**Terraform Modules**

**TerraWeek Day 5**

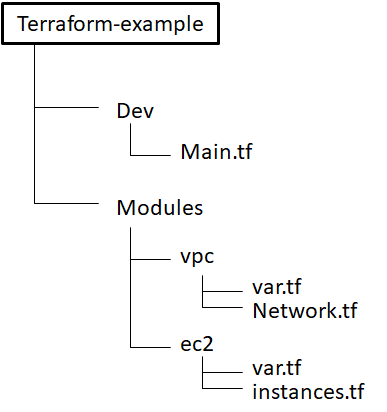


**Task 1:**

* **What are modules in Terraform and why do we need modules in Terraform?**

In Terraform, modules are like building blocks or templates that allow us to encapsulate and reuse parts of our infrastructure code. They help organize and structure our configurations, making them more modular and manageable.

Imagine you're building a house. Instead of starting from scratch every time and manually constructing every room, you can use pre-designed modules like a kitchen module, bedroom module, and bathroom module. These modules contain all the necessary details and configurations for their respective components. You can simply reuse them in different projects or environments, saving time and effort.



We need modules in Terraform for a few key reasons. Firstly, modules promote code reuse. Instead of writing the same configuration code over and over again, we can create a module for a specific infrastructure component and reuse it across multiple projects. This saves time and effort, and ensures consistency in our infrastructure deployments.

Secondly, modules enhance code organization and maintainability. By breaking down our infrastructure into smaller, self-contained modules, we can easily manage and update specific parts of our infrastructure without affecting the rest. This modular approach makes it easier to understand and maintain our configurations, especially as our infrastructure grows in complexity.

* What are the benefits of using modules in Terraform?

Benefits of using modules in Terraform:

1. **Reusability:** Modules promote code reuse by encapsulating infrastructure components, allowing them to be used across different projects or environments.
2. **Abstraction:** Modules provide a simplified interface to configure and manage complex infrastructure components, making configurations more manageable and understandable.
3. **Encapsulation:** Modules encapsulate configuration details and dependencies, improving code organization and making it easier to maintain and update infrastructure code.
4. **Collaboration:** Modules facilitate collaboration among team members by defining clear boundaries and interfaces, enabling parallel development and easier integration of different components.
5. **Versioning and Testing:** Modules can be versioned and tested independently, ensuring controlled changes and updates to infrastructure components while maintaining stability.
6. **Readability and Maintainability:** Modules improve code readability and maintainability by providing a structured and modular approach to infrastructure configuration, making it easier to understand and troubleshoot.
7. **Consistency and Standardization:** Modules enable the creation of standardized infrastructure patterns, ensuring consistent deployment practices and reducing configuration errors.

**Task 2:**

* **Create/Define a module in Terraform to encapsulate reusable infrastructure configuration in a modular and scalable manner. For e.g. EC2 instance in AWS.**

Step1 : Create a new directory named "terraweek\_day5" and navigate into it.

mkdir terraweek\_day5

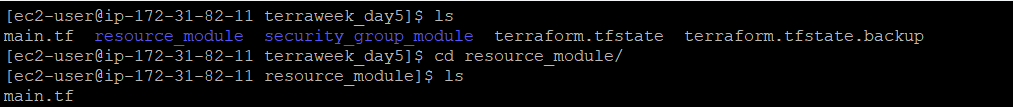
cd terraweek\_day5

Step2 : Create a new directory named "resource\_module" and navigate into it. Create a main.tf file within this directory.

mkdir resource\_module

cd resource\_module

touch main.tf



Step3 : Open the main.tf file within the resource\_module directory and add the following code to define the EC2 instance resource:

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]

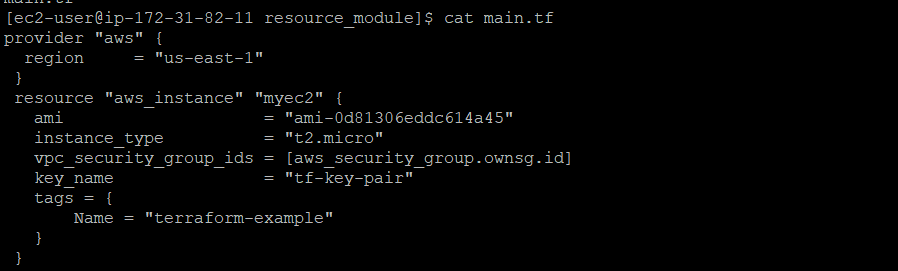
key\_name = "tf-key-pair"

tags = {

Name = "terraform-example"

}

}



Step4 : Navigate back to the parent directory (terrawee\_day5).

