Project Title: Multi-Cloud Deployment and Management with Terraform

Problem Statement

In multi-cloud environments, managing resources across multiple cloud providers (AWS, Azure, GCP) can be complex and error-prone, especially when attempting to maintain consistent configurations and enable failover across clouds. Traditional deployment methods often lack scalability, are prone to human error, and do not offer high availability across regions or providers. This project will create a Terraform-based solution that enables automated multi-cloud deployments, consistent configuration management, and high availability, with support for disaster recovery in case of regional failures.

Project Goals

1. Set up a multi-cloud deployment using Terraform, deploying resources across AWS, Azure, and GCP to ensure high availability.
2. Automate infrastructure creation and configuration management across cloud providers with Terraform modules, focusing on scalability and modularity.
3. Implement load balancing and failover mechanisms to automatically handle outages or high traffic by shifting workloads to alternative clouds.
4. Enable disaster recovery by replicating resources and databases across clouds and testing failover capabilities.
5. Provide monitoring and alerts for resource usage and costs across all environments to optimize and manage resources efficiently.

Tools Used

- Programming Languages: HCL (HashiCorp Configuration Language) for Terraform scripts,
Python (optional) for automation scripts
- Terraform: Infrastructure as Code tool to automate multi-cloud deployment and resource management.
- Cloud Providers: AWS, Azure, and GCP to host and manage resources across multiple cloud platforms.
- Vault: For secure storage and management of access keys and sensitive credentials across multiple providers.
- Monitoring: Prometheus and Grafana to track resource health and utilization across cloud providers.

- Alerting: PagerDuty or Slack API for real-time alerts on cloud resource status and usage.
- GitHub/GitLab: Version control and CI/CD integration to manage Terraform scripts and automate deploymentsProject Sprints.

Each sprint has a 20-hour workload, organized into specific tasks to achieve incremental progress.

Sprint 1: Initial Setup and Cloud Access Configuration
- **Tasks**:
- Set up the project repository, define the folder structure, and initialize the Terraform configuration.
- Create cloud provider accounts for AWS, Azure, and GCP, and configure the necessary permissions and IAM roles.
- Set up Vault or a similar tool to securely store and manage cloud provider credentials.
- Define Terraform provider configurations for AWS, Azure, and GCP in a modular format.
- **Goal**: Establish connectivity with each cloud provider and ensure Terraform can access and manage resources securely.

Sprint 2: Basic Multi-Cloud Resource Deployment
- **Tasks**:
- Define Terraform modules for common resources (e.g., VMs, storage, load balancers) that can be reused across AWS, Azure, and GCP.
- Deploy basic infrastructure across all three cloud providers, such as VMs and storage buckets, using the Terraform modules.
- Test the Terraform deployment process to confirm that resources are consistently created across providers.
- Set up basic networking and security configurations, ensuring resources can communicate securely across cloud boundaries.
- **Goal**: Build and test the initial infrastructure deployment across AWS, Azure, and GCP, laying the foundation for multi-cloud management.

Sprint 3: Load Balancing and Failover Configuration
- **Tasks**:

- Implement load balancers (e.g., AWS ELB, Azure Load Balancer, GCP Load Balancer) for each cloud provider to manage traffic across instances.
- Configure DNS-based routing with services like AWS Route 53 or GCP Cloud DNS to direct traffic across cloud providers.
- Set up failover mechanisms to redirect traffic in case of a failure in one provider, with health checks to monitor service availability.
- Test failover by simulating a service outage in one provider and ensuring traffic is redirected to the remaining providers.
- **Goal**: Enable load balancing and failover across clouds, ensuring high availability and reliability for the application.

Sprint 4: Disaster Recovery and Data Replication
- **Tasks**:
- Set up automated data replication across clouds for critical resources like databases and storage (e.g., AWS RDS, Azure SQL, GCP Cloud SQL).
- Implement backup and restore procedures using Terraform, storing backups across providers for redundancy.
- Test disaster recovery procedures by failing over to backups in a secondary provider and verifying data integrity and consistency.
- Document disaster recovery plans and procedures.
- **Goal**: Ensure disaster recovery by replicating data across providers, minimizing data loss and downtime in case of outages.

Sprint 5: Monitoring, Alerts, and Cost Management
- **Tasks**:
- Configure Prometheus to monitor resource health, usage, and performance metrics across AWS, Azure, and GCP.
- Set up Grafana dashboards to visualize multi-cloud resource utilization, availability, and cost metrics.
- Integrate alerts with PagerDuty or Slack to notify the DevOps team of high resource usage, outages, or cost threshold breaches.
- Implement cost tracking and analysis to help manage and optimize expenses across all providers.
- **Goal**: Provide visibility into the health and cost of multi-cloud resources, with proactive alerts to help manage and optimize cloud usage.

Sprint 6: Documentation, Final Testing, and Deployment Automation

- **Tasks**:
- Write comprehensive documentation, including setup guides, usage instructions, disaster recovery steps, and troubleshooting tips.
- Develop CI/CD pipelines using GitHub Actions or GitLab CI/CD to automate Terraform deployments and updates across environments.
- Perform final testing, simulating outages and disaster recovery scenarios to ensure system resilience and reliability.
- Collect feedback from team members or stakeholders and make necessary improvements.
- **Goal**: Deliver a production-ready, fully automated multi-cloud deployment system with documentation and deployment automation.

Summary of Deliverables by End of Project
- Automated Multi-Cloud Deployment System: A Terraform-based deployment system for AWS, Azure, and GCP.
- High Availability and Load Balancing: Load balancing and failover across cloud providers to ensure continuous availability.
- Disaster Recovery and Data Replication: Backup and restore processes to handle outages or data loss.
- Monitoring and Cost Management Dashboard: Grafana dashboards for visibility into resource usage, costs, and alerts.
- Comprehensive Documentation: Documentation for setup, usage, disaster recovery, and troubleshooting.