

✦ Member-only story

Top 10 Terraform Scripts to Supercharge Your AWS Setups

Boost Efficiency and Simplify Your AWS Infrastructure with These Essential Terraform Scripts



Usama Malik · Follow

Published in AWS in Plain English · 7 min read · 6 days ago



61



Managing cloud infrastructure on AWS can be complex, especially as projects scale and resource requirements grow.

Terraform, an infrastructure-as-code (IaC) tool by HashiCorp, simplifies this challenge by automating resource management, making it easier to deploy, maintain, and scale infrastructure. With Terraform, AWS resources are defined in code, offering powerful configuration, reusability, and control.

This article will discuss the top 10 Terraform scripts that can streamline and optimize your AWS setups, covering everything from provisioning EC2 instances to configuring CloudWatch alarms for monitoring.

Understanding Terraform for AWS Infrastructure Automation

Terraform allows developers and DevOps engineers to create and manage AWS infrastructure through declarative code, minimizing the manual work of resource configuration. By defining resources in `.tf` files, Terraform brings consistency, versioning, and scalability to AWS resource management.

Terraform's IaC model not only saves time but also reduces errors, making it possible to manage complex environments with ease.

Here's a brief on why Terraform is ideal for AWS:

- **Version Control:** Infrastructure can be versioned like application code, making rollbacks and updates more manageable.
- **Modular Configurations:** Code reuse is simplified through modules, which can standardize configurations across teams and projects.

- **Automatic Dependency Management:** Terraform manages dependencies between resources, automatically creating resources in the correct order.

Essential Setup: Getting Started with Terraform on AWS

Before diving into specific Terraform scripts, it's essential to have Terraform configured for AWS. Here's a step-by-step guide for setting up Terraform and connecting it to your AWS environment.

- **Install Terraform:** Download Terraform from Terraform's official site and install it according to your system's requirements.
- **Configure AWS CLI:** Ensure the AWS Command Line Interface (CLI) is installed and configured with the necessary permissions.

```
aws configure
```

Enter your AWS access key, secret key, region, and output format.

- **Set Up Terraform Provider for AWS:** In a `.tf` file, declare the AWS provider and configure your region:

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

- **Initialize Terraform:** Run the following command in your terminal to initialize Terraform for your project.

```
terraform init
```

- This downloads the necessary plugins, including the AWS provider.

With Terraform initialized, you're ready to begin using the scripts to supercharge your AWS infrastructure!

Top 10 Terraform Scripts for AWS Setups

Each of these scripts provides a valuable function within AWS, offering a mix of scalability, security, and reliability for your infrastructure.

1. Provisioning an EC2 Instance

EC2 instances are the backbone of many AWS environments. Terraform can be used to define and launch EC2 instances with specific configurations, such as the AMI, instance type, and tags for easy identification.

- **Code Snippet:** Here's an example `ec2.tf` configuration to create a basic EC2 instance:

```
resource "aws_instance" "my_ec2" {  
  ami           = "ami-0c55b159cbf1f0" # Amazon Linux 2 AMI  
  instance_type = "t2.micro"  
  tags = {  
    Name = "MyEC2Instance"  
  }  
}
```

```
}  
}
```

- **Commands:** To apply this configuration, run:

```
terraform plan  
terraform apply
```

- **Benefits:** Automates EC2 provisioning, allowing you to deploy instances with consistent configurations quickly.

2. Setting Up an S3 Bucket with Policies

Amazon S3 is essential for storage needs, and Terraform can automate the creation and configuration of S3 buckets, including versioning and policies to enhance security.

- **Code Snippet:** Example configuration for creating an S3 bucket with versioning and a basic bucket policy:

```
resource "aws_s3_bucket" "my_bucket" {  
  bucket = "my-terraform-bucket"  
  acl    = "private"  
  
  versioning {  
    enabled = true  
  }  
}  
  
resource "aws_s3_bucket_policy" "my_bucket_policy" {  
  bucket = aws_s3_bucket.my_bucket.id  
  policy = <<POLICY
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "${aws_s3_bucket.my_bucket.arn}/*"
    }
  ]
}
POLICY
}
```

- **Commands:**

```
terraform plan
terraform apply
```

- **Benefits:** Automates bucket creation with custom policies for security, ensuring that your storage is both accessible and protected.

3. Configuring a VPC with Public and Private Subnets

Setting up a VPC with public and private subnets allows you to control network access, essential for isolating sensitive resources. Terraform can help automate this, streamlining the process.