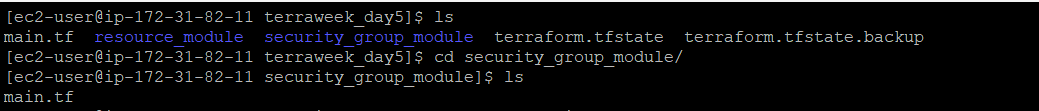
cd ..

Step 5: Create a new directory named "security\_group\_module" and navigate into it. Create a main.tf file within this directory.

mkdir security\_group\_module

cd security\_group\_module

touch main.tf



Step 6: Open the main.tf file within the security\_group\_module directory and add the following code to define the security group resource:

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

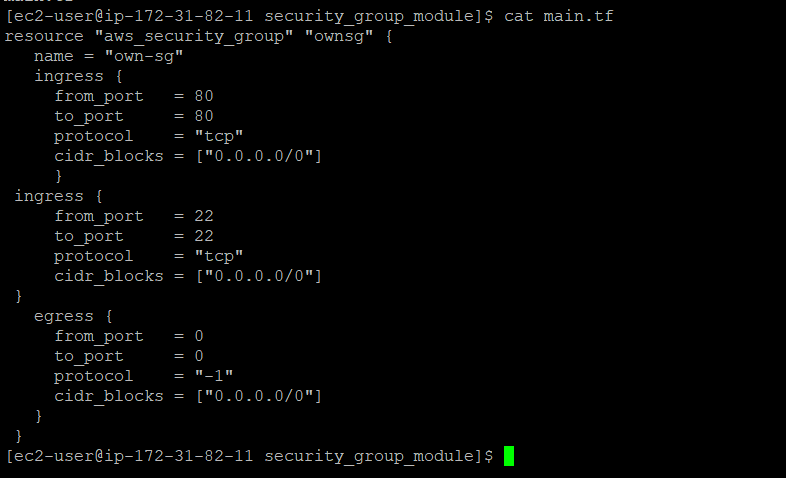
to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

}



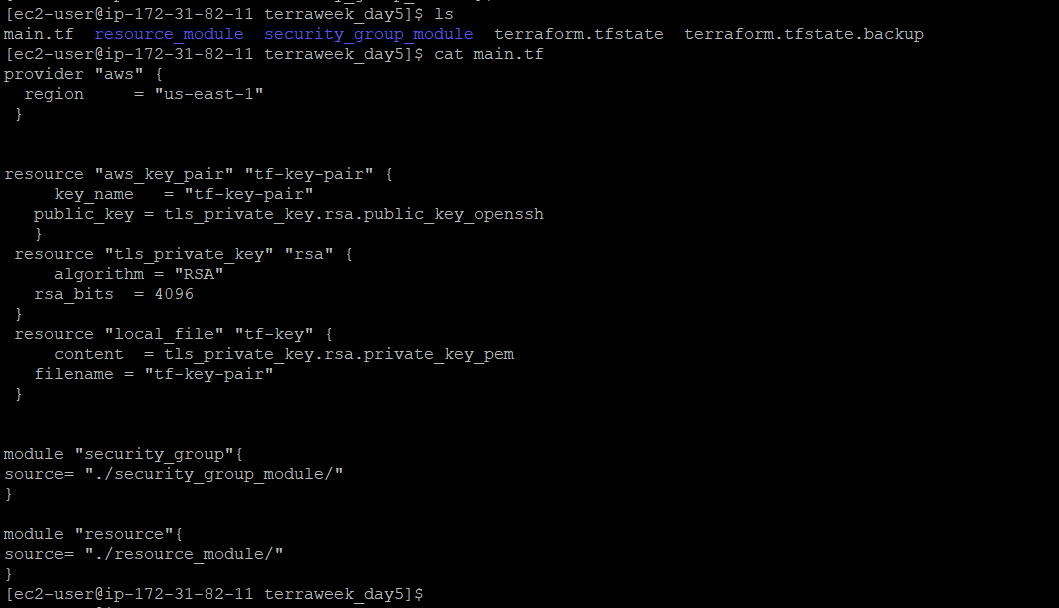
Step7 : Navigate back to the parent directory (terrawee\_day5).

cd ..

