Introduction of Terraform,

Why Use Terraform?,

What Can You Do with Terraform?

Terraform Installation Documentation,

Step-by-Step Configuration Guide(With AWS EC2 Instance),

Terraform commands, Terraform Best Practices,

Terraform Configurations with AWS Azure, GCP, Oracle,

1. Subnet Configuration, 1.1 Subnet Configuration (Public and Private),

2. Networking Services, 3. COMPUTE, 4. Platform as a Service (PaaS),

5. Object Storage, 6. Block Storage, 7. Database as a Service (DBaaS),

8. Serverless Computing, 9. Content Delivery Network (CDN),

10. Container Orchestration, 11. Identity and Access Management,

12. Load Balancing, 13. API Management, 14. Monitoring and Management,

15. Message Queuing and Event Streaming, 16. Security and Compliance,

17. Data Warehousing, 18. Backup and Recovery, 19. CI/CD Services,

20. Infrastructure as Code, 21. Container Services, 22. Configuration Management,

23. Developer Tools, 24. Business Analytics and Reporting, 25. Artificial Intelligence (AI) and
Natural Language Processing (NLP), 26. Machine Learning

Introduction of Terraform

Terraform is an open-source tool that allows you to define and manage infrastructure as code. It automates the creation, updating, and management of resources (e.g., virtual machines, networks, and databases) across various cloud providers (AWS, Azure, Google Cloud). You define the desired state of your infrastructure in configuration files, and Terraform handles provisioning and management.

Why Use Terraform?

Easy Setup: Write a list of what you need, and Terraform takes care of creating it.

Automation: Terraform can make changes, remove old resources, or update things automatically.

Consistency: Ensures all environments (like testing, staging, and production) are set up the same way.

Works with Everything: Supports most cloud platforms (AWS, Azure, Google Cloud) and many other tools.

Team Collaboration: Multiple people can work on the same setup without conflicts.

What Can You Do with Terraform?

Manage multiple clouds in one place.

Create infrastructure faster and easier.

Follow security and cost rules automatically.

Help teams deploy resources on their own safely.

Terraform Installation Documentation:

Official link:

https://developer.hashicorp.com/terraform/install

Ubuntu/Debian

```
wget -O - https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg
```

```
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list
```

```
sudo apt update && sudo apt install terraform
```

AWS CLI

AWS CLI Installation link:

https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

---

To install the AWS CLI ON UBUNTU, run the following commands.

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
```

```
unzip awscliv2.zip
```

```
sudo ./aws/install
```

Kubectl Installation link:(for kubernetes) on ubuntu

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
```

Step-by-Step Configuration Guide(With AWS EC2 Instance)

1. Initial Configuration: main.tf

The main.tf file is where you define your AWS provider and the resources you want to create, like an EC2 instance.

Terraform Configuration:

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.16"
    }
  }
  required_version = ">= 1.2.0"
}

provider "aws" {
  region = "us-west-2"
}

resource "aws_instance" "app_server" {
  ami           = "ami-08d70e59c07c61a3a"
  instance_type = "t2.micro"

  tags = {
    Name = var.instance_name
  }
}
```

- **Terraform Block**: Sets the required AWS provider and Terraform version.

- **Provider "aws"**: Defines AWS as the cloud provider and sets the region (us-west-2).
- **Resource "aws_instance"**: Creates an EC2 instance using the specified Amazon Machine Image (AMI) and instance type. It also adds a name tag from the variable instance_name.

---

2. Input Variables: variables.tf

The variables.tf file allows you to define input variables, making your Terraform configuration more flexible.

Example Input Variable Configuration:

```
variable "instance_name" {
  description = "Value of the Name tag for the EC2 instance"
  type        = string
  default     = "ExampleAppServerInstance"
}
```

- **instance_name Variable**: This variable is used to set the Name tag of the EC2 instance. You can change this value in your configuration without editing main.tf.

3. Output Values: outputs.tf

The outputs.tf file defines output values that show useful information after your resources are created.

Example Output Configuration:

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

output "instance_public_ip" {
  description = "Public IP address of the EC2 instance"
  value       = aws_instance.app_server.public_ip
}
```

- **instance_id**: Displays the EC2 instance's unique ID.
- **instance_public_ip**: Displays the public IP address of the EC2 instance.

4. Running the Configuration

To create and manage your AWS EC2 instance using Terraform, follow these steps:

Initialize Terraform: Make sure you're in the correct directory (where your main.tf, variables.tf, and outputs.tf files are stored) and initialize your Terraform configuration:

```
terraform init
```

Apply the Configuration: Use terraform apply to create the EC2 instance. Confirm the changes by typing yes when prompted:

```
terraform apply
```

After applying, Terraform will output the instance_id and instance_public_ip.

Inspect Output Values: To view the output values, run the following command:

```
terraform output
```

Destroy the Infrastructure: To delete all resources created by Terraform, run:

```
terraform destroy
```

Confirm by typing yes when prompted.

5. Terraform Advanced Configuration Use Cases

Provider Configuration: Specifies the cloud provider and region.

```
provider "aws" {
  region = "us-west-2"
}
```

Resource Creation: Defines infrastructure resources like an EC2 instance.

```
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"

  tags = {
    Name = "ExampleInstance"
  }
}
```

Variable Management: Use variables to centralize configuration and manage resources more easily.

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

provider "aws" {
  region = var.region
}
```

Output Values: Useful for exporting information about created resources for other configurations or visibility.

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

State Management: Terraform stores the state of resources in a .tfstate file. You can also use a remote backend, like an S3 bucket, to store this state.

```
terraform {
  backend "s3" {
    bucket         = "my-tfstate-bucket"
    key            = "terraform/state"
    region         = "us-west-2"
    encrypt        = true
    dynamodb_table = "terraform-locks"
  }
}
```

Modules: Encapsulate reusable configurations into modules for easy management.

```
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"
  name   = "my-vpc"
  cidr   = "10.0.0.0/16"

  azs             = ["us-west-2a", "us-west-2b"]
  public_subnets  = ["10.0.1.0/24", "10.0.2.0/24"]
  private_subnets = ["10.0.3.0/24", "10.0.4.0/24"]
}
```

Provisioners: Execute scripts or commands on your resources.

```
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"

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

Data Sources: Fetch information about existing infrastructure.

```
data "aws_ami" "example" {
  most_recent = true

  filter {
    name   = "name"
    values = ["amzn-ami-hvm-*"]
  }

  owners = ["137112412989"] # Amazon
}
```

Dynamic Blocks: Create multiple instances of nested blocks dynamically.

```
resource "aws_security_group" "example" {
  name = "example"

  dynamic "ingress" {
    for_each = var.ingress_rules
    content {
      from_port   = ingress.value.from_port
      to_port     = ingress.value.to_port
      protocol    = ingress.value.protocol
      cidr_blocks = ingress.value.cidr_blocks
    }
  }
}
```

Lifecycle Rules: Manage the lifecycle of resources, such as preventing destruction.

```
resource "aws_s3_bucket" "example" {
  bucket = "example-bucket"

  lifecycle {
    prevent_destroy = true
  }
}
```

6. Environment Variables

Set environment variables to pass sensitive information, such as AWS credentials:

```
export AWS_ACCESS_KEY_ID="your-access-key-id"
export AWS_SECRET_ACCESS_KEY="your-secret-access-key"
```

This guide provides a comprehensive overview for setting up and managing AWS EC2 instances with Terraform, from basic to advanced configurations. With these steps, you'll be able to efficiently manage your infrastructure using Terraform.

Terraform commands:

1. **terraform init**:
   - Initializes the working directory containing Terraform configuration files. It installs required provider plugins and sets up the backend for storing the state file.
2. **terraform fmt**:
   - Automatically formats the Terraform configuration files to ensure consistent style and indentation.
3. **terraform validate**:
   - Validates the configuration files to check for syntax errors and whether the configuration is logically correct.

4. **terraform plan**:
   - Previews changes that Terraform will make to the infrastructure based on the current configuration.
5. **terraform apply**:
   - Applies the changes required to reach the desired state defined in the configuration files.
6. **terraform show**:
   - Displays the current state of the infrastructure, typically showing a human-readable version of the state file.
7. **terraform destroy**:
   - Destroys the infrastructure managed by Terraform and removes it from the state.
8. **terraform state list**:
   - Lists all resources currently managed by Terraform in the state file.
9. **terraform state show <resource>**:
   - Displays detailed information about a specific resource in the state file.
10. **terraform state rm <resource>**:
    - Removes a resource from the Terraform state without destroying the resource itself.
11. **terraform taint <resource>**:
    - Marks a resource for recreation during the next terraform apply. It forces Terraform to recreate a resource.
12. **terraform untaint <resource>**:
    - Removes the "tainted" state from a resource, meaning it will no longer be recreated during the next apply.
13. **terraform import <resource> <resource_id>**:
    - Imports existing infrastructure into Terraform's state. Useful for bringing resources that were not created by Terraform under Terraform management.
14. **terraform providers**:
    - Lists the providers used in the configuration and their version constraints.
15. **terraform providers mirror**:
    - Downloads and caches the providers locally for offline use.
16. **terraform refresh**:
    - Updates the state with the latest values from the actual infrastructure without applying any changes.
17. **terraform version**:
    - Displays the installed version of Terraform.
18. **terraform force-unlock <LOCK_ID>**:
    - Forces the unlock of a Terraform state lock, typically used when a Terraform process was interrupted and left the state file locked.
19. **terraform state mv <source> <destination>**:

- Moves a resource within the state file from one address to another.

20. **terraform state pull**:
    - Retrieves the latest state file from the backend and outputs it to standard output.
21. **terraform state push <state_file>**:
    - Pushes a local state file to the remote backend.
22. **terraform fmt -check**:
    - Checks if the configuration files are formatted correctly without making changes.
23. **terraform graph**:
    - Generates a visual representation of the resources and their dependencies.
24. **terraform graph -type=plan**:
    - Generates a graph based on the execution plan to visualize what changes will happen when terraform apply is run.
25. **terraform output**:
    - Shows the values of outputs defined in the Terraform configuration.
26. **terraform workspace**:
    - Manages multiple workspaces for different environments.
27. **terraform workspace list**:
    - Lists all available workspaces in the current configuration.
28. **terraform workspace select <workspace>**:
    - Switches to a different workspace, making it the current workspace.
29. **terraform workspace new <workspace>**:
    - Creates a new workspace.
30. **terraform plan -out=planfile**:
    - Saves the execution plan to a file, which can later be applied with terraform apply planfile.
31. **terraform plan -detailed-exitcode**:
    - Runs the plan command and returns an exit code indicating the state of the infrastructure:
        - Exit code 0: No changes required.
        - Exit code 1: An error occurred.
        - Exit code 2: Changes are required.
32. **terraform console**:
    - Opens an interactive console to query the current state or experiment with expressions.
33. **terraform console -state=<path>**:
    - Opens an interactive console for querying the state file at a specified path.
34. **terraform state list**:
    - Lists all the resources in the current state file.
35. **terraform validate -var-file=<filename>**:

- Validates the configuration files using variable values from a specific file, such as terraform.tfvars.

36. **terraform validate -check-variables=true**:
    - Ensures the values provided for variables are valid, in addition to syntax validation.
37. **terraform output <output_name>**:
    - Displays a specific output value from the configuration.
38. **terraform providers lock**:
    - Updates the provider lock file to ensure consistency across different environments.
39. **terraform destroy -auto-approve**:
    - Automatically approves and destroys the infrastructure without requiring manual confirmation.
40. **terraform refresh**:
    - Refreshes the state with the most recent values from the infrastructure.
41. **terraform fmt -write=false**:
    - Checks if the configuration files are correctly formatted, without writing changes.
42. **terraform import**:
    - Imports an existing resource into the state.
43. **terraform destroy -target=<resource>**:
    - Destroys a specific resource without affecting others.
44. **terraform console -help**:
    - Displays help information for the Terraform console command.

Terraform Best Practices:

1. **Use Version Control**: Store your Terraform code in Git.
2. **Use Modules**: Break your code into reusable modules.
3. **Remote State**: Store state files remotely (e.g., AWS S3, Terraform Cloud).
4. **Use Variables**: Make your code flexible with variables.
5. **Use Data Sources**: Avoid hardcoding values; use data sources to fetch information.
6. **Separate Environments**: Use separate directories or workspaces for dev, staging, and prod.
7. **Run Plan First**: Always run terraform plan before terraform apply to check changes.
8. **Use Outputs**: Show important data like IPs using outputs.
9. **Lock Provider Versions**: Set specific provider versions to avoid unexpected changes.
10. **Avoid Hardcoding Secrets**: Use environment variables or secret management tools for sensitive data.

11. **Use terraform fmt & validate**: Format and validate your code for readability and correctness.
12. **Manage State Carefully**: Do not commit state files to Git, and keep them secure.
13. **Detect Drift**: Regularly run terraform plan to spot configuration drift.
14. **Minimize Dependencies**: Keep resource dependencies simple.
15. **Use terraform import**: Import existing resources into Terraform without recreating them.
16. **Document Your Code**: Add comments to explain your configurations.
17. **Use State Backends**: Store state files on a remote backend for collaboration (e.g., S3).
18. **Write Tests for Modules**: Test your Terraform modules with tools like Terratest.
19. **Use Workspaces for Environment Isolation**: Keep different environments isolated using workspaces.
20. **Keep Resources Organized**: Group related resources together in directories.
21. **Version Locking for Dependencies**: Lock the versions of modules and providers.
22. **Implement CI/CD for Terraform**: Automate terraform plan and terraform apply in CI/CD pipelines.
23. **Use terraform taint**: Mark resources for recreation if they're in a bad state.
24. **Review the Plan Output**: Always check the terraform plan output before applying changes.
25. **Avoid force_new**: Only use force_new when it's absolutely necessary.
26. **Minimize Hardcoding Values**: Use variables and data sources instead of hardcoded values.
27. **Refactor Over Time**: Regularly improve and refactor your Terraform code.
28. **Use terraform state rm**: Remove resources from the state if needed without destroying them
29. **Use Module Versions**: Lock specific versions of your modules.
30. **Use Terraform Cloud's Plans and Policies**: Set up policy checks in Terraform Cloud to enforce standards.
31. **Use terraform graph**: Visualize dependencies between resources to understand relationships.
32. **Validate Infrastructure Before Deployment**: Use terraform validate to check for errors.
33. **Simplify Conditional Logic**: Avoid complex conditionals that make code hard to read.
34. **Automate Cleanup**: Regularly clean up unused resources to save costs.
35. **Consider Resource Limits**: Be mindful of cloud provider limits and adjust configurations.
36. **Implement Drift Management**: Regularly check for and address configuration drift.
37. **Analyze Dependency Graph**: Use terraform graph to optimize resource creation order.
38. **Use Outputs for Documentation**: Display key information as outputs for easy reference.
39. **Use Data Sources for Reusability**: Fetch values that can be reused multiple times.

40. **Lock Provider and Module Versions**: Ensure stability with locked versions in .terraform.lock..
41. **Automate Testing**: Regularly test your Terraform code using tools like Terratest.
42. **Set Up Workflows for Alerts**: Set up alerts in your CI/CD pipeline for errors.
43. **Monitor Resource Usage**: Track your cloud resources to optimize cost and performance.
44. **Document State Backups**: Regularly back up your state files.
45. **Integrate with Other Tools**: Combine Terraform with tools like Ansible for broader management.
46. **Test with Varying Inputs**: Use different input values to test your code with varied data.
47. **Plan for Rollbacks**: Have a rollback plan for any issues during deployment.
48. **Use Change Automation**: Automate change management processes, like tagging resources.
49. **Monitor Terraform Runs**: Keep an eye on Terraform executions to catch issues quickly.
50. **Audit Terraform Configurations**: Periodically review your code for improvements and best practices.