- **Code Snippet:** Basic configuration for a VPC with two subnets.

```
resource "aws_vpc" "my_vpc" {
  cidr_block = "10.0.0.0/16"
}
```

```
resource "aws_subnet" "public_subnet" {  
  vpc_id      = aws_vpc.my_vpc.id  
  cidr_block  = "10.0.1.0/24"  
  map_public_ip_on_launch = true  
}  
  
resource "aws_subnet" "private_subnet" {  
  vpc_id      = aws_vpc.my_vpc.id  
  cidr_block  = "10.0.2.0/24"  
}
```

- **Commands:**

```
terraform plan  
terraform apply
```

- **Benefits:** Quickly configures a VPC with public and private subnets, ideal for separating application tiers and securing backend services.

4. Creating IAM Roles and Policies

IAM roles allow secure access control for users and applications in AWS. Terraform can be used to automate IAM role creation, reducing the manual setup of permissions and policies.

- **Code Snippet:** Configuration for an IAM role with a policy that grants read-only access to S3.

```
resource "aws_iam_role" "my_iam_role" {  
  name = "myReadOnlyRole"  
  assume_role_policy = jsonencode({
```

```
Version = "2012-10-17",
Statement = [{
  Action    = "sts:AssumeRole",
  Effect    = "Allow",
  Principal = { Service = "ec2.amazonaws.com" },
}]
})
}

resource "aws_iam_policy" "read_only_policy" {
  name = "ReadOnlyPolicy"
  policy = jsonencode({
    Version = "2012-10-17",
    Statement = [{
      Action    = ["s3:GetObject", "s3:ListBucket"],
      Effect    = "Allow",
      Resource = "*",
    }]
  })
}

resource "aws_iam_role_policy_attachment" "attach_policy" {
  role          = aws_iam_role.my_iam_role.name
  policy_arn    = aws_iam_policy.read_only_policy.arn
}
```

- **Commands:**

```
terraform plan
terraform apply
```

- **Benefits:** Automates role and policy management, making it easy to enforce access control across AWS resources.

5. Automating RDS (Relational Database Service) Deployment

Setting up RDS instances with Terraform allows for quick deployment of databases, including multi-AZ configurations for high availability.

- **Code Snippet:** Example configuration for an RDS instance.

```
resource "aws_db_instance" "my_db_instance" {  
    allocated_storage    = 20  
    engine               = "mysql"  
    instance_class       = "db.t2.micro"  
    name                = "mydatabase"  
    username             = "admin"  
    password             = "password"  
    parameter_group_name = "default.mysql5.7"  
    multi_az             = true  
}
```

- **Commands:**

```
terraform plan  
terraform apply
```

- **Benefits:** Provides an efficient way to set up and configure databases with minimal manual intervention.

6. Setting Up Auto Scaling Groups for EC2 Instances

Auto Scaling ensures that EC2 instances scale according to demand. Terraform can help configure Auto Scaling Groups, enabling high availability and fault tolerance.

- **Code Snippet:** Auto Scaling Group and Launch Configuration.

```
resource "aws_launch_configuration" "app_launch_config" {  
  image_id      = "ami-0c55b159cbfafa1f0"  
  instance_type = "t2.micro"  
}  
  
resource "aws_autoscaling_group" "app_asg" {  
  desired_capacity = 2  
  max_size         = 3  
  min_size         = 1  
  launch_configuration = aws_launch_configuration.app_launch_config.id  
  vpc_zone_identifier = [aws_subnet.public_subnet.id]  
}
```

- **Commands:**

```
terraform plan  
terraform apply
```

- **Benefits:** Automatically manages scaling based on load, helping meet traffic demand without manual intervention.

7. Deploying an Application Load Balancer (ALB)

An Application Load Balancer (ALB) distributes traffic across instances, improving availability. Terraform enables fast ALB configuration, essential for traffic management.

- **Code Snippet:** Basic configuration for an ALB.

```
resource "aws_lb" "app_lb" {  
  name = "my-app-lb"
```

```
internal          = false
load_balancer_type = "application"
security_groups   = [aws_security_group.lb_sg.id]
subnets          = [aws_subnet.public_subnet.id]
}
```

- **Commands:**

```
terraform plan
terraform apply
```

- **Benefits:** Streamlines ALB setup to balance load across instances and maintain a high availability structure.

8. Configuring CloudWatch Alarms

CloudWatch Alarms provide monitoring and alerting for specified AWS metrics, essential for resource and cost management. Terraform can automate this configuration, improving visibility into infrastructure.

- **Code Snippet:** Alarm for monitoring CPU usage of an EC2 instance.

```
resource "aws_cloudwatch_metric_alarm" "cpu_alarm" {
  alarm_name          = "high_cpu_alarm"
  comparison_operator = "GreaterThanEqualToThreshold"
  evaluation_periods  = "2"
  metric_name         = "CPUUtilization"
  namespace           = "AWS/EC2"
  period              = "120"
  statistic            = "Average"
  threshold           = "75"
  alarm_actions       = [aws_sns_topic.alerts.arn]
  dimensions = {
```

```
    InstanceId = aws_instance.my_ec2.id
  }
}
```

- **Commands:**

```
terraform plan
terraform apply
```

- **Benefits:** Provides proactive alerting and monitoring, enabling quick response to performance issues.

