10 TERRAFORM

REALTIME PROJECTS

By
Siva Sankari B R

@learnwithsankari

1. Provisioning AWS EC2 Instances with Security Groups

1. Problem Scenario:
Manually launching EC2 instances causes inconsistent setups and is time-consuming for repeated deployments.

2. Scope of the Use Case:
Automate the deployment of EC2 instances with predefined security groups and configurations using Terraform.

3. Approach:
Use Terraform to declare infrastructure and provision EC2 instances with associated security groups.

4. Pre-requisites:

- AWS Account
- Terraform Installed
- IAM user with EC2 permissions
- SSH Key Pair

5. Step-by-Step Implementation:

- Create `main.tf` with AWS provider and EC2 configuration.
- Define `security_groups.tf` for ports (e.g., 22, 80, 443).
- Use `variables.tf` for instance type and region.
- Initialize: `terraform init`
- Preview changes: `terraform plan`
- Apply infrastructure: `terraform apply`

6. Conclusion:
Successfully automated EC2 provisioning improves deployment speed and consistency across environments.

2. Deploy a Multi-Tier Application (Web + App + DB) on AWS

1. Problem Scenario:
Complex apps require separate tiers (web, app, DB) and are hard to manage manually.

2. Scope of the Use Case:
Provision a multi-tier architecture in AWS using Terraform modules and resource dependencies.

3. Approach:
Define separate modules for each tier and manage networking, security, and instance provisioning.

4. Pre-requisites:

- VPC Design Knowledge
- AWS Account & Terraform Setup
- Private Key for Bastion Access

5. Step-by-Step Implementation:

- Create modules: `web`, `app`, `db`.
- Create VPC, subnets, route tables.
- Add NAT gateway for private subnet internet access.
- Launch EC2 instances in each tier.
- Use security groups to restrict access between tiers.

6. Conclusion:
Provisioning a complete application stack becomes repeatable and scalable using Terraform.

3. Automate S3 Bucket with Versioning and Lifecycle Rules

1. Problem Scenario:
Manual bucket creation doesn't enforce versioning or lifecycle policies, risking data loss and storage costs.

2. Scope of the Use Case:
Create S3 buckets with versioning and automatic data archival using Terraform.

3. Approach:
Use Terraform's `aws_s3_bucket` resource with `versioning` and `lifecycle_rule` blocks.

4. Pre-requisites:

- AWS CLI configured
- Terraform Installed
- S3 Full Access IAM Policy

5. Step-by-Step Implementation:

- Create `s3.tf` defining the bucket.
- Enable versioning.
- Define a lifecycle rule to transition old objects to Glacier.
- Add outputs for bucket names.
- Run: `terraform init`, `plan`, `apply`

6. Conclusion:
S3 automation with policies ensures data management is cost-efficient and secure.

4. Build an Azure Kubernetes Cluster (AKS) with Terraform

1. Problem Scenario:

Manual setup of AKS is error-prone and doesn't scale well with repeated environment provisioning.

2. Scope of the Use Case:

Provision a managed Kubernetes cluster in Azure with node pools and RBAC using Terraform.

3. Approach:

Use Terraform Azure provider and AKS module to deploy the cluster and output kubeconfig.

4. Pre-requisites:

- Azure CLI Logged in
- Terraform Installed
- Azure Subscription

5. Step-by-Step Implementation:

- Create `main.tf` with Azure provider.
- Add AKS resources with agent pools.
- Enable role-based access control.
- Output `kube_config` for kubectl use.
- Deploy using `terraform apply`.

6. Conclusion:

Terraform automates consistent AKS cluster deployment, reducing configuration overhead.

5. Setup Load Balanced Auto-Scaling Web App on AWS

1. Problem Scenario:
High-traffic websites fail without load balancing and auto-scaling in place.

2. Scope of the Use Case:
Deploy a scalable web app with ALB and Auto Scaling Group via Terraform.

3. Approach:
Use Terraform to define launch templates, ASG, and ALB configuration.

4. Pre-requisites:

- AWS Account
- Basic EC2 AMI Image
- Terraform Installed

5. Step-by-Step Implementation:

- Define a launch template.
- Create an Auto Scaling Group (min/max size).
- Add an Application Load Balancer with the target group.
- Attach ASG to the target group.
- Apply infrastructure and test scaling by simulating load.

6. Conclusion:
Terraform enables on-demand infrastructure scaling for high availability and performance.

6. Provision an AWS RDS MySQL Database

1. Problem Scenario:
Manual RDS setup lacks consistency, and credentials are hardcoded insecurely.

2. Scope of the Use Case:
Create a managed RDS instance with secure credentials and parameter groups.

3. Approach:
Use Terraform to define RDS resources, subnet groups, and secrets for DB credentials.

4. Pre-requisites:

- AWS Account
- Terraform Installed
- VPC and Subnets

5. Step-by-Step Implementation:

- Define `aws_db_instance` in `rds.tf`.
- Create a subnet group.
- Add variables for DB name, password.
- Enable backup and multi-AZ.
- Apply Terraform and connect DB from EC2.

