# Comprehensive Guide to Infrastructure as Code (IaC) with Terraform

## Introduction

Infrastructure as Code (IaC) is a paradigm shift in how organizations manage and provision their IT infrastructure. Traditionally, infrastructure management involved manual configuration, which was not only time-consuming but also prone to errors and inconsistencies. With the advent of cloud computing and the rise of DevOps practices, the need for automated, scalable, and consistent infrastructure management became paramount. IaC emerged as a solution to these challenges by enabling infrastructure to be defined, deployed, and managed using code.

IaC allows organizations to automate the process of provisioning and managing infrastructure by using machine-readable files that describe the desired state of their infrastructure. These files can be versioned, shared, and reused, making it easier to maintain consistency across different environments (e.g., development, staging, production). By treating infrastructure as code, organizations can apply the same software development practices—such as version control, testing, and continuous integration—to their infrastructure.

Terraform, an open-source tool developed by HashiCorp, is one of the most widely used platforms for implementing IaC. Terraform allows users to define infrastructure in a declarative configuration language called HashiCorp Configuration Language (HCL). With Terraform, users can create, manage, and update infrastructure resources across various cloud providers, including AWS, Azure, Google Cloud, and on-premises environments. Terraform's versatility, combined with its powerful features, makes it a cornerstone of modern infrastructure management.

## The Evolution of Infrastructure Management

Before diving deeper into Terraform, it’s essential to understand the evolution of infrastructure management. Initially, infrastructure was managed manually, with system administrators configuring each server, network, and application individually. This approach was labor-intensive and led to "configuration drift," where different environments would become inconsistent over time.

With the rise of virtualization, tools like VMware and VirtualBox allowed for the creation of virtual machines (VMs), reducing the hardware footprint and enabling more flexible resource management. However, managing these VMs still required manual intervention, leading to similar challenges as physical infrastructure.

The advent of cloud computing brought about a new era in infrastructure management. Cloud providers like AWS, Azure, and Google Cloud offered on-demand resources, but managing these resources at scale required a new approach. This is where IaC came into play. By automating the provisioning and configuration of cloud resources, IaC tools like Terraform provided a way to manage infrastructure efficiently and consistently.

## Why Terraform?

Terraform stands out among IaC tools due to its broad support for cloud providers, its declarative approach, and its ability to manage the entire infrastructure lifecycle. Here are some key reasons why Terraform is widely adopted:

- **Provider Agnostic:** Terraform's architecture is provider-agnostic, meaning it can manage resources across different cloud providers and even on-premises environments. This flexibility allows organizations to use a single tool to manage hybrid or multi-cloud environments.
- **Declarative Language:** Terraform uses a declarative language (HCL) to define infrastructure. In a declarative approach, users describe the desired state of their infrastructure, and Terraform handles the details of how to achieve that state. This simplifies the process of managing infrastructure and reduces the risk of errors.
- **State Management:** Terraform keeps track of the state of the infrastructure in a state file. This file is used to determine the difference between the current state of the infrastructure and the desired state described in the configuration files. By storing the state file remotely, teams can collaborate on infrastructure management without conflicting changes.
- **Plan and Apply Workflow:** Terraform's workflow includes a "plan" phase, where users can preview the changes that will be made to the infrastructure before applying them. This feature provides a safety net, allowing users to review and confirm changes before they are executed.
- **Modularity and Reusability:** Terraform allows users to create reusable modules that encapsulate specific infrastructure components or services. These modules can be shared across projects, making it easier to manage complex infrastructure and promote best practices.
- **Extensibility with Providers:** Terraform's architecture is extensible through providers, which are plugins that enable Terraform to interact with APIs of cloud providers, SaaS applications, and other services. With a rich ecosystem of providers, Terraform can manage a wide range of resources, from virtual machines to DNS records, databases, and beyond.

## Benefits of Terraform

In addition to the key features highlighted above, Terraform offers several benefits that make it a preferred tool for IaC: 1. **Automation and Efficiency:**

   - Terraform automates the process of provisioning, configuring, and managing infrastructure. This automation reduces manual intervention, accelerates deployment times, and minimizes the risk of human errors. Automation is especially critical in dynamic environments where infrastructure needs to be scaled or modified frequently.

2. **Consistency and Predictability:**
   - One of the main advantages of IaC is the ability to maintain consistency across different environments. Terraform ensures that infrastructure is provisioned in a consistent and predictable manner, reducing configuration drift. This consistency is crucial for ensuring that development, testing, and production

    environments are identical, which in turn improves the reliability of applications.