9. Launching Lambda Functions with Terraform

AWS Lambda provides serverless computing. With Terraform, you can deploy Lambda functions efficiently, reducing the setup time and keeping code organized.

- **Code Snippet:** Lambda function configuration.

```
resource "aws_lambda_function" "my_lambda" {
  filename      = "lambda_function_payload.zip"
  function_name = "MyLambdaFunction"
  handler       = "index.handler"
  runtime       = "nodejs14.x"
  role          = aws_iam_role.lambda_exec_role.arn
}
```

- **Commands:**

```
terraform plan
terraform apply
```

[Open in app](#)**Medium** Write

10. Managing Route 53 for Domain Name Services

Terraform can automate Route 53 configurations, simplifying DNS management for custom domain names in your AWS environment.

- **Code Snippet:** DNS configuration for a Route 53 hosted zone.

```
resource "aws_route53_zone" "my_domain" {
  name = "example.com"
}

resource "aws_route53_record" "www" {
  zone_id = aws_route53_zone.my_domain.zone_id
  name    = "www.example.com"
  type    = "A"
  ttl     = "300"
  records = [aws_instance.my_ec2.public_ip]
}
```

- **Commands:**

```
terraform plan
terraform apply
```

- **Benefits:** Centralizes DNS management, allowing quick setup and management of domain name services.

. . .

Best Practices for Using Terraform with AWS

To get the most from Terraform, consider these best practices:

- **Version Control:** Track changes to your Terraform configurations using a version control system like Git.
- **Modularization:** Break down configurations into reusable modules to simplify management across projects.
- **State Management:** Manage Terraform's state file securely, especially in a shared environment, by using remote state storage options like AWS S3.
- **Security Considerations:** Secure sensitive data, like access keys and passwords, using tools like AWS Secrets Manager or environment variables.

Conclusion

Terraform is a powerful tool for managing AWS resources efficiently, making it possible to automate, version, and scale your infrastructure. These top 10 Terraform scripts cover a variety of essential AWS services and setups, from EC2 and RDS to Lambda and Route 53, providing a strong foundation to streamline and optimize your AWS environment.

By incorporating these Terraform configurations into your AWS setups, you can boost efficiency, reduce errors, and save valuable time, allowing you to focus on building robust applications and services.

In Plain English

Thank you for being a part of the **In Plain English** community! Before you go:

- Be sure to **clap** and **follow** the writer 🙌
- Follow us: [X](#) | [LinkedIn](#) | [YouTube](#) | [Discord](#) | [Newsletter](#) | [Podcast](#)
- **Create a free AI-powered blog on Differ.**
- More content at [PlainEnglish.io](#)

AWS

Cloud Computing

Terraform

Script

DevOps



Written by Usama Malik

Follow




2.4K Followers · Writer for AWS in Plain English

AWS Devops Engineer | 2x AWS Certified | AWS Consultant | Serverless Computing | Linux Administrator | Docker | Kubernetes | Terraform | Automation Engineer

More from Usama Malik and AWS in Plain English

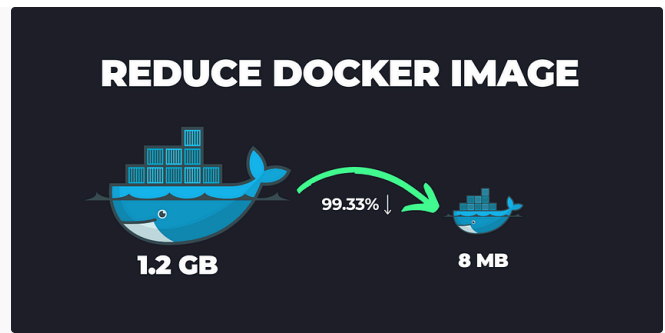



 Usama Malik in DevOps.dev

10 Essential SSH Server Security Tips & Best Practices

This article will outline 10 essential SSH server security tips and best practices that will...

★ Sep 3 🖱️ 285 💬 4 📌 ⋮




 Dipanshu in AWS in Plain English

Docker pros are shrinking images by 99%: The hidden techniques yo...

Unlock the secrets to lightning-fast deployments and slashed costs—before yo...

