IaC Report

Architecture

The designed IaC includes a CLI tool for planning, applying, and destroying infrastructure. The tool contains the following modules:

1. **CLI**
    a. An interface for supporting the aforementioned IaC lifecycle
    b. The main login has been implemented in *cli.py*
2. **Config**
    a. Specifies resources in a YAML file. The sample *infrastructure.yaml* file contains the following settings:
        i. Three resources, each with a type, name, and config section.
            1. The virtual machine configuration includes instance type, AMI ID, region, tags, and references to a security group and key pair.
            2. The network configuration defines a VPC with CIDR block, two subnets in different availability zones, and a route table with a route to an Internet Gateway.
            3. The database configuration specifies engine details, instance class, storage settings, credentials, backup settings, and network configuration.
3. **Configuration parser**
    a. Parses and validates the YAML configuration. It verifies that all mandatory properties of the resources are properly defined.
    b. The logic has been implemented in *config_parser.py*
4. **Resource managers**
    a. The tool can create, delete, or update a database, virtual machine (VM), or network using the provider SDK.
    b. Although the tool currently supports AWS, it is designed to be generic and easily extendable to other providers.
    c. The database, network, and VM managers can be found in the *resource_manager* folder.
    d. The tool uses the Boto3 library to connect to AWS. It automatically collects AWS credentials from local environments or AWS CLI credentials.
    e. **Note:** The tool does not support updating networks, as VPC manipulation is often complex.
    f. **Note:** The delete method removes all components within the VPC before deleting the VPC itself.

    g. **Note:** Connection to AWS has been commented to simplify running the tool.
5. **State manager**
    a. This implementation provides persistent state storage using SQLite, which allows the tool to keep track of the created resources across different runs. The state is stored in a file named *iac_state.db*.
    b. The tool uses a state manager during the planning phase to compare the current state of resources with the desired states specified in the YAML file. It then identifies any changes that need to be made. Based on this comparison, the tool determines whether each resource needs to be updated or created.
6. **Cloud Provider API Interface** (the logic is still empty)
7. **Unit Test** (the logic is still empty)

Improvements

- Update the planning phase to determine necessary changes in more details.
- Improve the state manager to store the history of actions/resources rather than overwriting the past states.

Installation

- Details have been explained in the README.md file.