**Terraform Manage Resources**

**TerraWeek Day 3**



**Task 1:**

**Create a Terraform configuration file to define a resource of AWS EC2 instance, Azure storage account, Google Compute Engine, etc. (any one)**

provider "aws" {

region = "us-east-1"

}

resource "aws\_instance" "myec2" {

ami = "ami-0d81306eddc614a45"

instance\_type = "t2.micro"

vpc\_security\_group\_ids = [aws\_security\_group.ownsg.id]

key\_name = "tf-key-pair"

tags = {

Name = "terraform-example"

}

}

resource "aws\_security\_group" "ownsg" {

name = "own-sg"

ingress {

    from\_port   = 80

    to\_port     = 80

    protocol    = "tcp"

    cidr\_blocks = ["0.0.0.0/0"]

    }

ingress {

from\_port = 22

to\_port = 22

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"]

}

}

resource "aws\_key\_pair" "tf-key-pair" {

key\_name = "tf-key-pair"

public\_key = tls\_private\_key.rsa.public\_key\_openssh

}

resource "tls\_private\_key" "rsa" {

algorithm = "RSA"

rsa\_bits = 4096

}

resource "local\_file" "tf-key" {

content = tls\_private\_key.rsa.private\_key\_pem

filename = "tf-key-pair"

}

This Terraform configuration provisions an EC2 instance in the AWS region "us-east-1". Here's a summary of what each section does:

1. **Provider Block:**

* Sets the provider as AWS and specifies the region as "us-east-1".

1. **Resource Block (aws\_instance):**

* Creates an EC2 instance resource.
* Uses the specified AMI ID to launch the instance.
* Sets the instance type as t2.micro.
* Associates the instance with a security group defined by aws\_security\_group.ownsg.id.
* Specifies the key pair name for SSH access.
* Adds a tag to the instance to identify it.

1. **Resource Block (aws\_security\_group):**

* Creates a security group resource and sets the name of the security group.
* Defines ingress rules to allow inbound traffic on a specific port 80 from any IP
* Defines an additional ingress rule to allow SSH traffic on port 22 from any IP.
* Defines an egress rule to allow all outbound traffic from the instance.

1. **Resource Block (aws\_key\_pair):**

* Creates an AWS key pair resource and sets the key pair name.
* Associates the public key from the tls\_private\_key.rsa resource with the key pair.

1. **Resource Block (tls\_private\_key):**

* Generates a TLS private key resource and uses the RSA algorithm.
* Sets the key length to 4096 bits.

1. Creates a local file resource.

* Writes the private key in PEM format from the tls\_private\_key.rsa resource to a file named "tf-key-pair".

In summary, this configuration sets up an EC2 instance with a security group that allows inbound traffic on a specific port and SSH traffic, creates a key pair for SSH access, and generates a local file containing the private key.

**Task 2:**

**Check state files before running plan and apply commands & Use validate command to validate your tf file for errors and provide the Output generated by each command.**

To use the above configuration, save it in a file with a .tf extension (e.g., main.tf), and then run the following Terraform commands in the same directory:

terraform init

terraform plan

terraform apply

Terraform will download the necessary plugins and then create the EC2 instance based on the defined configuration. You can customize the configuration according to your specific requirements.

The state files contain information about the current state of your infrastructure, and Terraform uses this information to determine what changes need to be made. If the state files are not up-to-date, Terraform may make incorrect changes to your infrastructure.

**Check State Files:** To check the state files, you can use the **terraform state list** command. It lists all the resources managed by Terraform and their current state.

Command: terraform state list

Output:

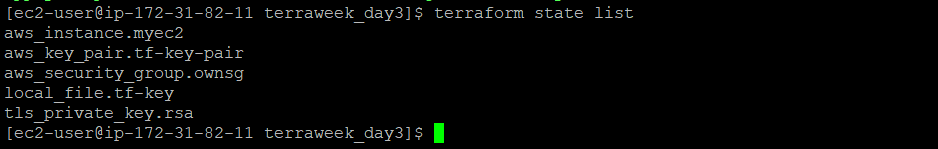
aws\_instance.myec2

aws\_key\_pair.tf-key-pair

aws\_security\_group.ownsg

local\_file.tf-key

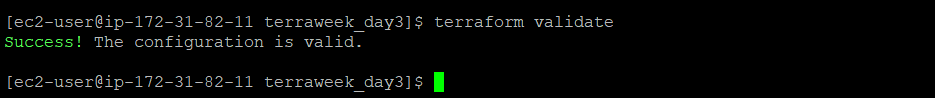
tls\_private\_key.rsa



**Validate Configuration File:** To validate the configuration file for errors, you can use the **terraform validate** command. It checks the syntax and structure of the Terraform files and reports any errors or warnings.