3. **Scalability and Flexibility:**
    - As organizations grow, so does the complexity of their infrastructure. Terraform's modular design and support for multiple providers make it easy to scale infrastructure as needed. Whether you're managing a small set of resources or a large-scale distributed system, Terraform can handle the complexity, allowing you to scale up or down with minimal effort.
4. **Collaboration and Version Control:**
    - By treating infrastructure as code, Terraform enables teams to collaborate more effectively. Infrastructure configurations can be stored in version control systems like Git, allowing multiple team members to work on the same infrastructure, track changes, and roll back to previous versions if necessary. This versioning capability is essential for maintaining a clear history of changes and ensuring that deployments are repeatable.
5. **Cost Management:**
    - Terraform's ability to automate infrastructure provisioning can lead to cost savings. By automating the teardown of resources when they are no longer needed (e.g., through the `terraform destroy` command), Terraform helps prevent unnecessary spending on idle resources. Additionally, by using Terraform's plan phase, teams can estimate the cost of infrastructure changes before applying them, enabling better budgeting and cost management.
6. **Disaster Recovery and Backup:**
    - Terraform's state management and version control capabilities make it easier to implement disaster recovery strategies. In the event of a failure, Terraform configurations can be used to quickly recreate the infrastructure in a different environment. Additionally, by storing state files in remote backends, organizations can ensure that they have up-to-date backups of their infrastructure state, which is critical for disaster recovery.
7. **Security and Compliance:**
    - Terraform helps enforce security best practices by allowing organizations to codify their security policies into the infrastructure configurations. For example, Terraform can be used to ensure that only approved AMIs are used, that encryption is enabled for sensitive data, and that security groups follow the principle of least privilege. Additionally, Terraform's ability to automate compliance checks and integrate with security tools makes it easier to maintain compliance with industry regulations.
8. **Integration with CI/CD Pipelines:**
    - Terraform can be integrated into CI/CD pipelines, enabling continuous delivery of infrastructure changes. By automating the deployment process, teams can ensure that infrastructure changes are tested, validated, and deployed in a consistent manner. This integration with CI/CD pipelines also facilitates rapid iteration and feedback loops, allowing teams to respond to changing requirements more quickly.

## Implementation Steps

Implementing Terraform in an organization involves several steps, from setting up the environment to writing configuration files, deploying infrastructure, and managing it over time.

1. **Set Up Terraform** o **Install Terraform:**
    - Terraform can be installed on various operating systems, including macOS, Windows, and Linux. The installation process involves downloading the appropriate binary from the Terraform website and placing it in a directory included in your system's PATH. For macOS users, Terraform can also be installed via Homebrew by running `brew install terraform`.
    - Once installed, verify the installation by running `terraform -version` in the terminal. This command should return the installed version of Terraform, confirming that it is correctly set up. o **Initialize the Terraform Workspace:**
    - Before you can start using Terraform, you need to initialize a working directory where your configuration files will reside. Navigate to the directory where you plan to store your Terraform configurations and run `terraform init`. This command initializes the directory, downloads the necessary provider plugins, and sets up the backend for state management.
    - During initialization, Terraform may ask for additional information, such as the authentication credentials for the cloud provider or the location of the state file. This information can be provided via environment variables, configuration files, or interactive prompts.
2. **Create Terraform Configuration Files** o **Define Providers:**  In your project directory, create a file named `main.tf`.
    - Specify the cloud provider (e.g., Azure) and the region you wish to deploy to.
    - Example of configuring Azure as the provider:

```
hcl Copy code
provider "azurerm" {
features {} }
```

o **Define Resources:**
- Use the resource block to define infrastructure components such as virtual machines, databases, and network configurations.
- Example of creating a Virtual Machine on Azure:

```
hcl Copy
code
resource "azurerm_resource_group" "example" {
name     = "example-resources"   location =
"West Europe"
}
```

```
resource "azurerm_virtual_network" "example" {
name                       =  "example-network"
address_space      = ["10.0.0.0/16"]   location
=
azurerm_resource_group.example.location
  resource_group_name =
azurerm_resource_group.example.name
}

resource "azurerm_subnet" "example" {
name                 = "example-subnet"
resource_group_name  =
azurerm_resource_group.example.name
virtual_network_name =
azurerm_virtual_network.example.name
address_prefixes     = ["10.0.2.0/24"]
}

resource "azurerm_network_interface" "example" {
name                = "example-nic"   location
= azurerm_resource_group.example.location
resource_group_name =
azurerm_resource_group.example.name

  ip_configuration {
    name                          = "example"
subnet_id                     =
azurerm_subnet.example.id
    private_ip_address_allocation = "Dynamic"
  }
}

resource "azurerm_linux_virtual_machine" "example" {
name             = "example-vm"   resource_group_name
=  azurerm_resource_group.example.name      location
=  azurerm_resource_group.example.location      size
=  "Standard_DS1_v2"      admin_username           =
"adminuser"   network_interface_ids = [
    azurerm_network_interface.example.id,
  ]   admin_ssh_key {
username   = "adminuser"
    public_key = file("~/.ssh/id_rsa.pub")
  }
  source_image_id = data.azurerm_image.example.id }
```

- **Define Outputs:**
   - Outputs are used to display information about the infrastructure after deployment. This can include resource IDs, IP addresses, or other important details.
   - Example of defining an output:

```
hcl Copy code output
"public_ip" {
value =
azurerm_linux_virtual_machine.example.public_ip_address
}
```

3. **Manage Infrastructure** ○ **Run Terraform Commands:**
    -  **Plan:** Use the `terraform plan` command to preview changes before applying them. This command generates an execution plan that shows the actions Terraform will take to achieve the desired state defined in your configuration files.
    -  **Apply:** Run `terraform apply` to execute the changes defined in the execution plan. Terraform will create, update, or delete resources as necessary to match the desired state.
    -  **Destroy:** Use `terraform destroy` to remove all resources defined in your configuration files. This command is useful for cleaning up resources that are no longer needed. ○ **Manage State Files:**
    -  Terraform uses state files to keep track of the current state of your infrastructure. By default, state files are stored locally in the project directory. However, for team collaboration and remote state management, it is recommended to use remote backends, such as AWS S3 or Azure Storage.
    -  To configure a remote backend, update the `backend` block in your `main.tf` file with the appropriate backend configuration. For example, to use an S3 bucket as a remote backend:

```
hcl Copy code
terraform {
backend "s3" {
    bucket         = "my-terraform-state"
key               =  "terraform.tfstate"
region         = "us-east-1"
  }
}
```

