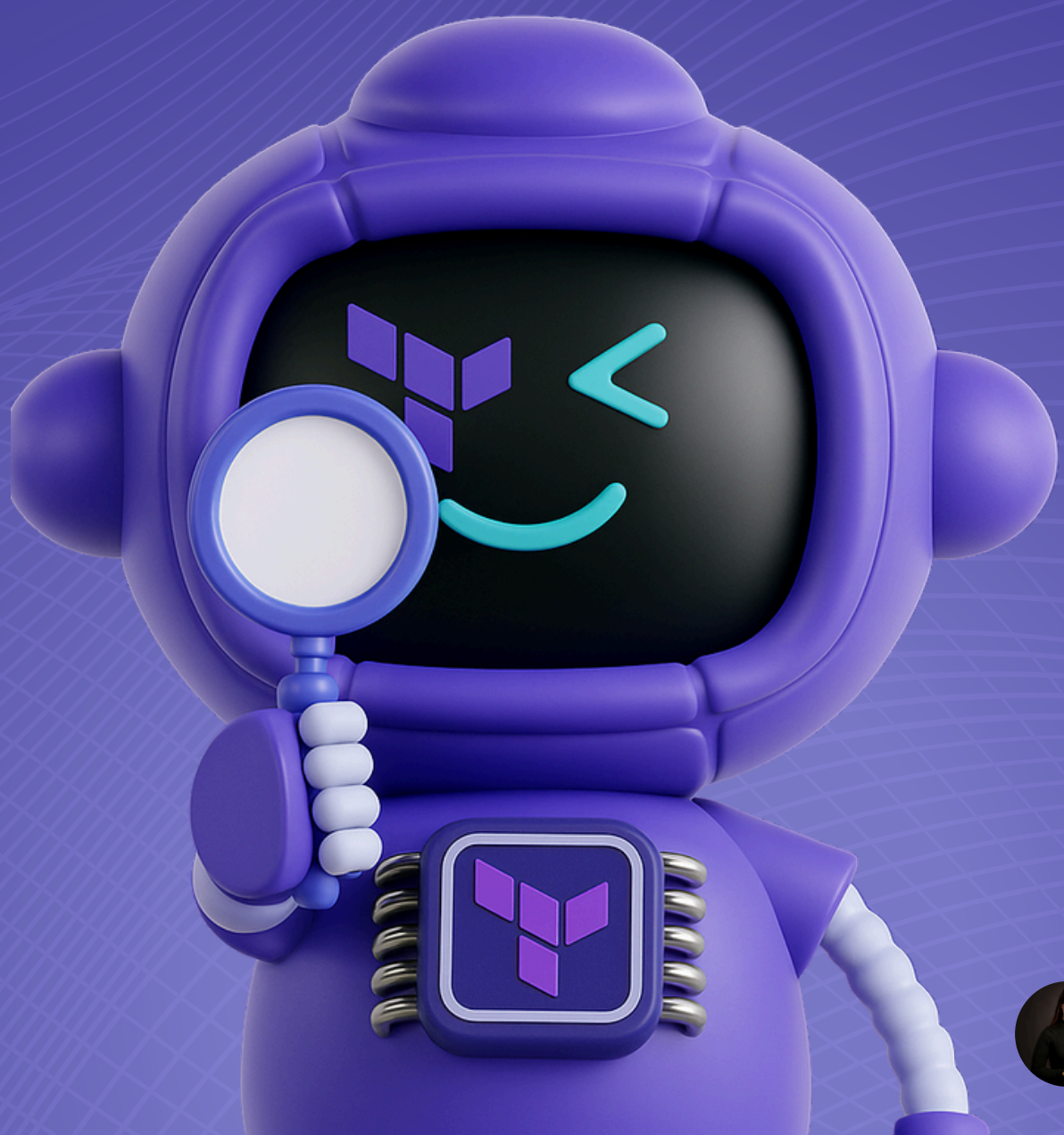
Terraform:
An Introduction to Infrastructure as Code

Manage your infrastructure with code, automation, and collaboration



Assma Fadhli
DevSecOps Engineer

Introduction

- Terraform, by **HashiCorp**, is an open-source **Infrastructure as Code (IaC)** tool.
- Lets you define and provision infra with a declarative language.
- Benefits: versioning, testing, automation.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

What is Terraform?

