Terraform + AWS for beginners

a

Introduction

Agenda

What is Terraform?

What is Terraform?

"Terraform is an ~~open-source~~ tool developed by HashiCorp, and it's designed for building, changing, and version-controlling infrastructure efficiently."

Terraform allows you to define and manage the entire lifecycle of your infrastructure using a declarative configuration language.

The typical Terraform workflow: **Write**, **Plan**, **Apply**.
You start by writing your infrastructure code, then run '*terraform plan*' to see what changes will be made, and finally, '*terraform apply*' executes those changes.

What is AWS?

What is AWS?

AWS and IaC

"As AWS offers a multitude of services, managing them manually can be daunting. That's where Infrastructure as Code, and specifically Terraform, come into play."

AWS Console vs. IaC

"While AWS has an intuitive Management Console, as your architecture grows, you'll need more scalable, automated, and standardized approaches like Terraform for effective management."

Benefits of Combining AWS and Terraform

"When you combine Terraform's capabilities with AWS's extensive service offerings, you get a robust, flexible, and automated infrastructure management solution."

Why Use Terraform with AWS?

Why Use Terraform with AWS?

Advantages Over AWS CloudFormation

"AWS does offer its own IaC solution, CloudFormation, but Terraform provides more flexibility. For instance, Terraform can manage resources across multiple cloud providers, not just AWS."

Multi-cloud Capabilities

"Being cloud-agnostic means you're not locked into one vendor. You could manage resources in AWS, Google Cloud, and Azure, all within the same Terraform configuration."

Flexibility and Modularity

"Terraform's modular architecture enables you to write reusable, composable infrastructure as code components, making it easier to standardize and scale your infrastructure."

Why Use Terraform with AWS?

Speed and Efficiency

"Terraform's efficient planning stage allows you to preview changes before they happen, making the entire process faster and reducing the scope for errors."

Collaboration Features

"Terraform supports a collaborative workflow via features like state locking and workspaces, making it easier for teams to work on the same infrastructure without conflicts."

Ease of Use

"Terraform uses a simple, human-readable language called HCL (HashiCorp Configuration Language). This makes it relatively easy to get started with, especially if you're already familiar with AWS services."

Basic Terraform Syntax

Basic Terraform Syntax

Advantages Over AWS CloudFormation

"A basic Terraform configuration file usually has three main sections: **Providers**, **Resources**, and **Outputs**."

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_instance" "my_instance" {
  ami = "ami-0abcdef1234567890"
  instance_type = "t2.micro"
}

output "instance_ip" {
  value = aws_instance.my_instance.public_ip
}
```

(A little less) Basic Terraform Syntax

```
variable "availability_zone_names" {
  type    = list(string)
  default = ["us-west-1a"]
}


locals {
  service_name = "forum"
  owner        = "Community Team"
}


output "instance_ip" {
  value       = aws_db_instance.db.password
  description = "The password for logging in to the database."
  sensitive   = true

}
```

Creating AWS Resources with Terraform

Creating AWS Resources with Terraform

Step 1: Initialize the Project

```
terraform init
```

Step 2: Writing the Configuration

Terraform configurations are placed in files with the `.tf` extension.

Step 3: Terraform Plan

"`terraform plan` is your next step. It allows you to preview the changes, showing you what resources will be created, modified, or destroyed."

Step 4: Terraform Apply

"After confirming the plan, you execute `terraform apply`. This creates the resources on AWS as per your configuration. You'll get real-time feedback in the console."

Destroy

"If needed, you can destroy the resources using `terraform destroy`.
This command removes all resources that were created."

Best Practices for Using Terraform with AWS

Best Practices for Using Terraform with AWS

Version Control

"Always keep your Terraform configurations in a version control system like Git. This provides a history of changes and facilitates collaboration."

Modularize Your Code

"Create modular Terraform configurations. This makes it easier to manage and scale your infrastructure."
Terraform Registry

Use Remote Backends

"Store your Terraform state files in a secure, remote backend like AWS S3 with state locking and encryption enabled."

Best Practices for Using Terraform with AWS

Secure Your Secrets

"Never hardcode sensitive information in your .tf files. Use AWS Secrets Manager or environment variables to securely manage secrets."

Implement IAM Policies

"Use AWS IAM policies to restrict what actions Terraform can perform. This follows the principle of least privilege and enhances security."

Validate Configurations

"Always run terraform validate to ensure that your configurations are syntactically valid and internally consistent."

Best Practices for Using Terraform with AWS

Test Before Deploying

"Before deploying any changes to production, thoroughly test them in a staging environment. Consider using tools like **terraform workspace** to manage multiple environments."

Automated Pipelines

"For larger projects, use automated CI/CD pipelines to test and deploy your Terraform configurations. This ensures that your infrastructure is as code, fully tested, and automatically deployed."

Monitoring and Logging with Terraform and AWS

Best Practices for Using Terraform with AWS

AWS CloudWatch

Collects metrics, sets alarms, and triggers actions based on those alarms.

AWS CloudTrail

Tracks all API calls, including the originating IP addresses and actions taken.

AWS Config

Captures detailed resource configuration histories and determines overall compliance against the configurations specified.