

# Terraform Interview Questions

## Q1- What is Terraform providers & Example?

**Ans-** Terraform providers are plugins that enable Terraform to interact with various APIs and services. Providers are responsible for managing the lifecycle of a resource: create, read, update, and delete. They allow Terraform to work with cloud platforms, SaaS providers, and other services

### Example

#### # Configure the Azure provider

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

#### # Create a resource group

```
resource "azurerm_resource_group" "example" {  
  name     = "example-resources"  
  location = "East US"  
}
```

## Q2- What is Terraform state and Terraform file?

**Ans-** **Terraform state** is a critical aspect of how Terraform operates. It records the state of your infrastructure managed by Terraform. The state is stored in a file, often called the **state file** (**terraform.tfstate**).

**This file serves several important purposes:**

1. **Mapping Real Resources to Configuration:** Terraform uses the state file to map the resources in your configuration to real-world resources.
2. **Metadata Storage:** The state file stores metadata for resources, which helps Terraform improve performance.
3. **Dependency Graph:** Terraform uses the state to determine the order of operations and relationships between resources.
4. **Concurrency Control:** The state file helps manage concurrent updates to your infrastructure.

**Terraform files** are the configuration files that define your infrastructure. These files use the HashiCorp Configuration Language (HCL) or JSON and typically have a **.tf extension**. These files describe the desired state of your infrastructure and are used by Terraform to plan and apply changes.

- **Main.tf**- This file typically contains the main configuration for your infrastructure
- **Variable.tf**

```
variable "region" {  
  description = "The AWS region to create resources in"
```

```
type    = string
default = "us-west-2"
```

- **Output.tf**

```
output "instance_id" {
  description = "The ID of the EC2 instance"
  value       = aws_instance.example.id
}
```

### Q3- What is Terraform modules?

**Ans-** Terraform modules are a way to group multiple resources together and reuse this configuration across different parts of your infrastructure

```
terraform-azure-example/
├── main.tf
├── outputs.tf
├── variables.tf
├── modules/
│   └── network/
│       ├── main.tf
│       ├── outputs.tf
│       └── variables.tf
```

We can call modules in root module via providing path of child modules.

```
module "network" {
  source          = "../modules/network"
  resource_group_name = var.resource_group_name
  location        = var.location
  vnet_name       = var.vnet_name
  address_space   = var.address_space
  subnet_name     = var.subnet_name
  subnet_prefixes = var.subnet_prefixes
}
```

- **Root Module:** The main entry point for Terraform, located in the directory where **terraform apply** is run.
- **Child Module:** Modules called by another module, usually stored in a subdirectory or sourced remotely.
- **Remote Module:** Modules sourced from remote locations such as Git repositories, Terraform Registry, or other remote storage.

**Example:**

```
module "network" {
  source = "git::https://github.com/username/repo.git/modules/network?ref=v1.0.0"
```

#### Q4- Why we need terraform?

**Ans:** Terraform is required due to its reusability, efficiency, and its open source so there is no cost involved so there is no extra cost we need to wear.

- **Declarative Configuration:** Terraform uses a declarative language (HashiCorp Configuration Language, HCL) to define infrastructure. You describe the desired state of your infrastructure, and Terraform takes care of achieving that state.
- **Version Control:** Infrastructure configurations can be stored in version control systems (like Git), enabling versioning, rollback, and collaboration.

#### Q5- What is difference b/w imperative and declarative method?

**Ans:** Below is basic difference

**Imperative Approach** - The imperative approach involves explicitly describing the steps required to achieve a desired outcome. It focuses on **how** to perform tasks, detailing the sequence of commands or operations.

Example - Shell scripts, Python scripts, etc

# Bash script to set up a web server

```
apt-get update
apt-get install -y nginx
systemctl start nginx
systemctl enable nginx
```

**Declarative Approach** - The declarative approach involves specifying the desired state of the system without describing the steps to achieve that state. It focuses on **what** the end state should be, letting the underlying system figure out the necessary operations.

Example:- Terraform, Kubernetes YAML manifests, AWS CloudFormation

Aspect	Imperative	Declarative
Focus	How to achieve the result	What the end result should be
Control	Fine-grained control over each step	Abstraction of steps, focusing on the desired state
State Management	Manual	Automatic
Complexity	Can handle complex logic	Simpler, more maintainable
Error Prone	Higher due to manual steps	Lower due to automation
Use Cases	Scripts, procedural programming	Infrastructure as Code (IaC), configuration management
Examples	Shell scripts, Ansible (procedural)	Terraform, Kubernetes manifests, AWS

Aspect	Imperative	Declarative
		CloudFormation

#### Q6- Differentiate between implicit and explicit dependency?

Ans:-

- **Implicit Dependency:** Terraform automatically figures out the order in which resources need to be created or updated based on how they're connected in your code. You don't have to explicitly tell Terraform about these relationships; it understands them by looking at your configuration.
- **Explicit Dependency:** With explicit dependency, you directly tell Terraform which resources depend on others by using specific syntax. You explicitly declare the relationships between resources in your code, leaving no room for ambiguity.

```
resource "azurerm_network_interface" "example" {
  ...
  depends_on = [azurerm_virtual_network.example] # Explicit dependency declaration
}
```

#### Q7- How do you handle explicit dependency in terraform?

**Ans:** you can handle explicit dependencies between resources using the **depends\_on** attribute. This attribute allows you to specify a list of resources that a particular resource depends on, ensuring that Terraform creates or updates the dependent resources before processing the resource with the dependency

#### Q8- How do you handle secrets in terraform?

We have multiple ways to handle secret in terraform

```
1. Set environment variables:
export ARM_CLIENT_ID="your_client_id"
export ARM_CLIENT_SECRET="your_client_secret"
export ARM_SUBSCRIPTION_ID="your_subscription_id"
export ARM_TENANT_ID="your_tenant_id"
```

#### 2. Configure Terraform to use these environment variables:

```
provider "azurerm" {
  features {}

  client_id = var.client_id
```

```

client_secret = var.client_secret
subscription_id = var.subscription_id
tenant_id     = var.tenant_id
}

```

```

variable "client_id" {}
variable "client_secret" {
  sensitive = true
}
variable "subscription_id" {}
variable "tenant_id" {}

```

### 3. Azure Key Vault

Azure Key Vault is a secure way to store and manage sensitive information. You can integrate Azure Key Vault with Terraform using the Azure Key Vault provider.