---

1. Terraform Configurations with AWS Azure, GCP, Oracle

1. Subnet Configuration

Official Terraform Hashicorp link:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/subnet

1. AWS:

```
provider "aws" {
  region = "us-east-1"  # specify your region
}

resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"
}

resource "aws_subnet" "subnet" {
  vpc_id     = aws_vpc.main.id
  cidr_block = "10.0.1.0/24"
  availability_zone = "us-east-1a"  # specify availability zone
  map_public_ip_on_launch = true
}
```

---

2. Azure:

Official Terraform Hashicorp link:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/subnet

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

resource "azurerm_virtual_network" "main" {
  name                = "example-vnet"
  location            = "East US"
  resource_group_name = "example-resources"
  address_space       = ["10.0.0.0/16"]
}

resource "azurerm_subnet" "subnet" {
  name                 = "example-subnet"
  resource_group_name  = azurerm_virtual_network.main.resource_group_name
  virtual_network_name = azurerm_virtual_network.main.name
  address_prefixes     = ["10.0.1.0/24"]
}
```

---

3. GCP:

Official Terraform Hashicorp link:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/compute_subnetwork.html

```
provider "google" {
  project = "your-project-id"
  region  = "us-central1"
}

resource "google_compute_network" "vpc" {
  name = "example-vpc"
}

resource "google_compute_subnetwork" "subnet" {
  name          = "example-subnet"
  region        = "us-central1"
  network       = google_compute_network.vpc.id
  ip_cidr_range = "10.0.1.0/24"
  private_ip_google_access = true
}
```

---

4. Oracle Cloud:

Official Terraform Hashicorp link:

https://registry.terraform.io/providers/oracle/oci/latest/docs/resources/core_subnet

```
provider "oci" {
  tenancy_ocid     = "your-tenancy-id"
  user_ocid        = "your-user-id"
  fingerprint      = "your-fingerprint"
  private_key_path = "path/to/your/private-key.pem"
  region           = "us-phoenix-1"
}

resource "oci_core_vcn" "vcn" {
  cidr_block = "10.0.0.0/16"
  compartment_id = "your-compartment-id"
  display_name = "example-vcn"
}

resource "oci_core_subnet" "subnet" {
  cidr_block = "10.0.1.0/24"
  compartment_id = oci_core_vcn.vcn.compartment_id
  vcn_id = oci_core_vcn.vcn.id
  display_name = "example-subnet"
  availability_domain = "Uocm:PHX-AD-1"
}
```

Steps for Configuration:

1. **Set up your provider credentials** (API keys, configuration files, etc.) for each cloud provider.
2. **Create the appropriate VPC/network** resource before creating the subnet.
3. **Apply the Terraform configuration** to create the subnet with:

```
terraform init
terraform plan
terraform apply
```

1.1 Subnet Configuration (Public and Private)

AWS

```
provider "aws" {

  region = "us-west-2"

}


# VPC creation
```

```
resource "aws_vpc" "main_vpc" {

  cidr_block = "10.0.0.0/16"

}


# Public subnet creation

resource "aws_subnet" "public_subnet" {

  vpc_id                  = aws_vpc.main_vpc.id

  cidr_block              = "10.0.1.0/24"

  availability_zone       = "us-west-2a"

  map_public_ip_on_launch = true

  tags = {

    Name = "Public Subnet"

  }

}


# Private subnet creation

resource "aws_subnet" "private_subnet" {

  vpc_id                  = aws_vpc.main_vpc.id

  cidr_block              = "10.0.2.0/24"

  availability_zone       = "us-west-2b"

  tags = {

    Name = "Private Subnet"

  }
}
```

Azure Subnet Configuration (Public and Private)

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

# Virtual Network creation
resource "azurerm_virtual_network" "main_vnet" {
  name                = "main-vnet"
  location            = "East US"
  resource_group_name = "myResourceGroup"
  address_space       = ["10.0.0.0/16"]
}

# Public subnet creation
resource "azurerm_subnet" "public_subnet" {
  name                 = "public-subnet"
  resource_group_name  = "myResourceGroup"
  virtual_network_name = azurerm_virtual_network.main_vnet.name
  address_prefixes     = ["10.0.1.0/24"]
}
```

```
# Private subnet creation

resource "azurerm_subnet" "private_subnet" {

  name                 = "private-subnet"

  resource_group_name  = "myResourceGroup"

  virtual_network_name = azurerm_virtual_network.main_vnet.name

  address_prefixes     = ["10.0.2.0/24"]

}
```

GCP Subnet Configuration (Public and Private)

```
provider "google" {

  project = "my-gcp-project"

  region  = "us-west1"

}


# VPC creation

resource "google_compute_network" "main_vpc" {

  name                    = "main-vpc"

  auto_create_subnetworks = false

}


# Public subnet creation

resource "google_compute_subnetwork" "public_subnet" {

  name          = "public-subnet"
  region        = "us-west1"
  network       = google_compute_network.main_vpc.name
  ip_cidr_range = "10.0.1.0/24"
  private_ip_google_access = true
}

# Private subnet creation
resource "google_compute_subnetwork" "private_subnet" {
  name          = "private-subnet"
  region        = "us-west1"
  network       = google_compute_network.main_vpc.name
  ip_cidr_range = "10.0.2.0/24"
  private_ip_google_access = false
}
```

Oracle Cloud Subnet Configuration (Public and Private)

```
provider "oci" {
  region = "us-phoenix-1"
}

# Virtual Cloud Network (VCN) creation
resource "oci_core_virtual_network" "main_vcn" {
  compartment_id = "ocid1.compartment.oc1..xxxxxEXAMPLExxxxx"
  cidr_block     = "10.0.0.0/16"
  display_name   = "Main VCN"
}

# Public subnet creation
resource "oci_core_subnet" "public_subnet" {
  compartment_id      = "ocid1.compartment.oc1..xxxxxEXAMPLExxxxx"
  vcn_id              = oci_core_virtual_network.main_vcn.id
  cidr_block          = "10.0.1.0/24"
  display_name        = "Public Subnet"
  availability_domain = "Uocm:PHX-AD-1"
  route_table_id      = "ocid1.routetable.oc1..xxxxxEXAMPLExxxxx"
}

# Private subnet creation
resource "oci_core_subnet" "private_subnet" {
  compartment_id      = "ocid1.compartment.oc1..xxxxxEXAMPLExxxxx"
  vcn_id              = oci_core_virtual_network.main_vcn.id
  cidr_block          = "10.0.2.0/24"
  display_name        = "Private Subnet"
  availability_domain = "Uocm:PHX-AD-2"
  route_table_id      = "ocid1.routetable.oc1..xxxxxEXAMPLExxxxx"
}
```

Notes:

- **AWS**: For public subnets, the map_public_ip_on_launch property is set to true to allow instances launched in this subnet to get public IPs.
- **Azure**: For Azure, subnets are created with specific address prefixes.
- **GCP**: In GCP, the subnet type is determined by whether private_ip_google_access is set to true (for private subnet) or false (for public).
- **Oracle Cloud**: Oracle's subnet configuration requires defining the route_table_id, typically pointing to a route table for each subnet type (public or private).

---

2. **Networking Services**

- **AWS:** VPC (Virtual Private Cloud)
- **Azure:** Virtual Network
- **GCP:** VPC (Virtual Private Cloud)
- **Oracle Cloud:** Oracle Cloud Virtual Cloud Network (VCN)

1. AWS: Virtual Private Cloud (VPC)

Official Terraform Hashicorp Documentation:
https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/vpc

Introduction:
AWS VPC allows you to create isolated network environments within AWS. You can define your own IP address range, create subnets, and configure route tables, network gateways, and security settings to control access and communication within your network.

Terraform Configuration:

```
provider "aws" {

  region = "us-west-2"

}
```

```
resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"
  enable_dns_support = true
  enable_dns_hostnames = true
}
```

---

2. Azure: Virtual Network

Official Terraform Hashicorp Documentation:
https://registry.terraform.io/providers/hashicorp/azurerm/4.0.0/docs/resources/virtual_network

Introduction:
Azure Virtual Network (VNet) provides private network functionality to your Azure resources, allowing secure communication between them. You can segment the network into subnets, control traffic flow, and configure security rules.

Terraform Configuration:

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


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

---

3. GCP: Virtual Private Cloud (VPC)

Official Terraform Hashicorp Documentation:
https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/compute_network

Introduction:
GCP VPC allows you to create a private network with global reach, spanning across multiple regions. You can control your IP range, routing, and configure firewalls for access control.

Terraform Configuration:

```
provider "google" {
  region = "us-central1"
}


resource "google_compute_network" "default" {
  name = "my-vpc"
  auto_create_subnetworks = true
}
```

---

4. Oracle Cloud: Virtual Cloud Network (VCN)

Official Terraform Hashicorp Documentation:
https://registry.terraform.io/providers/oracle/oci/latest/docs/resources/core_vcn

Introduction:
Oracle Cloud's VCN provides a logically isolated network environment in Oracle Cloud, with

the ability to create subnets, route tables, internet gateways, and configure security lists and policies.

Terraform Configuration:

```
provider "oci" {
  tenancy_ocid = "your_tenancy_ocid"
  user_ocid    = "your_user_ocid"
  fingerprint  = "your_fingerprint"
  private_key_path = "your_private_key.pem"
}


resource "oci_core_virtual_network" "main" {
  compartment_id = "your_compartment_id"
  display_name   = "my_vcn"
  cidr_block     = "10.0.0.0/16"
```

---

3. COMPUTE

- **AWS:** EC2 (Elastic Compute Cloud)
- **Azure:** Virtual Machines
- **GCP:** Compute Engine
- **Oracle Cloud:** Compute

Infrastructure as a Service (IaaS) provides virtualized computing resources over the internet. It allows users to rent computing infrastructure such as virtual machines, storage, and networking on-demand. IaaS is flexible, scalable, and helps reduce the capital expenses associated with physical hardware.

Here’s a brief overview of IaaS offerings from major cloud providers with Terraform configuration examples:

1. AWS EC2 (Elastic Compute Cloud)

- EC2 is a scalable compute service that lets users run virtual servers (instances) in the cloud.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance

Terraform Configuration:

```
provider "aws" {
  region = "us-east-1"
}


resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
}
```

---

2. Azure Virtual Machines

- Azure Virtual Machines provide on-demand scalable computing resources with full control over the OS.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/virtual_machine

Terraform Configuration:

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


resource "azurerm_virtual_machine" "example" {
  name                  = "example-vm"
  location              = "East US"
  resource_group_name   = "example-resources"
  size                  = "Standard_B1s"
  network_interface_ids = [azurerm_network_interface.example.id]
  vm_os_simple {
    license_type = "Windows_Client"
  }
}
```

---

3. GCP Compute Engine

- Google Cloud's Compute Engine allows users to run virtual machines in Google's data centers with high performance.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/compute_instance

Terraform Configuration:

```
provider "google" {
  project = "your-project-id"
  region  = "us-central1"
}

resource "google_compute_instance" "default" {
  name         = "example-instance"
  machine_type = "f1-micro"
  zone         = "us-central1-a"
  boot_disk {
    initialize_params {
      image = "debian-9-stretch-v20190729"
    }
  }
  network_interface {
    network = "default"
    access_config {
      // Include this block to assign an external IP
    }
  }
}
```

---

4. Oracle Cloud Compute

- Oracle Cloud Infrastructure (OCI) Compute lets you run virtual machines in the cloud with high performance and security.

  Official Terraform Hashicorp Documentation:

  https://registry.terraform.io/providers/oracle/oci/latest/docs/resources/core_instance

Terraform Configuration:

```
provider "oci" {
  tenancy_ocid       = "ocid1.tenancy.oc1..example"
  user_ocid          = "ocid1.user.oc1..example"
  fingerprint        = "your-fingerprint"
  private_key_path   = "~/.oci/oci_api_key.pem"
  region             = "us-phoenix-1"
}

resource "oci_core_instance" "example" {
  availability_domain = "Uocm:PHX-AD-1"
  compartment_id      = "ocid1.compartment.oc1..example"
  shape               = "VM.Standard2.1"

  display_name        = "example-instance"

  image               = "ocid1.image.oc1..example"

}
```

---

4. Platform as a Service (PaaS)

- **AWS:** Elastic Beanstalk
- **Azure:** App Service
- **GCP:** App Engine
- **Oracle Cloud:** Oracle Cloud Applications (for SaaS and PaaS solutions)

Platform as a Service (PaaS) provides a framework for developers to build, run, and manage applications without worrying about the underlying infrastructure. It abstracts the operating system and infrastructure layers, allowing developers to focus solely on the application code. PaaS offerings from different cloud providers help in automating deployment, scaling, and management of applications.

Here are brief introductions to PaaS offerings from major cloud providers and their Terraform configurations:

AWS: Elastic Beanstalk

Elastic Beanstalk is a fully managed PaaS that makes it easy to deploy and manage applications in the cloud. It supports multiple programming languages such as Java, .NET, PHP, Node.js, and Python.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/elastic_beanstalk_environment

Terraform Configuration:

```
provider "aws" {

  region = "us-east-1"

}


resource "aws_elastic_beanstalk_application" "my_app" {

  name        = "my-app"

  description = "My Elastic Beanstalk application"

}


resource "aws_elastic_beanstalk_environment" "my_env" {

  name                = "my-app-env"

  application         = aws_elastic_beanstalk_application.my_app.name

  solution_stack_name = "64bit Amazon Linux 2 v3.3.6 running Node.js"

}
```

Azure: App Service

Azure App Service is a fully managed PaaS for building, deploying, and scaling web apps. It supports multiple languages and frameworks and provides features like autoscaling, security, and easy integration with other Azure services.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/app_service.html

Terraform Configuration:

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

resource "azurerm_app_service_plan" "example" {
  name                = "example-asp"
  location            = "East US"
  resource_group_name = "example-resources"
  kind                = "App"
  reserved            = false
  sku {
    tier = "Standard"
    size = "S1"
  }
}

resource "azurerm_web_app" "example" {
  name                = "example-web-app"
  location            = "East US"
  resource_group_name = "example-resources"
  app_service_plan_id = azurerm_app_service_plan.example.id
}
```

GCP: App Engine

Google App Engine is a fully managed platform for building and deploying applications. It provides automatic scaling, load balancing, and high availability.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/app_engine_application

Terraform Configuration:

```
provider "google" {

  project = "my-project"

  region  = "us-central1"

}


resource "google_app_engine_application" "example" {

  location_id = "us-central"

}


resource "google_app_engine_standard_app_version" "example" {

  service = "default"

  version_id = "v1"

  runtime = "python39"

  entrypoint = "python app.py"
}
```

Oracle Cloud: Oracle Cloud Applications (for SaaS and PaaS solutions)

Oracle Cloud offers a range of PaaS and SaaS solutions. Oracle PaaS includes services like Oracle Integration Cloud and Oracle Autonomous Database that help developers quickly build applications without managing the underlying infrastructure.

Oracle Autonomous Database

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/oracle/oci/latest/docs/data-sources/database_autonomous_db_versions

```
provider "oci" {
  region = "us-phoenix-1"
}


resource "oci_application" "example" {
  compartment_id = "ocid1.compartment.oc1..example"  # Replace with actual compartment OCID
  display_name   = "example-application"
  description    = "Oracle Cloud Application"
}
```

# Example for creating an Oracle Autonomous Database

```
resource "oci_database_autonomous_database" "example_db" {
  compartment_id       = "ocid1.compartment.oc1..example"  # Replace with actual compartment OCID
  db_name              = "exampledb"
  cpu_core_count       = 1
  db_workload          = "OLTP"
  admin_password       = "YourSecurePasswordHere"
  data_storage_size_in_tbs = 1

  # Optional: Create a backup policy
  backup_policy {
    enabled = true
    time_in_minutes = 15
  }
}
```

# Example for Oracle Integration Cloud

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/oracle/oci/latest/docs/resources/integration_integration_instance

```
resource "oci_application_integration_instance" "example_integration" {
  compartment_id = "ocid1.compartment.oc1..example"  # Replace with actual compartment OCID
  display_name   = "example-integration"
  description    = "Oracle Integration Cloud Instance"
  edition        = "STANDARD"

}
```

Key Components:

- **Oracle Cloud Application (oci_application)**: This creates an Oracle Cloud Application in a given compartment with a display name and description.
- **Oracle Autonomous Database (oci_database_autonomous_database)**: This resource configures an Autonomous Database, specifying CPU count, workload type, and storage.
- **Oracle Integration Cloud (oci_application_integration_instance)**: This resource creates an instance of Oracle Integration Cloud.