    ○ **Version Control:**
    -  Store Terraform configuration files in version control systems like Git to track changes and collaborate with team members. Commit your configurations and use branching strategies to manage different environments (e.g., development, staging, production).
4. **Best Practices** ○ **Organize Configurations:**
     Organize Terraform configurations into logical modules to promote reusability and maintainability. Group related resources into separate module files and use variables to parameterize module inputs. ○ **Use Modules:**
     Create and use Terraform modules to encapsulate and reuse infrastructure components. Modules allow you to define a set of resources that can be instantiated multiple times with different configurations. ○
    **Implement Security Controls:**
     Follow security best practices by defining security policies in your configurations. Use Terraform's built-in features to manage sensitive information securely, such as using environment variables for credentials and encrypting state files.
    ○ **Automate Testing and Validation:**

   Integrate Terraform configurations into CI/CD pipelines to automate testing and validation. Use tools like Terraform validate and linting tools to ensure configurations meet quality standards before applying changes.

- **Monitor and Audit Changes:**
   -  Implement monitoring and auditing practices to track changes made to your infrastructure. Use logging and alerting mechanisms to detect and respond to unexpected changes or issues.
- **Document Your Infrastructure:**
   -  Maintain documentation for your Terraform configurations and infrastructure setup. Document the purpose of each module, the variables used, and any dependencies or constraints. This documentation helps with onboarding new team members and maintaining the infrastructure over time.

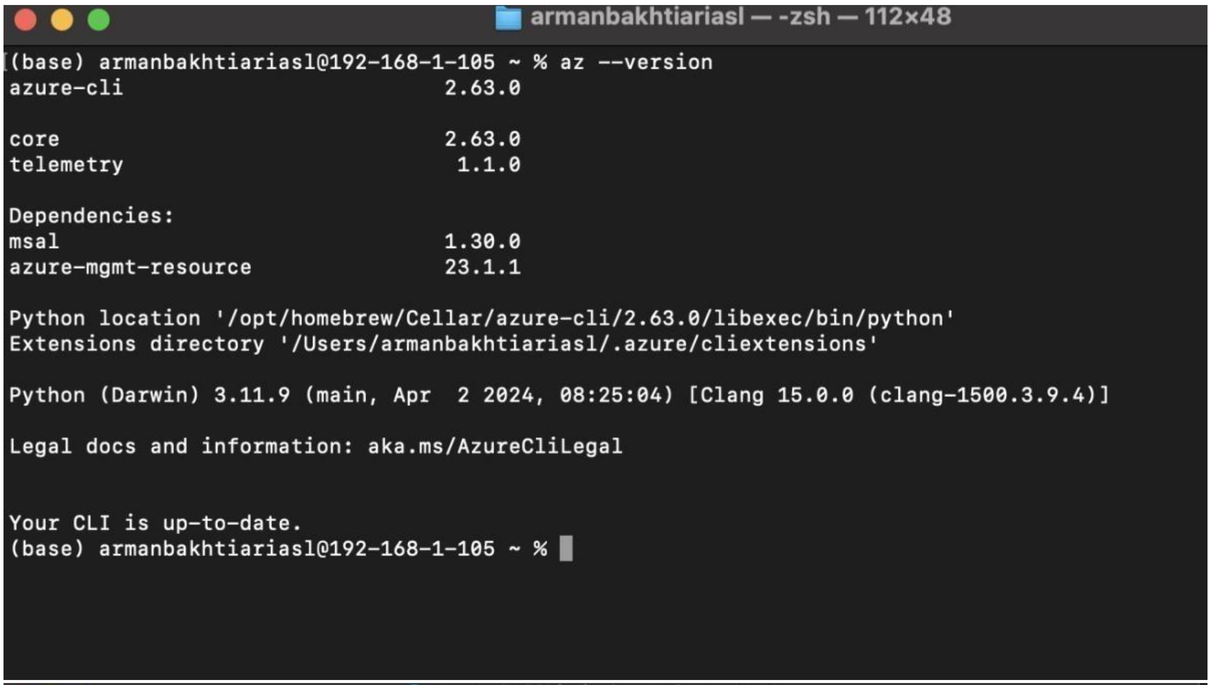
## Conclusion

Terraform is a powerful tool that revolutionizes infrastructure management by treating infrastructure as code. Its declarative approach, provider-agnostic architecture, and robust features make it a valuable asset for organizations seeking to automate, manage, and scale their infrastructure efficiently. By implementing Terraform and adhering to best practices, organizations can achieve consistency, agility, and cost-effectiveness in their infrastructure management processes.

