Documentation: Setting Up and Deploying the Python App Project

A. Step-by-Step Guide

1. Prerequisites

- **AWS Account**: You need an AWS account to provision the resources.
- **Terraform**: Ensure you have Terraform installed. Follow the installation guide here.
- **AWS CLI**: Ensure you have AWS CLI installed and configured on your machine or make use of terraform cloud and add the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY as variables
- **SSH Key Pair**: Create an SSH key pair to access the EC2 instances.

2. Setup the Repository

Configure the repo to use on terraform cloud and add all your scripts to the repo

3. Initialize Terraform

Initialize Terraform to install necessary providers and modules.

```
terraform init
```

4. Plan the Infrastructure

Review the infrastructure plan to ensure resources are correctly defined.

```
terraform plan
```

5. Apply the Infrastructure

Apply the Terraform configuration to provision the resources.

```
terraform apply
```

6. Verify EC2 Instances and ELB

- After the resources are provisioned, navigate to the AWS Management Console.
- Verify that the EC2 instances are running in the private subnet.
- Check that the Elastic Load Balancer (ELB) is set up and associated with the EC2 instances.

7. Access the Application

- Find the DNS name of the ELB in the AWS Management Console.
- Access the application by navigating to the ELB DNS name in your web browser.

B. Assumptions Made and Troubleshooting Steps

Assumptions

1. **IAM Role and Policies**: It is assumed that the IAM roles and policies required for EC2 and DynamoDB access are correctly set up.
2. **VPC Configuration**: The VPC is correctly configured with public and private subnets.
3. **Docker Image**: The Docker image for the application is correctly built and pushed to Docker Hub.

Troubleshooting Steps

1. **EC2 Instance Initialization**: If the EC2 instances take too long to initialize, ensure that the `user_data.sh` script has the correct commands to install Docker and run the Docker container.
2. **Application Not Accessible**:
   - Verify the security group rules to ensure HTTP/HTTPS traffic is allowed.
   - Check the ELB health checks to ensure they are configured correctly.
3. **DynamoDB Table Not Found**:
   - Ensure the DynamoDB table name is correctly set in the environment variables.
   - Check the IAM roles to confirm they have the necessary permissions to access DynamoDB.
4. **Docker Container Logs**:
   - SSH into the EC2 instance using the SSH key pair.
   - Check the Docker container logs using `docker logs <container_id>` to debug any issues with the application.

File Structure Overview

- **`app.py`**: Python application file.
- **`Dockerfile`**: Dockerfile to build the application image.
- **`dynamodb.tf`**: Terraform file for DynamoDB provisioning (remove if not used).
- **`ec2.tf`**: Terraform file for EC2 instance provisioning.
- **`elb.tf`**: Terraform file for ELB setup.
- **`iam.tf`**: Terraform file for IAM roles and policies.
- **`internet_gateway.tf`**: Terraform file for Internet Gateway setup.
- **`main.tf`**: Main Terraform configuration file.
- **`outputs.tf`**: Terraform file to define output values.
- **`route_tables.tf`**: Terraform file for route table configurations.
- **`security_groups.tf`**: Terraform file for security group configurations.
- **`subnets.tf`**: Terraform file for subnet configurations.
- **`terraform.tfvars`**: Terraform variables file.
- **`variables.tf`**: Terraform file to define variable inputs.
- **`vpc.tf`**: Terraform file for VPC setup.
- **`user_data.sh`**: Script to install Docker and run the Docker container on EC2 instances.

This documentation provides a comprehensive guide to setting up and deploying the Python application using Terraform and Docker, with assumptions and troubleshooting steps to help address common issues.