#### Example:

1. **Create an Azure Key Vault and a secret:**

```

az keyvault create --name myKeyVault --resource-group myResourceGroup --location westus
az keyvault secret set --vault-name myKeyVault --name mySecret --value "mysecretvalue"

```

2. **Configure Terraform to use the Azure Key Vault provider:**

```

provider "azurerm" {
  features {}
}

data "azurerm_key_vault" "example" {
  name            = "myKeyVault"
  resource_group_name = "myResourceGroup"
}

data "azurerm_key_vault_secret" "example" {
  name      = "mySecret"
  key_vault_id = data.azurerm_key_vault.example.id
}

resource "azurerm_example_resource" "example" {
  secret_value = data.azurerm_key_vault_secret.example.value
}

```

### Q9- What is terraform resource graph?

**Ans:** The Terraform resource graph is a conceptual and visual representation of the resources defined in a Terraform configuration and their relationships to one another. It helps you understand how resources are interdependent and the order in which Terraform will create, update, or destroy them during the execution of a plan or apply operation.

### Generating and Visualizing the Resource Graph

To generate and visualize the resource graph, follow these steps:

1. **Generate the Graph:** Run the following command to generate the resource graph in DOT format:

```
terraform graph > graph.dot
```

2. **Visualize the Graph:** Use Graphviz or an online DOT file viewer to visualize the graph. For example, if you have Graphviz installed:

```
dot -Tpng graph.dot -o graph.png
```

### Q10- What is terraform refresh?

**Ans: terraform refresh** is a Terraform command used to reconcile the state of the Terraform-managed infrastructure with the actual state of the resources. This command updates the state file to reflect the current reality of the resources in the infrastructure.

```
terraform refresh
```

### Q11- What is null resource in terraform??

**Ans:- null\_resource** in Terraform is a way to execute scripts or commands and manage dependencies without creating any actual infrastructure. This is useful for running custom scripts after certain resources are created.

### Use null\_resource to Run a Local Script After VM Creation

Now, let's use a null\_resource to run a script after the VM is created. Let's assume you have a script called `configure.sh` that you want to run.

```
resource "null_resource" "run_script" {  
  provisioner "local-exec" {  
    command = "sh configure.sh"  
  }  
}
```

```
# Ensure this runs after the VM is created  
depends_on = [  
  azurerm_linux_virtual_machine.example  
]
```

**Q12- Someone deleted resources manually in portal how can you recover that through terraform???**

Ans:- **1. Identify Deleted Resources**

First, identify which resources have been deleted. This information can be gathered from the Azure portal or resource logs.

**2. Update Terraform Configuration**

Ensure that your Terraform configuration files (.tf files) still describe the desired state of your infrastructure, including the deleted resources.

**3. Refresh the State**

Run terraform refresh to update the Terraform state file with the current state of your infrastructure. This will reflect the deletion of resources in the state file.

```
terraform refresh
```

**4. Plan to Recreate Resources**

Run terraform plan to see what changes Terraform will make to align the actual state with the desired state described in your configuration files. The plan should indicate that the deleted resources will be recreated.

```
terraform plan
```

**5. Apply Changes**

Run terraform apply to execute the plan and recreate the deleted resources.

```
terraform apply
```

**Q13- Explain Terraform workspace?**

**Ans:** In Terraform, workspaces are a feature that allows you to manage multiple instances of the same infrastructure configuration within a single directory. Each workspace maintains its own state file, allowing you to manage different environments (such as development, staging, and production) or different configurations (such as different regions or configurations with different variables) independently

## Common Commands for Terraform Workspaces

Create a Workspace:

```
terraform workspace new <workspace-name>
```

- **Select a Workspace:**

```
terraform workspace select <workspace-name>
```

- **List Workspaces:**

```
terraform workspace list
```

- **Delete a Workspace:**

```
terraform workspace delete <workspace-name>
```

- **Show Current Workspace:**

```
terraform workspace show
```

## Example Use Case

Suppose you have a Terraform configuration for deploying a web application to different environments:

```
|— main.tf  
|— variables.tf  
|— terraform.tfvars
```

- You can create different workspaces for development, staging, and production:

```
terraform workspace new dev  
terraform workspace new staging  
terraform workspace new prod
```

## Q14- Explain commands in terraform?

Ans:-

- **terraform init:** Initializes a Terraform working directory by downloading plugins and modules.
- **terraform plan:** Generates an execution plan describing infrastructure changes.
- **terraform apply:** Applies the changes described in the execution plan.
- **terraform destroy:** Destroys Terraform-managed infrastructure.



- **terraform validate:** Validates Terraform configuration files for syntax errors.
- **terraform refresh:** Updates the state file with the current real-world state of resources.
- **terraform state:** Manages Terraform state files, allowing inspection and modification.
- **terraform workspace:** Manages Terraform workspaces for multiple environments or configurations.
- **terraform output:** Prints outputs defined in Terraform configuration files.
- **terraform fmt:** Rewrites Terraform configuration files to a canonical format.
- **terraform providers:** Prints a tree of the providers used in the configuration.
- **terraform version:** Prints the Terraform version currently in use.