Notes:

- **Compartment OCID**: Replace ocid1.compartment.oc1..example with the actual compartment OCID from your Oracle Cloud account.
- **Admin Password**: Ensure the admin password meets the required security standards.
- **Edition**: Adjust the Oracle Integration Cloud edition based on your needs (STANDARD, ENTERPRISE, etc.).

---

5. Object Storage

- **AWS:** S3 (Simple Storage Service)
- **Azure:** Blob Storage
- **GCP:** Cloud Storage
- **Oracle Cloud:** Oracle Cloud Object Storage

1. AWS: S3 (Simple Storage Service)
AWS S3 is a scalable object storage service designed to store and retrieve any amount of data. It provides high durability, availability, and security for applications like backup, archiving, and data storage.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/s3_bucket

Terraform Configuration for AWS S3:

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_s3_bucket" "my_bucket" {
  bucket = "my-unique-bucket-name"
  acl    = "private"
}
```

---

2. Azure: Blob Storage

Azure Blob Storage is designed for storing large amounts of unstructured data, such as text, images, or binary data. It supports scalable, durable, and cost-effective storage solutions.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/storage_blob

Terraform Configuration for Azure Blob Storage:

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

```
resource "azurerm_storage_account" "my_storage_account" {

  name                     = "mystorageaccount"

  resource_group_name      = "my_resource_group"

  location                 = "East US"

  account_tier             = "Standard"

  account_replication_type = "LRS"

}


resource "azurerm_storage_container" "my_container" {

  name                  = "mycontainer"

  storage_account_name  = azurerm_storage_account.my_storage_account.name

  container_access_type = "private"

}
```

---

3. GCP: Cloud Storage
Google Cloud Storage offers object storage with scalability, low latency, and high availability. It's useful for large-scale data analytics, media, and backup storage.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/wiardvanrij/ipv4google/latest/docs/resources/storage_bucket

Terraform Configuration for GCP Cloud Storage:

```
provider "google" {
  project = "my-project-id"
  region  = "us-central1"
}

resource "google_storage_bucket" "my_bucket" {
  name          = "my-unique-bucket-name"
  location      = "US"
  force_destroy = true
}
```

---

4. Oracle Cloud: Oracle Cloud Object Storage

Oracle Cloud Object Storage provides scalable storage solutions for large volumes of data. It’s designed for security, high durability, and seamless integration with other Oracle Cloud services.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/oracle/oci/latest/docs/resources/objectstorage_bucket

Terraform Configuration for Oracle Cloud Object Storage:

```
provider "oci" {
  region = "us-phoenix-1"
}
```

```
resource "oci_objectstorage_bucket" "my_bucket" {

  compartment_id = "my-compartment-id"

  name           = "my-unique-bucket-name"

  storage_tier   = "Standard"

}
```

---

6. Block Storage

- **AWS:** EBS (Elastic Block Store)
- **Azure:** Azure Disk Storage
- **GCP:** Persistent Disks
- **Oracle Cloud:** Oracle Cloud Block Volumes

AWS: EBS (Elastic Block Store)

Introduction:
AWS Elastic Block Store (EBS) provides persistent block-level storage volumes for use with Amazon EC2 instances. EBS volumes are highly available and scalable, allowing you to store data persistently, independent of EC2 instances. They can be attached to instances, formatted, and mounted for use.

Official Terraform Hashicorp Documentation:
https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/ebs_volume

Terraform Configuration:

```
resource "aws_ebs_volume" "example" {

  availability_zone = "us-west-2a"

  size              = 10

  tags = {

    Name = "MyEBSVolume"
```

```
 }

}
```

Azure: Azure Disk Storage

Introduction:
Azure Disk Storage offers high-performance block storage for Azure Virtual Machines. It provides options for both standard and premium disks, supporting different use cases based on the needs of the workloads. Azure Disk Storage is durable and scalable, with managed and unmanaged disk options.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/managed_disk

Terraform Configuration:

```
resource "azurerm_managed_disk" "example" {

  name                 = "example-disk"

  resource_group_name  = "example-resources"

  location             = "East US"

  size                 = 10

  storage_account_type = "Standard_LRS"

}
```

GCP: Persistent Disks

Introduction:
Google Cloud Persistent Disks are durable, high-performance block storage devices that can be attached to Google Cloud Compute Engine instances. Persistent disks can be resized dynamically and provide both standard and SSD-backed storage for a variety of applications.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/compute_disk

Terraform Configuration:

```
resource "google_compute_disk" "example" {
  name  = "example-disk"
  type  = "pd-standard"
  size  = 10
  zone  = "us-central1-a"
}
```

Oracle Cloud: Oracle Cloud Block Volumes

Introduction:
Oracle Cloud Block Volumes offer high-performance, scalable storage that can be used with Oracle Cloud Compute instances. These volumes provide durable and flexible storage that can be attached and detached from instances as required, with both standard and high-performance options available.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/oracle/oci/latest/docs/resources/core_volume.html

Terraform Configuration:

```
resource "oci_core_volume" "example" {
  availability_domain = "Uocm:PHX-AD-1"
  compartment_id      = "ocid1.compartment.oc1..example"
  size_in_gbs         = 10
  display_name        = "example-block-volume"
}
```

---

7. Database as a Service (DBaaS)

- **AWS:** RDS (Relational Database Service)
- **Azure:** Azure SQL Database
- **GCP:** Cloud SQL
- **Oracle Cloud:** Oracle Autonomous Database

Database as a Service (DBaaS) Overview

Database as a Service (DBaaS) is a cloud-based service that provides database management and hosting solutions without the need for customers to handle the infrastructure. It simplifies database management by automating tasks such as backup, scaling, patching, and monitoring. Each cloud provider offers a DBaaS solution that caters to various types of databases, from relational to NoSQL.

AWS: RDS (Relational Database Service)

Introduction: AWS RDS is a fully managed relational database service that supports multiple database engines, including MySQL, PostgreSQL, Oracle, SQL Server, and MariaDB. It automates database management tasks like backups, software patching, and scaling.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/db_instance

Terraform Configuration:

```
resource "aws_db_instance" "example" {
  allocated_storage    = 20
  db_name              = "exampledb"
  engine               = "mysql"
  instance_class       = "db.t2.micro"
  username             = "admin"
  password             = "password123"
  parameter_group_name = "default.mysql5.7"
  multi_az             = false
  publicly_accessible = true
  tags = {
    Name = "example-db"
  }
}
```

Azure: Azure SQL Database

Introduction: Azure SQL Database is a fully managed relational database service in the Microsoft Azure cloud. It provides high availability, scalability, and automated backups for SQL Server databases.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/mssql_database

Terraform Configuration:

```
resource "azurerm_sql_server" "example" {
  name                         = "example-server"
  resource_group_name          = "example-resources"
  location                     = "East US"
  version                      = "12.0"
  administrator_login          = "adminuser"
  administrator_login_password = "password123"
}


resource "azurerm_sql_database" "example" {
  name                = "exampledb"
  resource_group_name = azurerm_sql_server.example.resource_group_name
  server_name         = azurerm_sql_server.example.name
  sku_name            = "S1"
  collation           = "SQL_Latin1_General_CP1_CI_AS"
}
```

GCP: Cloud SQL

Introduction: Google Cloud SQL is a fully managed relational database service that supports MySQL, PostgreSQL, and SQL Server. It offers automated backups, scaling, and high availability features.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/google/5.0.0/docs/resources/sql_database_instance

Terraform Configuration:

```
resource "google_sql_database_instance" "example" {

  name             = "example-db-instance"

  region           = "us-central1"

  database_version = "MYSQL_8_0"

  tier             = "db-f1-micro"


  root_password = "password123"

}


resource "google_sql_database" "example" {

  name     = "exampledb"

  instance = google_sql_database_instance.example.name

}
```

Oracle Cloud: Oracle Autonomous Database

Introduction: Oracle Autonomous Database is a fully managed database service that automates key database tasks like provisioning, patching, backup, and scaling. It is available in both transaction processing and data warehousing configurations.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/oracle/oci/latest/docs/data-sources/database_autonomous_database

Terraform Configuration:

```
resource "oci_db_autonomous_database" "example" {

  compartment_id = var.compartment_id

  db_name        = "exampledb"

  cpu_core_count = 1

  data_storage_size_in_tbs = 1

  db_workload    = "DW"

  admin_password = "password123"

  db_version     = "19c"

}
```

---

8. Serverless Computing

- **AWS:** Lambda
- **Azure:** Azure Functions
- **GCP:** Cloud Functions
- **Oracle Cloud:** Oracle Functions

Serverless Computing allows you to run applications without managing the infrastructure. Instead of provisioning, scaling, and managing servers, you write code and deploy it, and the cloud provider takes care of all the operational aspects, including scaling and infrastructure management. This enables developers to focus more on writing business logic and less on maintaining servers.

AWS Lambda

AWS Lambda lets you run code without provisioning or managing servers. It scales automatically and charges only for the compute time consumed.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/lambda_function

Terraform Configuration for AWS Lambda:

```
resource "aws_lambda_function" "example" {

  function_name = "example_lambda_function"

  handler       = "index.handler"

  runtime       = "nodejs14.x"

  role          = aws_iam_role.lambda_role.arn

  filename      = "function.zip"

}


resource "aws_iam_role" "lambda_role" {

  name = "lambda_role"


  assume_role_policy = jsonencode({

    Version = "2012-10-17"

    Statement = [

      {

        Action    = "sts:AssumeRole"
        Effect    = "Allow"
        Principal = {
          Service = "lambda.amazonaws.com"
        }
      }
    ]
  })
}
```

Azure Functions

Official Terraform Hashicorp Documentation:

Azure Functions allows you to run event-triggered code without having to explicitly provision or manage infrastructure. It integrates easily with other Azure services.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/function_app

Terraform Configuration for Azure Functions:

```
resource "azurerm_function_app" "example" {
  name                       = "example-function-app"
  location                   = "East US"
  resource_group_name        = azurerm_resource_group.example.name
  app_service_plan_id        = azurerm_app_service_plan.example.id
  storage_connection_string  = azurerm_storage_account.example.primary_connection_string
```

```
}

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

resource "azurerm_app_service_plan" "example" {
  name                = "example-service-plan"
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  kind                = "FunctionApp"
  sku {
    tier = "Dynamic"
    size = "Y1"
  }
}

resource "azurerm_storage_account" "example" {
  name                     = "examplestorageacct"
  resource_group_name      = azurerm_resource_group.example.name
  location                 = azurerm_resource_group.example.location
  account_tier             = "Standard"
  account_replication_type = "LRS"

}
```

Google Cloud Functions

Google Cloud Functions allows you to run your code in response to events on Google Cloud services, HTTP requests, or other triggers, without provisioning servers.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/cloudfunctions_function

Terraform Configuration for Google Cloud Functions:

```
resource "google_cloudfunctions_function" "example" {

  name        = "example-function"

  description = "A simple function"

  runtime     = "nodejs16"

  entry_point = "helloWorld"

  source_archive_bucket = google_storage_bucket.example.name

  source_archive_object = google_storage_bucket_object.example.name

  trigger_http = true

}


resource "google_storage_bucket" "example" {
  name     = "example-function-bucket"
  location = "US"
}

resource "google_storage_bucket_object" "example" {
  name   = "function.zip"
  bucket = google_storage_bucket.example.name
  source = "function.zip"
}
```

Oracle Functions

Oracle Functions is a serverless compute service that allows you to run code in response to events with automatic scaling and high availability.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/oracle/oci/latest/docs/resources/functions_function

Terraform Configuration for Oracle Functions:

```
resource "oci_functions_function" "example" {
  compartment_id = var.compartment_id
  display_name   = "example-function"
  image          = "your-container-image"
  memory_in_mbs  = 256
  timeout_in_seconds = 60
}


resource "oci_functions_application" "example" {
  compartment_id = var.compartment_id
  display_name   = "example-app"
  virtual_network_id = var.virtual_network_id
}


resource "oci_functions_api_gateway" "example" {
  compartment_id = var.compartment_id
  display_name   = "example-api-gateway"
  application_id = oci_functions_application.example.id
}
```

---

9. Content Delivery Network (CDN)

- **AWS:** CloudFront
- **Azure:** Azure CDN
- **GCP:** Cloud CDN
- **Oracle Cloud:** Oracle Cloud CDN

Content Delivery Network (CDN)

A Content Delivery Network (CDN) is a system of distributed servers designed to deliver content, such as web pages, images, videos, and other assets, to users based on their geographic location. CDNs optimize the delivery of content by caching it in multiple locations worldwide, improving performance, reducing latency, and enhancing the overall user experience. Below are the CDN services provided by major cloud providers, along with Terraform configurations for deploying them.

AWS: CloudFront

AWS CloudFront is a fast content delivery network service that distributes content globally with low latency and high transfer speeds. It integrates seamlessly with AWS services like S3, EC2, and Lambda.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/cloudfront_distribution

Terraform Configuration:

```
resource "aws_cloudfront_distribution" "example" {
  origin {
    domain_name = "example-bucket.s3.amazonaws.com"
    origin_id   = "S3-example-bucket"
  }

  enabled = true

  default_cache_behavior {
    target_origin_id = "S3-example-bucket"
    viewer_protocol_policy = "redirect-to-https"
    allowed_methods {
      methods = ["GET", "HEAD"]
    }
  }

  price_class = "PriceClass_100"
}
```

Azure: Azure CDN

Azure CDN is a global content delivery network for distributing content to users with low latency. It supports multiple providers like Akamai, Microsoft, and Verizon to improve speed and scalability.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/cdn_profile

Terraform Configuration:

```
resource "azurerm_cdn_profile" "example" {
  name                = "example-cdn-profile"
  resource_group_name = "example-resources"
  location            = "Central US"
  sku                 = "Standard_Akamai"
}
```

```
resource "azurerm_cdn_endpoint" "example" {
  name                = "example-cdn-endpoint"
  resource_group_name = azurerm_cdn_profile.example.resource_group_name
  cdn_profile_name    = azurerm_cdn_profile.example.name
  origin_host_header  = "example.com"
  origin {
    name = "example-origin"
    host_name = "example-origin.com"
  }
}
```

GCP: Cloud CDN

Google Cloud CDN leverages Google's global edge points of presence to accelerate content delivery for websites and applications, reducing latency and improving the user experience.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/compute_backend_bucket

Terraform Configuration:

```
resource "google_compute_backend_service" "example" {
  name        = "example-backend-service"
  protocol    = "HTTP"
  backends {
    group = "example-instance-group"
  }
}

resource "google_compute_url_map" "example" {
  name            = "example-url-map"
  default_service = google_compute_backend_service.example.id
}

resource "google_compute_global_forwarding_rule" "example" {
  name       = "example-forwarding-rule"
  target     = google_compute_url_map.example.id
  port_range  = "80"
  IP_address  = "0.0.0.0"
}

resource "google_compute_backend_bucket" "example" {
  name = "example-backend-bucket"
  bucket_name = "example-bucket"
}
```

Oracle Cloud: Oracle Cloud CDN

Oracle Cloud CDN is a global CDN service that enhances web application performance by caching content at edge locations, reducing server load, and accelerating content delivery.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/oracle/oci/5.8.0/docs/resources/media_services_stream_cdn_config

Terraform Configuration(simple):

```
resource "oci_cdn_control" "example" {

  compartment_id = "ocid1.compartment.oc1..example"

  display_name   = "example-cdn"

  origin_domain  = "example-origin.com"

  is_enabled     = true

}


resource "oci_cdn_origin" "example" {

  cdn_control_id = oci_cdn_control.example.id

  origin_domain  = "example-origin.com"

  origin_type    = "CUSTOM"