Command: terraform validate

Output: Success! The configuration is valid.



These commands help you check the state files, validate the configuration, and get insights into the changes that Terraform will make to your infrastructure before actually applying them.

**Task 3:**

**Add a provisioner to the configuration file to configure the resource after it is created and use Terraform commands to apply for changes and destroy to remove resources.**

To configure a resource after it is created, you can use provisioners in Terraform. Provisioners allow you to run scripts or execute commands on the resource during creation or destruction. Here's an example of adding a provisioner to an AWS EC2 instance resource:

provider "aws" {

region = "us-east-1"

}

resource "aws\_instance" "myec2" {

ami = "ami-04a0ae173da5807d3"

instance\_type = "t2.micro"

vpc\_security\_group\_ids = [aws\_security\_group.ownsg.id]

key\_name = "tf-key-pair"

tags = {

Name = "terraform-example"

}

}

resource "aws\_security\_group" "ownsg" {

name = "own-sg"

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"]

}

}

resource "aws\_key\_pair" "tf-key-pair" {

key\_name = "tf-key-pair"

public\_key = tls\_private\_key.rsa.public\_key\_openssh

}

resource "tls\_private\_key" "rsa" {

algorithm = "RSA"

rsa\_bits = 4096

}

resource "local\_file" "tf-key" {

content = tls\_private\_key.rsa.private\_key\_pem

filename = "tf-key-pair"

}

**In this example, the** aws\_instance **resource is configured to use the remote-exec provisioner. The** inline **argument to the** remote-exec **provisioner specifies the command that will be run on the remote resource. In this case, the command is** echo 'Hello, world!' > /home/ec2-user/terraweek\_day3/hello.txt**.**

To apply changes and provision the resources, you can use the following command:

terraform apply

This command will create the EC2 instance and execute the provisioner, which in this case will print the message "Resource created. Configuring...".

To destroy the resources created by the Terraform configuration, you can use the following command:

terraform destroy

This command will remove the EC2 instance and any associated resources.

**Task 4:**

**Add lifecycle management configurations to the configuration file to control the creation, modification, and deletion of the resource and use Terraform commands to apply the changes.**

To control the lifecycle management of resources in Terraform, you can use lifecycle blocks. These blocks allow you to define specific behavior for resource creation, modification, and deletion. Here's an example of adding lifecycle management configurations to an AWS EC2 instance resource:

provider "aws" {

region = "us-east-1"

}

resource "aws\_instance" "myec2" {

ami = "ami-04a0ae173da5807d3"

instance\_type = "t2.micro"

vpc\_security\_group\_ids = [aws\_security\_group.ownsg.id]

key\_name = "tf-key-pair"

tags = {

Name = "terraform-example"

}

lifecycle {

create\_before\_destroy = true

prevent\_destroy = false

}

}

resource "aws\_security\_group" "ownsg" {

name = "own-sg"

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"]

}

}

resource "aws\_key\_pair" "tf-key-pair" {

key\_name = "tf-key-pair"

public\_key = tls\_private\_key.rsa.public\_key\_openssh

}

resource "tls\_private\_key" "rsa" {

algorithm = "RSA"

rsa\_bits = 4096

}

resource "local\_file" "tf-key" {

content = tls\_private\_key.rsa.private\_key\_pem

filename = "tf-key-pair"

}

provisioner "remote-exec" {

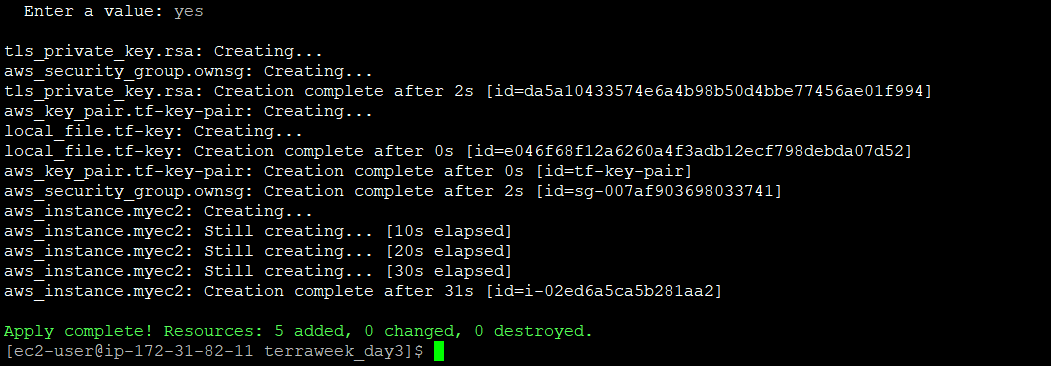
inline = ["echo 'Hello, world!' > /home/ubuntu/hello.txt"]