#### Q15- What are terraform provisioners?

Execute scripts or commands on local or remote resources after they are provisioned by Terraform, handling tasks not managed directly by Terraform's resource declarations.

```
resource "azurerm_virtual_machine" "example" {
  name           = "example-vm"
  location       = "East US"
  resource_group_name = azurerm_resource_group.example.name
  vm_size        = "Standard_DS1_v2"
```

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

```
  os_profile {
    computer_name = "hostname"
    admin_username = "adminuser"
```

```
    admin_ssh_key {
      username = "adminuser"
      public_key = file("~/ssh/id_rsa.pub")
    }
  }
```

#### # Local-Exec provisioner example

```
provisioner "local-exec" {
  command = "echo Virtual machine ${self.name} provisioned"
}
```

### # Remote-Exec provisioner example

```
provisioner "remote-exec" {  
  inline = [  
    "sudo apt-get update",  
    "sudo apt-get install -y nginx",  
    "sudo systemctl start nginx"  
  ]  
}
```

### # File provisioner example

```
provisioner "file" {  
  source      = "path/to/local/file.txt"  
  destination = "/path/on/remote/file.txt"  
}
```

### Q16- What is terraform taint ??

**Terraform taint** is a command used to mark a specific resource managed by Terraform as "tainted," forcing it to be destroyed and recreated during the next terraform apply. This is useful for handling situations where you want to recreate a resource due to configuration changes or troubleshooting issues.

```
terraform taint azurerm_virtual_machine.example
```

#### Key Points:

- **Selective Recreate:** Allows you to recreate specific resources without affecting others.
- **State Management:** Updates Terraform's state file to reflect the tainted status for the resource.
- **Use Case:** Useful for troubleshooting resource configuration issues or applying specific changes that require recreation rather than modification.

### Q17- How can we import a resource ?

```
terraform import <resource_type>.<resource_name> <resource_id>
```

- **<resource\_type>:** The type of resource as defined in your Terraform configuration.
- **<resource\_name>:** The name of the resource instance in Terraform.
- **<resource\_id>:** The unique identifier of the existing resource in the provider's format.

```
Terraform import azurerm_virtual_machine.example-vm  
/subscriptions/subscription_id/resourceGroups/resource_group_name/providers/Microsoft.Compu  
te/virtualMachines/example-vm
```

### Q18- What is the purpose of lockfile in terraform ?

The purpose of the lockfile in Terraform is to ensure exclusive access and prevent concurrent modifications to the state file (terraform.tfstate), maintaining consistency and integrity during infrastructure operations.

- **Prevents Concurrent Modifications:** Terraform uses a lockfile to prevent concurrent modifications to the state file (terraform.tfstate). This ensures that only one Terraform operation (like terraform apply or terraform destroy) can modify the state at any given time.
- **Exclusive Access:** When one user or process is modifying infrastructure with Terraform, other users or processes are prevented from simultaneously applying changes that could conflict or overwrite state data.

#### Q19- When state locking occurs in Terraform ?

In Terraform, state locking occurs when a command that modifies the Terraform state (terraform apply, terraform destroy, etc.) is executed.

#### Q20- How can we deploy multiple subscription using same terraform code ?

##### Ans

To deploy multiple subscriptions using the same Terraform code, you can use the concept of workspaces or modules and parameterize your Terraform configuration. Here's how you can do it:

##### 1. Using Workspaces

Workspaces allow you to manage multiple environments or configurations within the same Terraform configuration. Here's a general approach:

###### 1. Initialize Terraform:

```
terraform init
```

###### 2. Create and Select Workspaces: Create a workspace for each subscription.

```
terraform workspace new dev
terraform workspace new prod
```

###### 3. Define Variables: Define subscription-specific variables in your variables.tf file:

```
variable "subscription_id" {
  description = "The subscription ID"
}
```

###### 4. Use Variables in Providers: Use the variables in your provider configuration:

```
provider "azurerm" {
  subscription_id = var.subscription_id
  features {} }
```

###### 5. Configure Variables for Workspaces: Create terraform.tfvars files for each workspace, e.g., dev.tfvars, prod.tfvars, containing the subscription-specific values:

```
# dev.tfvars
subscription_id = "your-dev-subscription-id"
```

```
# prod.tfvars
subscription_id = "your-prod-subscription-id"
```

6. **Apply Configuration:** Select the appropriate workspace and apply the configuration:

```
terraform workspace select dev
terraform apply -var-file="dev.tfvars"
```

```
terraform workspace select prod
terraform apply -var-file="prod.tfvars"
```

## 2. Using Modules and Different Variable Files

You can also use modules and different variable files for different subscriptions. Here's a general approach:

1. **Create a Module:** Define your infrastructure as a module in a modules directory.

```
# modules/your_module/main.tf
resource "azurerm_resource_group" "example" {
  name     = var.resource_group_name
  location = var.location
}
```

```
# modules/your_module/variables.tf
variable "resource_group_name" {}
variable "location" {}
variable "subscription_id" {}
```

2. **Create a Root Module:** In your root module, call the module and pass the necessary variables.

```
# main.tf
module "your_module" {
  source = "../modules/your_module"

  resource_group_name = var.resource_group_name
  location            = var.location
  subscription_id     = var.subscription_id
}
```

3. **Define Variables:** Define the variables in your root module.

```
# variables.tf
variable "resource_group_name" {}
variable "location" {}
variable "subscription_id" {}
```

4. **Create Variable Files:** Create variable files for each subscription, e.g., dev.tfvars, prod.tfvars.

```
# dev.tfvars
resource_group_name = "dev-rg"
location            = "West Europe"
subscription_id     = "your-dev-subscription-id"
```

```
# prod.tfvars
resource_group_name = "prod-rg"
location            = "East US"
subscription_id     = "your-prod-subscription-id"
```

5. **Apply Configuration:** Apply the configuration using the appropriate variable file.

```
terraform apply -var-file="dev.tfvars"
terraform apply -var-file="prod.tfvars"
```

## Q20- How can we call data of an existing resource?

**Ans:** To reference existing resources in Terraform, you use the data block. The data block allows you to query existing resources and use their attributes in your Terraform configuration.