}
```

---

10. Container Orchestration

- **AWS:** ECS (Elastic Container Service) / EKS (Elastic Kubernetes Service)

- **Azure:** Azure Kubernetes Service (AKS)
- **GCP:** Google Kubernetes Engine (GKE)
- **Oracle Cloud:** Oracle Kubernetes Engine (OKE)

1. AWS ECS (Elastic Container Service)

AWS ECS is a fully managed container orchestration service that allows you to run and scale Docker containers in a highly scalable and secure environment. ECS works with EC2 instances or AWS Fargate for serverless compute, simplifying container management and scalability.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/ecs_service.html

Terraform Configuration Example for ECS:

```
provider "aws" {

  region = "us-west-2"

}


resource "aws_ecs_cluster" "example" {

  name = "example-cluster"

}


resource "aws_ecs_task_definition" "example" {

  family                   = "example-task"

  network_mode             = "awsvpc"

  requires_compatibilities = ["FARGATE"]
  container_definitions = jsonencode([{
    name      = "example-container"
    image     = "nginx:latest"
    essential = true
    portMappings = [{
      containerPort = 80
      hostPort      = 80
    }]
  }])
}

resource "aws_ecs_service" "example" {
  name            = "example-service"
  cluster         = aws_ecs_cluster.example.id
  task_definition = aws_ecs_task_definition.example.arn
  desired_count   = 2
}
```

---

2. AWS EKS (Elastic Kubernetes Service)

EKS is a fully managed Kubernetes service, enabling users to deploy, manage, and scale containerized applications using Kubernetes. It integrates seamlessly with other AWS services like IAM and CloudWatch for monitoring.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/modules/terraform-aws-modules/eks/aws/latest

Terraform Configuration Example for EKS:

```
# Define the AWS provider and region
provider "aws" {
  region = "us-west-2"
}

# EKS module configuration
module "eks" {
  source          = "terraform-aws-modules/eks/aws"

  # Define the cluster name and Kubernetes version
  cluster_name    = "my-eks-cluster"
  cluster_version = "1.26"

  # Specify the VPC and subnets for the cluster
  vpc_id          = "vpc-12345678" # Replace with your actual VPC ID
  subnets         = [
    "subnet-abcdef01", # Replace with actual Subnet IDs
    "subnet-abcdef02",
    "subnet-abcdef03"
  ]
  # Enable desired features and configurations
  enable_irsa = true # Enable IAM Roles for Service Accounts

  # Define managed node groups
  node_groups = {
    eks_nodes = {
      desired_capacity = 2
      max_capacity     = 3
      min_capacity     = 1

      instance_type = "t3.medium"
      key_name      = "my-ssh-key" # Replace with your actual key pair name
    }
  }

  # Tags for resources
  tags = {
    Environment = "production"
    Project     = "eks-demo"
  }
}
```

Provider Configuration: Ensure the AWS provider is correctly initialized by including the required provider block and credentials if necessary.

VPC and Subnet IDs: Replace placeholder IDs (e.g., vpc-12345678 and subnet-abcdef01) with the actual IDs from your AWS environment.

Node Groups:

- Ensure the node group block has all required configurations, like ami_type, disk_size, or launch_template, if applicable, depending on your AWS setup.
- The node_groups attribute is correct for the **terraform-aws-modules/eks/aws** module.

Tags: Tags are optional but very helpful for identifying resources. The ones you've added (Environment and Project) are useful.

---

3. Azure AKS (Azure Kubernetes Service)

AKS is a fully managed Kubernetes service in Azure that simplifies the deployment, management, and scaling of containerized applications. AKS provides native integration with Azure Active Directory and monitoring tools.

Official Terraform Hashicorp Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/kubernetes_cluster

Terraform Configuration Example for AKS:

```
resource "azurerm_resource_group" "example" {

  name     = "example"

  location = "West Europe" # Ensure this matches the region for your AKS cluster

}


resource "azurerm_private_dns_zone" "example" {
  name                = "privatelink.westeurope.azmk8s.io" # Align with the cluster's region
  resource_group_name = azurerm_resource_group.example.name
}

resource "azurerm_user_assigned_identity" "example" {
  name                = "aks-example-identity"
  resource_group_name = azurerm_resource_group.example.name
  location            = azurerm_resource_group.example.location
}

resource "azurerm_role_assignment" "example" {
  scope                = azurerm_private_dns_zone.example.id
  role_definition_name = "Private DNS Zone Contributor"
  principal_id         = azurerm_user_assigned_identity.example.principal_id

  depends_on = [
    azurerm_user_assigned_identity.example,
    azurerm_private_dns_zone.example,
  ]
}

resource "azurerm_kubernetes_cluster" "example" {
  name                    = "aksexamplewithprivatednszone1"
  location                = azurerm_resource_group.example.location
  resource_group_name     = azurerm_resource_group.example.name
  dns_prefix              = "aksexamplednsprefix1"
  private_cluster_enabled = true
  private_dns_zone_id     = azurerm_private_dns_zone.example.id

  # Additional configurations (e.g., default_node_pool, identity, network_profile)
  default_node_pool {
    name       = "default"
    vm_size    = "Standard_DS2_v2"
    node_count = 2
  }

  identity {
    type = "UserAssigned"
    user_assigned_identity = azurerm_user_assigned_identity.example.id
  }

  depends_on = [
    azurerm_role_assignment.example,
  ]
}
```

Review Points:

1. **Location Consistency**: Ensure that the location used in azurerm_resource_group, azurerm_private_dns_zone, and other resources is consistent with your actual deployment region. You're using West Europe for the resource group but defining a private DNS zone for eastus2. This might lead to resource placement issues.
2. **Private DNS Zone Name**: Verify that privatelink.eastus2.azmk8s.io is appropriate for your AKS cluster's private DNS zone. The eastus2 in the name must match the cluster's actual region unless it's intentional.
3. **Role Assignment Dependencies**: The azurerm_role_assignment depends implicitly on the azurerm_user_assigned_identity and azurerm_private_dns_zone, but adding explicit depends_on can make dependencies clearer, especially if provisioning order is crucial.
4. **AKS Configuration**: The azurerm_kubernetes_cluster configuration mentions omitted parts. Ensure you have specified critical configurations like default_node_pool, identity, and network_profile.
5. **Terraform Provider Version**: Use a required_providers block with a pinned version for the azurerm provider to avoid compatibility issues.
6. **Resource Naming**: Consider naming resources more dynamically if you're automating or parameterizing this configuration for multiple environments.

---

4. GCP GKE (Google Kubernetes Engine)

GKE is a managed Kubernetes service provided by Google Cloud, offering powerful features like automated scaling and load balancing for running containerized applications.

Official Hashicorp Terraform configuration:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/container_cluster

```
provider "google" {

  project = "your-gcp-project-id"

  region  = "us-central1"
```

```
}

resource "google_service_account" "default" {
  account_id   = "service-account-id"
  display_name = "Service Account"
}

resource "google_container_cluster" "primary" {
  name                     = "my-gke-cluster"
  location                 = "us-central1"
  remove_default_node_pool = true
  initial_node_count       = 1
}

resource "google_container_node_pool" "primary_preemptible_nodes" {
  name       = "my-node-pool"
  location   = "us-central1"
  cluster    = google_container_cluster.primary.name
  node_count = 1

  node_config {
    preemptible  = true
    machine_type = "e2-medium"
    service_account = google_service_account.default.email

    oauth_scopes    = [

      "https://www.googleapis.com/auth/cloud-platform"

    ]

  }

}
```

Key Features:

1. **Provider Configuration**: Directly includes project and region values.
2. **Custom Service Account**: Configures a service account with an account ID and display name for use by the node pool.
3. **Cluster Configuration**:
   - Removes the default node pool to allow custom node pool management.
   - Creates a minimal default configuration with initial_node_count = 1 for the cluster.
4. **Node Pool Management**:
   - Creates a node pool with preemptible nodes to save costs.
   - Configures the machine_type and associates it with the custom service account.
5. **OAuth Scopes**: Grants cloud-platform access scope for flexibility in interacting with Google Cloud resources.

---

5. Oracle Cloud OKE (Oracle Kubernetes Engine)

OKE is Oracle's managed Kubernetes service, allowing users to run Kubernetes clusters with Oracle Cloud's robust security, scalability, and monitoring features.

Official Hashicorp Terraform configuration:

https://registry.terraform.io/providers/oracle/oci/latest/docs/resources/containerengine_cluster

Terraform Configuration Example for OKE:

```
resource "oci_containerengine_cluster" "test_cluster" {
  # Required
  compartment_id     = "ocid1.compartment.oc1..example"  # Replace with your OCI compartment OCID
  kubernetes_version  = "v1.24.8"  # Replace with the desired Kubernetes version
  name               = "test-cluster"  # Replace with your desired cluster name
  vcn_id             = oci_core_vcn.test_vcn.id  # Replace with your VCN OCID

  # Optional
  cluster_pod_network_options {
    cni_type = "CNI_PLUGIN_TYPE"  # Replace with the desired CNI type
  }

  defined_tags = {
    "Operations.CostCenter" = "42"  # Replace with your tag details
  }
  freeform_tags = {
    "Department" = "Finance"  # Replace with your tag details
  }

  endpoint_config {
    is_public_ip_enabled = true  # Set to true/false based on whether you want a public IP
    nsg_ids            = ["ocid1.networksecuritygroup.oc1..example"]  # Replace with your NSG OCID
    subnet_id          = oci_core_subnet.test_subnet.id  # Replace with your Subnet OCID
  }

  image_policy_config {
    is_policy_enabled = true  # Set to true if you want to enable image policies
    key_details {
      kms_key_id = oci_kms_key.test_key.id  # Replace with your KMS Key OCID
    }
  }

  kms_key_id = oci_kms_key.test_key.id  # Replace with your KMS Key OCID

  options {
    add_ons {
      is_kubernetes_dashboard_enabled = true  # Set to true to enable the Kubernetes dashboard
      is_tiller_enabled               = true  # Set to true to enable Tiller
    }

    admission_controller_options {
      is_pod_security_policy_enabled = true  # Set to true to enable pod security policies
    }
    kubernetes_network_config {
      pods_cidr     = "10.244.0.0/16"  # Replace with your pod CIDR
      services_cidr = "10.96.0.0/12"   # Replace with your services CIDR
    }

    open_id_connect_token_authentication_config {
      is_open_id_connect_auth_enabled = true  # Set to true to enable OpenID Connect
      ca_certificate                  = "ca-certificate"  # Replace with actual certificate if needed
      client_id                       = oci_containerengine_client.test_client.id  # Replace with your client ID
      groups_claim                    = "groups"  # Replace with actual group claim
      groups_prefix                   = "prefix"  # Replace with actual group prefix
      issuer_url                      = "https://example.com"  # Replace with actual issuer URL
      required_claims {
        key   = "claim_key"  # Replace with actual claim key
        value = "claim_value"  # Replace with actual claim value
      }
      signing_algorithms = ["RS256"]  # Replace with desired signing algorithms
      username_claim     = "username"  # Replace with actual username claim
      username_prefix    = "prefix"  # Replace with actual username prefix
    }

    open_id_connect_discovery {
      is_open_id_connect_discovery_enabled = true  # Set to true if you want OpenID Connect discovery enabled
    }


    persistent_volume_config {
      defined_tags  = {
        "Operations.CostCenter" = "42"  # Replace with your tag details
      }
      freeform_tags = {
        "Department" = "Finance"  # Replace with your tag details
      }
    }


    service_lb_config {
      defined_tags  = {
        "Operations.CostCenter" = "42"  # Replace with your tag details
      }
      freeform_tags = {
        "Department" = "Finance"  # Replace with your tag details
      }
    }


    service_lb_subnet_ids = ["ocid1.subnet.oc1..example"]  # Replace with your subnet OCID for service load balancer
```

```
  }


  type = "K8S"  # Set the desired cluster type, usually "K8S"

}
```

Ensure Resource Existence: Make sure the referenced resources (e.g., VCN, Subnet, KMS key) are already created or defined in your Terraform configuration to avoid errors during deployment.

Use Variables for Flexibility: While you asked not to use variables, incorporating variables can make the configuration more flexible and reusable across different environments.

Review Optional Configurations: Double-check that all optional configurations (like OpenID Connect, Kubernetes add-ons, and image policies) align with your specific requirements before enabling them.

Tagging Strategy: Consider adding more meaningful tags for better cost allocation and resource management (e.g., environment, project name).

Security Considerations: Ensure that any sensitive information, such as credentials or keys, is securely managed, possibly by using oci_secret or similar mechanisms.

---

11. Identity and Access Management

- **AWS:** IAM (Identity and Access Management)
- **Azure:** Azure Active Directory
- **GCP:** Cloud Identity
- **Oracle Cloud:** Oracle Identity and Access Management (IAM)

1. AWS IAM (Identity and Access Management)

AWS IAM allows you to control access to AWS resources securely. You can create and manage AWS users and groups, and assign permissions to allow or deny access to resources.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/iam_role

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/iam_user

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/iam_group

Terraform Configuration:

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_iam_user" "example_user" {
  name = "example_user"
}

resource "aws_iam_group" "example_group" {
  name = "example_group"
}

resource "aws_iam_policy" "example_policy" {
  name        = "example_policy"
  description = "A custom policy to access S3"
  policy      = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action   = "s3:ListBucket"
        Effect   = "Allow"
        Resource = "arn:aws:s3:::example-bucket"
      },
      {
        Action   = "s3:GetObject"
        Effect   = "Allow"
        Resource = "arn:aws:s3:::example-bucket/*"
      },
    ]
  })
}

resource "aws_iam_group_policy_attachment" "example_group_attachment" {
  group      = aws_iam_group.example_group.name
  policy_arn = aws_iam_policy.example_policy.arn
}

resource "aws_iam_user_group_membership" "example_group_membership" {
  user   = aws_iam_user.example_user.name
  groups = [aws_iam_group.example_group.name]
```

```
}


resource "aws_iam_access_key" "example_access_key" {

  user = aws_iam_user.example_user.name

}
```

1. **IAM User**: Represents an individual user who can access AWS services.
    - Example: aws_iam_user resource.
2. **IAM Group**: A collection of users that can share permissions.
    - Example: aws_iam_group resource.

An **IAM Group** in AWS is a collection of IAM users. It allows you to manage permissions for multiple users at once by attaching policies to the group. When a user is added to a group, the user automatically inherits all the permissions associated with the group.

Key Points about IAM Groups:

- **Organize Users**: Groups help organize IAM users based on roles or permissions. For example, you can have a Admins group and a Developers group, each with different permissions.
- **Assign Policies to Groups**: You can attach **AWS-managed** or **customer-managed policies** to a group to define what the members can or cannot do in AWS.
- **Easier Management**: Instead of attaching individual policies to each user, you attach policies to groups. This simplifies managing permissions across many users.
    -

IAM Role: Used to delegate access to AWS services or cross-account access.

- Example: aws_iam_role resource.

IAM Policy: Defines permissions (e.g., access to S3, EC2).

- Example: aws_iam_policy resource.

Attach Policies: You can attach AWS-managed or custom policies to users, groups, or roles.

- Example: aws_iam_role_policy_attachment, aws_iam_group_policy_attachment.

- **Access Keys**: Allow programmatic access for users.
  - Example: aws_iam_access_key.

---

2. Azure Active Directory (Azure AD)

Azure AD provides identity services to manage user access to Azure resources and other Microsoft services. It includes features like user authentication, role-based access control, and multi-factor authentication.

