Assignment - management of multiple deployments

Rationale: Currently we have multiple deployments that span across multiple accounts. Each deployment has a seperate terraform codebase, with some TF modules shared. As our deployments grow, managing each terraform configuration separately becomes less and less viable. We need to move to a more scalable solution and the goal of this assignment is to create a simple demo of some approach for the solution.

Please publish your solution in a publicly available git repository. All code, scripts and possibly documentation should be committed into this repository. If manual steps are performed (eg. start a docker-compose or terraform commands), they should be documented so everybody familiar with the technologies should be able to reproduce the steps. Each task should be represented as one or more commits in the repository.

Tasks:

All the stuff should run on a local machine, no cloud environment should be involved for simplicity.

1. Initial setup, simulate single deployment.

Goal summary: Create an instance of postgres that is managed by terraform.

Create a docker-compose.yaml file which spins-up instances of Postgresql database. The DB should be accessible from localhost only.

Then create a Terraform module which creates databases and users in the running Postgresql instance. There should be 3 databases and 3 user pairs, each user should have full access to only one of the databases. Also create another user which will have access to all the databases but only for read-only operations. Users should be authenticated by strong passwords. Choose database and user names freely. It is ok to leave the root passwords for the databases in the docker-compose file (for simplicity, it is just simulation), but any passwords configured via terraform should not be visible in git. You can use the local terraform state file.

2. Step two, multiple deployments.

Goal summary: Create a scalable solution for managing multiple deployments with terraform.

Update the docker-compose file so it spins up a few more (eg 3) Postgresql instances, each listening on a different port. Each postgresql instance represents a separate deployment, imagine each instance being in a separate AWS account for example. Each should have a setup (databases and users) identical to the first one, only the passwords should be unique for each database. The emphasis is to be able to add another instance with minimal effort.