### • Define the Data Source:

In your Terraform configuration, define a data block to query the existing resource. For example, to get details of an existing Azure resource group:

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

data "azurerm_resource_group" "example" {
  name = "existing-resource-group-name"
}
```

### • Use the Data Source:

You can use the attributes of the data source in your resource definitions. For example, you might want to use the location of the existing resource group when creating a new resource:

```
resource "azurerm_storage_account" "example" {
  name                = "examplestoracc"
  resource_group_name = data.azurerm_resource_group.example.name
  location            = data.azurerm_resource_group.example.location
  account_tier        = "Standard"
  account_replication_type = "LRS"
}
```

## Q21- When the plugins download in terraform ?

**Ans** When you run **terraform init** for the first time in a new configuration directory, Terraform downloads the necessary provider plugins based on the providers specified in your configuration files.

## Q22- What is difference between terraform refresh and terraform plan.?

Ans

- **terraform refresh:** Updates the state file to reflect the actual state of resources without proposing any changes.
- **terraform plan:** Generates an execution plan showing the proposed changes to reach the desired state defined in the configuration.

## Q22- What type of automation tools you have used other than terraform?

Ans:-We are using many different tool other than Terraform

- Azure CLI
- PowerShell
- ARM Template

## Q23- Which types of resources you have created in Azure via terraform?

I have create most of all type of service Which we are using for IAAS & PAAS.

### ▪ Resource Groups

```
resource "azurerm_resource_group" "example" {  
  name      = "example-resources"  
  location = "West US"  
}
```

### ▪ Virtual Networks

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

### ▪ Subnets

```
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     = ["10.0.1.0/24"]  
}
```

### ▪ Network Security Groups

```
resource "azurerm_network_security_group" "example" {  
  name = "example-nsg"
```

```

location          = azurerm_resource_group.example.location
resource_group_name = azurerm_resource_group.example.name
}

```

## • Virtual Machines

```

resource "azurerm_virtual_machine" "example" {
  name                = "example-vm"
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  network_interface_ids = [azurerm_network_interface.example.id]
  vm_size             = "Standard_DS1_v2"

```

```

  storage_os_disk {
    name          = "example-os-disk"
    caching       = "ReadWrite"
    create_option = "FromImage"
    managed_disk_type = "Standard_LRS"
  }

```

```

  os_profile {
    computer_name = "hostname"
    admin_username = "adminuser"
    admin_password = "Password1234!"
  }

```

```

  os_profile_linux_config {
    disable_password_authentication = false
  }

```

```

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

```

## Q24- Suppose a resource is created using terraform and someone made changes manually using portal, How to sync the changes with terraform state file?

Ans:- When changes are made manually in the Azure portal to resources managed by Terraform, you'll need to sync those changes back into your Terraform state file to maintain consistency and manage them effectively.

- **Import Resources:** For existing resources that were modified manually, you can import them into Terraform state. Use the terraform import command to bring them under Terraform's management. For example:

```

terraform import azurerm_resource_group.example /subscriptions/000000000-0000-0000-0000-000000000000/resourceGroups/rg-example

```

Replace `azurerm_resource_group.example` with your resource type and name, and `/subscriptions/0000...` with the actual Azure resource ID.

- **Terraform State Refresh:** After importing, perform a Terraform state refresh to update the state file with

Terraform refresh

### Q25- What are locals in terraform?

Ans:-In Terraform, locals allow you to define reusable values within a module without creating additional resources. They're useful for avoiding repetitive expressions and enhancing readability.

#### Example in Azure:

```
locals {  
  resource_group_name = "myResourceGroup"  
  location             = "West Europe"  
}  
  
resource "azurerm_resource_group" "example" {  
  name     = local.resource_group_name  
  location = local.location  
}
```

- `locals` block defines `resource_group_name` and `location` as reusable values.
- `azurerm_resource_group` resource uses these locals to specify the name and location of the Azure resource group.

### Q26- What is Landing Zones in azure and how we can deploy via terraform?

Ans:

**Azure Landing Zones** are architectural blueprints that facilitate the deployment of well-organized, compliant, and secure Azure environments. They typically include:

1. **Management Groups:** Hierarchical structure for organizing subscriptions.
2. **Resource Groups:** Logical containers for managing Azure resources.
3. **Policies and Governance:** Azure Policy for compliance and RBAC for access control.
4. **Networking:** Virtual Networks (VNETs) for connectivity and isolation.

**Deploying via Terraform:** Use Terraform to define and provision Azure resources such as management groups, VNETs, policies, and more, ensuring consistent deployment and management of Azure landing zones.

### Q26- What is terraform flow ??

- **Initialization (terraform init):** Initializes the working directory and downloads necessary providers and modules.
- **Planning (terraform plan):** Generates an execution plan showing what Terraform will do to reach the desired state defined in configuration files.



- **Application (terraform apply):** Applies the changes defined in Terraform configuration files to provision, modify, or delete infrastructure resources.

#### Q26- What will happen when we run terraform destroy command?

##### Ans

- **Plan Destruction:** Generate a destruction plan (terraform plan -destroy) to show which resources will be deleted.
- **Resource Deletion:** Delete all resources managed by Terraform in the current configuration.
- **State Management:** Update the Terraform state file (terraform.tfstate) to reflect the removal of resources.

#### Q26- When a statefile will be created ?

A Terraform state file (**terraform.tfstate**) is created:

1. **During Initialization:** The state file is initially created when you run terraform apply for the first time after defining infrastructure resources in Terraform configuration files.
2. **Resource Tracking:** It tracks the current state of deployed resources, including IDs, metadata, and dependencies.
3. **Update Operations:** The state file is updated automatically whenever you apply changes (terraform apply) or destroy resources (terraform destroy), reflecting the current state of your infrastructure.

#### Q26- You have a requirement to create a scalable web app infrastructure using the terraform , that app consist of a LB and DB , what would be your design and what type of structures of your terraform configuration is going to be. how are you going to deploy these infrastructure.

- **Frontend:** The user interface (UI) that users interact with through their browsers.
- **Backend:** The server-side logic that processes requests, interacts with databases (like the DB mentioned earlier), and performs computations.
- **Database Integration:** Integration with databases for storing and retrieving data.
- **Scalability Considerations:** Configuration to handle varying loads and traffic spikes efficiently.