Official Hashicorp Terraform configuration:

https://registry.terraform.io/providers/hashicorp/azuread/latest/docs/resources/user

https://registry.terraform.io/providers/hashicorp/azuread/latest/docs/resources/directory_role

https://registry.terraform.io/providers/hashicorp/azuread/latest/docs/resources/group

```
# Define the Azure AD user
resource "azuread_user" "example_user" {
  user_principal_name = "example_user@domain.com"
  display_name        = "Example User"
  password            = "SecurePassword123!"
  force_password_change = false # Set to true if you want the user to change password on next login
  mail_nickname       = "exampleuser"
  user_type           = "Member" # Can be "Member" or "Guest" depending on user type
}
```

```
# Define the Azure AD group
resource "azuread_group" "example_group" {
  display_name = "Example Group"
  security_enabled = true # Set to false if the group is an Office 365 group
  mail_enabled = false   # Set to true if this is a mail-enabled group
  visibility = "Private" # Options: "Private", "Public"
}

# Add the user to the group
resource "azuread_group_member" "example_member" {
  group_object_id   = azuread_group.example_group.id
  member_object_id  = azuread_user.example_user.id
}

# Optionally, assign a directory role to the user (e.g., Global Administrator)
resource "azuread_directory_role" "example_role" {
  display_name = "Global Administrator"
  role_template_id = data.azuread_directory_role.global_admin.id
}

# Assign the role to the user
resource "azuread_directory_role_member" "example_role_member" {
  role_object_id   = azuread_directory_role.example_role.id
  principal_object_id = azuread_user.example_user.id
}

# Define a data source to retrieve the role template ID for Global Administrator role
data "azuread_directory_role" "global_admin" {
  display_name = "Global Administrator"
}
```

Changes and additions:

1. **User Configuration:**
   - I added force_password_change (set to false for no password reset on first login) and user_type (can be Member or Guest).
   - Added mail_nickname for the user to ensure uniqueness.
2. **Group Configuration:**
   - Added the security_enabled flag to indicate whether the group is security-enabled (set to true for security groups).
   - Included the visibility and mail_enabled flags to control the group's behavior (whether it's a mail-enabled or security group).
3. **Role Assignment:**
   - Added a directory role resource (azuread_directory_role) to assign a directory role like "Global Administrator" to the user.
   - Introduced the azuread_directory_role_member resource to associate the role with the user.

---

3. GCP Cloud Identity

Cloud Identity is a service provided by Google Cloud for managing users, devices, and apps across GCP resources. It allows you to set up identity services such as Single Sign-On (SSO), Multi-Factor Authentication (MFA), and more.

Terraform Configuration:

```
resource "google_identity_platform_project" "example_project" {
  project_id = "example-project-id"
}

resource "google_identity_platform_user" "example_user" {
  email    = "example_user@example.com"
  password = "SecurePassword123!"
}
```

---

4. Oracle Cloud IAM

Oracle Identity and Access Management (IAM) enables you to control user access to Oracle Cloud Infrastructure (OCI) resources. It provides capabilities for user authentication, role-based access control, and policies for managing resources.

Terraform Configuration:

```
resource "oci_identity_user" "example_user" {
  compartment_id = "compartment_ocid"
  name           = "example_user"
  description    = "Example User"
}
```

```
resource "oci_identity_group" "example_group" {
  compartment_id = "compartment_ocid"
  name           = "example_group"
}

resource "oci_identity_group_membership" "example_membership" {
  group_id = oci_identity_group.example_group.id
  user_id  = oci_identity_user.example_user.id
}
```

These configurations are just basic examples. IAM services allow you to create roles, policies, and permissions tailored to your security needs. Adjust the configurations according to your organization's specific requirements.

---

12. Load Balancing

- **AWS:** Elastic Load Balancing
- **Azure:** Azure Load Balancer
- **GCP:** Cloud Load Balancing
- **Oracle Cloud:** Oracle Cloud Load Balancing

Elastic Load Balancing (AWS)

Introduction: AWS Elastic Load Balancing (ELB) automatically distributes incoming application traffic across multiple targets, such as Amazon EC2 instances, containers, and IP addresses. It helps improve the fault tolerance of your application and ensures high availability.

Official Hashicorp Terraform configuration:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/elb

Terraform Configuration:

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_lb" "example" {
  name               = "example-lb"
  internal           = false
  load_balancer_type = "application"
  security_groups    = ["sg-xxxxxxxx"]
  subnets            = ["subnet-xxxxxxxx", "subnet-yyyyyyyy"]

  enable_deletion_protection = false
}

resource "aws_lb_target_group" "example" {
  name     = "example-tg"
  port     = 80
  protocol = "HTTP"
  vpc_id   = "vpc-xxxxxxxx"
}
```

```
resource "aws_lb_listener" "example" {
  load_balancer_arn = aws_lb.example.arn
  port              = 80
  protocol          = "HTTP"

  default_action {
    type             = "fixed-response"
    fixed_response {
      status_code = 200
      content_type = "text/plain"
      message_body = "OK"
    }
  }
}
```

Azure Load Balancer

Introduction: Azure Load Balancer distributes network traffic across multiple servers, ensuring high availability of your services. It supports both inbound and outbound traffic for applications.

Official Hashicorp Terraform configuration:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/lb

Terraform Configuration:

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

resource "azurerm_lb" "example" {
  name                = "example-lb"
  location            = "East US"
  resource_group_name = "example-rg"
}

resource "azurerm_lb_backend_address_pool" "example" {
  name                = "example-backend-pool"
  resource_group_name = "example-rg"
  loadbalancer_id     = azurerm_lb.example.id
}

resource "azurerm_lb_probe" "example" {
  name                = "example-probe"
  resource_group_name = "example-rg"
  loadbalancer_id     = azurerm_lb.example.id
  port                = 80
  protocol            = "Http"
  request_path        = "/health"
}
```

Cloud Load Balancing (GCP)

Introduction: Google Cloud Load Balancing enables users to distribute traffic to resources globally, ensuring low-latency access to services. It supports HTTP(S), TCP, and SSL proxy load balancing.

Official Hashicorp Terraform configuration:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/compute_forwarding_rule

Terraform Configuration:

```
provider "google" {
  region = "us-central1"
}


resource "google_compute_forwarding_rule" "example" {
  name                  = "example-lb"
  target                = google_compute_backend_service.example.self_link
  port_range            = "80"
  ip_address            = "0.0.0.0"
  load_balancing_scheme = "EXTERNAL"
  ip_protocol           = "TCP"
}

resource "google_compute_backend_service" "example" {
  name        = "example-backend-service"
  protocol    = "HTTP"
  backends    = [
    {
      group = google_compute_instance_group.example.self_link
    }
  ]
}

resource "google_compute_instance_group" "example" {
  name = "example-instance-group"
  zone = "us-central1-a"

  instances = [
    google_compute_instance.example.self_link,
  ]
}
```

Oracle Cloud Load Balancing

Introduction: Oracle Cloud Load Balancing provides a highly available, scalable, and fault-tolerant solution to distribute traffic across backend servers. It supports TCP and HTTP(S) protocols and ensures seamless traffic management.

Official Hashicorp Terraform configuration:

https://registry.terraform.io/providers/oracle/oci/latest/docs/resources/load_balancer_load_balancer.html

Terraform Configuration:

```
provider "oci" {}


resource "oci_load_balancer" "example" {
  compartment_id = "ocid1.compartment.oc1..example"
  display_name   = "example-lb"
  shape          = "100Mbps"


  subnet_ids = ["ocid1.subnet.oc1..example"]


  backend_set {
    name               = "example-backend-set"
    policy             = "ROUND_ROBIN"
    protocol           = "HTTP"
    health_checker {
      interval_in_seconds = 10
      timeout_in_seconds  = 5
      port                = 80
    }

    backends {
      ip_address = "192.168.1.1"
      port       = 80
    }
  }
}
```

---

13. API Management

AWS: API Gateway

Azure: Azure API Management

GCP: Apigee API Platform

Oracle Cloud: Oracle API Gateway

AWS: API Gateway

AWS API Gateway is a fully managed service that allows developers to create, publish, maintain, monitor, and secure APIs at any scale. It provides features like traffic management, authorization, access control, and API monitoring.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/api_gateway_rest_api

Terraform Configuration:

```
resource "aws_api_gateway_rest_api" "example" {
  name        = "example-api"
  description = "This is an example API"

  body = jsonencode({
    openapi = "3.0.1"
    info = {
      title   = "example"
      version = "1.0"
    }
    paths = {
      "/path1" = {
        get = {
          x-amazon-apigateway-integration = {
            httpMethod           = "GET"
            payloadFormatVersion = "1.0"
            type                 = "HTTP_PROXY"
            uri                  = "https://ip-ranges.amazonaws.com/ip-ranges.json"
          }
        }
      }
    }
  })

  endpoint_configuration {
    types = ["REGIONAL"]
  }
}

resource "aws_api_gateway_deployment" "example" {
  rest_api_id = aws_api_gateway_rest_api.example.id

  triggers = {
    redeployment = sha1(jsonencode(aws_api_gateway_rest_api.example.body))
  }

  lifecycle {
    create_before_destroy = true
  }
}
```

```
resource "aws_api_gateway_stage" "example" {
  deployment_id = aws_api_gateway_deployment.example.id
  rest_api_id   = aws_api_gateway_rest_api.example.id
  stage_name    = "example-stage"

  variables = {
    exampleVar = "exampleValue"
  }
}
```

Azure: Azure API Management

Azure API Management is a fully managed API gateway that enables organizations to publish, secure, and analyze APIs. It provides tools to create and manage APIs, control access, and monitor performance.

Official Terraform Documentation;

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/api_management

Terraform Configuration:

```
resource "azurerm_api_management" "example" {
  name                = "example-apim"
  location            = "East US"
  resource_group_name = azurerm_resource_group.example.name
  publisher_name      = "Example Publisher"
  publisher_email     = "publisher@example.com"
}


resource "azurerm_api_management_api" "example" {
  name                = "example-api"
  api_management_name = azurerm_api_management.example.name
  revision            = "1"
  display_name        = "Example API"
  path                = "example-path"
  protocols           = ["https"]
}
```

GCP: Apigee API Platform

Introduction:

Apigee API Platform by Google Cloud enables API providers to manage the full API lifecycle. It helps in creating, securing, monitoring, and scaling APIs.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/api_gateway_api

Terraform Configuration for **Apigee with VPC networking**:

```
resource "google_compute_network" "apigee_network" {
  name = "apigee-network"
}

resource "google_apigee_organization" "org" {
  analytics_region   = "us-central1"
  project_id         = "your-project-id"
  authorized_network = google_compute_network.apigee_network.id
}
```

Oracle Cloud: Oracle API Gateway

Oracle API Gateway provides a way to securely create and manage APIs across different cloud environments. It offers API rate limiting, access control, logging, and monitoring.

Official Terraform Documentation:

https://registry.terraform.io/providers/oracle/oci/latest/docs/resources/apigateway_gateway

Terraform Configuration:

```
resource "oci_apigateway_gateway" "test_gateway" {
  # Required
  compartment_id = var.compartment_id
  endpoint_type  = var.gateway_endpoint_type
  subnet_id      = oci_core_subnet.test_subnet.id
```

```
# Optional
certificate_id = oci_apigateway_certificate.test_certificate.id

ca_bundles {
  # Required
  type = var.gateway_ca_bundles_type

  # Optional
  ca_bundle_id = oci_apigateway_ca_bundle.test_ca_bundle.id
  certificate_authority_id = oci_apigateway_certificate_authority.test_certificate_authority.id
}

defined_tags = {
  "Operations.CostCenter" = "42"
}

display_name = var.gateway_display_name
freeform_tags = {
  "Department" = "Finance"
}

network_security_group_ids = var.gateway_network_security_group_ids
```

```
  response_cache_details {
    # Required
    type = var.gateway_response_cache_details_type

    # Optional
    authentication_secret_id             = oci_vault_secret.test_secret.id
    authentication_secret_version_number  = var.gateway_response_cache_details_authentication_secret_version_number
    connect_timeout_in_ms                 = var.gateway_response_cache_details_connect_timeout_in_ms
    is_ssl_enabled                       = var.gateway_response_cache_details_is_ssl_enabled
    is_ssl_verify_disabled               = var.gateway_response_cache_details_is_ssl_verify_disabled
    read_timeout_in_ms                   = var.gateway_response_cache_details_read_timeout_in_ms
    send_timeout_in_ms                   = var.gateway_response_cache_details_send_timeout_in_ms

    servers {
      # Optional
      host = var.gateway_response_cache_details_servers_host
      port = var.gateway_response_cache_details_servers_port
    }
  }
}
```

---

14. Monitoring and Management

AWS: CloudWatch, CloudTrail

Azure: Azure Monitor, Azure Log Analytics

GCP: Stackdriver (now part of Google Cloud Operations)

Oracle Cloud: Oracle Cloud Monitoring

AWS: CloudWatch & CloudTrail

AWS CloudWatch provides monitoring for AWS cloud resources and applications. It collects and tracks metrics, collects and monitors log files, and sets alarms. CloudTrail records AWS API calls and events to help monitor and maintain security and compliance.

Terraform Configuration:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/cloudwatch_metric_alarm

```
# AWS CloudWatch Alarm

resource "aws_cloudwatch_metric_alarm" "foobar" {

  alarm_name                = "terraform-test-foobar5"

  comparison_operator       = "GreaterThanOrEqualToThreshold"

  evaluation_periods        = 2

  metric_name               = "CPUUtilization"

  namespace                 = "AWS/EC2"

  period                    = 120

  statistic                 = "Average"
  threshold           = 80
  alarm_description   = "This metric monitors ec2 cpu utilization"
  insufficient_data_actions = []
}
```

# AWS CloudTrail

Official Terraform Documentation

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/cloudtrail

Terraform Configuration:

```
resource "aws_s3_bucket" "example" {
  bucket        = "tf-test-trail"
  force_destroy = true
}

resource "aws_cloudtrail" "example" {
  depends_on = [aws_s3_bucket.example]

  name           = "example"
  s3_bucket_name = aws_s3_bucket.example.id
}

resource "aws_s3_bucket_policy" "example" {
  bucket = aws_s3_bucket.example.id
  policy = jsonencode({
    Version   = "2012-10-17"
    Statement = [
      {
        Sid       = "AWSCloudTrailAclCheck"
        Effect    = "Allow"
        Principal = {
          Service = "cloudtrail.amazonaws.com"
        }
        Action    = "s3:GetBucketAcl"
        Resource  = aws_s3_bucket.example.arn
        Condition = {
          StringEquals = {
            "aws:SourceArn" = "arn:aws:cloudtrail:${data.aws_region.current.name}:${data.aws_caller_identity.current.account_id}:trail/example"
          }
        }
      },
      {
        Sid       = "AWSCloudTrailWrite"
        Effect    = "Allow"
        Principal = {
          Service = "cloudtrail.amazonaws.com"
        }
        Action    = "s3:PutObject"
        Resource  =
"${aws_s3_bucket.example.arn}/prefix/AWSLogs/${data.aws_caller_identity.current.account_id
}/*"
        Condition = {
          StringEquals = {
            "s3:x-amz-acl" = "bucket-owner-full-control"
            "aws:SourceArn" =
"arn:aws:cloudtrail:${data.aws_region.current.name}:${data.aws_caller_identity.current.account
_id}:trail/example"
          }
        }
      }
    ]
  })
}

data "aws_caller_identity" "current" {}

data "aws_partition" "current" {}

data "aws_region" "current" {}
```

Azure: Azure Monitor & Log Analytics

Introduction:

Azure Monitor provides a comprehensive solution for collecting, analyzing, and acting on telemetry from Azure resources. Azure Log Analytics helps you query and analyze logs from different sources within your environment.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/monitor_metric_alert

Terraform Configuration:

```
# Azure Monitor Log Analytics Workspace

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


resource "azurerm_storage_account" "to_monitor" {
  name                     = "examplestorageaccount"
  resource_group_name      = azurerm_resource_group.example.name
  location                 = azurerm_resource_group.example.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
}

resource "azurerm_monitor_action_group" "main" {
  name                = "example-actiongroup"
  resource_group_name = azurerm_resource_group.example.name
  short_name          = "exampleact"

  webhook_receiver {
    name        = "callmyapi"
    service_uri = "http://example.com/alert"
  }
}

resource "azurerm_monitor_metric_alert" "example" {
  name                = "example-metricalert"
  resource_group_name = azurerm_resource_group.example.name
  scopes              = [azurerm_storage_account.to_monitor.id]
  description         = "Action will be triggered when Transactions count is greater than 50."

  criteria {
    metric_namespace = "Microsoft.Storage/storageAccounts"
    metric_name      = "Transactions"
    aggregation      = "Total"

    operator         = "GreaterThan"

    threshold        = 50


    dimension {

      name     = "ApiName"

      operator = "Include"

      values   = ["*"]

    }

  }


  action {

    action_group_id = azurerm_monitor_action_group.main.id

  }

}
```

GCP: Stackdriver (Google Cloud Operations)

Introduction:

Stackdriver (now part of Google Cloud Operations suite) provides monitoring, logging, and diagnostics for applications on Google Cloud Platform (GCP). It helps you monitor, troubleshoot, and improve the performance of your applications.