}

In the above example, a lifecycle block is added to the AWS EC2 instance resource. The **create\_before\_destroy** parameter is set to **true**, which means that Terraform will create a new instance before destroying the old one during updates. The **prevent\_destroy** parameter is set to **false**, which allows the instance to be destroyed using the **terraform destroy** command.

To apply changes and create or update the resource with the lifecycle management configurations, you can use the following command:

1. terraform apply



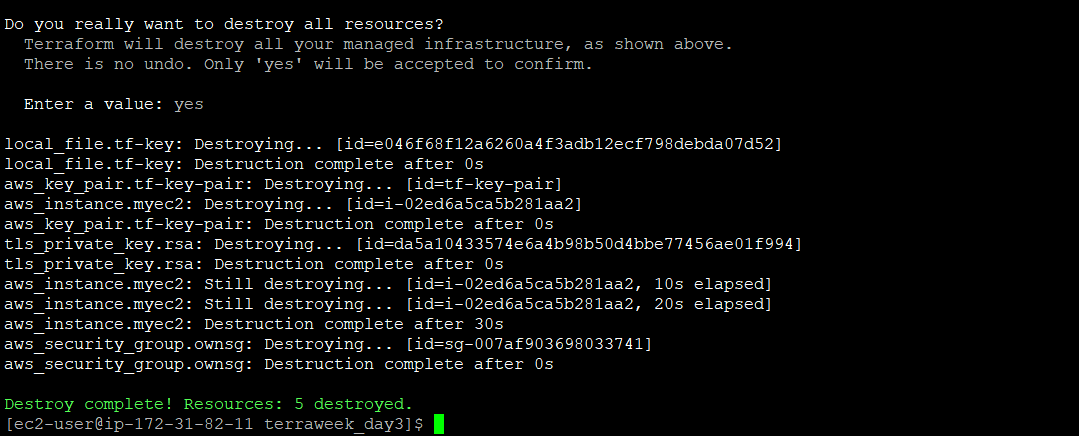
This command will create or update the EC2 instance according to the configuration and the specified lifecycle management behaviour.

Please note that lifecycle management configurations should be used carefully, as they can impact the behaviour of resource updates and destruction. It's important to understand the implications of these configurations and test them thoroughly before applying them to production environments.

To destroy the resources created by the Terraform configuration, you can use the following command:

terraform destroy

This command will remove the EC2 instance and any associated resources.



Happy Learning🎉🚀

**TerraWeek Day 3 – Terraform Manage Resources**🚀

Today's tasks involved creating an AWS EC2 instance, managing state files, adding provisioners, and implementing lifecycle management. #TerraWeek #InfrastructureAsCode #AWS #Terraform

Task 1: Let's get started by creating a Terraform configuration file for your dream infrastructure. Define AWS EC2 instances, Azure storage accounts, or Google Compute Engine resources—take your pick! #Terraform #InfrastructureAsCode

Task 2: But before you hit the ground running, it's crucial to check your state files and validate your tf file for errors. Run these commands:

* terraform state list: Get a comprehensive list of managed resources and their current state.
* terraform validate: Ensure your configuration file is error-free. 🚀

Task 3: Want to take your configuration to the next level? Add a provisioner to customize your resources after creation. Then, use Terraform's apply command for changes and destroy command to remove resources. #InfrastructureAutomation #Provisioning

Task 4: Take control of your resource lifecycle with lifecycle management configurations. Use Terraform commands to apply changes and witness the magic of creation, modification, and deletion. #ResourceManagement #TerraformMagic

Remember, every step brings you closer to mastering Terraform's immense capabilities. Stay curious, keep learning, and embrace the power of Infrastructure as Code! 💪💼

#TerraWeek #Terraform #InfrastructureAsCode #Automation #DevOps