Of course! Here are some real-world Terraform examples with code snippets for you to practice:

1. \*\*Provisioning an AWS EC2 Instance\*\*:

- This example demonstrates how to use Terraform to provision a simple AWS EC2 instance.

```hcl

provider "aws" {

region = "us-east-1"

}

resource "aws\_instance" "example" {

ami = "ami-0c55b159cbfafe1f0"

instance\_type = "t2.micro"

}

```

2. \*\*Creating a VPC with Subnets and Security Groups\*\*:

- This example creates a Virtual Private Cloud (VPC) with public and private subnets, and associated security groups.

```hcl

provider "aws" {

region = "us-west-2"

}

resource "aws\_vpc" "example" {

cidr\_block = "10.0.0.0/16"

}

resource "aws\_subnet" "public" {

vpc\_id = aws\_vpc.example.id

cidr\_block = "10.0.1.0/24"

availability\_zone = "us-west-2a"

}

resource "aws\_subnet" "private" {

vpc\_id = aws\_vpc.example.id

cidr\_block = "10.0.2.0/24"

availability\_zone = "us-west-2b"

}

resource "aws\_security\_group" "example" {

name = "example-security-group"

description = "Example security group"

ingress {

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

}

```

3. \*\*Deploying an AWS Lambda Function\*\*:

- This example demonstrates deploying an AWS Lambda function using Terraform.

```hcl

provider "aws" {

region = "us-west-2"

}

resource "aws\_lambda\_function" "example" {

filename = "lambda\_function\_payload.zip"

function\_name = "example\_lambda\_function"

role = aws\_iam\_role.example.arn

handler = "exports.handler"

runtime = "nodejs14.x"

}

resource "aws\_iam\_role" "example" {

name = "example\_lambda\_role"

assume\_role\_policy = <<EOF

{

"Version": "2012-10-17",

"Statement": [

{

"Action": "sts:AssumeRole",

"Principal": {

"Service": "lambda.amazonaws.com"

},

"Effect": "Allow",

"Sid": ""

}

]

}

EOF

}

```

4. \*\*Creating an Azure Virtual Machine\*\*:

- This example demonstrates provisioning an Azure virtual machine using Terraform.

```hcl

provider "azurerm" {

features {}

}

resource "azurerm\_resource\_group" "example" {

name = "example-resources"

location = "East US"

}

resource "azurerm\_virtual\_network" "example" {

name = "example-network"

address\_space = ["10.0.0.0/16"]

location = azurerm\_resource\_group.example.location

resource\_group\_name = azurerm\_resource\_group.example.name

}

resource "azurerm\_subnet" "example" {

name = "example-subnet"

virtual\_network\_name = azurerm\_virtual\_network.example.name

address\_prefixes = ["10.0.1.0/24"]

resource\_group\_name = azurerm\_resource\_group.example.name

}

resource "azurerm\_network\_interface" "example" {

name = "example-nic"

location = azurerm\_resource\_group.example.location

resource\_group\_name = azurerm\_resource\_group.example.name

ip\_configuration {

name = "internal"

subnet\_id = azurerm\_subnet.example.id

private\_ip\_address\_allocation = "Dynamic"

}

}

resource "azurerm\_linux\_virtual\_machine" "example" {

name = "example-vm"

resource\_group\_name = azurerm\_resource\_group.example.name

location = azurerm\_resource\_group.example.location

size = "Standard\_DS1\_v2"

admin\_username = "adminuser"

network\_interface\_ids = [azurerm\_network\_interface.example.id]

admin\_ssh\_key {

username = "adminuser"

public\_key = file("~/.ssh/id\_rsa.pub")

}

os\_disk {

caching = "ReadWrite"

storage\_account\_type = "Premium\_LRS"

}

source\_image\_reference {

publisher = "Canonical"

offer = "UbuntuServer"

sku = "16.04-LTS"

version = "latest"

}

}

```

These examples cover some common use cases of Terraform, including provisioning cloud resources on AWS and Azure. You can try deploying these configurations in your own environment to get hands-on experience with Terraform. Remember to replace placeholder values (like AMI IDs, resource names, etc.) with actual values applicable to your environment.

