Terraform Real-Time Project Question for a Team of Four

1 Project Title

Deploy a Scalable Web Application Infrastructure on AWS with Terraform, Including State Management Using S3 and DynamoDB

2 Objective

As a team of four, design and deploy a scalable web application infrastructure on AWS using Terraform. The infrastructure should include state management with a remote backend in an S3 bucket and a DynamoDB table for state locking to ensure safe collaboration. The project simulates a real-world scenario where multiple team members work on the same infrastructure codebase.

3 Project Requirements

3.1 Infrastructure Components

- **VPC**: Create a Virtual Private Cloud (VPC) with public and private subnets across at least two Availability Zones (AZs).
- **EC2 Instances**: Deploy two EC2 instances (web servers) in the public subnets behind an Application Load Balancer (ALB).
- **RDS**: Set up an RDS instance (e.g., MySQL or PostgreSQL) in the private subnets for the application database.
- **Auto Scaling Group**: Configure an Auto Scaling Group for the EC2 instances to ensure scalability.
- **Security Groups**: Define appropriate security groups for the ALB, EC2 instances, and RDS to control traffic.
- **S3 Bucket**: Create an S3 bucket to store static assets for the web application (e.g., images, CSS files).

3.2 Terraform State Management

- **S3 Backend**: Configure Terraform to store the state file in an S3 bucket to enable collaboration among team members.
- **DynamoDB Locking**: Set up a DynamoDB table to manage state locking, preventing concurrent modifications to the state file.
- **Bucket Security**: Enable versioning and encryption on the S3 bucket to secure the state file.

3.3 Collaboration and Modularity

- Use Terraform modules to organize the code (e.g., separate modules for VPC, EC2, RDS, ALB, etc.).
- Ensure the codebase is stored in a Git repository (e.g., GitHub or GitLab) for version control.
- Implement a branching strategy (e.g., feature branches for each team member) to simulate collaborative development.

3.4 Testing and Validation

- Validate the infrastructure using `terraform plan` and `terraform apply`.
- Simulate concurrent state access by having multiple team members apply changes simultaneously to test DynamoDB locking.
- Verify the web application is accessible via the ALB URL and that static assets are served from the S3 bucket.
- **Note**: Do not test destructive actions like `terraform destroy` in a production-like environment without proper safeguards.

3.5 Documentation

- Document the architecture diagram (can be created using tools like Lucidchart or Draw.io).
- Provide a README with setup instructions:
  - Instructions for setting up the Terraform backend (S3 and DynamoDB).
  - Steps to deploy the infrastructure.
  - Include a troubleshooting guide for common issues (e.g., state lock errors, connectivity issues).

4 Team Roles and Suggested Task Division

- **Team Member 1 (Infrastructure Lead)**:
  - Design VPC, subnets, and ALB configurations.
  - Create the Terraform module for VPC and networking components.
  - Validate connectivity between components.
- **Team Member 2 (Compute and Scaling)**:
  - Configure EC2 instances, Auto Scaling Group, and launch templates.
  - Create the Terraform module for EC2 and Auto Scaling Group.
  - Test auto-scaling by simulating traffic to trigger scaling events.

- **Team Member 3 (Database and Storage)**:
  - Set up the RDS instance and S3 bucket for static assets.
  - Create Terraform modules for RDS and S3.
  - Ensure secure access to the RDS and S3 buckets.
- **Team Member 4 (State Management and CI/CD)**:
  - Configure the S3 backend and DynamoDB table for state locking.
  - Set up the Git repository and branching strategy.
  - (Optional) Implement a simple CI/CD pipeline using GitHub Actions or GitLab CI to run `terraform plan` and `terraform apply` on pull requests.

5 Terraform Code Example for S3 Backend and DynamoDB Locking

```
# main.tf
terraform {
  required_version = ">= 1.0.0"

  backend "s3" {
    bucket         = "my-terraform-state-bucket"
    key            = "prod/terraform.tfstate"
    region         = "us-east-2"
    dynamodb_table = "terraform-state-lock"
    encrypt        = true
  }
}

# S3 bucket for Terraform state
resource "aws_s3_bucket" "terraform_state" {
  bucket = "my-terraform-state-bucket"

  versioning {
    enabled = true
  }

  server_side_encryption_configuration {
    rule {
      apply_server_side_encryption_by_default {
        sse_algorithm = "AES256"
      }
    }
  }
}
```

```
# DynamoDB table for state locking
resource "aws_dynamodb_table" "terraform_state_lock" {
  name           = "terraform-state-lock"
  billing_mode   = "PAY_PER_REQUEST"
  hash_key       = "LockID"

  attribute {
    name = "LockID"
    type = "S"
  }
}
```

6 Deliverables

1. Terraform code repository with modular structure and backend configuration.
2. Architecture diagram of the deployed infrastructure.
3. README with setup, deployment, and troubleshooting instructions.
4. Demonstration of the deployed infrastructure (e.g., access the web application via ALB URL).
5. Evidence of successful state locking (e.g., logs showing lock acquisition/release).

7 Evaluation Criteria

- Correctness of the Terraform code and infrastructure deployment.
- Proper configuration of S3 backend and DynamoDB locking.
- Modularity and reusability of Terraform code.
- Collaboration effectiveness (e.g., Git commits, branching strategy).
- Quality of documentation and troubleshooting guide.
- Scalability and security of the infrastructure.

8 Time Estimate

2–3 days (assuming 4–6 hours per day per team member).

9 Tips for Success

- Start by setting up the S3 bucket and DynamoDB table manually or via Terraform to configure the backend early.
- Use `terraform init` to initialize the backend and verify state locking.
- Test small changes incrementally to avoid state conflicts.

- Communicate frequently to avoid overlapping changes in the same Terraform resources.
- Use `terraform fmt` and `terraform validate` to ensure code quality.

10 Conclusion

This project simulates a real-world DevOps scenario, focusing on collaboration, state management, and scalable infrastructure deployment. For further clarification or additional resources, please reach out.