```
provider "azurerm" {  
  features {}  
}  
  
resource "azurerm_resource_group" "example" {  
  name     = "myResourceGroup"  
  location = "West Europe"}
```

```

}
resource "azurerm_public_ip" "lb_public_ip" {
  name          = "myPublicIP"
  location      = azurerm_resource_group.example.location
  allocation_method = "Dynamic"
  sku           = "Standard"
}

resource "azurerm_lb" "example" {
  name          = "myLB"
  resource_group_name = azurerm_resource_group.example.name
  location      = azurerm_resource_group.example.location

  frontend_ip_configuration {
    name          = "PublicIPAddress"
    public_ip_address_id = azurerm_public_ip.lb_public_ip.id
  }
  backend_address_pool {
    name = "myBackendPool"
  }
}

resource "azurerm_mysql_server" "example" {
  name          = "myMySQLServer"
  location      = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  sku_name      = "GP_Gen5_2"
  storage_profile {
    storage_mb = 5120
  }
  administrator_login      = "mysqladmin"
  administrator_login_password = "P@ssw0rd!"
}

```

## Q26- How do you use the depends\_on attribute in Terraform?

The **depends\_on** attribute establishes explicit dependencies between resources in Terraform, ensuring that certain resources are created or updated before others, effectively controlling the order of resource provisioning and ensuring correct resource dependencies.

```

resource "azurerm_network_interface" "example" {
  depends_on = [azurerm_virtual_network.example] # Explicit dependency declaration
}

```

## Q26- compare the usage of count[index] and for\_each[map/list] ?

Ans: Both **count[index]** and **for\_each[map/list]** are used to create multiple instances of resources in Terraform.

While **count[index]** is based on a numerical index and creates a fixed number of resource instances, **for\_each[map/list]** is based on a map or list of values and allows for dynamic creation of resource instances based on variable input.

### Example:

```
resource "azurerm_virtual_machine" "example" {
  count = 3
  # Configuration for each
  VM name = "vm-${count.index}"
  # Other settings...
}
```

### For Each--:

```
variable "storage_accounts" {
  type = map(object({
    location = string
    sku      = string
  }))
}

resource "azurerm_storage_account" "example" {
  for_each = var.storage_accounts
  location = each.value.location
  sku      = each.value.sku
  # Other settings...
}
```

### Q27-How do variables (`var`) and outputs (`output`) differ in Terraform?

### Ans:-

#### Variables (var):

- **Purpose:** Variables in Terraform are used to define and parameterize configurations, allowing for dynamic inputs that can be customized during deployment.
- **Usage:** They are declared in .tf files (e.g., variables.tf) and can be set via command-line flags (-var) or terraform.tfvars files.

```
variable "region" {
  description = "AWS region where resources will be deployed"
  type        = string
  default     = "us-east-1"
}
```

#### Outputs (output):

- **Purpose:** Outputs in Terraform are used to expose information about resources after deployment, such as IP addresses or resource IDs.
- **Usage:** They are defined in .tf files (e.g., outputs.tf) and accessed using terraform output to fetch values from the Terraform state.

```
output "instance_id" {
  value = aws_instance.example.id
}
```

### Q27-What types of variables we use in Terraform?

- **String:** Used for defining textual values.

```
variable "region" {  
  type = string  
  default = "us-west-2"  
}
```

- **Number:** Used for defining numeric values, which can be integers or floating-point numbers.

```
variable "instance_count" {  
  type = number  
  default = 3  
}
```

- **Boolean:** Used for defining true/false values.

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

- **List:** Used for defining ordered collections of values.

```
variable "availability_zones" {  
  type = list(string)  
  default = ["us-west-1a", "us-west-1b"]  
}
```

- **Map:** Used for defining collections of key-value pairs.

```
variable "tags" {  
  type = map(string)  
  default = {  
    Environment = "Production"  
    Owner       = "DevOps Team"  
  }  
}
```

### Q27-What are Static & dynamic blocks in terraform?

**Ans:**

**Static Block:-** Static blocks in Terraform are used to define fixed configurations within resource blocks, where the configuration is predefined and does not change based on input variables.

```
resource "azurerm_virtual_machine" "example" {  
  name          = "my-vm"  
  location      = "West Europe"  
  resource_group_name = azurerm_resource_group.example.name  
  vm_size       = "Standard_DS1_v2"  
}
```

**Dynamic Block:** - Dynamic blocks in Terraform allow for flexible configurations based on dynamically-generated values or lists, where the configuration can vary based on input variables or conditions.

```
variable "vm_sizes" {  
  type = list(string)  
  default = ["Standard_DS1_v2", "Standard_DS2_v2", "Standard_DS3_v2"]  
}  
  
resource "azurerm_virtual_machine" "example" {  
  for_each = { for idx, size in var.vm_sizes : idx => size }  
  name     = "vm-${each.key}"  
  location = "West Europe"  
  resource_group_name = azurerm_resource_group.example.name  
  vm_size   = each.value  
  os_profile {  
    computer_name = "vm-${each.key}"  
    admin_username = "adminuser"  
    admin_password = "P@ssw0rd!"  
  }  
}
```

## DOCKER INTERVIEW QUESTION

### 1- How do you build a Docker image from a Dockerfile?

Ans - Use the docker build command. For example:

```
bash  
  
docker build -t my-image:latest .
```

### 2- How do you start Docker containers using Docker Compose?

Ans - Use the docker-compose up command. For example:

```
bash  
  
docker-compose up
```

### 3- How do you create and use a Docker volume?

Ans - Create a volume with docker volume create and use it in a container with the -v flag:

```
bash
```

```
docker volume create my-volume  
docker run -v my-volume:/app/data my-image
```

4- **How do you create a Docker network?**

Ans- Use the docker network create command. For example:

```
bash
```

```
docker network create my-network
```

5- **How do you connect a container to a Docker network?**

Ans- Use the --network flag when running a container or the docker network connect command. For example:

```
bash
```

```
docker run --network my-network my-image  
docker network connect my-network my-container
```

6- **What is the difference between CMD and ENTRYPOINT in a Dockerfile?**

Ans- Both **CMD** and **ENTRYPOINT** specify the command to run when a container starts. CMD sets default commands and arguments but can be overridden. ENTRYPOINT defines the executable to run and cannot be overridden.

7- **What is the purpose of the docker exec command?**

Ans- The docker exec command is used to run a new command in a running container. For example:

```
bash
```

```
docker exec -it my-container /bin/bash
```



8- **How do you monitor Docker containers?**

Ans -Docker provides several commands for monitoring, such as docker stats for real-time performance data and docker logs for container logs.

9- **Your team has decided to migrate a legacy application to Docker. The application consists of a web server, a database, and a cache service. How would you approach this migration?**

Ans- **Analyze the application:** Understand the dependencies and requirements of each component.

**Create Dockerfiles:** Write Dockerfiles for the web server, database, and cache service, specifying the necessary dependencies.

**Use Docker Compose:** Define a docker-compose.yml file to manage the multi-container application.

**Network configuration:** Ensure proper networking between containers using Docker networks.

**Data persistence:** Use Docker volumes to persist data for the database and cache services.

**Testing:** Test each container individually and then as a whole using Docker Compose.

**Deployment:** Deploy the application to the desired environment using Docker Compose or Docker Swarm/Kubernetes for orchestration.

10- **Your Dockerized web application is experiencing increased traffic, and you need to scale it horizontally. How would you achieve this using Docker?**

Ans- **Docker Compose:** Update the docker-compose.yml file to scale the web service

```
yml
services:
  web:
    image: my-web-app
    deploy:
      replicas: 5
```

11- **You need to run a MySQL database in a Docker container, but the data should persist even if the container is removed. How would you set this up?**

Ans - Create a Docker volume:

```
bash

docker volume create mysql-data
```

Run the MySQL container with the volume:

**docker run -d --name mysql -e MYSQL\_ROOT\_PASSWORD=root -v mysql-data:/var/lib/mysql mysql:latest**

(This ensures that the data stored in /var/lib/mysql inside the container is persisted in the mysql-data volume.)



12- One of your containers is failing, and you need to inspect the logs and debug it. What steps would you take?

Ans- **Check logs:** Use the docker logs command to view the container's logs

```
bash
docker logs my-container
```

**Inspect container:** Use the docker inspect command to get detailed information about the container.

```
bash
docker inspect my-container
```

**Execute commands:** Use the docker exec command to run commands inside the running container.

```
bash
docker exec -it my-container /bin/bash
```

**Check Docker daemon logs:** If needed, check the Docker daemon logs on the host system for additional information.

13- Your containers need to communicate with each other, but they are unable to do so. How would you troubleshoot and resolve this issue?

Ans - **Check network configuration:** Ensure that the containers are on the same Docker network

```
bash
docker network ls
docker network inspect my-network
```

**Connect containers to network:** If not, connect them to the same network

```
bash
```

```
docker network connect my-network my-container1  
docker network connect my-network my-container2
```

**Test connectivity:** Use `docker exec` to run network commands (e.g., `ping`, `curl`) inside the containers to test connectivity.

```
bash
```

```
docker exec my-container1 ping my-container2
```

14- Your application requires sensitive information such as API keys and passwords. How would you securely manage these secrets in Docker?

Ans- Use Docker secrets to store sensitive data securely

```
bash
```

```
echo "my-secret" | docker secret create my_secret -
```

```
yaml
```

```
services:  
  web:  
    image: my-web-app  
    secrets:  
      - my_secret
```

Use environment variables with caution, preferably injecting them at runtime.

```
bash
```

```
docker run -e API_KEY=my-secret-key my-web-app
```

15- You need to update your running application without any downtime. How would you achieve this using Docker?

Ans - Use Docker Swarm for rolling updates.

```
bash
```

```
docker service update --image new-image:latest my-web-service
```

Define update configurations in the docker-compose.yml for Swarm

```
yaml
```

```
services:
  web:
    image: my-web-app
    deploy:
      update_config:
        parallelism: 2
        delay: 10s
```

#### 16- What are Docker labels?

Ans- Docker labels are key-value pairs that you can add to Docker objects (such as images, containers, and volumes) to organize and manage them more effectively. Labels can be used for filtering and grouping.