Terraform Configuration:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/monitoring_notification_channel

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/monitoring_alert_policy

# GCP Stackdriver Monitoring Notification Channel

```
# GCP Stackdriver Monitoring Notification Channel

resource "google_monitoring_notification_channel" "email" {

  display_name = "Email Notification Channel"

  type         = "email"

  labels = {

    email_address = "your-email@example.com"

  }

}
```

# GCP Monitoring Alert Policy

```
resource "google_monitoring_alert_policy" "example" {

  display_name = "High CPU Alert"

  notification_channels = [google_monitoring_notification_channel.email.id]


  conditions {

    display_name = "CPU Utilization High"

    condition_threshold {

      comparison = "COMPARISON_GT"
```

```
   threshold_value = 80
   aggregations {
    alignment_period = "60s"
    per_series_aligner = "ALIGN_RATE"
   }
   filter = "metric.type=\"compute.googleapis.com/instance/disk/write_bytes_count\"" # Replace this with CPU metric filter
  }
 }
}
```

Oracle Cloud: Oracle Cloud Monitoring

Oracle Cloud Monitoring provides capabilities to monitor cloud infrastructure, track resource utilization, and receive alerts for resource changes or failures, enabling proactive management of cloud resources.

Official Terraform Documentation:

https://registry.terraform.io/providers/oracle/oci/latest/docs/data-sources/monitoring_alarms

Terraform Configuration:

```
# Oracle Cloud Monitoring Alarms
resource "oci_monitoring_alarm" "example" {
 display_name = "High CPU Alarm"
 metric_name  = "CPUUtilization"
```

```
  namespace    = "oci_computeagent"
  compartment_id = "ocid1.compartment.oc1..example"

  condition {
    comparison_operator = "GREATER_THAN"
    threshold          = 85
  }

  action {
    action_type = "NOTIFY"
    notify {
      email {
        recipients = ["your-email@example.com"]
      }
    }
  }
}
```

---

15. Message Queuing and Event Streaming

AWS: SQS (Simple Queue Service), SNS (Simple Notification Service), Kinesis

Azure: Azure Service Bus, Event Grid, Event Hub

GCP: Pub/Sub, Cloud Tasks

Oracle Cloud: Oracle Cloud Messaging Service (OCMS)

AWS

SQS (Simple Queue Service): A fully managed message queuing service that enables decoupling of microservices, distributed systems, and serverless applications.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/sqs_queue

Terraform Configuration:

```
resource "aws_sqs_queue" "terraform_queue" {
  name                      = "terraform-example-queue"
  delay_seconds             = 90
  max_message_size          = 2048
  message_retention_seconds = 86400
  receive_wait_time_seconds = 10
  redrive_policy = jsonencode({
    deadLetterTargetArn = aws_sqs_queue.terraform_queue_deadletter.arn
    maxReceiveCount     = 4
  })



  tags = {
    Environment = "production"
  }
}
```

SNS (Simple Notification Service): A fully managed pub/sub messaging service that allows you to decouple microservices and send notifications to subscribers.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/sns_topic

Terraform Configuration:

```
resource "aws_sns_topic" "my_topic" {
  name = "my-topic"
}
```

Kinesis: A platform to collect, process, and analyze real-time streaming data, including video, audio, application logs, and social media feeds.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/kinesis_stream

Terraform Configuration:

```
resource "aws_kinesis_stream" "test_stream" {
  name             = "terraform-kinesis-test"
  shard_count      = 1
  retention_period = 48

  shard_level_metrics = [
    "IncomingBytes",
    "OutgoingBytes",
  ]

  stream_mode_details {
    stream_mode = "PROVISIONED"
  }

  tags = {
    Environment = "test"
  }
}
```

Azure

Azure Service Bus: A fully managed enterprise message broker with message queuing and publish/subscribe topics.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/servicebus_topic

Terraform Configuration:

```
resource "azurerm_resource_group" "example" {
  name     = "tfex-servicebus-topic"
  location = "West Europe"
}

resource "azurerm_servicebus_namespace" "example" {
  name                = "tfex-servicebus-namespace"
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  sku                 = "Standard"

  tags = {
    source = "terraform"
  }
}

resource "azurerm_servicebus_topic" "example" {
  name         = "tfex_servicebus_topic"
  namespace_id = azurerm_servicebus_namespace.example.id
  partitioning_enabled = true
}
```

Event Grid: A fully managed event routing service that allows you to easily integrate applications using event-driven architecture.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/eventgrid_topic

Terraform Configuration:

```
resource "azurerm_eventgrid_topic" "example" {
  name                = "example-eventgrid"
  resource_group_name = azurerm_resource_group.example.name
  location            = "East US"
}
```

Event Hub: A highly scalable data streaming platform that ingests large amounts of data in real-time, including logs and telemetry.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/eventhub.html

Terraform Configuration:

```
resource "azurerm_eventhub_namespace" "example" {
  name                = "example-eventhub"
  location            = "East US"
  resource_group_name = azurerm_resource_group.example.name
  sku                 = "Standard"
}
```

GCP

Pub/Sub: A fully managed real-time messaging service for event-driven systems, offering reliable message delivery.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/pubsub_topic

Terraform Configuration:

```
resource "google_pubsub_topic" "example" {
  name = "example-topic"

  labels = {
    foo = "bar"
  }

  message_retention_duration = "86600s"
}
```

Cloud Tasks: A fully managed service for managing the execution of background tasks and deferred work.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/cloud_tasks_queue

Terraform Configuration:

```
resource "google_cloudtasks_queue" "my_queue" {

  name     = "my-queue"

  location = "us-central1"

}
```

Oracle Cloud

Oracle Cloud Messaging Service (OCMS): A fully managed messaging platform to facilitate communication between cloud services, microservices, and applications.

Official Terraform Documentation:

https://registry.terraform.io/providers/oracle/oci/latest/docs/resources/ons_notification_topic

Terraform Configuration:

```
resource "oci_osms_topic" "my_topic" {
  display_name = "my-topic"

  compartment_id = var.compartment_id

}
```

---

16. Security and Compliance

AWS: AWS Shield (DDoS Protection), AWS WAF

Azure: Azure DDoS Protection, Azure Security Center

GCP: Cloud Armor, Identity-Aware Proxy, Cloud Security Command Center

Oracle Cloud: Oracle Cloud Security

AWS Shield: AWS Shield provides DDoS (Distributed Denial of Service) protection for AWS services. It offers both Standard and Advanced protection. AWS WAF: The AWS Web Application Firewall (WAF) helps protect applications from common web exploits. IAM (Identity and Access Management): IAM enables you to manage access to AWS services and resources securely.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/shield_protection

Terraform Configuration :

```
resource "aws_shield_protection" "example" {

  name = "example-shield-protection"

  resource_arn = aws_elb.example.arn # Example for a Load Balancer

}
```

AWS WAF:

AWS WAF (Web Application Firewall) is a managed service that protects web applications from common threats like SQL injection and cross-site scripting (XSS). It allows you to create custom security rules to filter malicious traffic based on factors like IP addresses or HTTP headers. AWS WAF integrates with services like CloudFront and Application Load Balancer, offering scalable and cost-effective protection for your web applications

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/wafv2_ip_set

Terraform Configuration:

```
resource "aws_wafv2_ip_set" "example" {
  name               = "example"
  description        = "Example IP set"
  scope              = "REGIONAL"
  ip_address_version = "IPV4"
  addresses          = ["1.2.3.4/32", "5.6.7.8/32"]

  tags = {
    Tag1 = "Value1"
    Tag2 = "Value2"
  }
}
```

Azure:

Azure DDoS Protection: Azure DDoS Protection provides protection against volumetric, state-exhaustion, and small-scale application-layer attacks. Azure Security Center: A unified security management system to monitor and protect your Azure resources.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/network_ddos_protection_plan

Terraform Configuration:

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


resource "azurerm_network_ddos_protection_plan" "example" {
  name                = "example-protection-plan"
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
}
```

Terraform Configuration Example (Azure DDoS Protection):

```
resource "azurerm_ddos_protection_plan" "example" {
  name                = "example-ddos-plan"
  resource_group_name = azurerm_resource_group.example.name
  location            = azurerm_resource_group.example.location
}

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

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"]
}

resource "azurerm_subnet_ddos_protection" "example" {
  subnet_id              = azurerm_subnet.example.id
  ddos_protection_plan_id = azurerm_ddos_protection_plan.example.id
}
```

GCP:

Cloud Armor: Provides DDoS protection for Google Cloud services. Identity-Aware Proxy: Ensures that only authenticated users can access your applications. Cloud Security Command Center: Provides security and data risk visibility across Google Cloud services.

Terraform Configuration Example (Cloud Armor):

```
resource "google_compute_security_policy" "example" {
  name = "example-security-policy"
}


resource "google_compute_backend_service" "example" {
  name            = "example-backend-service"
  security_policy = google_compute_security_policy.example.id
  protocol        = "HTTP"
  health_checks   = [google_compute_health_check.example.id]
}


resource "google_compute_health_check" "example" {
  name            = "example-health-check"
  http_health_check {
    port = 80
    request_path = "/"
  }
}
```

Oracle Cloud:

Oracle Cloud Security: Oracle Cloud Infrastructure (OCI) offers security features like identity and access management, firewall rules, and security monitoring.

Official Terraform Documentation:

https://registry.terraform.io/providers/oracle/oci/latest/docs/resources/identity_user

Terraform Configuration Example (OCI IAM):

```
resource "oci_identity_user" "example" {
  name        = "example-user"
  description = "A user for OCI"
}

resource "oci_identity_policy" "example" {
  name           = "example-policy"
  compartment_id = oci_identity_compartment.example.id
  statements     = ["allow group example-group to manage all-resources in compartment example"]
}
```

---

17. Data Warehousing

AWS: Redshift

Azure: Azure Synapse Analytics

GCP: BigQuery

Oracle Cloud: Oracle Autonomous Data Warehouse

1. AWS Redshift

Amazon Redshift is a fully managed, petabyte-scale data warehouse service in the cloud. It allows you to run complex queries and analytics on large datasets quickly and efficiently. Redshift integrates seamlessly with a wide range of AWS services, providing scalable and cost-effective data warehousing solutions.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/redshift_cluster

Terraform Configuration for AWS Redshift:

```
resource "aws_redshift_cluster" "example" {

  cluster_identifier = "my-redshift-cluster"

  database_name      = "dev"

  master_username    = "admin"

  master_password    = "Password123"

  node_type          = "dc2.large"
  cluster_type      = "single-node"
  port              = 5439
}
```

2. Azure Synapse Analytics

Azure Synapse Analytics (formerly SQL Data Warehouse) is an integrated analytics service that accelerates time to insight from big data and data warehousing. It combines enterprise data warehousing and big data analytics into one unified platform, allowing seamless integration with other Azure services.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/4.2.0/docs/resources/synapse_workspace

Terraform Configuration for Azure Synapse Analytics:

```
resource "azurerm_synapse_workspace" "example" {
  name                        = "my-synapse-workspace"
  resource_group_name         = azurerm_resource_group.example.name
  location                    = "East US"
  managed_resource_group_name = "synapse-rg"
}

resource "azurerm_synapse_sql_pool" "example" {
  name                          = "my-synapse-sql-pool"
  resource_group_name          = azurerm_resource_group.example.name
  synapse_workspace_name       = azurerm_synapse_workspace.example.name
  location                     = "East US"
  sku_name                     = "DW100c"
  performance_level            = "DW100c"
}
```

---

3. Google Cloud BigQuery

BigQuery is a fully-managed, serverless data warehouse that allows for scalable analysis of large datasets. It is integrated with Google Cloud's data services and can handle real-time analytics and petabyte-scale data processing.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/4.2.0/docs/resources/synapse_workspace

Terraform Configuration for BigQuery:

```
resource "google_bigquery_dataset" "example" {
  dataset_id = "my_dataset"
  project    = "my-project"
  location   = "US"
}

resource "google_bigquery_table" "example" {
  table_id   = "my_table"

  dataset_id = google_bigquery_dataset.example.dataset_id

  schema     = jsonencode([

    {

      name = "id"

      type = "STRING"

    },

    {

      name = "name"

      type = "STRING"

    }

  ])

}
```

---

4. Oracle Autonomous Data Warehouse

Oracle Autonomous Data Warehouse (ADW) is a fully-managed cloud data warehouse service that delivers high performance, scalability, and automation, designed to reduce the complexity of managing a data warehouse. ADW is optimized for business analytics, machine learning, and reporting workloads.

Official Terraform Documentation:

https://registry.terraform.io/providers/oracle/oci/latest/docs/data-sources/database_autonomous_database

Terraform Configuration for Oracle Autonomous Data Warehouse:

```
resource "oci_data_safe" "example" {
  compartment_id = "<compartment_id>"
  display_name   = "my_data_safe"
}

resource "oci_warehouse" "example" {
  compartment_id = "<compartment_id>"
  db_version     = "19c"
  shape          = "VM.Standard2.4"
  display_name   = "my-adw-instance"
  admin_password = "Password123"
}
```

---

18. Backup and Recovery

AWS: Backup

Azure: Azure Backup (for an Azure VM)

GCP: Backup and DR (Disaster Recovery)

Oracle Cloud: Oracle Cloud Backup

AWS: Backup

AWS Backup is a fully managed backup service that allows you to automate backup tasks for AWS services like EC2, RDS, DynamoDB, and more. It provides centralized backup management, automated backup schedules, and compliance reports.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/backup_plan

Terraform Configuration for AWS Backup:

```
resource "aws_backup_plan" "example" {
  name = "example-backup-plan"
  backup_vault_name = "example-vault"

  rule {
    rule_name         = "daily-backup"
    target_vault_name = "example-vault"
    schedule          = "cron(0 12 * * ? *)"
    lifecycle {
      cold_storage {
        days = 30
      }
    }
  }
}
```

```
resource "aws_backup_vault" "example" {

  name = "example-vault"

}
```

Azure Backup for an Azure VM

Azure Backup is a cloud-based service that protects data in the Azure cloud and on-premises environments. It offers backup for virtual machines, databases, and other resources, along with recovery options for disaster recovery scenarios.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/backup_protected_vm

1. **azurerm_resource_group:**
   - Creating a resource group in the "West Europe" region to contain your Recovery Services Vault.
2. **azurerm_recovery_services_vault:**
   - A Recovery Services Vault (for backup management) within the resource group.
3. **azurerm_backup_policy_vm:**
   - A backup policy that specifies daily backups at 23:00 and retains backups for 10 days.
4. **data "azurerm_virtual_machine" "example"**:
   - A data source to fetch the existing virtual machine you wish to back up. This avoids hardcoding the VM ID and makes your configuration more flexible.
5. **azurerm_backup_protected_vm:**
   - The actual backup configuration for the VM, linking it to the Recovery Services Vault and applying the backup policy.

Here is a slightly refined version of your code to ensure it works smoothly:

```
# Define Resource Group
resource "azurerm_resource_group" "example" {
  name     = "tfex-recovery_vault"
  location = "West Europe"
}

# Define Recovery Services Vault
resource "azurerm_recovery_services_vault" "example" {
  name                = "tfex-recovery-vault"
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  sku                 = "Standard"
}

# Define Backup Policy for VM
resource "azurerm_backup_policy_vm" "example" {
  name                = "tfex-recovery-vault-policy"
  resource_group_name = azurerm_resource_group.example.name
  recovery_vault_name = azurerm_recovery_services_vault.example.name

  backup {
    frequency = "Daily"
    time      = "23:00"
  }

  retention_daily {
    count = 10
  }
}

# Fetch the VM to be backed up
data "azurerm_virtual_machine" "example" {
  name                = "example-vm"
  resource_group_name = azurerm_resource_group.example.name
}