Step 8: Create a new main.tf file in the terraweek\_day5 directory to use the modules.

touch main.tf

Step 9: Open the main.tf file and add the following code to use the modules:



By following these steps, you have created a Terraform module for the EC2 instance (resource\_module) and another module for the security group (security\_group\_module). The main.tf file in the terraweek\_day5 directory references these modules

**Step10 :** Run the terraform init command to initialize Terraform and configure the backend to use AWS S3

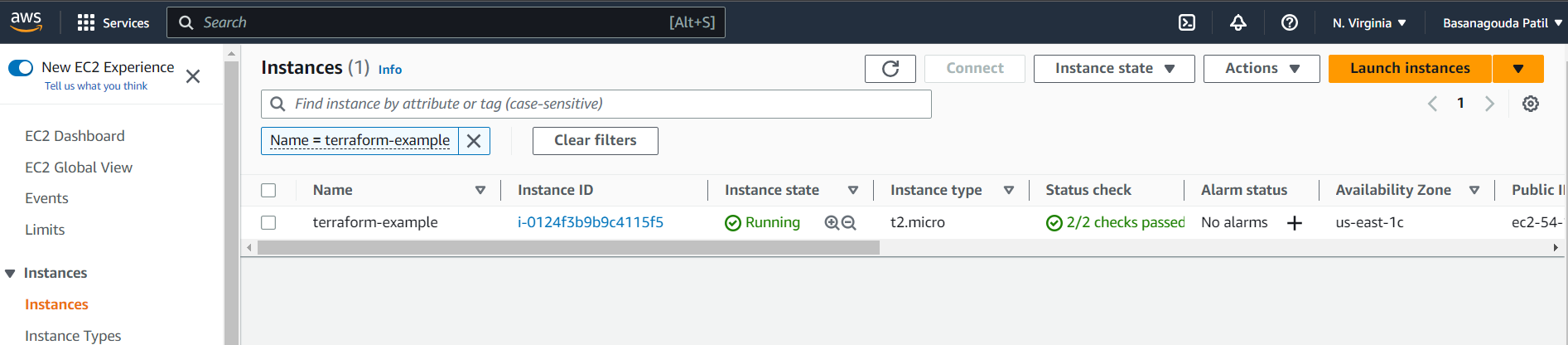
terraform init

**Step11 :** Run Terraform commands such as terraform plan and terraform a0pply, the state will be stored in AWS S3, enabling remote state management and collaboration.

terraform plan

terraform apply

**Step 12:** Then finally, check the ec2 in AWS management console.



**Task 3:**

* **Modular composition** :

Modular composition in Terraform refers to the practice of breaking down infrastructure code into smaller, reusable modules. Modules are self-contained units that represent specific components or resources, such as a virtual machine or a database. They allow you to organize and structure your code in a more modular and manageable way.

By using modules, you can create reusable building blocks for your infrastructure. Each module can have its own inputs and outputs, allowing you to customize its behavior and easily integrate it into different projects. This promotes code reuse, reduces duplication, and makes your codebase more maintainable.

* **Module versioning**.

Module versioning is the process of assigning versions to your modules. Versioning helps you track and manage changes to your modules over time. It allows you to control the updates and ensure that deployments are consistent and predictable. With version constraints, you can specify which module versions are acceptable for your infrastructure. This helps prevent unexpected changes and ensures that your deployments remain stable.

By using module versioning, you can lock the version of a module that you want to use in your infrastructure. This ensures that even if the module gets updated in the future, your deployments will still use the specific version you have locked. It provides stability and allows you to manage updates in a controlled manner, ensuring that your infrastructure remains reliable and predictable.

Conclusion:

In this blog, we discussed the concept of modules in Terraform and their benefits. We explored how to create a module for an EC2 instance in AWS by following step-by-step instructions. We also touched upon the importance of modular composition and how it helps in breaking down infrastructure code into smaller, reusable components.

By leveraging Terraform modules, you can efficiently build and manage your infrastructure, promote code reuse, and maintain consistency across projects. With the ability to lock module versions, you have control over updates and can confidently manage changes to your infrastructure configurations.

Happy Terraforming! And Happy Learning 😊

🚀 Excited to share my latest blog post on Terraform Modules! 🌱

In TerraWeek Day 5, we dive into the power of modules in Terraform and why they are essential for managing infrastructure. Modules act as building blocks, enabling us to encapsulate and reuse parts of our code. Just like assembling a house, modules provide pre-designed templates for components like EC2 instances and security groups, saving time and effort.

Check out the step-by-step guide on creating a module for an EC2 instance in AWS. Save time and effort by encapsulating and reusing infrastructure components. #AWS #EC2 #InfrastructureManagement

Read the full blog to supercharge your Terraform skills and optimize your infrastructure deployments. Happy Terraforming! 🌟🚀 #InfrastructureAutomation #DevCommunity

Join the TerraWeek challenge led by [Shubham Londhe](https://www.linkedin.com/in/ACoAABhZ4kMBt55axHJpEnVRp0UOUl-_JwwmPwk) and unlock the power of Terraform. Don't miss this opportunity to enhance your skills and boost your infrastructure management game. See you there! 👩‍💻👨‍💻

Remember to like 👍, comment 💬, and share this post to spread the knowledge! Let's empower each other in the world of #DevOps and #CloudComputing. Happy Learning! 😊🎓

#TerraWeek #InfrastructureAsCode #DevOpsJourney #CloudNative #TechCommunity