Course 2: Introduction & IaC Fundamentals (Hands-On with Terraform)

1. Introduction

In Course 1, you learned the **concepts of IaC, Terraform, and GitOps**.
Now, it's time to get hands-on. By the end of this module, you will:

- Deploy a **VPC and subnet** on your cloud provider using Terraform.
- Use a **remote backend** for Terraform state.
- Test your setup **locally** with reproducible and disposable infrastructure.
- Learn to import, destroy, and redeploy your infrastructure with just a few commands.

⚠️ **Important**: Infrastructure must be managed responsibly. Always handle credentials with care. Each **region** has its own **costs (💸)**, **latency (⚡)**, and **environmental impact (🌍)**. Choose your deployment region carefully, balancing price, performance, and sustainability.

---

2. Prerequisites

Each group (4 students) must ensure the following before starting:

- **Local tools installed**:
    - Terraform CLI
    - Git CLI
    - Cloud provider CLI (AWS, GCP, or Azure)
- **Cloud project ready**:
    - One project per group (for **dev** environment).
    - Example: `student-team1-dev`.
    - Credentials configured locally via the provider CLI:

```
aws configure

gcloud auth login

az login
```

    - ⚠️ **Never hardcode access keys or service account JSONs** in Terraform or GitHub.
        - In production, you would use **Identity Federation** with short-lived credentials.
        - 👉 **For simplicity, we will not use identity federation in this course.**
- **GitHub repository**: one per group, containing Terraform code.
- **Team organization**: groups are **autonomous** on how to split the work (providers, variables, backend setup, etc.).
- Feel free to ask if your group needs assistance but **RTFM** ! Hashicorp Terraform and Cloud providers documentation are well written and contain all necessary information to achieve this exercise.

3. Terraform Basics Refresher

Providers

```
provider "aws" {
  region = var.region
}
```

Variables

`variables.tf`:

```
variable "project_id" {
  type        = string
  description = "Cloud project ID"
}

variable "region" {
  type        = string
  description = "Region for resources"
}
```

```
variable "vpc_name" {
  type        = string
  description = "Name of the VPC"
}

variable "cidr_block" {
  type        = string
  description = "CIDR block for the VPC"
}
```

4. VPC & Subnet Example

In this example, we'll deploy a **VPC with one subnet**.
👉 Every resource type (like `aws_vpc`, `aws_subnet`, `google_compute_network`, etc.) is documented in the Terraform Providers Registry.
This registry is your **reference for all resource arguments and attributes**.

Example (AWS):

```
resource "aws_vpc" "main" {
  cidr_block = var.cidr_block
  tags = {
    Name = var.vpc_name
  }
}

resource "aws_subnet" "main" {
  vpc_id            = aws_vpc.main.id
  cidr_block        = "10.0.1.0/24"
  tags = {
    Name = "${var.vpc_name}-subnet"
  }
}

output "vpc_id" {
  value       = aws_vpc.main.id
  description = "The ID of the created VPC"
}
```

---

5. Terraform State Management

- **State file** tracks resources under Terraform’s control.
- **Local state** → default, but unsafe for teamwork.
- **Remote state** → shared backend (e.g., bucket) is required.

⚠️ The backend bucket **must be created before `terraform init`**.
You can:

- Create it **manually**, OR
- Use a **script** to create it automatically.
  👉 After that, you can use `terraform import` to bring the bucket into Terraform state.

Example AWS S3 backend:

```
terraform {
  backend "s3" {
    bucket = "team-terraform-state"
    key    = "global/vpc/terraform.tfstate"
    region = var.region
  }
}
```

⚠️ A backend block cannot refer to named values (like input variables, locals, or data source attributes). Later, you'll need to use one backend per environment, so use a separate file in a different folder for your backend configuration and pass it to your terraform CLI:

GCP example:

File: main.tf:

```
 terraform {

   backend "gcs" {
   }
}
```

File: backends/dev.config

```
bucket  = "tfstates-gh-demo"

prefix  = "terraform/state"
```

terraform init -backend-config="./backends/dev.config"

📌 **Notes**

- **State lock (`tfstate.lock`)**
  - Created automatically when running `terraform plan` or `terraform apply`.
  - Prevents multiple concurrent operations against the same state file.
- **`terraform import`**
  - Brings existing infra (like your backend bucket) under Terraform control.

    - Import syntax is always documented in the **Terraform Providers documentation**.

- `terraform destroy`
    - Deletes all resources defined in your code.
    - Always test in **dev** first before destroying production resources.

- **Dependencies**
    - Terraform builds a **dependency graph** automatically based on references between resources.
      Example: if a subnet uses a VPC ID, Terraform knows to create the VPC first
    - You can override or enforce explicit dependencies using the **`depends_on`** meta-argument:

None

```
          resource "aws_subnet" "main" {
            vpc_id      = aws_vpc.main.id
            cidr_block  = "10.0.1.0/24"
            depends_on  = [aws_vpc.main]
          }
```

    - Use `depends_on` only when necessary (e.g., when Terraform cannot infer the dependency automatically, such as with provisioners or outputs).

---

6. Local Setup Testing

Initialize Terraform:

```
terraform init
```

Do a terraform plan and check the result:

```
terraform plan -var-file=dev.tfvars
```

Deploy **dev** environment:

```
terraform apply -var-file=dev.tfvars
```

Check outputs:

```
vpc_id = vpc-123456
subnet_id = subnet-abcdef
```

Destroy when done:

```
terraform destroy -var-file=dev.tfvars
```

👉 The infrastructure must be **fully reproducible**:

- One command to create → `terraform apply`
- One command to delete → `terraform destroy`

---

7. Security & Best Practices

- Always use variables for **region, project_id, and CIDRs**.
- Use `sensitive = true` for secrets:

  ```
  variable "db_password" {
    type      = string
    sensitive = true
  ```

```
    }
```

- Credentials:
    - Local: authenticate with the **cloud CLI**.
    - **Never commit keys**.
- Prefer **short-lived credentials** with identity federation.
  👉 **For this course, we will not use identity federation (too advanced), but remember this is best practice in production.**

---

8. Advanced Notes

- **Null resources**:
    - Allow executing arbitrary scripts.
    - Use only as a **last resort** (e.g., unsupported features).

---

9. Key Takeaways

- Each group works in **one dev project**.
- Backend bucket must be created **manually or via script**, then imported.
- Infra must be **reproducible** (`apply`) and **disposable** (`destroy`).

- Handle credentials securely and consider **region impact on cost, latency, and sustainability**.

- Test your setup progressively (`init/plan`)

- `tfstate.lock` = prevents concurrent operations.

- **Terraform Providers Docs** = your official reference for resources and arguments.

👉 **Next Chapter**: We'll extend this with **multiple environments (dev & prd)** and **CI/CD automation**.