# Apply Backup to the VM
resource "azurerm_backup_protected_vm" "vm1" {
  resource_group_name = azurerm_resource_group.example.name
  recovery_vault_name = azurerm_recovery_services_vault.example.name
  source_vm_id        = data.azurerm_virtual_machine.example.id
  backup_policy_id    = azurerm_backup_policy_vm.example.id
}
```

GCP: Backup and DR (Disaster Recovery)

Google Cloud's Backup and DR solutions help organizations to backup and restore data across GCP services, as well as integrate disaster recovery for both cloud-based and on-premises environments.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/google-beta/latest/docs/resources/backup_dr_backup_plan

Terraform Configuration:

```
resource "google_backup_dr_backup_vault" "my_backup_vault" {
  provider = google-beta
  location = "us-central1"
  backup_vault_id = "bv-bp-test"
  backup_minimum_enforced_retention_duration = "86400s" # example value, 1 day
}

resource "google_backup_dr_backup_plan" "my-backup-plan-1" {
  provider = google-beta
  location = "us-central1"
  backup_plan_id = "backup-plan-simple-test"
  resource_type = "compute.googleapis.com/Instance"
  backup_vault = google_backup_dr_backup_vault.my_backup_vault.id
  backup_rules {
    rule_id = "rule-1"
    backup_retention_days = 5

    standard_schedule {
      recurrence_type = "HOURLY"
      hourly_frequency = 6
      time_zone = "UTC"

      backup_window {
        start_hour_of_day = 0
        end_hour_of_day = 6 # example, a 6-hour window for backup
      }
    }
  }
}
```

---

19. CI/CD Services

AWS: CodePipeline

Azure: Azure DevOps Pipelines

GCP: Cloud Build

Oracle - Developer Services Build Pipeline

Introduction

CI/CD Services streamline the software development lifecycle by automating code integration, testing, and deployment to production environments.

1. AWS - CodePipeline

AWS CodePipeline: A fully managed service that automates the build, test, and deploy phases for fast and reliable application delivery on AWS.

Official Terraform Documentation:
https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/codepipeline

Terraform Configuration:

```
provider "aws" {

  region = "us-east-1"

}


resource "aws_s3_bucket" "artifact_store" {

  bucket = "codepipeline-artifacts"

}


resource "aws_iam_role" "codepipeline_role" {

  name = "codepipeline-role"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Action    = "sts:AssumeRole"
      Effect    = "Allow"
      Principal = { Service = "codepipeline.amazonaws.com" }
    }]
  })
}


resource "aws_codepipeline" "pipeline" {
  name     = "example-pipeline"
  role_arn = aws_iam_role.codepipeline_role.arn

  artifact_store {
    type     = "S3"
    location = aws_s3_bucket.artifact_store.bucket
  }

  stage {
    name = "Source"
    action {
      name             = "Source"
      category         = "Source"
      owner            = "AWS"
      provider         = "S3"
      version          = "1"
      output_artifacts = ["source_output"]

      configuration = {
        S3Bucket = aws_s3_bucket.artifact_store.bucket
        S3ObjectKey = "source.zip"
      }
    }
  }

  stage {
    name = "Build"

    action {
      name             = "Build"
      category         = "Build"
      owner            = "AWS"
      provider         = "CodeBuild"
      version          = "1"
      input_artifacts  = ["source_output"]
      output_artifacts = ["build_output"]

      configuration = {
        ProjectName = aws_codebuild_project.example.name
      }
    }
  }
}

resource "aws_codebuild_project" "example" {
  name          = "example-codebuild-project"
  service_role  = aws_iam_role.codepipeline_role.arn
  artifacts {
    type = "S3"
  }

  environment {
    compute_type                = "BUILD_GENERAL1_SMALL"
    image                       = "aws/codebuild/standard:5.0"
    type                        = "LINUX_CONTAINER"
    privileged_mode             = true
  }

  source {
    type = "S3"
    location = aws_s3_bucket.artifact_store.bucket
  }
}
```

---

2. azuredevops_pipeline_authorization

Azure DevOps Pipeline Authorization is the process of granting Azure DevOps pipelines permission to access resources within your Azure environment. This is typically done by using a **Service Connection** that securely authenticates the pipeline to Azure services and resources, enabling tasks like resource provisioning, deployments, and infrastructure management.

In Terraform, pipeline authorization is managed through service principal authentication or Managed Identity, enabling Azure DevOps pipelines to interact with Azure resources securely.

Official Terraform Documentation:

https://registry.terraform.io/providers/microsoft/azuredevops/latest/docs/resources/pipeline_authorization

Terraform configuration to create a **Service Principal** for Azure DevOps to use in a pipeline:

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

resource "azurerm_service_principal" "example" {
  application_id = azurerm_azuread_application.example.application_id
}

resource "azurerm_azuread_application" "example" {
  name                       = "example-app"
  homepage                   = "https://example.com"
  identifier_uris            = ["https://example.com"]
  reply_urls                 = ["https://example.com/response"]
}

resource "azurerm_role_assignment" "example" {
  principal_id   = azurerm_service_principal.example.id
  role_definition_name = "Contributor"
  scope          = "/subscriptions/${data.azurerm_client_config.example.subscription_id}"
}
```

3. GCP - Cloud Build

GCP Cloud Build: A serverless CI/CD platform for automating builds and deployments on Google Cloud.

Official Terraform Documentation:
https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/cloudbuild_trigger

GCP uses **Cloud Build** for CI/CD:

```
provider "google" {
  project = "example-project"
  region  = "us-central1"
}


resource "google_storage_bucket" "build_artifacts" {
  name     = "cloud-build-artifacts"
  location = "US"
}


resource "google_cloudbuild_trigger" "example" {
  name = "example-trigger"


  trigger_template {
    project_id = "example-project"
    branch_name = "main"
    repo_name   = "example-repo"
  }
  build {
    steps {
      name = "gcr.io/cloud-builders/docker"
      args = ["build", "-t", "gcr.io/${var.project_id}/example", "."]
    }

    images = ["gcr.io/${var.project_id}/example"]
  }
}
```

---

4. oci_devops_build_pipeline

The oci_devops_build_pipeline resource in Terraform enables you to create and manage CI/CD pipelines on Oracle Cloud Infrastructure (OCI). It automates the process of building, testing, and deploying applications, integrating with OCI services like Object Storage, Compute, and Kubernetes.

Example Terraform Configuration:

```
resource "oci_devops_build_pipeline" "example_pipeline" {
  compartment_id = "<compartment_ocid>"
  display_name   = "Example Build Pipeline"

  project_id = "<devops_project_ocid>"
  build_spec {
    build_steps {
      display_name = "Build Step"
      script {
        source = "<script_source>"
      }
    }
  }

  deploy_spec {
    deploy_steps {
      display_name = "Deploy Step"
      type = "OCI_COMPUTE"
      deploy_parameters {
        instance_id = "<compute_instance_id>"
      }
    }
  }
}
```

This Terraform configuration defines a simple build pipeline with build and deploy steps. You can modify the configuration to match your specific use case, such as adding stages for tests, setting up notifications, or configuring different deployment targets.

---

20. Infrastructure as Code

AWS: CloudFormation

Azure: Azure Resource Manager (ARM) Templates

GCP: Deployment Manager

Oracle: OCI Resource Manager

CloudFormation (AWS)

AWS CloudFormation is an infrastructure-as-code (IaC) service that allows you to define and manage AWS resources using JSON or YAML templates. It automates resource provisioning, scaling, and configuration management, ensuring consistent environments.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/cloudformation_stack

Terraform Configuration Example:

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_cloudformation_stack" "network" {
  name = "networking-stack"

  parameters = {
    VPCCidr = "10.0.0.0/16"
```

```
}

template_body = jsonencode({
  AWSTemplateFormatVersion = "2010-09-09"

  Description = "CloudFormation template to create a VPC"

  Parameters = {
    VPCCidr = {
      Type        = "String"
      Default     = "10.0.0.0/16"
      Description = "Enter the CIDR block for the VPC. Default is 10.0.0.0/16."
    }
  }

  Resources = {
    myVpc = {
      Type = "AWS::EC2::VPC"
      Properties = {
        CidrBlock = {
          Ref = "VPCCidr"
        }
        Tags = [
            {
              Key   = "Name"
              Value = "Primary_CF_VPC"
            }
          ]
        }
      }
    }
  })
}
```

Azure Resource Manager (ARM) Templates

Introduction: Azure Resource Manager (ARM) Templates enable declarative management of Azure resources. Using JSON files, ARM Templates facilitate the automation of resource provisioning and management, ensuring consistent deployments across environments.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/resource_group_template_deployment

Terraform Configuration Example:

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

resource "azurerm_storage_account" "example" {
  name                     = "mystorageaccount"
  resource_group_name      = azurerm_resource_group.example.name
  location                 = azurerm_resource_group.example.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
  kind                     = "StorageV2"
}

resource "azurerm_resource_group" "example" {
  name     = "example-resource-group"
  location = "eastus"
}
```

Deployment Manager (GCP)

Introduction: Google Cloud Deployment Manager is an IaC tool that allows you to specify GCP resources in YAML, Python, or Jinja2 templates. It helps automate the provisioning and management of Google Cloud resources in a repeatable and predictable manner.

Terraform Configuration Example:

```
provider "google" {
  project = "my-gcp-project"
  region  = "us-central1"
}


resource "google_compute_instance" "example" {
  name         = "example-instance"
  machine_type = "e2-micro"
  zone         = "us-central1-a"


  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-11"
    }
  }


  network_interface {
    network = "default"
  }
}
```

OCI Resource Manager (Oracle)

Introduction: Oracle Cloud Infrastructure (OCI) Resource Manager is a managed service that helps you provision OCI resources using Terraform. It simplifies resource lifecycle management while ensuring consistency through Terraform templates.

Official Terraform Documentation:

https://registry.terraform.io/providers/oracle/oci/latest/docs/resources/resourcemanager_private_endpoint

Terraform Configuration Example:

```
provider "oci" {

  tenancy     = "ocid1.tenancy.oc1..example"

  user        = "ocid1.user.oc1..example"

  fingerprint = "example-fingerprint"

  private_key = file("~/.oci/oci_api_key.pem")

  region      = "us-ashburn-1"

}


resource "oci_core_vcn" "example" {

  cidr_block     = "10.0.0.0/16"

  display_name   = "example-vcn"
  compartment_id = "ocid1.compartment.oc1..example"
```

---

21. Container Services

AWS: ECS / EKS / Fargate

Azure: Azure Container Instances / AKS

GCP: Google Kubernetes Engine (GKE)

Oracle: Oracle Container Engine for Kubernetes (OKE)

AWS

ECS (Elastic Container Service)

A highly scalable and secure container orchestration service that supports Docker containers, allowing you to run and manage containers without the need for a separate orchestrator.

EKS (Elastic Kubernetes Service)

Managed Kubernetes service that simplifies the deployment, management, and scaling of Kubernetes clusters.

Fargate

A serverless compute engine for containers that works with ECS and EKS, enabling you to run containers without managing the underlying infrastructure.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/ecs_cluster

Terraform Example:

```
provider "aws" {
  region = "us-east-1"
}
```

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/eks_cluster

Terraform Configuration:

```
resource "aws_ecs_cluster" "example" {
  name = "example-ecs-cluster"
}


resource "aws_eks_cluster" "example" {
  name     = "example-eks-cluster"
  role_arn = aws_iam_role.example.arn
  vpc_config {
    subnet_ids = aws_subnet.example.*.id
  }
}
```

```
}
```

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/3.14.1/docs/resources/ecs_service

Terraform Configuration:

```
resource "aws_ecs_service" "example" {
  name            = "example-service"
  cluster         = aws_ecs_cluster.example.id
  launch_type     = "FARGATE"
  desired_count   = 1
  task_definition = aws_ecs_task_definition.example.arn
}
```

Azure

AKS (Azure Kubernetes Service)

A managed Kubernetes service that simplifies Kubernetes cluster management, including upgrades, scaling, and security patches.

Terraform Example:

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

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/kubernetes_cluster

Terraform Configuration:

```
resource "azurerm_kubernetes_cluster" "example" {
  name                = "example-aks-cluster"
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  dns_prefix          = "exampleaks"

  default_node_pool {
    name       = "default"
    node_count = 2
    vm_size    = "Standard_DS2_v2"
  }
}
```

Azure Container Instances (ACI)

A quick and efficient way to run containers on Azure without managing virtual machines.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/1.43.0/docs/resources/container_group

```
resource "azurerm_container_group" "example" {
  name                = "example-container-group"
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name

  container {
    name   = "example"
    image  = "nginx"
    cpu    = "0.5"
    memory = "1.5"
  }
}
```

GCP

Google Kubernetes Engine (GKE)

Managed Kubernetes service to deploy, manage, and scale containerized applications using Kubernetes on Google Cloud.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/container_cluster

Terraform Example:

```
provider "google" {
  project = "my-gcp-project"
  region  = "us-central1"
}

resource "google_container_cluster" "example" {
  name     = "example-gke-cluster"
  location = "us-central1"

  node_config {
    machine_type = "e2-medium"
    disk_size_gb = 100
    oauth_scopes = ["https://www.googleapis.com/auth/cloud-platform"]
  }
}
```

Oracle

Oracle Container Engine for Kubernetes (OKE)

A managed Kubernetes service that lets you deploy and manage Kubernetes clusters on Oracle Cloud Infrastructure.

Official Terraform Documentation:

https://registry.terraform.io/providers/oracle/oci/latest/docs/resources/containerengine_cluster

Terraform Example:

```
resource "oci_containerengine_cluster" "test_cluster" {
    # Required
    compartment_id = var.compartment_id
    kubernetes_version = var.cluster_kubernetes_version
    name = var.cluster_name
    vcn_id = oci_core_vcn.test_vcn.id

    # Cluster Pod Network Options (Required)
    cluster_pod_network_options {
        cni_type = var.cluster_cluster_pod_network_options_cni_type
    }

    # Required tags
    defined_tags = {"Operations.CostCenter" = "42"}
    freeform_tags = {"Department" = "Finance"}

    # Type
    type = var.cluster_type
}
```

---

22. Configuration Management

AWS: Systems Manager

Azure: Azure Automation

GCP: Cloud Deployment Manager

Oracle: OCI Configuration Management

AWS: Systems Manager

AWS Systems Manager (SSM) is a unified interface to manage and automate infrastructure tasks across AWS resources. It allows you to configure and maintain the system without needing to manually interact with each instance.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/ssm_parameter

Terraform Example:

```
resource "aws_ssm_parameter" "example" {

  name  = "/example/parameter"

  type  = "String"

  value = "This is a parameter"

}
```

Azure: Azure Automation

Azure Automation allows you to automate repetitive tasks and configuration management for your Azure infrastructure. It integrates with Azure Resource Manager (ARM) to automate processes such as patch management and configuration drift.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/automation_account

Terraform Example:

```
resource "azurerm_automation_account" "example" {
  name                = "example-automation"
  location            = "East US"
  resource_group_name = azurerm_resource_group.example.name
}
```

GCP: Cloud Deployment Manager

Google Cloud Deployment Manager is used to automate the deployment of Google Cloud resources. It allows you to define infrastructure in YAML, JSON, or Python files, which can then be deployed through Terraform or directly.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/deployment_manager_deployment

Terraform Example:

```
resource "google_deployment_manager_deployment" "example" {
  name = "example-deployment"
  target {
    config = "config.yaml"
  }
}
```

Oracle: OCI Configuration Management

Oracle Cloud Infrastructure (OCI) Configuration Management provides tools to manage the configuration of your cloud infrastructure, ensuring consistency and compliance across your resources.

Terraform Example:

```
resource "oci_identity_dynamic_group" "example" {
  compartment_id = var.compartment_id
  description   = "Example Dynamic Group"
  name          = "example_dynamic_group"
}
```

23. Developer Tools

AWS: AWS CodePipeline, CodeBuild, CodeDeploy, CodeCommit

Azure: Azure DevOps, Azure Repos, Azure Pipelines

GCP: Cloud Build, Cloud Source Repositories

Oracle Cloud: Oracle Developer Cloud Service

1. AWS Services

AWS CodePipeline

AWS CodePipeline is a fully managed continuous delivery service for fast and reliable application and infrastructure updates. It automates the build, test, and deployment phases of your release process.