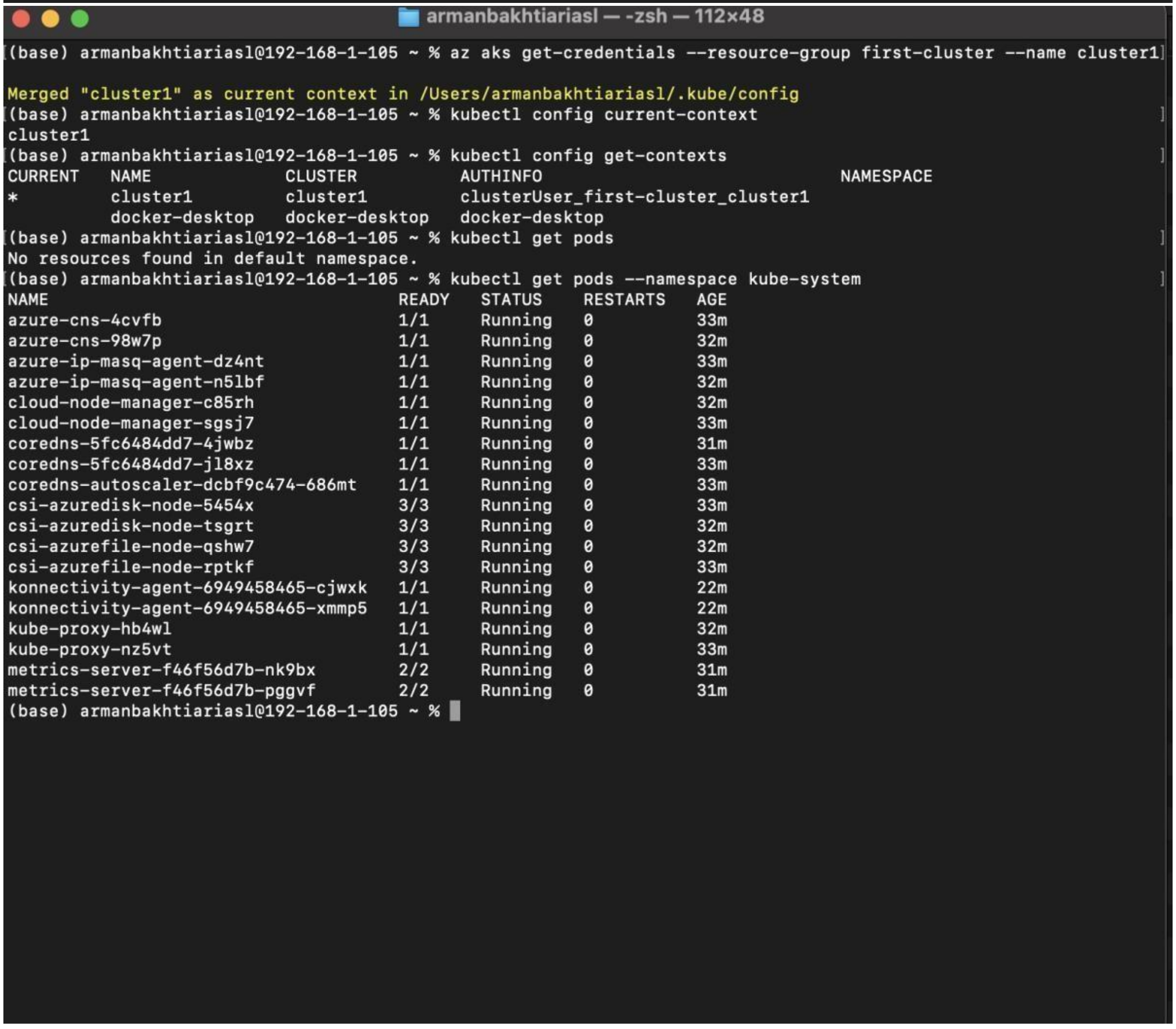
This document is provided by Arman Bakhtiariasl and THAMASHA GALAHAHENA MUDIYANSELAGE

This work is 70% by Arman Bakhtiariasl and 30% by THAMASHA GALAHAHENA MUDIYANSELAGE

Implementing Azure on my laptop;

