Smarter



# Exercise:

- ▶ Jenkins help your release to be
  - ▶ Faster
  - ▶ Safer
  - ▶ Easier
  - ▶ Smarter

# Exercise:

- ▶ Link:

- ▶ <https://atgensoft.com/training/JenkinsSetup.zip>
- ▶ <https://atgensoft.com/training/cgi-terraform-28032024-ci.pdf>

# Contact us

- ▶ Contact @ <http://www.atgensoft.com/>
- ▶ Linkedin: @atgensoft
- ▶ FaceBook: @atgensoft
- ▶ YouTube: @atgensoft
- ▶ Email: [SAGAR.MEHTA@ATGENSOFT.COM](mailto:SAGAR.MEHTA@ATGENSOFT.COM)
- ▶ Doc: <https://atgensoft.com/cgi-terraform-28032024>

Thank You !!

