Infrastructure Provisioning Using Terraform for AWS Resources

Overview

This paper outlines a plan to provision a Kubernetes cluster, Elastic Container Registry (ECR), and a MySQL Database on Amazon Web Services (AWS) using Terraform. The purpose of this project is to automate the infrastructure setup process and improve the efficiency, scalability, and security of our systems. By adopting Terraform, we can define and manage our infrastructure as code, reducing manual tasks and ensuring consistency across environments.

Project Goals

1. Create a Kubernetes cluster for container orchestration to deploy and manage our applications efficiently.
2. Set up an Elastic Container Registry (ECR) for storing Docker container images securely.
3. Provision a MySQL Database for our application's data storage.
4. Utilize variables in Terraform for secure access key management and authentication.

AWS Services Used

1. Elastic Kubernetes Service (EKS): Managed Kubernetes service to deploy, scale, and manage containerized applications.
2. Elastic Container Registry (ECR): A fully managed Docker container registry for securely storing and managing container images.
3. Relational Database Service (RDS): A managed MySQL database for storing application data.

Terraform Advantages

1. Infrastructure as Code: Terraform enables us to define our infrastructure using declarative configuration files, making it versionable, repeatable, and shareable.
2. Scalability: We can easily scale our infrastructure by modifying the Terraform configuration and applying the changes.
3. Consistency: Terraform ensures consistent provisioning across development, testing, and production environments.

4. Security: Using variables for access key management and authentication helps keep our sensitive information secure.

IAM Access Key Management

To provision AWS resources using Terraform, we will generate IAM (Identity and Access Management) access keys. These keys consist of an Access Key ID and a Secret Access Key, acting as the user's credentials for programmatic access to the AWS API.

IAM User Creation

We will create a dedicated IAM user with the necessary permissions to interact with AWS services and provision the resources for this project. The IAM user will be granted the following minimum permissions:

1. AmazonEKSClusterPolicy: To create and manage the Amazon EKS cluster.
2. AmazonEC2FullAccess: To manage the underlying EC2 instances and networking resources.
3. AmazonECRFullAccess: To manage the Elastic Container Registry for Docker images.
4. AmazonRDSFullAccess: To create and manage the MySQL database on Amazon RDS.

We will adhere to the principle of least privilege and only grant the IAM user the permissions required for this specific project.

Best Practices for IAM Access Key Management

To ensure the security of our infrastructure, we will follow these best practices for IAM access key management:

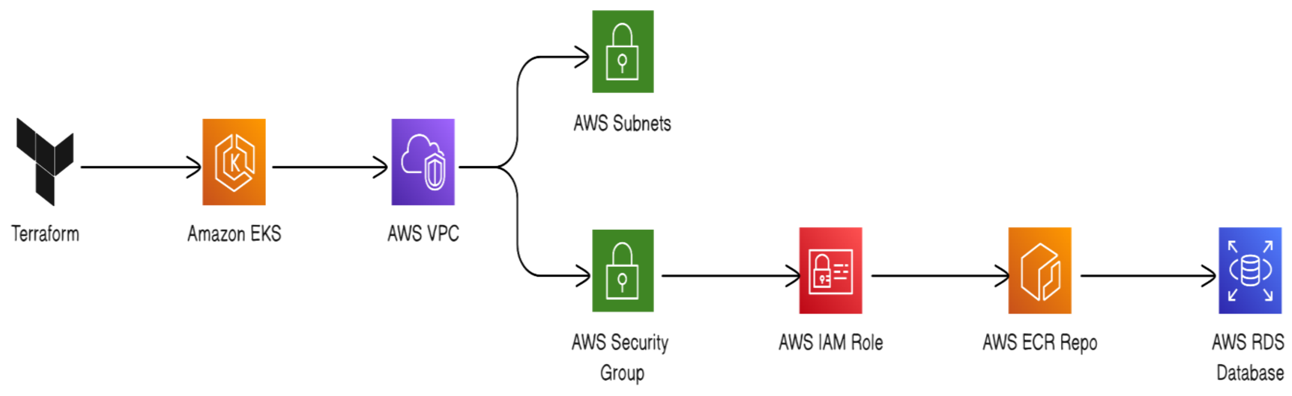
1. **Secure Storage**: IAM access keys will be stored securely, preferably using a secure credential manager or AWS Vault to avoid accidental exposure.
2. **Rotating Access Keys**: We will regularly rotate IAM access keys to mitigate the impact of potential key compromises.
3. **Least Privilege**: The IAM user will be granted only the necessary permissions required for this project. We will avoid using root-level or excessive permissions.

4. **Restrict Access**: Access to the IAM access keys will be restricted by setting proper IAM policies and applying the principle of least privilege.

Using IAM Access Keys with Terraform

In the **terraform.tfvars** file, we will provide the IAM user's Access Key ID and Secret Access Key. Terraform will use these credentials to authenticate with AWS and provision the infrastructure. We will ensure that these values are kept confidential and are not shared in public repositories or insecure locations.

Architecture Overview



Steps to Provision Infrastructure

1. Set up AWS Credentials: Create an IAM user with appropriate permissions and generate Access Key ID and Secret Access Key. Store them securely on the local machine.
2. Write Terraform Configuration: Define the infrastructure resources in **main.tf** and use variables for access keys and other sensitive information in **variables.tf**.
3. Create Kubernetes Cluster: Provision the EKS cluster, VPC, subnets, and security groups using Terraform.
4. Set up ECR Repository: Create a private ECR repository to securely store Docker images.
5. Deploy MySQL Database: Provision the RDS instance and specify the database username and password using variables.

6. Apply Terraform Configuration: Run **terraform init**, **terraform plan**, and **terraform apply** to provision the resources.

Conclusion

By adopting Terraform for infrastructure provisioning, we can achieve a more efficient, scalable, and secure environment for our applications. The ability to manage our infrastructure as code ensures consistency and simplifies the deployment process across multiple environments. Additionally, using variables for access key management enhances security by keeping sensitive information isolated.