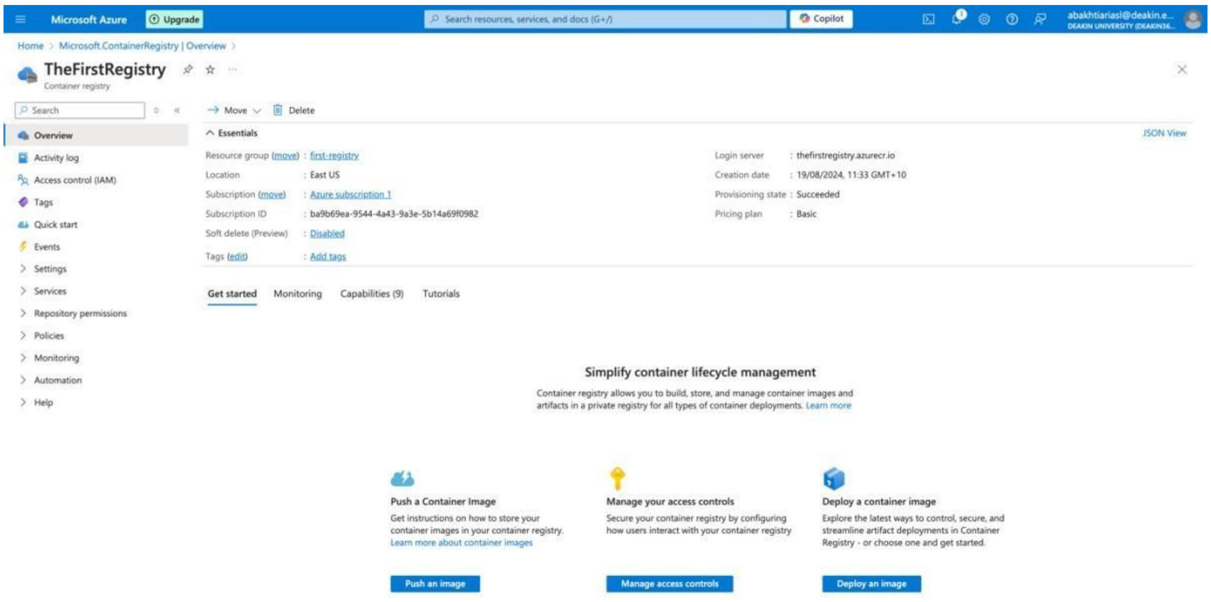


```
(base) armanbakhtiariasl@192-168-1-105 ~ % az --version
azure-cli                         2.63.0

core                              2.63.0
telemetry                          1.1.0

Dependencies:
msal                              1.30.0
azure-mgmt-resource               23.1.1

Python location '/opt/homebrew/Cellar/azure-cli/2.63.0/libexec/bin/python'
Extensions directory '/Users/armanbakhtiariasl/.azure/cliextensions'

Python (Darwin) 3.11.9 (main, Apr  2 2024, 08:25:04) [Clang 15.0.0 (clang-1500.3.9.4)]

Legal docs and information: aka.ms/AzureCliLegal


Your CLI is up-to-date.
(base) armanbakhtiariasl@192-168-1-105 ~ %
```



```
(base) armanbakhtiariasl@192-168-1-105 ~ % az aks get-credentials --resource-group first-cluster --name cluster1

Merged "cluster1" as current context in /Users/armanbakhtiariasl/.kube/config
(base) armanbakhtiariasl@192-168-1-105 ~ % kubectl config current-context
cluster1
(base) armanbakhtiariasl@192-168-1-105 ~ % kubectl config get-contexts
CURRENT   NAME             CLUSTER          AUTHINFO                                NAMESPACE
*         cluster1         cluster1         clusterUser_first-cluster_cluster1
          docker-desktop   docker-desktop   docker-desktop
(base) armanbakhtiariasl@192-168-1-105 ~ % kubectl get pods
No resources found in default namespace.
(base) armanbakhtiariasl@192-168-1-105 ~ % kubectl get pods --namespace kube-system
NAME                                  READY   STATUS    RESTARTS   AGE
azure-cns-4cvfb                       1/1     Running   0          33m
azure-cns-98w7p                       1/1     Running   0          32m
azure-ip-masq-agent-dz4nt             1/1     Running   0          33m
azure-ip-masq-agent-n5lbf             1/1     Running   0          32m
cloud-node-manager-c85rh              1/1     Running   0          32m
cloud-node-manager-sgsj7              1/1     Running   0          33m
coredns-5fc6484dd7-4jwbz              1/1     Running   0          31m
coredns-5fc6484dd7-jl8xz              1/1     Running   0          33m
coredns-autoscaler-dcbf9c474-686mt    1/1     Running   0          33m
csi-azuredisk-node-5454x              3/3     Running   0          33m
csi-azuredisk-node-tsgrt              3/3     Running   0          32m
csi-azurefile-node-qshw7              3/3     Running   0          32m
csi-azurefile-node-rptkf              3/3     Running   0          33m
konnectivity-agent-6949458465-cjwxk   1/1     Running   0          22m
konnectivity-agent-6949458465-xmmp5   1/1     Running   0          22m
kube-proxy-hb4wl                      1/1     Running   0          32m
kube-proxy-nz5vt                      1/1     Running   0          33m
metrics-server-f46f56d7b-nk9bx        2/2     Running   0          31m
metrics-server-f46f56d7b-pggvf        2/2     Running   0          31m
(base) armanbakhtiariasl@192-168-1-105 ~ %
```

Creating group registry on Azure:



TheFirstRegistry
Container registry

Essentials

Resource group (move) : first-registry
Location : East US
Subscription (move) : Azure subscription 1
Subscription ID : ba9b69ea-9544-4a43-9a3e-5b14a69f0982
Soft delete (Preview) : Disabled
Tags (edit) : Add tags

Login server : thefirstregistry.azurecr.io
Creation date : 19/08/2024, 11:33 GMT+10
Provisioning state : Succeeded
Pricing plan : Basic