Official Terraform Documentation:
https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/codepipeline

Terraform Configuration:

```
resource "aws_codepipeline" "example" {
  name     = "example-pipeline"
  role_arn = aws_iam_role.codepipeline_role.arn

  artifact_store {
    location = "s3-bucket-name"
    type     = "S3"
  }
  stages {
    name = "Source"
    action {
      name             = "SourceAction"
      category         = "Source"
      owner            = "AWS"
      provider         = "S3"
      version          = "1"
      output_artifacts = ["source_output"]
      configuration = {
        S3Bucket    = "source-bucket"
        S3ObjectKey = "source.zip"
      }
    }
  }
}
```

AWS CodeBuild

AWS CodeBuild is a fully managed build service that compiles source code, runs tests, and produces software packages. It integrates with other AWS services like CodePipeline to provide a full CI/CD pipeline.

Official Terraform Documentation:
https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/codebuild_project

Terraform Configuration:

```
resource "aws_codebuild_project" "example" {
  name          = "example-project"
  description   = "Example CodeBuild project"
  build_timeout = 5

  environment {
    compute_type = "BUILD_GENERAL1_SMALL"
    image        = "aws/codebuild/standard:4.0"
    type         = "LINUX_CONTAINER"
  }

  service_role = aws_iam_role.codebuild_role.arn

  source {
    type            = "S3"
    location        = "s3://source-bucket/source.zip"
    buildspec       = "buildspec.yml"
  }
}
```

AWS CodeDeploy

AWS CodeDeploy automates code deployments to any instance, including EC2, on-premises servers, and Lambda. It ensures smooth application updates with minimal downtime.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/codedeploy_deployment_group

Terraform Configuration:

```
resource "aws_codedeploy_app" "example" {
  name = "example-app"
  compute_platform = "Server"
}

resource "aws_codedeploy_deployment_group" "example" {
  app_name              = aws_codedeploy_app.example.name
  deployment_group_name  = "example-deployment-group"
  service_role_arn       = aws_iam_role.codedeploy_role.arn
}
```

---

2. Azure Services

Azure DevOps

Azure DevOps provides a suite of development tools and services for continuous integration, deployment, version control, and project management.

Terraform Configuration:
https://registry.terraform.io/providers/microsoft/azuredevops/latest/docs

Terraform Configuration:

```
terraform {
  required_providers {
    azuredevops = {
      source  = "microsoft/azuredevops"
      version = ">= 0.1.0"
    }
  }
}

provider "azuredevops" {
  # Optional: Authentication settings like `org_service_url` and `personal_access_token` can be defined here.
  org_service_url     = "https://dev.azure.com/your_organization"
  personal_access_token = var.azure_devops_pat
}

resource "azuredevops_project" "project" {
  name        = "Project Name"
  description = "Project Description"
  visibility  = "private" # Options: "private" or "public", default is "private"
}
```

Azure Repos

Azure Repos offers Git repositories or Team Foundation Version Control (TFVC) for source control.

Official Terraform Documentation:
https://registry.terraform.io/providers/microsoft/azuredevops/latest/docs/resources/git_repository

Terraform Configuration:

```
resource "azuredevops_git_repository" "example" {
  project_id = azuredevops_project.example.id
  name       = "example-repository"
  initialization {
    init_type = "Clean"
  }
}
```

Azure Pipelines

Azure Pipelines is a cloud-based CI/CD service for building, testing, and deploying code to any platform.

Official Terraform Documentation:

https://registry.terraform.io/providers/microsoft/azuredevops/latest/docs/resources/pipeline_authorization

Terraform Configuration:

```
resource "azuredevops_pipeline" "example" {
  project_id     = azuredevops_project.example.id
  name           = "example-pipeline"
  yaml_path      = "azure-pipelines.yml"
  folder         = "\\example-folder"     # Optional: Specify folder path in Azure DevOps for organization
  repository {
    repository_id = azuredevops_git_repository.example.id
    branch        = "main"                # Specify the branch to use
    type          = "TfsGit"              # Type of repository (e.g., TfsGit, GitHub, Bitbucket, etc.)
  }
  variables {
    name  = "Environment"
    value = "Production"                  # Example variable definition
  }
  triggers {
    branch_filter = ["main"]              # Configure branch filters for triggers
    batch         = true                  # Batch changes into a single build
  }
}
```

3. GCP Services

Cloud Build

Google Cloud Build is a fast and scalable CI/CD service for automating the build and deployment of applications on Google Cloud.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/cloudbuild_trigger

Terraform Configuration:

```
resource "google_cloudbuild_trigger" "example" {
  name = "example-trigger"
  github {
    owner  = "your-github-owner"
    repo   = "your-repo"
    push {
      branch = "main"
    }
  }

  build {
    steps {
      name = "gcr.io/cloud-builders/mvn"
      args = ["clean", "install"]
    }

  }

}
```

Cloud Source Repositories

Google Cloud Source Repositories is a fully-featured Git repository hosted on Google Cloud for managing your source code.

Terraform Configuration:

```
resource "google_sourcerepo_repository" "example" {

  name = "example-repo"

}
```

---

4. Oracle Cloud Services

Oracle Developer Cloud Service

Oracle Developer Cloud Service provides a comprehensive set of DevOps tools, including source code management, build automation, and deployment pipelines.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/sourcerepo_repository

Terraform Configuration:

```
provider "oci" {
  region = "us-phoenix-1"
}

resource "oci_devops_repository" "example" {
  repository_name = "example-repository"
  project_id      = oci_devops_project.example.id
}

resource "oci_devops_project" "example" {
  project_name = "example-project"
}
```

---

24. Business Analytics and Reporting

AWS: QuickSight

Azure: Power BI

GCP: Looker

Oracle Cloud: Oracle Analytics Cloud

1. AWS: QuickSight

Amazon QuickSight is a scalable, business intelligence service built for the cloud that allows you to create and publish interactive dashboards. It enables fast data analysis and visualizations with a variety of data sources.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/quicksight_group

Terraform Configuration for AWS QuickSight:

```
resource "aws_quicksight_group" "example" {
  aws_account_id = "your_account_id"
  namespace      = "default"
  group_name     = "example-group"
  description    = "Example QuickSight group"
}

resource "aws_quicksight_user" "example" {
  aws_account_id = "your_account_id"
  namespace      = "default"
  user_name      = "example-user"
  email          = "user@example.com"
  identity_type  = "IAM"
  group_name     = aws_quicksight_group.example.group_name
}
```

---

2. Azure: Power BI

Power BI is a Microsoft tool for business analytics that helps visualize data, share insights, and collaborate in real time. Power BI can connect to a wide range of data sources to generate meaningful reports and dashboards.

This Terraform configuration defines resources for deploying an Azure Resource Group and a Power BI Embedded resource. The azurerm_resource_group block creates a resource group in the West Europe region, which serves as a container for managing related Azure resources. The azurerm_powerbi_embedded block provisions a Power BI Embedded capacity within this resource group, enabling the embedding of interactive Power BI reports and dashboards into applications. This configuration ensures proper linkage between the resources by referencing the resource group details dynamically in the Power BI resource definition.

```
resource "azurerm_resource_group" "example" {

  name     = "example-resources"

  location = "West Europe"

}


resource "azurerm_powerbi_embedded" "example" {

  name                = "examplepowerbi"

  location            = azurerm_resource_group.example.location

  resource_group_name = azurerm_resource_group.example.name

  sku_name            = "A1"

  administrators      = ["azsdktest@microsoft.com"]

}
```

3. GCP: Looker

Looker is a data exploration and business intelligence tool on Google Cloud that helps users to analyze and visualize data, creating interactive dashboards and reports. It integrates with BigQuery and other data warehouses on GCP.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/looker_instance

Terraform Configuration for GCP Looker:

```
resource "google_looker_instance" "looker-instance" {
  name             = "my-instance"
  platform_edition  = "LOOKER_CORE_STANDARD_ANNUAL"
  region           = "us-central1"
  oauth_config {
    client_id = "my-client-id"
    client_secret = "my-client-secret"
  }
  deletion_policy = "DEFAULT"
}
```

---

4. Oracle Cloud: Oracle Analytics Cloud

Oracle Analytics Cloud (OAC) provides comprehensive analytics tools including data visualization, reporting, and dashboards. It integrates with Oracle Cloud services to enable deep data insights across different business functions.

Official Terraform Documentation:

https://registry.terraform.io/providers/oracle/oci/latest/docs/resources/analytics_analytics_instance

Terraform Configuration for Oracle Analytics Cloud:

```
provider "oci" {

  region = "us-phoenix-1"

}


resource "oci_analytics_instance" "example" {

  compartment_id = "your_compartment_id"

  display_name   = "example-analytics-instance"

  shape          = "oc4"

  license_type   = "LICENSE_INCLUDED"

  subnet_id      = "your_subnet_id"

}
```

25. Artificial Intelligence (AI) and Natural Language Processing (NLP)

AWS: Amazon Lex, Polly, Comprehend, Rekognition

Azure: Azure Cognitive Services (Speech, Vision, Language)

GCP: Cloud Natural Language, Cloud Vision, Dialogflow

Oracle Cloud: Oracle AI Services

Artificial Intelligence (AI) and Natural Language Processing (NLP)

AI and NLP involve using machine learning models and algorithms to process and analyze human language, enabling machines to understand, interpret, and respond to text and speech data. These services are widely used in applications like chatbots, voice assistants, sentiment analysis, image recognition, and more.

AWS AI Services:

Amazon Lex: A service for building conversational interfaces using voice and text. It powers chatbots and virtual assistants.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/lex_bot

- Terraform Configuration:

```
resource "aws_lex_bot" "example" {

  name        = "exampleBot"

  description = "Sample bot"

  locale      = "en-US"

  clarification_prompt {
    messages = ["Sorry, I didn't get that. Could you repeat?"]

  }

  intent {

    intent_name = "HelloIntent"

    intent_version = "$LATEST"

  }

}
```

Amazon Polly: A service that turns text into lifelike speech, enabling applications to read aloud content.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/data-sources/polly_voices

- Terraform Configuration:

```
resource "aws_polly_vocabulary" "example" {

  name = "exampleVocabulary"

  language_code = "en-US"

  vocabulary_file = "s3://bucket-name/vocabulary.json"

}
```

Amazon Comprehend: A natural language processing (NLP) service that analyzes text for key phrases, entities, and sentiment.

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/comprehend_document_classifier

- Terraform Configuration:

```
resource "aws_comprehend_document_classification_model" "example" {
  model_name = "exampleModel"

  input_data_config {
    s3_uri = "s3://bucket-name/data/"
  }

  output_data_config {
    s3_uri = "s3://bucket-name/output/"
  }

  role_arn = "arn:aws:iam::account-id:role/comprehend-role"
}
```

2. **Amazon Rekognition**: A service that provides image and video analysis, including facial recognition, object detection, and text recognition.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/5.56.1/docs/resources/rekognition_collection

- **Terraform Configuration:**

```
resource "aws_rekognition_collection" "example" {
  name = "exampleCollection"
}
```

Azure AI Services:

1. **Azure Cognitive Services - Speech**: A set of APIs for speech-to-text, text-to-speech, and speech translation.

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/cognitive_account

- Terraform Configuration:

```
resource "azurerm_cognitive_account" "example" {
  name                = "exampleCognitiveAccount"
  resource_group_name = "exampleResourceGroup"
  location            = "East US"
  kind                = "Speech"
  sku_name            = "S1"
}
```

2. **Azure Cognitive Services - Vision**: Offers tools for image recognition, object detection, and facial recognition.
    - Terraform Configuration:

```
resource "azurerm_cognitive_account" "example" {

  name                = "exampleVisionAccount"

  resource_group_name = "exampleResourceGroup"

  location            = "East US"

  kind                = "Vision"

  sku_name            = "S1"

}
```

3. **Azure Cognitive Services - Language**: Provides language understanding tools such as text analytics, sentiment analysis, and entity extraction.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/cognitive_account

- **Terraform Configuration:**

```
resource "azurerm_cognitive_account" "example" {

  name                = "exampleLanguageAccount"

  resource_group_name = "exampleResourceGroup"

  location            = "East US"
  kind                = "Language"

  sku_name            = "S1"

}
```

GCP AI Services:

1. **Cloud Natural Language**: Analyzes and understands text through sentiment analysis, entity recognition, and content classification.
   - ○ Terraform Configuration:

```
resource "google_project" "example" {

  name       = "example-project"

  project_id = "example-project-id"

}
```

2. **Cloud Vision**: Offers image recognition capabilities like detecting labels, text, and faces.

   Official Terraform Documentation:

   https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/dialogflow_agent

   - ○
   - ○ Terraform Configuration:

```
resource "google_project" "example" {

  name       = "example-project"

  project_id = "example-project-id"
}
```

3. **Dialogflow**: A platform for building conversational interfaces, supporting text and voice-based interactions.
   - Terraform Configuration:

```
resource "google_dialogflow_agent" "example" {
  display_name = "exampleAgent"
  project_id   = "example-project-id"
}
```

4.

Oracle Cloud AI Services:

1. **Oracle AI Services**: Includes various AI capabilities such as vision, speech, and language understanding.

Official Terraform Documentation:

https://registry.terraform.io/providers/oracle/oci/latest/docs

Terraform Configuration:

```
resource "oci_ai_vision_service" "example" {
  compartment_id = "compartment_id"
  display_name   = "exampleAIService"
  description    = "AI Vision Service"
          }
```

26. Machine Learning

AWS: SageMaker

Azure: Azure Machine Learning

GCP: AI Platform

Oracle Cloud: Oracle Cloud AI and Machine Learning

1. AWS: SageMaker

Introduction: Amazon SageMaker is a fully managed service that enables developers and data scientists to quickly build, train, and deploy machine learning models at scale. It provides pre-built algorithms, model training, and deployment capabilities, making it easier to develop robust ML solutions.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/sagemaker_notebook_instance

Terraform Configuration Example:

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_sagemaker_notebook_instance" "example" {
  name                = "my-notebook-instance"
  instance_type       = "ml.t2.medium"
  role_arn            = "arn:aws:iam::account-id:role/my-sagemaker-role"
  notebook_instance_lifecycle_config_name = "lifecycle-config"
}
```

---

2. Azure: Azure Machine Learning

Introduction: Azure Machine Learning is a cloud-based platform by Microsoft that enables data scientists and developers to build, train, and deploy machine learning models. It supports a variety of frameworks like TensorFlow, PyTorch, and Scikit-learn.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/machine_learning_workspace

Terraform Configuration Example:

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

resource "azurerm_machine_learning_workspace" "example" {
  name                = "example-workspace"
  location            = "East US"
  resource_group_name = "example-rg"
}
```

---

3. GCP: AI Platform

Introduction: Google Cloud AI Platform provides a suite of machine learning tools to help you develop, train, and deploy ML models. It supports deep learning and integrates well with TensorFlow, Scikit-learn, and other libraries.

Official Terraform Documentation:

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/notebooks_instance

Terraform Configuration Example:

```
provider "google" {
  project = "your-project-id"
  region  = "us-central1"
}

resource "google_ai_platform_notebook_instance" "example" {
  name   = "my-instance"
  zone   = "us-central1-a"
  machine_type = "n1-standard-4"
  software_config {
    notebook_version = "tfa-ml-1"
  }
}
```

4. Oracle Cloud: Oracle Cloud AI and Machine Learning

Introduction: Oracle Cloud AI and Machine Learning services offer a broad set of tools for building AI and machine learning models, including data science workspaces and pre-built models. It supports both AutoML and custom model creation.

Official Terraform Documentation:

https://registry.terraform.io/providers/oracle/oci/latest/docs

Terraform Configuration Example:

```
provider "oci" {
  tenancy_ocid     = "your-tenancy-ocid"
  user_ocid        = "your-user-ocid"
  fingerprint      = "your-fingerprint"
  private_key_path = "path-to-your-private-key.pem"
}

resource "oci_ai_services_model" "example" {
  compartment_id = "your-compartment-id"
  display_name   = "my-ai-model"
  model_type     = "tensorflow"
}
```