Infrastructure as Code (IaC) Deployment Report

Why I Chose Terraform Over CloudFormation

I chose Terraform for this project over AWS CloudFormation due to its cloud-agnostic nature, flexibility, and modular approach. Terraform allows for reusable modules, making the codebase maintainable and scalable. Its declarative syntax and open-source ecosystem provide better community support and tooling. In contrast to CloudFormation, which is tightly coupled with AWS, Terraform can manage infrastructure across multiple cloud providers, which is beneficial for hybrid and future-proof deployments.

Key Learnings

- **Infrastructure as Code (IaC):** Learned how to manage and provision complete infrastructure using code. Gained experience in creating modular, reusable, and version-controlled configurations.
- **Amazon ECS (Fargate):** Understood how to deploy containerized workloads without managing servers. Learned to set up ECS clusters, services, and task definitions.
- **Prefect Workers on ECS:** Learned how to run Prefect workers on ECS Fargate. Configured containers with secrets from AWS Secrets Manager, enabling secure workflow execution.
- **IAM and Networking:** Learned to define fine-grained IAM roles and secure networking practices, including VPC, private/public subnets, and NAT Gateway.
- **CI/CD with GitHub Actions:** Integrated GitHub Actions to automate Terraform workflows, including applying and destroying infrastructure using manual triggers.
- **Terraform Remote State with S3 & DynamoDB:** Implemented remote backend using S3 for state storage and DynamoDB for state locking, ensuring safe team collaboration and avoiding state corruption.

Challenges Faced and Solutions

- **Module Integration:** Initially faced issues referencing outputs from one module in another. Resolved this by carefully defining and calling outputs using output.tf in each module.
- **Service Discovery Configuration:** Setting up ECS service discovery was tricky. Fixed it by explicitly defining a private DNS namespace and linking it to the ECS cluster.
- **Secrets Management:** Accessing Prefect Cloud credentials securely was complex. Used AWS Secrets Manager and added necessary IAM permissions to the task execution role.

- **Terraform State Management:** Initially used local state, which posed collaboration risks. Migrated to remote state storage using S3 and DynamoDB for consistent, team-safe infrastructure management.
- **CI/CD Automation:** Manual deployment was time-consuming and error-prone. Integrated GitHub Actions to automate Terraform provisioning and destruction using workflow dispatch triggers.

Demo Link

Demo of the full deployment and functionality:

https://drive.google.com/file/d/1OUuquLJoCnPx-xXD6-2mzmPTonOroK-l/view?usp=sharing

Suggestions for Improvement

- **Auto-scaling:** Implement ECS Service Auto Scaling based on CPU and memory usage to handle dynamic workloads efficiently.
- **Monitoring and Alerts:** Integrate with CloudWatch for container logs, metrics, and alerting to monitor system health.
- **Parameter Store:** Store configuration parameters in SSM Parameter Store for better management and auditing.