Simplify container lifecycle management

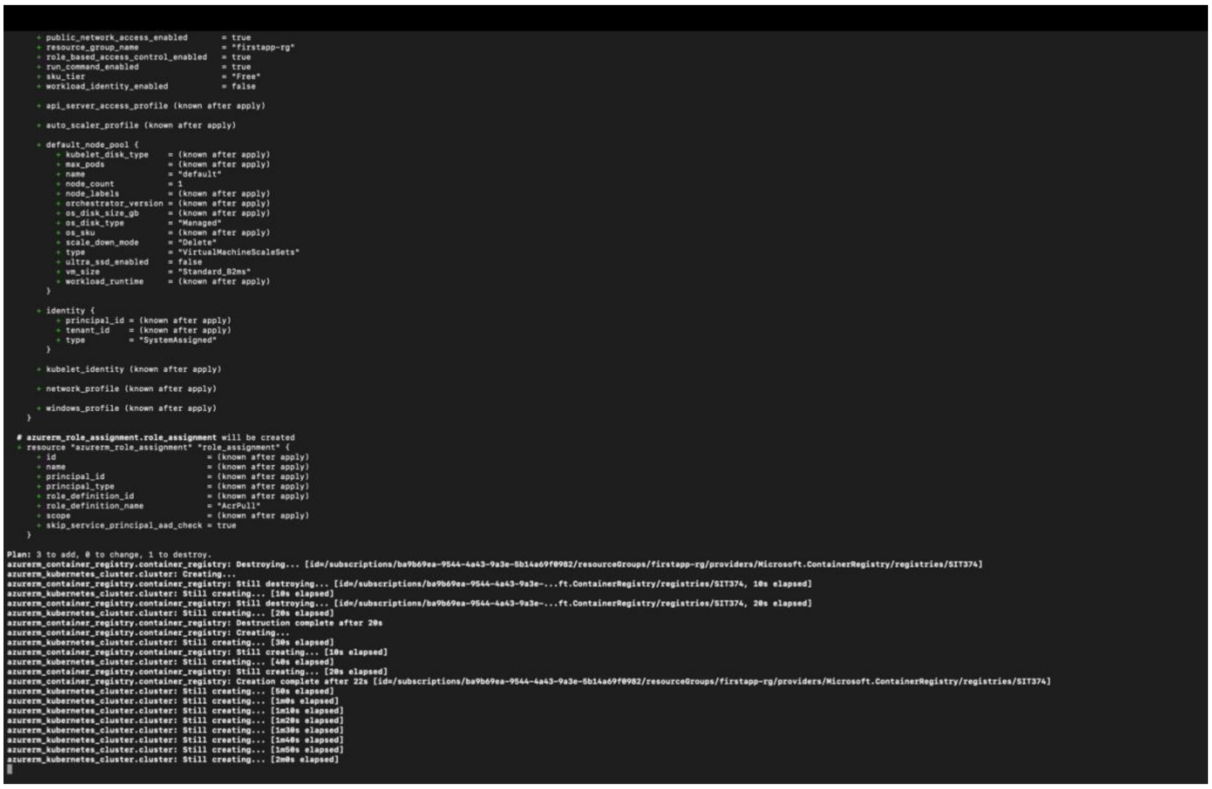
Container registry allows you to build, store, and manage container images and artifacts in a private registry for all types of container deployments. Learn more

Push a Container Image
Get instructions on how to store your container images in your container registry. Learn more about container images

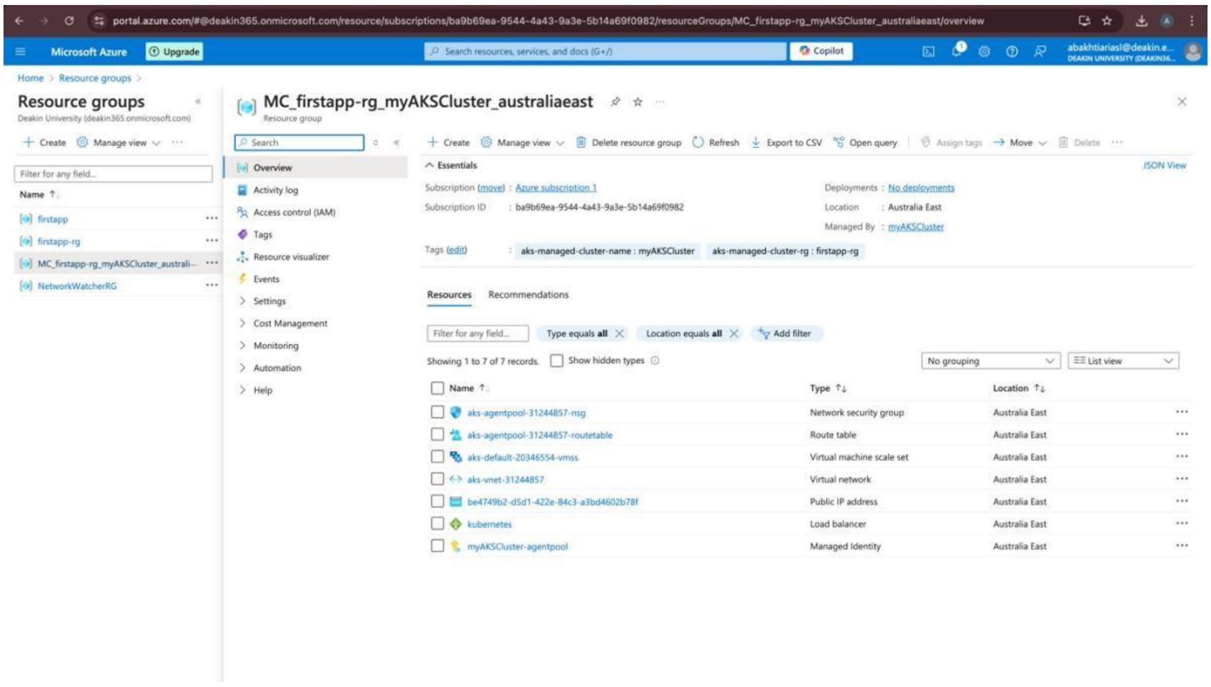
Manage your access controls
Secure your container registry by configuring how users interact with your container registry

Deploy a container image
Explore the latest ways to control, secure, and streamline artifact deployments in Container Registry - or choose one and get started.

erraform is actively applying changes to Azure resources, including destroying an existing container registry and setting up a new AKS cluster with specific configurations.