- **Define** and **manage** infra using **config files**.
- Works across **multi-cloud providers** (AWS, Azure, GCP).
- **Shareable**, **reusable**, **automated** approach.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Key Benefits

- Full lifecycle orchestration.
- Multi-cloud compatibility.
- Automation → fewer errors.
- Version control → tracking & collaboration.
- Modularity → reusable modules.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Terraform Workflow

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Terraform Workflow

Init	Validate	Plan	Apply	Destroy
• Used to initialized a working directory containing terraform config files. • Downloads Providers	• Validates the terraform config files in that respective directory to ensure they are Syntacitically valid and internally consistent.	• Creates an execution plan • Terraform performs a refresh and determines what actions are necessary to achieve the desired state specificd in config files.	• Used to apply the changes required to reach the desired state of the configuration • By default, apply scan the current directory for the config and applies the changes	• Used to destroy the terraform managed infrastruccture. • This will ask for confirmation before destrrying.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

HashiCorp Configuration Language (HCL)

- HCL = human-readable & machine-friendly.
- Lets you clearly **describe infra**.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Example

```hcl
resource "aws_s3_bucket" "example" {
  bucket = "my-unique-terraform-bucket-12345"
  acl    = "private"

  tags = {
    Name        = "My Terraform Bucket"
    Environment = "Dev"
  }
}
```

- Defines an S3 bucket with a unique name, **private ACL**, and **tags**.
- Declarative → you describe the **desired state**, Terraform figures out the rest.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Providers

- **Plugins** that let Terraform **manage resources** across cloud platforms.
- Example: AWS provider → manage EC2, S3, VPC.
- terraform init downloads the needed providers.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

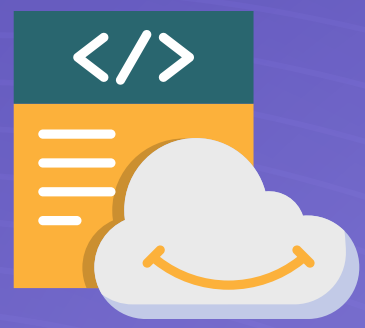
Assma Fadhli
DevSecOps Engineer

Example

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
}
```

This **specifies** the AWS provider and **configures** it for the us-east-1 region.



#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Modules

- Self-contained **packages** of Terraform configs.
- **Reusable** across projects → consistent & scalable infra.
- Sources: local path, Terraform Registry, Git, S3, etc.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Example

```
module "web_app" {
  source = "./modules/webserver"

  instance_type = "t2.micro"
  ami_id        = "ami-0abcdef1234567890"
  vpc_id        = "vpc-0123456789abcdef0"
}
```

This defines a **reusable web server setup**, improving scalability & maintainability.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

State File

- Stored in **terraform.tfstate**.
- **Maps configuration** to real resources.
- **Tracks resource attributes and metadata**.
- **Speeds up** operations on large infrastructures.
- Acts as the **source of truth** for Terraform.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Remote State and Backends

- Local state works for **individuals** but not ideal for teams or production.
- Remote backends define where Terraform **stores state**.
- Enable **collaboration without conflicts**.
- Provide better **security and durability**.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Example

```
terraform {
  backend "s3" {
    bucket         = "my-terraform-state-bucket"
    key            = "path/to/my/key/terraform.tfstate"
    region         = "us-east-1"
    encrypt        = true
    dynamodb_table = "my-terraform-locks"
  }
}
```

This configuration sets an S3 bucket for the state file, a unique key, AWS region, encryption, and a DynamoDB table for state locking to prevent concurrent changes and data corruption.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Remote backends

- **AWS S3** – Highly available, durable, integrates with AWS.
- **Azure Blob Storage** – Microsoft Azure's object storage for state files.
- **Google Cloud Storage** – GCP's object storage for state management.
- **HashiCorp Consul** – Distributed service mesh with state storage support.
- **Terraform Cloud/Enterprise** – Managed service with collaboration and policy features.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Securing the State File

Since the state file contains sensitive infrastructure data, secure it by:

- **Encryption**: Encrypt at rest and in transit.
- **Access Control:** Restrict read/write access.
- **State Locking**: Use backends with locking to prevent concurrent changes.
- **Version Control**: Keep configuration files in Git, but avoid committing the state file directly.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

CLI Commands

terraform init

Initialize a working directory with Terraform configuran-fiies and downl-oaded providers

terraform validate

Validate that the configuration is syntactically valid and internally consistent

terraform apply

Apply the planned changes to create, update, or delete Infrastructure

terraform destroy

Destroy all resources defined in the configuration (asks for confirmation)

terraform fmt

Format Terraform configuration files into a standard style

terraform output

Display outout values defined in the configuration

terraform show

Show the current state or a saved plan

terraform get

Download or update Terraform modules

terraform state

Advanced state management (list, move, pull, push, remove

terraform providers

Show information about the providers required by the configuration

terraform taint

Mark a resource for recreation during the next apply

terraform import

Import existing resources into Terraform state

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Input variables

- Act as parameters for **Terraform modules and configurations**.
- They allow you to c**ustomize behavior without changing the core code**.
- Promote **reusability of configurations**.
- Make **configurations more flexible**.

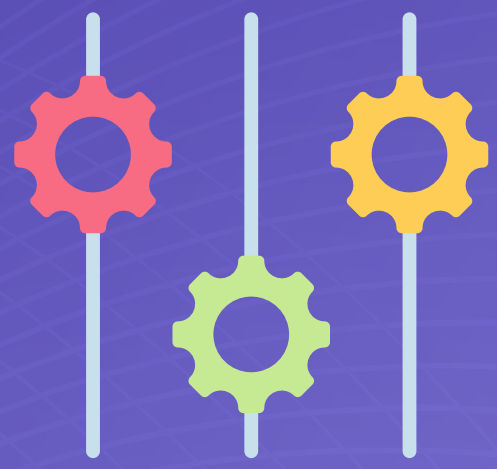
#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Example

