AS A developer at Alpine Inc,

I WANT an automated deployment process for our Foo app, including infrastructure,

SO THAT we can avoid human errors during the deployment process.

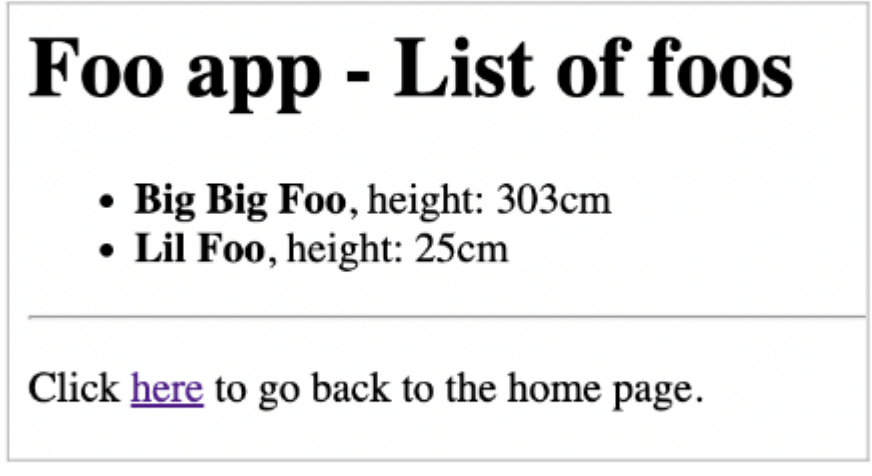
Acceptance criteria:

GIVEN credentials for an AWS environment, and a docker image already built and hosted in an image registry,

WHEN we run the deployment script in the repo (or, ideally, when a GitHub Actions workflow runs),

THEN we get the infrastructure we need, with the application running on it.

The Foo app is a nodejs web app with a PostgreSQL database backend. The dev team has built the app's docker image and hosted it at `patrmitacr.azurecr.io/assignment2app` (note: this is a docker pull URL, not a web address). They would like you to deploy this application to EC2. The current deployment process is described in `how-to-deploy.txt` in the starter repo. When running correctly, the UI of the Foo app looks like this:



Ideally they would like to achieve greater resiliency and automation, but as a minimum they need an automated process to get the app container and its database running on an EC2 instance. More details can be found in the 'Deliverable' section below.

The Approach

Alpine Inc's CTO (Chief Technology Officer) has prescribed the following tools for use in this project:

- GitHub (GitHub Classroom)
- GitHub Actions – used for creating the pipelines
- Terraform
- Ansible
- AWS
- Docker

You may use additional tools, libraries, etc as long as their inclusion is explained and justified in your README.md.

Section A

1. In README.md, describe and justify your solution, including appropriate use of diagrams (e.g. process diagram and infrastructure architecture diagram). (20%)

    a. Note: As you proceed through sections A-D, you should update your README file to cover your whole solution, not just your solution for section A.

2. Use Terraform to deploy the required infrastructure e.g. EC2 instance(s). (10%)

3. Use Ansible to configure the EC2 instance(s) to run the app and database containers. (10%)

4. Write a shell script to deploy the infrastructure and run the application on it. (10%)

Section B

5. To improve the resiliency of the application, deploy the app container on two identical EC2 instances behind a load balancer, with the database running on a separate EC2 instance. (10%)

Section C

6. In your Terraform configuration, use an S3 bucket remote backend for state. (10%)

    a. Note: You can clickops the state bucket (it is fairly normal in industry to clickops the state bucket) or you can use the Terraform configuration in `misc/state-bucket-infra.tf` (separate from your main Terraform configuration).

    b. Note #2: Don't get the Terraform configurations mixed up. You can't deploy the state bucket with the configuration that is using the state bucket. If this last point is too confusing, just clickops the state bucket and ignore `state-bucket-infra.tf`. :-)

Section D

Note: If you attempt this part of this assignment, also include a standalone deployment script as per "Section A". (The CTO wants to "avoid vendor lock-in".)

7. Create a GitHub Actions workflow which deploys the infrastructure and runs the application on it. (10%)

8. Make the GitHub Actions workflow run whenever the `main` branch is modified, and also able to be triggered via the GitHub Actions REST API. (10%)

9. Ensure that if the infrastructure is already deployed in the environment, then re-running the workflow is a no-op. (10%)