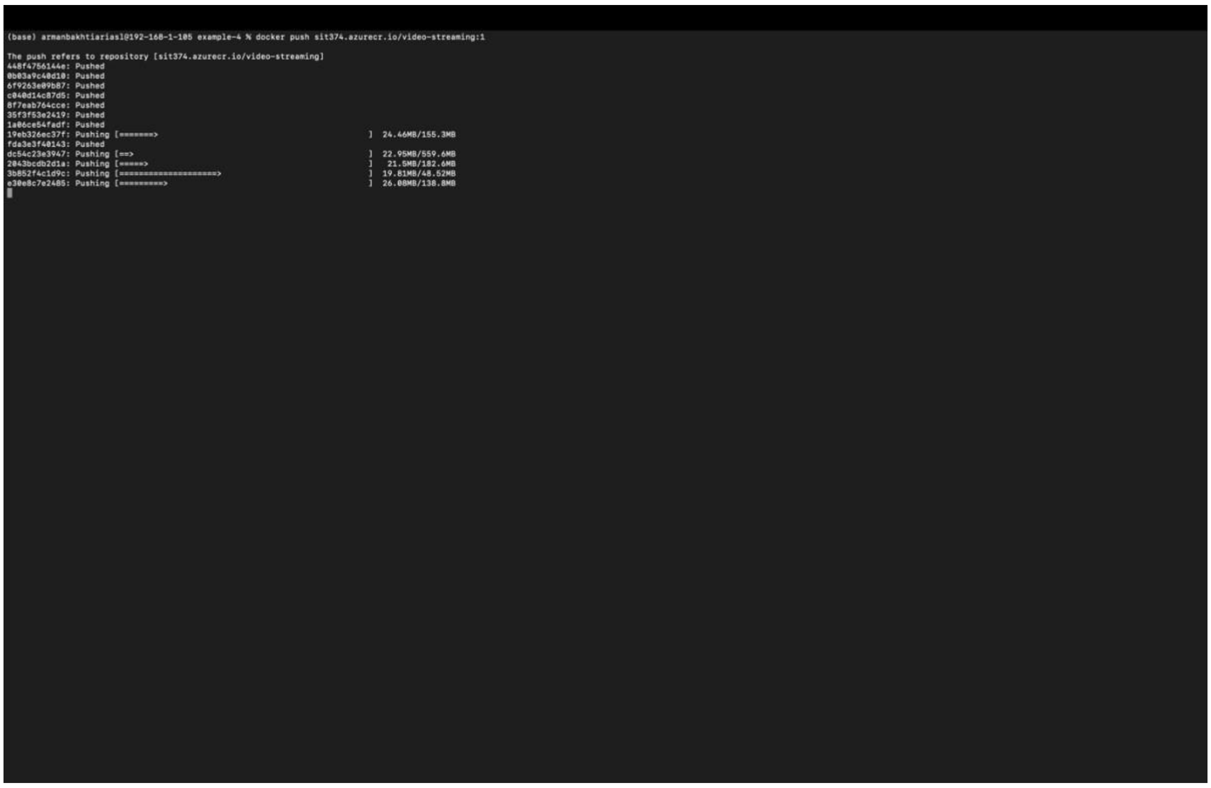
```
      + public_network_access_enabled       = true
      + resource_group_name                 = "firstapp-rg"
      + role_based_access_control_enabled   = true
      + run_command_enabled                 = true
      + sku_tier                            = "Free"
      + workload_identity_enabled           = false

      + api_server_access_profile (known after apply)

      + auto_scaler_profile (known after apply)

      + default_node_pool {
          + kubelet_disk_type     = (known after apply)
          + max_pods              = (known after apply)
          + name                  = "default"
          + node_count            = 1
          + node_labels           = (known after apply)
          + orchestrator_version = (known after apply)
          + os_disk_size_gb       = (known after apply)
          + os_disk_type          = "Managed"
          + os_sku                = (known after apply)
          + scale_down_mode       = "Delete"
          + type                  = "VirtualMachineScaleSets"
          + ultra_ssd_enabled     = false
          + vm_size               = "Standard_B2ms"
          + workload_runtime      = (known after apply)
        }

      + identity {
          + principal_id = (known after apply)
          + tenant_id    = (known after apply)
          + type         = "SystemAssigned"
        }

      + kubelet_identity (known after apply)

      + network_profile (known after apply)

      + windows_profile (known after apply)
    }

  # azurerm_role_assignment.role_assignment will be created
  + resource "azurerm_role_assignment" "role_assignment" {
      + id                               = (known after apply)
      + name                             = (known after apply)
      + principal_id                     = (known after apply)
      + principal_type                   = (known after apply)
      + role_definition_id               = (known after apply)
      + role_definition_name             = "AcrPull"
      + scope                            = (known after apply)
      + skip_service_principal_aad_check = true
    }

Plan: 3 to add, 0 to change, 1 to destroy.
azurerm_container_registry.container_registry: Destroying... [id=/subscriptions/ba9b69ea-9544-4a43-9a3e-5b14a69f0982/resourceGroups/firstapp-rg/providers/Microsoft.ContainerRegistry/registries/SIT374]
azurerm_kubernetes_cluster.cluster: Creating...
azurerm_container_registry.container_registry: Still destroying... [id=/subscriptions/ba9b69ea-9544-4a43-9a3e-...ft.ContainerRegistry/registries/SIT374, 10s elapsed]
azurerm_kubernetes_cluster.cluster: Still creating... [10s elapsed]
azurerm_container_registry.container_registry: Still destroying... [id=/subscriptions/ba9b69ea-9544-4a43-9a3e-...ft.ContainerRegistry/registries/SIT374, 20s elapsed]
azurerm_kubernetes_cluster.cluster: Still creating... [20s elapsed]
azurerm_container_registry.container_registry: Destruction complete after 20s
azurerm_container_registry.container_registry: Creating...
azurerm_kubernetes_cluster.cluster: Still creating... [30s elapsed]
azurerm_container_registry.container_registry: Still creating... [10s elapsed]
azurerm_kubernetes_cluster.cluster: Still creating... [40s elapsed]
azurerm_container_registry.container_registry: Still creating... [20s elapsed]
azurerm_container_registry.container_registry: Creation complete after 22s [id=/subscriptions/ba9b69ea-9544-4a43-9a3e-5b14a69f0982/resourceGroups/firstapp-rg/providers/Microsoft.ContainerRegistry/registries/SIT374]
azurerm_kubernetes_cluster.cluster: Still creating... [50s elapsed]
azurerm_kubernetes_cluster.cluster: Still creating... [1m0s elapsed]
azurerm_kubernetes_cluster.cluster: Still creating... [1m10s elapsed]
azurerm_kubernetes_cluster.cluster: Still creating... [1m20s elapsed]
azurerm_kubernetes_cluster.cluster: Still creating... [1m30s elapsed]
azurerm_kubernetes_cluster.cluster: Still creating... [1m40s elapsed]
azurerm_kubernetes_cluster.cluster: Still creating... [1m50s elapsed]
azurerm_kubernetes_cluster.cluster: Still creating... [2m0s elapsed]
```