```
variable "instance_type" {
  description = "The EC2 instance type"
  type        = string
  default     = "t2.micro"
}
```

Variables can include **default values**, **types**, and **descriptions**.



#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Output Values

- Expose **specific data** from **Terraform configurations** (e.g., IPs, DNS names, resource IDs).
- Provide information after **infrastructure provisioning**.
- Useful for **displaying important details** to users.
- Enable **data sharing** between different Terraform configurations.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Example

```
output "instance_ip" {
  description = "The public IP address of the EC2 instance"
  value       = aws_instance.web.public_ip
}
```

This Terraform code defines an output value named **instance_ip**.

- **description** provides context, explaining that it represents the public IP of the EC2 instance.
- **value** references aws_instance.web.public_ip, meaning it will display the public IP address of the web EC2 instance after Terraform provisions it.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Data Sources

- Data sources let Terraform **fetch information** about existing infrastructure or external data.

- Enable **use of retrieved data** within configurations.

- Useful for **integrating with resources** not managed by Terraform.

- Help in **retrieving dynamic or real-time information**

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Example

```
data "aws_ami" "ubuntu" {
  most_recent = true
  owners      = ["099720109477"] # Canonical

  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-*"]
  }

  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
}
```

This data source retrieves the most recent Ubuntu . AMI ID, which can then be used to launch an EC instance.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Workspaces & Loops

- Workspaces manage separate states for one configuration.
- Useful for multiple environments (dev, staging, prod).
- Each has its own state file to avoid conflicts.
- Default workspace is default.
- count / for_each = create multiple instances efficiently.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Null Resources

- Null resource = special Terraform resource type.
- It doesn't manage any real infrastructure object.
- Used as a placeholder to attach provisioners.
- Can trigger actions when other resources change.
- Helpful when no specific Terraform resource exists for that action.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Example

```
resource "null_resource" "example" {
  triggers = {
    always_run = timestamp()
  }

  provisioner "local-exec" {
    command = "echo \"Hello from null resource at $(date)\" >>
/tmp/null_resource_log.txt"
  }
}
```

In this example, the **null_resource** uses a local-exec provisioner to run a shell command on the local machine. **The triggers** argument with **timestamp()** ensures it executes on every **terraform apply**.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Provisioners

- Provisioners execute scripts or commands on local or remote machines.
- They run as part of a resource's lifecycle.
- Common uses:
  - Install software
  - Configure services
  - Upload files after resource creation
- Should be used sparingly.
- Prefer native Terraform resources or cloud-init scripts when possible.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Types of Provisioners

- **local-exec** → runs a command on the local machine (where Terraform is executed).
- **remote-exec** → runs a command on a remote resource (e.g., EC2) after creation; needs SSH/WinRM.
- **file** → copies files from the local machine to a remote resource.

⚠ Prefer cloud-init or config mgmt tools when possible.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Example

```
resource "aws_instance" "web" {
  ami           = "ami-0abcdef1234567890"
  instance_type = "t2.micro"

  provisioner "remote-exec" {
    inline = [
      "sudo apt-get update",
      "sudo apt-get install -y nginx",
      "sudo systemctl start nginx"
    ]
  }

  connection {
    type        = "ssh"
    user        = "ubuntu"
    private_key = file("~/.ssh/id_rsa")
    host        = self.public_ip
  }
}
```

This example uses a remote-exec provisioner to install and start Nginx on an EC instance after it's launched. The connection block specifies how Terraform should connect to the remote instance.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Best Practices

- Store configs in Git.
- Use modules.
- Remote, encrypted state.
- Always run plan.
- Don’t hardcode secrets.
- Detect & fix drift.
- Pin provider/module versions.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Conclusion

- Terraform = powerful IaC tool.
- Enables automation, consistency, and collaboration.
- Workflow: Write → Plan → Apply.
- Best practices = secure, scalable infra.

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer

Find my course on LinkedIn Learning: Automated Threat Detection with Splunk, TheHive, and Snort

Or click the link below to start learning:

👉Here is the link to my course

#Boost your skills with my LinkedIn Learning course: Automated Threat Detection with Splunk, TheHive, and Snort

Assma Fadhli
DevSecOps Engineer