Day 10: Exam Preparation and Industry Application

Final Review and Mock Exam

Today, we'll focus on reviewing key concepts covered throughout the course and prepare for the Terraform Associate certification exam.

We'll also discuss how Terraform is applied in industry settings.

Key Concepts Review:

1. Infrastructure as Code (IaC) principles

2. Terraform basics: providers, resources, and state management

3. Terraform configuration language (HCL)

4. Modules and reusability

5. Workspaces and environment management

6. Collaboration and version control with Terraform

7. Security and compliance in Terraform

8. Advanced Terraform features and troubleshooting

9. Multi-cloud management with Terraform

Industry Applications

How Companies Hire for Terraform-Related Roles

1. DevOps Engineer

2. Infrastructure Engineer

3. Cloud Architect

4. Site Reliability Engineer (SRE)

5. Platform Engineer

Skills and Experience Companies Look for in Terraform Professionals

1. Strong understanding of IaC principles

2. Proficiency in at least one major cloud platform (AWS, Azure, GCP)

3. Experience with version control systems (e.g., Git)

4. Knowledge of CI/CD pipelines

5. Familiarity with security best practices

6. Scripting skills (e.g., Python, Bash)

7. Understanding of networking concepts

8. Experience with containerization and orchestration (e.g., Docker, Kubernetes)

Industry Insight

1. Terraform has become a critical tool in modern infrastructure management.
2. Companies across various industries are leveraging Terraform to streamline their operations and improve efficiency.

Real-world scenario: E-commerce giant scales infrastructure globally

A leading e-commerce company faced challenges in scaling its infrastructure to meet growing global demand.

They implemented a comprehensive Terraform strategy to manage their multi-region, multi-cloud infrastructure.

The company:

1. Developed a modular Terraform structure for reusable components across regions

2. Implemented a custom Terraform wrapper for standardized deployments

3. Integrated Terraform with their existing CI/CD pipeline for automated infrastructure updates

4. Used Terraform workspaces to manage multiple environments (dev, staging, production) in each region

Results:

- 75% reduction in time-to-market for new regional deployments

- 50% decrease in infrastructure-related incidents

- Improved developer productivity with standardized infrastructure templates

- Enhanced compliance with automated policy checks in the deployment process

Hands-On Activity: Comprehensive Terraform Project

In this final hands-on activity, we'll create a comprehensive Terraform project that incorporates many of the concepts learned throughout the course.

This project will set up a multi-environment, multi-region infrastructure on AWS.

```
# main.tf

terraform {

  required_providers {

    aws = {

      source  = "hashicorp/aws"

      version = "~> 3.0"

    }

  }

  backend "s3" {

    bucket = "my-terraform-state-bucket"

    key    = "global/s3/terraform.tfstate"

    region = "us-west-2"

    dynamodb_table = "terraform-locks"

    encrypt        = true

  }

}

provider "aws" {

  region = var.region

}

module "vpc" {

  source = "./modules/vpc"

  region = var.region

  environment = terraform.workspace

}

module "ec2" {

  source = "./modules/ec2"

  vpc_id = module.vpc.vpc_id
```

```
  subnet_ids = module.vpc.public_subnet_ids
  environment = terraform.workspace
}
module "rds" {
  source = "./modules/rds"
  vpc_id = module.vpc.vpc_id
  subnet_ids = module.vpc.private_subnet_ids
  environment = terraform.workspace
}
# variables.tf
variable "region" {
  description = "AWS region"
  type        = string
  default     = "us-west-2"
}
# outputs.tf
output "vpc_id" {
  value = module.vpc.vpc_id
}
output "ec2_public_ips" {
  value = module.ec2.public_ips
}
output "rds_endpoint" {
  value = module.rds.endpoint
}
# modules/vpc/main.tf
resource "aws_vpc" "main" {
```

```
  cidr_block = "10.0.0.0/16"
  tags = {
    Name = "${var.environment}-vpc"
  }
}
resource "aws_subnet" "public" {
  count           = 2
  vpc_id          = aws_vpc.main.id
  cidr_block      = "10.0.${count.index + 1}.0/24"
  availability_zone = "${var.region}${count.index == 0 ? "a" : "b"}"
  tags = {
    Name = "${var.environment}-public-subnet-${count.index + 1}"
  }
}
resource "aws_subnet" "private" {
  count           = 2
  vpc_id          = aws_vpc.main.id
  cidr_block      = "10.0.${count.index + 10}.0/24"
  availability_zone = "${var.region}${count.index == 0 ? "a" : "b"}"
   tags = {
    Name = "${var.environment}-private-subnet-${count.index + 1}"
  }
}
# modules/ec2/main.tf
resource "aws_instance" "app_server" {
  count       = 2
  ami         = "ami-0c55b159cbfafe1f0"
```

```
  instance_type = "t2.micro"
  subnet_id     = var.subnet_ids[count.index]
 tags = {
    Name = "${var.environment}-app-server-${count.index + 1}"
  }
}
# modules/rds/main.tf
resource "aws_db_instance" "default" {
  allocated_storage    = 10
  engine               = "mysql"
  engine_version       = "5.7"
  instance_class       = "db.t3.micro"
  name                 = "${var.environment}db"
  username             = "admin"
  password             = random_password.db_password.result
  parameter_group_name = "default.mysql5.7"
  skip_final_snapshot  = true
  db_subnet_group_name = aws_db_subnet_group.default.name
 tags = {
    Environment = var.environment
  }
}
resource "random_password" "db_password" {
  length  = 16
  special = true
}
resource "aws_db_subnet_group" "default" {
```

```
  name       = "${var.environment}-db-subnet-group"
  subnet_ids = var.subnet_ids
  tags = {
    Name = "${var.environment} DB subnet group"
  }
}
```