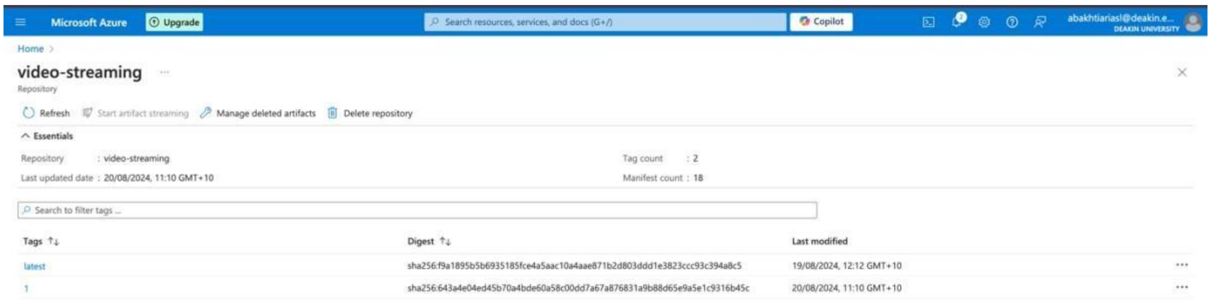
Subscription (move) : Azure subscription 1
Subscription ID : ba9b69ea-9544-4a43-9a3e-5b14a69f0982
Deployments : No deployments
Location : Australia East
Managed By : myAKSCluster
Tags (edit) : aks-managed-cluster-name : myAKSCluster  aks-managed-cluster-rg : firstapp-rg

Showing 1 to 7 of 7 records.

| Name | Type | Location |
|---|---|---|
| aks-agentpool-31244857-nsg | Network security group | Australia East |
| aks-agentpool-31244857-routetable | Route table | Australia East |
| aks-default-20346554-vmss | Virtual machine scale set | Australia East |
| aks-vnet-31244857 | Virtual network | Australia East |
| be4749b2-d5d1-422e-84c3-a3bd4602b78f | Public IP address | Australia East |
| kubernetes | Load balancer | Australia East |
| myAKSCluster-agentpool | Managed Identity | Australia East |



```
(base) armanbakhtiariasl@192-168-1-105 example-4 % docker push sit374.azurecr.io/video-streaming:1

The push refers to repository [sit374.azurecr.io/video-streaming]
448f4756144e: Pushed
0b03a9c40d10: Pushed
6f9263e09b87: Pushed
c040d14c87d5: Pushed
8f7eab764cce: Pushed
35f3f53e2419: Pushed
1a06ce54fadf: Pushed
19eb326ec37f: Pushing [=======>                                           ]  24.46MB/155.3MB
fda3e3f40143: Pushed
dc54c23e3947: Pushing [==>                                                ]  22.95MB/559.6MB
2043bcdb2d1a: Pushing [=====>                                             ]   21.5MB/182.6MB
3b852f4c1d9c: Pushing [====================>                              ]  19.81MB/48.52MB
e30e8c7e2485: Pushing [=========>                                         ]  26.08MB/138.8MB
```



video-streaming

Repository

Repository : video-streaming
Last updated date : 20/08/2024, 11:10 GMT+10
Tag count : 2
Manifest count : 18

| Tags | Digest | Last modified |
|---|---|---|
| latest | sha256:f9a1895b5b6935185fce4a5aac10a4aae871b2d803ddd1e3823ccc93c394a8c5 | 19/08/2024, 12:12 GMT+10 |
| 1 | sha256:643a4e04ed45b70a4bde60a58c00dd7a67a876831a9b88d65e9a5e1c9316b45c | 20/08/2024, 11:10 GMT+10 |