```
bash
```

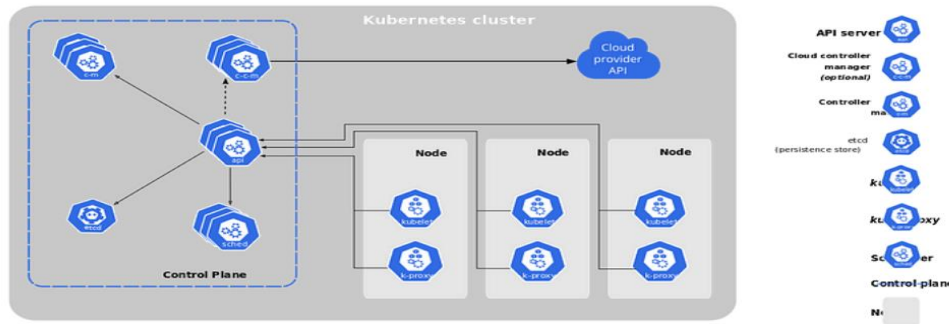
```
docker run -d --label env=production my-container
```

# **KUBERNETES QUESTION ANSWER**

## **1- What is Kubernetes Architecture**

1. **API Server:** In Simple terms, after installing the kubectl on the master node developers run the commands to create pods. So, the command will go to the API Server, and then, the API Server forwards it to that component which will help to create the pods. In other words, the API Server is an entry point for any Kubernetes task where the API Server follows the hierarchical approach to implement the things.
2. **Etcd:** Etcd is like a database that stores all the pieces of information of the Master node and Worker node(entire cluster) such as Pods IP, Nodes, networking configs, etc. Etcd stored data in key-value pair. The data comes from the API Server to store in etc.
3. **Controller Manager:** The controller Manager collects the data/information from the API Server of the Kubernetes cluster like the desired state of the cluster and then decides what to do by sending the instructions to the API Server.
4. **Scheduler:** Once the API Server gathers the information from the Controller Manager, the API Server notifies the Scheduler to perform the respective task such as increasing the number of pods, etc. After

getting notified, the Scheduler takes action on the provided work.



Kubernetes Architecture By Kubernetes

## 2- How does Kubernetes manage container networking?

Ans- Kubernetes uses a flat networking model where each Pod gets its own IP address. This enables containers in different Pods to communicate with each other directly without NAT. Kubernetes networking can be implemented using various network plugins (CNI) like Flannel, Calico, Weave, etc.

## 3- What is a ReplicaSet in Kubernetes?

Ans- A ReplicaSet is a Kubernetes object that ensures a specified number of pod replicas are running at any given time. It automatically replaces failed pods and scales the number of replicas up or down as needed.

## 4- What is Difference between Replica Set and Replication Controller

Ans-

S.No	Replica Set	Replication controller
1	A <b>ReplicaSet</b> is a newer and more advanced Kubernetes object that also ensures a specified number of pod replicas are running at any given time.	A <b>ReplicationController</b> is an older Kubernetes object that ensures a specified number of pod replicas are running at all times
2	<b>ReplicaSets</b> support both equality-based and set-based selectors.	<b>ReplicationControllers</b> use equality-based selectors.
3	Commonly used as part of Deployments, which offer additional benefits like rolling updates, rollbacks, and pause/resume functionality.	Not integrated with Deployments.

## 5- What is a PersistentVolume (PV) in Kubernetes?

Ans- A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically through a StorageClass. It is independent of the Pod lifecycle.

**6- What is a PersistentVolumeClaim (PVC) in Kubernetes?**

Ans- A PersistentVolumeClaim (PVC) is a request for storage by a user. It is used to dynamically provision PersistentVolumes and bind them to Pods.

**7- What is a DaemonSet in Kubernetes?**

Ans- A DaemonSet ensures that a copy of a Pod is running on all (or some) nodes in the cluster. It is commonly used for logging, monitoring, and other node-specific tasks.

**8- What is kubelet?**

Ans- kubelet is an agent that runs on each Node in the Kubernetes cluster. It ensures that containers are running in Pods as expected and communicates with the Kubernetes API server.

**9- What is a kube-proxy?**

Ans- kube-proxy is a network proxy that runs on each Node in the cluster. It maintains network rules and handles traffic routing to ensure that services can communicate with each other.

**10- What is the Difference between liveness probe and readiness probe**

Ans-

S.No	Liveness Probe	Readiness Probe
1	<b>Liveness probes</b> are used to determine if a container is alive or stuck in a state from which it cannot recover. If the liveness probe fails, Kubernetes will kill the container, and the container will be subjected to its restart policy.	<b>Readiness probes</b> are used to determine if a container is ready to start accepting traffic. If a readiness probe fails, Kubernetes will remove the pod's IP address from the list of endpoints for the corresponding service, ensuring that no traffic is sent to the container until it is ready.
2	Restart the container if the probe fails	Mark the pod as "NotReady" without restarting the container
3	No direct impact on traffic routing	Controls whether the pod receives traffic

**11- Scenario: You notice a pod consistently hitting high CPU usage.**

**Ans-** First, identify the pod using `kubectl get pods`. Then, use `kubectl describe pod <pod_name>` to check resource requests and limits. You can also use `kubectl top pods` to see CPU usage. Analyze the pod's workload and optimize code or adjust resource requests/limits. Consider horizontal pod autoscaling (HPA) for automatic scaling based on CPU usage.

**12- Scenario: A service isn't receiving traffic despite being deployed.**

**Ans-** Verify the service definition using `kubectl get service <service_name>`. Check if the selector matches the pods' labels. Use `kubectl get endpoints <service_name>` to see if any pods are backing the service. Troubleshoot network connectivity issues, firewall rules, or load balancer configuration.

**13- Scenario: A rolling update of a deployment fails.**

Analyze the deployment logs using `kubectl get deployments <deployment_name> -o yaml`. Check for errors related to health checks, container images, or resource limitations. You can also use `kubectl rollout history deployment <deployment_name>` to see the history and rollback if necessary.

**14- Scenario: A pod is stuck in a crash loop backoff state.**

**Ans -** Identify the pod using `kubectl get pods` and check its logs using `kubectl logs <pod_name>`. The logs might reveal application errors or resource issues. Fix the underlying problem in the container image or adjust resource requests/limits.

**15- How would you scale an application in Kubernetes to handle increased traffic?**

**Ans-** To scale an application, you can adjust the number of replicas in your Deployment. Manually scale using `kubectl scale`:

```
bash
```

```
kubectl scale deployment/my-deployment --replicas=10
```

Alternatively, you can configure Horizontal Pod Autoscaler (HPA) to automatically scale based on CPU utilization or other metrics.

```

yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: my-app-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-deployment
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 80

```

This HPA configuration will automatically scale the Deployment up or down based on CPU utilization.

#### 16- How would you implement a canary deployment in Kubernetes?

Ans - A canary deployment can be implemented by creating a new Deployment with a smaller number of replicas alongside the existing Deployment.

```

yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app-canary
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: my-app
        version: canary
    spec:
      containers:
        - name: my-container
          image: my-image:v2

```

Update the Service to route traffic to both the stable and canary deployments.



```
yaml

apiVersion: v1
kind: Service
metadata:
  name: my-app-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Gradually increase the replicas in the canary deployment while monitoring its performance.

**17- How would you deploy an application to multiple environments (e.g., dev, staging, prod) using Kubernetes?**

Ans- Use Kubernetes namespaces to separate environments and customize configurations using ConfigMaps and Secrets for each environment.

```
bash

kubectl create namespace dev
kubectl create namespace staging
kubectl create namespace prod
```

Example Deployment in different namespaces:

```
bash

kubectl apply -f deployment-dev.yaml -n dev
kubectl apply -f deployment-staging.yaml -n staging
kubectl apply -f deployment-prod.yaml -n prod
```

This allows you to deploy the same application to different environments with environment-specific configurations.

**18- How would you implement a blue-green deployment strategy in Kubernetes?**

Ans- Create two separate Deployments (blue and green) and switch traffic between them using a Service.