6. Conclusion:
Secure and scalable RDS deployment is easily manageable via Terraform scripts.

7. Create Infrastructure in Multiple Environments (Dev/Staging/Prod)

1. Problem Scenario:
Hard-coded environments lead to accidental resource overwrites across dev/staging/prod.

2. Scope of the Use Case:
Use workspaces or variable-based configurations to separate infrastructure for each environment.

3. Approach:
Use Terraform workspaces or `terraform.tfvars` files per environment.

4. Pre-requisites:

- Terraform
- Cloud provider credentials

5. Step-by-Step Implementation:

- Create `variables.tf` and `terraform.tfvars` for each env.
- Use `terraform workspace, new dev`, `staging`, etc.
- Deploy and test resources isolated per environment.

6. Conclusion:
Environment isolation using Terraform prevents cross-env misconfigurations.

8. Implement IAM Role and Policies for DevOps Teams

1. Problem Scenario:
Giving full admin access to everyone increases security risks.

2. Scope of the Use Case:
Create IAM roles and granular policies for least-privilege access using Terraform.

3. Approach:
Use `aws_iam_role`, `aws_iam_policy`, and `aws_iam_role_policy_attachment`.

4. Pre-requisites:

- AWS IAM Setup
- Terraform Installed

5. Step-by-Step Implementation:

- Define IAM policy JSON.
- Create an IAM role with a trusted entity.
- Attach policy to the role.
- Output role ARN for use.

6. Conclusion:
IAM automation enforces consistent access control, improving cloud security posture.

9. Monitor EC2 Instances with CloudWatch Alarms

1. Problem Scenario:
Lack of monitoring results in late detection of performance issues.

2. Scope of the Use Case:
Attach CloudWatch alarms to EC2 instances using Terraform for proactive alerting.

3. Approach:
Create `aws_cloudwatch_metric_alarm` with thresholds for CPU usage, disk, etc.

4. Pre-requisites:

- EC2 Instance Running
- SNS Topic (for notifications)
- Terraform Installed

5. Step-by-Step Implementation:

- Define CloudWatch alarm resource.
- Attach to instance ID.
- Create SNS topic and subscription.
- Test by simulating high CPU load.

6. Conclusion:
CloudWatch integration improves system observability and response time.

10. Create a GitHub Actions CI/CD Pipeline to Deploy Terraform

1. Problem Scenario:
Running Terraform manually introduces errors and delays in automated workflows.

2. Scope of the Use Case:
Use GitHub Actions to run `terraform plans` and `apply commits` to the main branch.

3. Approach:
Define GitHub workflow with Terraform setup and deploy job using OIDC or secrets.

4. Pre-requisites:

- GitHub Repo
- AWS Credentials Stored in GitHub Secrets
- Terraform Code

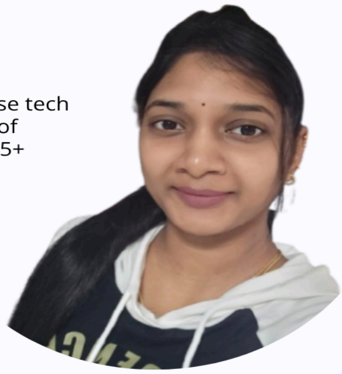
5. Step-by-Step Implementation:

- Create `.github/workflows/terraform.yml`
- Add jobs for `terraform fmt`, `init`, `plan`, `apply`
- Add manual approval before apply
- Push to repo and monitor Actions tab

6. Conclusion:
CI/CD integration ensures infrastructure is deployed and updated in a controlled, automated manner.

ABOUT ME

I'm Siva Sankari, Co-Founder of CareerByteCode, a global enterprise tech platform accelerating the success of 241,000+ IT professionals across 95+ countries.



MY ROLES

As a senior consultant in Cloud, Devops and AI/GenAI, i help enterprises reimagine infrastructure and operatins driving intelligent transformation through automation, cloud-native architectures and AI-powered pipelines.

HELPING CLIENTS ON

- Career Guidance
- Mock Interviews
- Interview Preparation Packages
- Resume Writing
- Linkedin Optimization
- Training - Upcoming
- Real-Time Projects
- Lead generation
- Consulting
- Support

careerbytecode.substack.com

CareerByteCode
Learning Made simple

As a passionate cloud DevOps consultant, I'm committed to uplifting IT professionals through personalized career services. **Connect with me on LinkedIn @learnwithsankari** to accelerate your tech career today!

Like
Share
Follow
Siva Sankari
Devops & Cloud Mentor
CAREERBYTECODE
in @learnwithsankari