Certainly! Here are some of the top Terraform interview questions, including scenario-based questions:

\*\*1. Terraform Basics:\*\*

- \*\*Question:\*\* What is Terraform, and how does it differ from other infrastructure as code (IaC) tools?

- \*\*Scenario:\*\* Explain how Terraform compares to tools like Ansible, Chef, and Puppet in terms of declarative vs. imperative configuration management.

- \*\*Question:\*\* How does Terraform manage state, and why is it important?

- \*\*Scenario:\*\* Describe a scenario where state management becomes critical in a Terraform project, such as managing multiple environments or collaborating with team members.

\*\*2. Terraform Configuration:\*\*

- \*\*Question:\*\* What are providers and resources in Terraform?

- \*\*Scenario:\*\* You need to provision an AWS EC2 instance using Terraform. Explain how you would define the provider and resource blocks in your Terraform configuration.

- \*\*Question:\*\* How do you use variables and outputs in Terraform?

- \*\*Scenario:\*\* You want to parameterize your Terraform configurations for reusability. Describe how you would define input variables and use output values in your Terraform project.

\*\*3. Infrastructure Provisioning:\*\*

- \*\*Question:\*\* How do you handle dependencies between resources in Terraform?

- \*\*Scenario:\*\* You need to provision a VPC with multiple subnets, security groups, and an EC2 instance. Explain how you would manage resource dependencies to ensure the correct creation order.

- \*\*Question:\*\* What is the difference between `terraform apply` and `terraform plan`?

- \*\*Scenario:\*\* You're deploying infrastructure changes to a production environment. Describe the steps you would take to preview changes using `terraform plan` and apply them using `terraform apply` safely.

\*\*4. Terraform State:\*\*

- \*\*Question:\*\* How does Terraform manage state locking?

- \*\*Scenario:\*\* Multiple team members are working on the same Terraform project simultaneously. Explain how Terraform's state locking mechanism prevents concurrent modifications and ensures data integrity.

- \*\*Question:\*\* What are the benefits of using remote state storage in Terraform?

- \*\*Scenario:\*\* You're working on a Terraform project with a distributed team. Describe how storing the Terraform state remotely in a shared backend (e.g., AWS S3, Terraform Cloud) improves collaboration and reduces the risk of data loss.

\*\*5. Terraform Modules:\*\*

- \*\*Question:\*\* What are Terraform modules, and why are they useful?

- \*\*Scenario:\*\* You're provisioning similar infrastructure components across multiple projects. Explain how you would create reusable Terraform modules to standardize configurations and promote code reuse.

- \*\*Question:\*\* How do you manage module dependencies in Terraform?

- \*\*Scenario:\*\* Your Terraform project relies on custom modules hosted in a Git repository. Describe the process of referencing these modules in your main configuration and managing version dependencies.

\*\*6. Best Practices and Advanced Topics:\*\*

- \*\*Question:\*\* What are some best practices for organizing Terraform configurations?

- \*\*Scenario:\*\* You're starting a new Terraform project. Discuss strategies for structuring directories, naming conventions, and separating environments (e.g., development, staging, production) in your Terraform codebase.

- \*\*Question:\*\* How do you handle secrets and sensitive data in Terraform?

- \*\*Scenario:\*\* You need to store sensitive information (e.g., API tokens, passwords) securely in your Terraform configurations. Explain different approaches for managing secrets, such as using environment variables, encrypted files, or external vaults.

These questions cover a range of topics and scenarios commonly encountered in Terraform interviews. Make sure to practice answering them thoroughly and be prepared to discuss your experiences and problem-solving approach in real-world situations.