Create blue and green Deployments:

yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app-blue
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: my-app
        version: blue
    spec:
      containers:
      - name: my-container
        image: my-image:v1
```

yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app-green
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: my-app
        version: green
    spec:
      containers:
      - name: my-container
        image: my-image:v2
```


Use a Service to route traffic to the current version (e.g., blue):

yaml

```
apiVersion: v1
kind: Service
metadata:
  name: my-app-service
spec:
  selector:
    app: my-app
    version: blue
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

Switch traffic to the green Deployment by updating the Service selector:

bash

 Copy code

```
kubectl patch service my-app-service -p '{"spec": {"selector": {"version": "green"}}}'
```

## 19- How to adjust the resource usage limits in Kubernetes?

Ans- In Kubernetes, you can adjust the resource usage limits of Pods using resource requests and limits. These settings ensure that your containers get the CPU and memory resources they need to function properly while preventing any single container from monopolizing resources.

Define Resource Requests and Limits in Pod Spec

You can specify resource requests and limits in the container specification within your Pod or Deployment YAML file.

```

apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: my-image:latest
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"

```

In this example:

- The container is guaranteed 64Mi of memory and 250m (0.25) of CPU.
- The container can use up to 128Mi of memory and 500m (0.5) of CPU.

Applying Resource Requests and Limits to a Deployment

You can also apply these settings to a Deployment, which manages a set of Pods.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-container
        image: my-image:latest
        resources:
          requests:
            memory: "64Mi"
            cpu: "250m"
          limits:
            memory: "128Mi"
            cpu: "500m"

```

## Checking Resource Usage

You can check the current resource usage and allocation using **kubectl** top commands:

```
bash

kubectl top pods
kubectl top nodes
```

To update the resource limits of a running deployment:  
Edit the deployment YAML file:

```
yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 3
  template:
    spec:
      containers:
      - name: my-container
        image: my-image:latest
        resources:
          requests:
            memory: "128Mi"
            cpu: "500m"
          limits:
            memory: "256Mi"
            cpu: "1"
```

Apply the updated configuration:

```
bash

kubectl apply -f my-deployment.yaml
```

## 20- A pod is stuck in a Pending state. How would you troubleshoot this issue?

Ans- Check Pod Description: (**kubectl describe pod <pod-name>**)

Check Node Resources: (**kubectl describe nodes**)

Check for Node Affinity/Anti-Affinity: Verify if there are any node affinity or anti-affinity rules that are preventing the pod from being scheduled.

**Check Taints and Tolerations:** Ensure that the pod has the necessary tolerations for any taints on the nodes.

### **kubectl describe nodes | grep -i taints**

Check Resource Quotas: Verify if there are any resource quotas in the namespace that are preventing the pod from being scheduled.

### **kubectl get resourcequotas**

## **21- A pod is continuously crashing and restarting. What steps would you take to diagnose the problem?**

Ans - Check Pod Logs: Get logs from the crashing pod to see any application errors.

### **kubectl logs <pod-name>**

For previous logs: **kubectl logs <pod-name> --previous**

Describe the Pod: Check the events and status for any clues.

### **kubectl describe pod <pod-name>**

Check Container Exit Code: Look at the exit code of the container to understand why it is crashing.

```
kubectl get pod <pod-name> -o jsonpath='{.status.containerStatuses[0].lastState.terminated.exitCode}'
```

Check Readiness and Liveness Probes: Misconfigured probes can cause containers to be killed and restarted.

### **kubectl describe pod <pod-name> | grep -i liveness**

### **kubectl describe pod <pod-name> | grep -i readiness**

Check Resource Limits:

### **kubectl describe pod <pod-name> | grep -i limits**

## **22- A node is marked as Not Ready. How would you troubleshoot this issue?**

Ans- Describe the Node: Check for events and conditions.

### **kubectl describe node <node-name>**

Check Kubelet Logs: Inspect the kubelet logs on the node for errors.

### **ssh <node-name>**

### **journalctl -u kubelet**

Check Disk Pressure:

Verify if the node is under disk pressure.

### **kubectl describe node <node-name> | grep -i pressure**

Check Network Connectivity: Ensure the node has network connectivity to the master and other nodes.

Check Node Status: Look for taints and other status conditions.

**kubectl get nodes <node-name> -o jsonpath='{.status.conditions}'**

**23- A pod cannot communicate with another pod in a different namespace. How would you troubleshoot this issue?**

Ans- Check Network Policies: Ensure there are no Network Policies blocking the traffic.

**kubectl get networkpolicies --all-namespaces**

Check Service and DNS:

Ensure the DNS service is working correctly and resolving the service names.

**kubectl exec <pod-name> -- nslookup <service-name>.<namespace>.svc.cluster.local**

Check Pod Network:

Verify the CNI (Container Network Interface) plugin is correctly configured and working.

**kubectl get pods -n kube-system | grep -i cni**

Check Firewall Rules:

Ensure there are no firewall rules blocking the communication between namespaces.

**24- You are unable to delete a namespace. What could be the possible reasons and how would you resolve it?**

Ans - Check for Finalizers: The namespace might have finalizers that prevent deletion.

**kubectl get namespace <namespace-name> -o jsonpath='{.spec.finalizers}'**

Remove Finalizers: Remove the finalizers manually if they are stuck.

**kubectl patch namespace <namespace-name> -p '{"spec":{"finalizers":null}}'**

Check for Resources: Ensure all resources within the namespace are deleted.

**kubectl get all -n <namespace-name>**

Check for Stuck Resources: Sometimes, resources can get stuck in terminating state.

**kubectl get pods -n <namespace-name> | grep Terminating**

Force Delete Stuck Resources:

**kubectl delete pod <pod-name> --grace-period=0 --force -n <namespace-name>**

**25- ConfigMap changes are not reflected in the running pods. What could be the issue?**

**Ans- Check ConfigMap: Ensure the ConfigMap has been updated correctly.**

**kubectl get configmap <configmap-name> -o yaml**

Restart Pods: Pods need to be restarted to pick up changes in the ConfigMap.

**kubectl rollout restart deployment <deployment-name> -n <namespace>**

Check Volume Mounts: If using volume mounts, ensure they are correctly configured to use the ConfigMap.

yaml

```
volumeMounts:
- name: config-volume
  mountPath: /etc/config
volumes:
- name: config-volume
  configMap:
    name: <configmap-name>
```