★ Sep 18 🖱️ 3.3K 💬 15 📌 ⋮

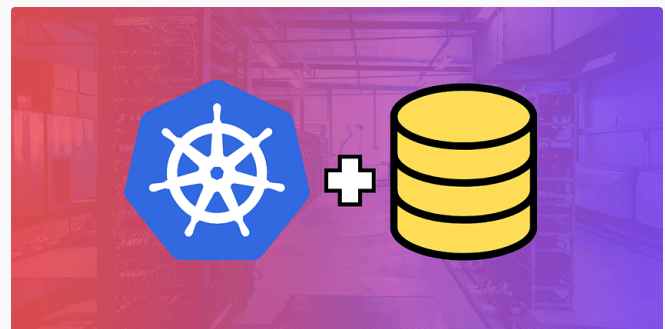



 Dipanshu in AWS in Plain English

Why Developers Are Ditching PostgreSQL, MySQL and MongoDB

The Database Revolution Your Project Is Missing Out On

★ Oct 1 🖱️ 2.1K 💬 47 📌 ⋮



 Usama Malik in DevOps.dev

7 Best Open Source Storage Solutions for Kubernetes

In this article, we'll explore 7 of the best open-source storage solutions available for...

★ Sep 30 🖱️ 238 💬 3 📌 ⋮

See all from Usama Malik

See all from AWS in Plain English

Recommended from Medium

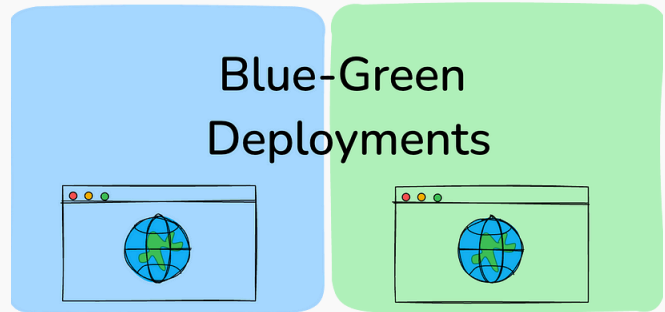



Z Zachary King in DevOps.dev

Practical Terraform: You're Doing it Wrong (Part 1)

We've all written Terraform IaC that we're not proud of before—it happens. These are...

★ Oct 14 🖱️ 62 💬 2 📌 ⋮



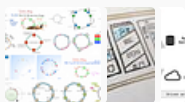
 David Bour in ITNEXT

Kubernetes Namespaces: The Secret Weapon for Zero-Risk Blue-Green Deployments

Kubernetes has made managing container images seamless with its multitude of...

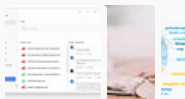
★ Nov 5 🖱️ 135 📌 ⋮

Lists



General Coding Knowledge

20 stories · 1737 saves



Productivity

242 stories · 613 saves



Natural Language Processing

1809 stories · 1424 saves



DevOpsDynamo

Ultimate Guide to Monitoring with Prometheus, Grafana, and Loki on...

I've been in Kubernetes setups for a long time, and I've seen the configuration of the...

6d ago 14

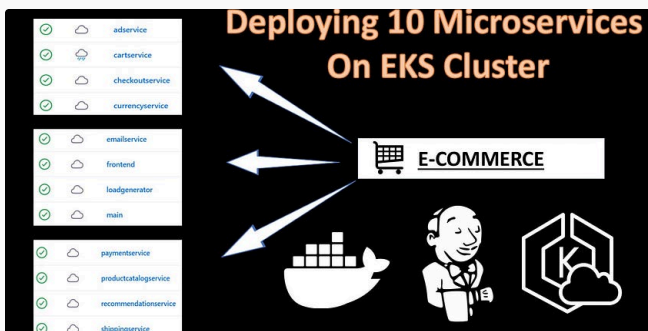


Obafemi in AWS Tip

10 Automation Scripts for Infrastructure Management Usin...

Automate important, repetitive tasks

Nov 7 43

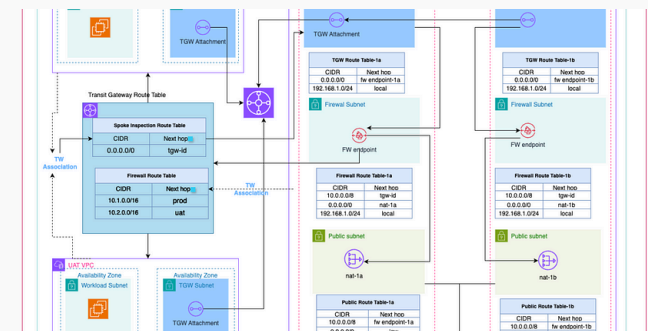


@Harsh

Deploying 10 Microservices Using CI/CD with Jenkins and Amazon...

In this guide, I'll walk you through how to set up a CI/CD pipeline using Jenkins to deploy 1...

Nov 6 40



Shivanshu Sharma in Searce

Configuring Traffic Inspection Using a Single Inspection VPC

Step by Step guide to configure centralized VPC for traffic inspection in AWS

Oct 8 25



See more recommendations