Beginner-Level Challenge: Single Cloud Infrastructure Deployment

Objective: Set up a basic infrastructure using Terraform on a single cloud provider (e.g., AWS, Azure, or GCP).

Project Details

1. Task:

- Deploy a simple web server* (e.g., Nginx or Apache) using Terraform on a cloud provider of choice.
- The infrastructure will include:
  - A **single virtual machine** (EC2 in AWS, VM in Azure/GCP).
  - A **security group** or **firewall rule** allowing HTTP (port 80) traffic.
  - Ensure that the machine can be accessed via a **public IP address**.

2. Steps:

- Create the Terraform configuration file defining:
  - The cloud provider.
  - VM resource (size, image, etc.).
  - Security group/firewall rules.
- Initialize the project with `terraform init`.
- Use `terraform plan` to ensure the configuration is correct.
- Deploy the infrastructure using `terraform apply`.
- Verify that the web server is accessible from the browser.

3. Skills Acquired :

- Basic **Terraform syntax** and **resource provisioning**.
- Use of **security groups** and managing **network traffic**.
- Introduction to the **terraform init, plan, apply** workflow.

Mid-Level Challenge: Multi-Tier Web Application with Autoscaling and Load Balancing

Objective :

Build a more complex infrastructure using Terraform with a multi-tier architecture that includes a load balancer , auto-scaling group , and multiple web servers .

Project Details :

1. Task:

- Deploy a multi-tier web application with:

  - An Elastic Load Balancer (ELB) to distribute traffic.

  - Auto-Scaling Group to manage the number of web servers based on traffic load.

  - A VPC with subnets, internet gateways , and appropriate routing.

  - S3 bucket to store website content or backups.

2. Steps:

- Create a Terraform configuration that defines:

  - A **VPC** and subnets.

  - An **auto-scaling group** with desired capacity settings.

  - An **Elastic Load Balancer (ELB)**.

  - **Launch configuration** to specify the web server setup (using user data scripts).

- Use `terraform init`, `terraform plan`, and `terraform apply` to deploy the infrastructure.

- Ensure traffic flows through the load balancer to the web servers and scales up/down based on load.

3. Skills Acquired:

- Understanding of **auto-scaling**, **load balancing**, and managing **network infrastructure**.

- **VPC creation**, subnets, and **internet gateway** setup.

- Managing multi-tier architectures using Terraform’s **modular approach**.

Advanced-Level Challenge: Multi-Cloud Infrastructure with CI/CD Integration

Objective:

Design a multi-cloud infrastructure using Terraform across multiple cloud providers (AWS, Azure, GCP) and integrate it with a CI/CD pipeline for automated deployment.

Project Details :

1. Task:

- Build an infrastructure where:

  - Web servers are hosted across **multiple cloud providers** (e.g., AWS EC2, Azure VM, and GCP Compute Engine).

- A **DNS service** (e.g., AWS Route 53 or Azure DNS) is configured to balance traffic across all cloud environments.

- A **CI/CD pipeline** (e.g., Jenkins, GitLab CI) automatically triggers Terraform scripts upon code changes (using `terraform apply` in the pipeline).

- Implement **state management** using a **remote backend** (e.g., AWS S3 with DynamoDB for state locking).

2. Steps:

- Create Terraform configuration for each cloud provider (AWS, Azure, GCP) to provision similar resources (e.g., VMs, security groups).

- Integrate DNS to distribute traffic across providers.

- Implement **remote state management** to ensure the infrastructure is managed from a single source of truth.

- Use a CI/CD tool to automate the deployment process:

- Whenever a change is pushed to the code repository, the CI/CD pipeline should trigger the Terraform plan and apply stages.

3. Skills Acquired:

- Advanced multi-cloud provisioning and managing **cross-cloud dependencies**.

- Setting up a **CI/CD pipeline** for Terraform-based infrastructure.

- Managing **Terraform remote state** and **locking mechanisms**.

Suggested Improvements for Day 1 Content:

1. Detailed Examples in Labs :

- Adding sample Terraform configuration files for learners to reference would be helpful.

This gives beginners a starting point and reduces confusion.

2. More Hands-on Labs:

- You could break down the hands-on lab into several mini-tasks.

For example, Day 1 could involve deploying only the virtual machine, and future lessons could introduce security groups, networking, and auto-scaling.

3. Real-World Problem Solving:

- After teaching Terraform’s plan and apply workflow, it would be valuable to include scenarios where students need to modify their infrastructure and handle state changes.

4. Extra Challenges for Each Level:

- After completing each challenge project, add additional optional tasks. For example:

  - **Beginner**: Add SSH access to the instance using Terraform provisioners.

  - **Mid-Level**: Integrate a basic **monitoring solution** (e.g., CloudWatch) into the project.

  - **Advanced**: Implement **security best practices** such as encryption of data in transit and at rest.

5. Industry Best Practices :

- Include a section on **best practices** for using Terraform, such as:

  - Avoiding hard-coded secrets (use of Vault or environment variables).

  - Structuring projects for scalability (using modules).

  - Testing Terraform configurations with **terraform validate** or **terratest**.

Conclusion:

- This set of challenge projects introduces practical, hands-on experience for learners at every skill level, gradually guiding them from basic to complex multi-cloud infrastructure setups.
- By completing these, students will gain the confidence to apply Terraform in real-world scenarios. The suggestions for improvements will further enhance Day 1 content by reinforcing concepts and building a strong foundation for more advanced topics in the following days.
- Would you like me to incorporate these into your course, or would you prefer more detailed walkthroughs of the challenge projects?