To use this Terraform project:

1. Set up an S3 bucket and DynamoDB table for remote state storage.

2. Initialize the Terraform working directory:

```
terraform init
```

3. Create workspaces for different environments:

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

4. Plan and apply the Terraform configuration for each environment:

```
terraform workspace select dev
terraform plan
terraform apply
terraform workspace select staging
terraform plan
terraform apply
terraform workspace select prod
terraform plan
terraform apply
```

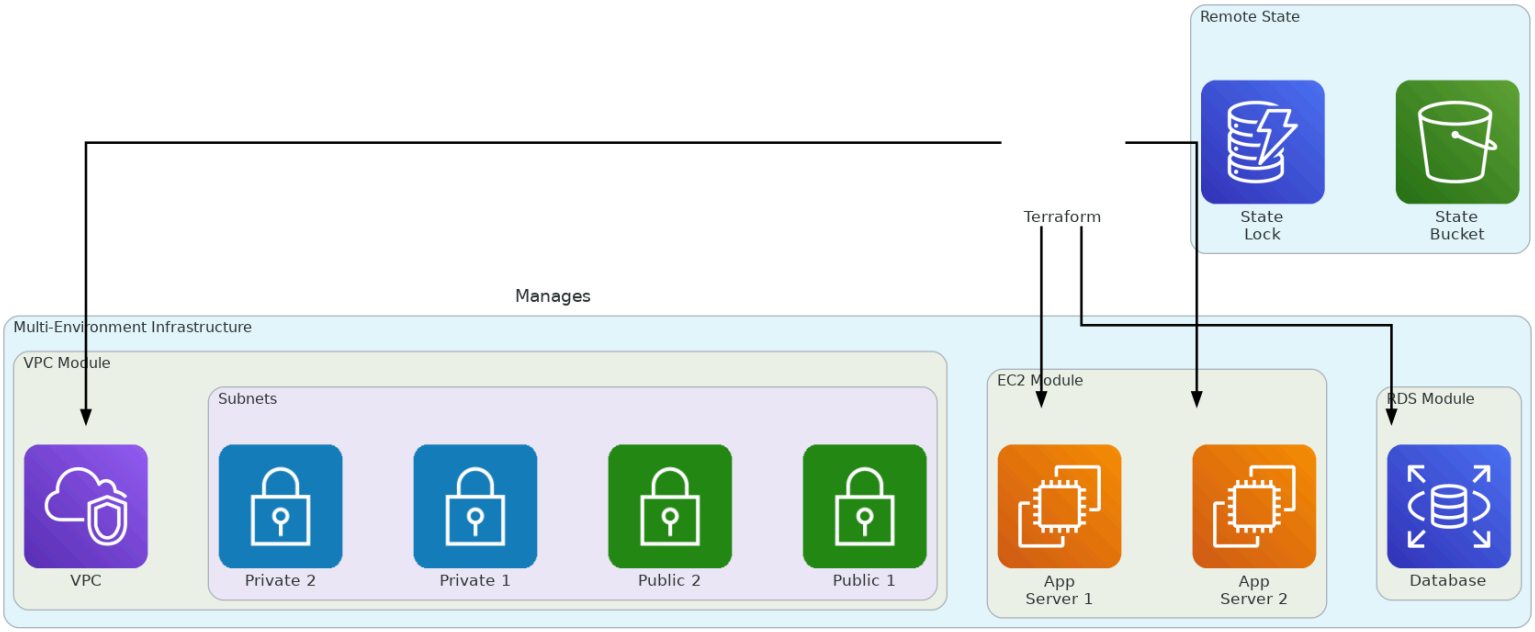
This project demonstrates:

- Modular Terraform structure
- Multi-environment management using workspaces

- Remote state management

- VPC, EC2, and RDS resource creation

- Use of variables and outputs



Comprehensive Terraform Project Architecture

To visualize the architecture of our comprehensive Terraform project, let's create a diagram using Python and the Diagrams library.

This diagram illustrates:

- The overall Terraform project structure

- Remote state management with S3 and DynamoDB

- VPC module with public and private subnets

- EC2 instances in public subnets

- RDS instance in private subnet

- Relationships between different components

Mock Exam

1. Which Terraform command is used to preview the changes that will be made to your infrastructure?

a) terraform show

b) terraform plan

c) terraform apply

d) terraform preview

2. What is the purpose of the `terraform.tfstate` file?

a) To store Terraform configuration

b) To lock the Terraform working directory

c) To store the current state of your infrastructure

d) To define provider configurations

3. How can you reuse Terraform configuration across multiple projects?

a) By copying and pasting code

b) By using modules

c) By using workspaces

d) By using multiple state files

4. What is the purpose of the `terraform init` command?

a) To apply changes to your infrastructure

b) To initialize a new Terraform working directory

c) To create a new Terraform configuration

d) To validate Terraform syntax

5. Which of the following is NOT a valid Terraform data type?

a) string

b) number

c) boolean

d) array

6. What is the purpose of Terraform workspaces?

a) To organize resources into logical groups

b) To manage multiple environments with the same configuration

c) To define reusable Terraform configurations

d) To store sensitive information securely

7. How can you ensure that Terraform resources are created in a specific order?

a) By using the `depends_on` attribute

b) By ordering resources in the configuration file

c) By using the `create_before_destroy` lifecycle rule

d) Terraform automatically determines the correct order

8. What is the purpose of the `terraform refresh` command?

a) To update the Terraform configuration

b) To update the state file to match the real-world infrastructure

c) To download the latest provider plugins

d) To recreate all resources in the configuration

9. Which of the following is a best practice for managing Terraform state in a team environment?

a) Using local state files and sharing them via email

b) Storing state files in version control

c) Using a remote backend like S3 with state locking

d) Recreating the state file before each apply

10. What is the purpose of the `terraform import` command?

a) To import modules from other Terraform configurations

b) To bring existing infrastructure under Terraform management

c) To import variables from external files

d) To import provider configurations

Answers:

1. b) terraform plan

2. c) To store the current state of your infrastructure

3. b) By using modules

4. b) To initialize a new Terraform working directory

5. d) array (Terraform uses list instead)

6. b) To manage multiple environments with the same configuration

7. a) By using the `depends_on` attribute

8. b) To update the state file to match the real-world infrastructure

9. c) Using a remote backend like S3 with state locking

10. b) To bring existing infrastructure under Terraform management

- This content covers the final review, industry applications, and exam preparation for the Terraform course, following the structure of the provided reference content.
- It includes industry insights, a real-world scenario, a comprehensive hands-on activity, a diagram, and mock exam questions.
- The hands-on activity and diagram are provided as separate artifacts for clarity and reusability.