Migration to Containerisation and the Cloud

We have achieved the team goal of migrating their amazing application away from our old infrastructure, and into AWS. We were tasked with providing a seamless local environment for developers; this was achieved by creating a simple dockerised NGINX load balancer, to mimic the live environment, and utilising docker-compose to spin up the application locally.

Once you have downloaded Docker Desktop from https://docs.docker.com/docker-for-windows/install/, you can now clone the repository and from the root directory run the following command, and docker will provision everything you need; you will be able to see the containers in your Docker Desktop Dashboard:

> **docker-compose up**

You can learn more about docker-compose here: https://docs.docker.com/compose/reference/up/

We wanted to ensure the new cloud infrastructure was managed through code, to provide some consistency across environments and make it easy to manage. All of the resources used to host our containers in AWS were written in Terraform, and can be provisioned by running the following command, from the *terraform/infra* or *terraform/ecr* directories:

> **terraform apply**

You can learn more about what each resource is by cross referencing the code with the official documentation here: https://www.terraform.io/docs/index.html

To trigger our deployments, GitHub Workflows has been utilised to provide the basis of a CI/CD pipeline. Continuous Deployment is achieved through listening for commits in the application directory on the master branch which triggers a deployment; deployments are also triggered on a schedule to ensure external dependencies are kept up to date.

As the application grows and we introduce environments, we will create actions which will run automated tests against the code that is committed to the repository as part of our Continuous Integration strategy.

The GitHub workflow action builds a Docker image with the latest application code, and pushes the image to the Elastic Container Repository in AWS; the image and task definition are then passed to ECS to deploy. The ECS deployment strategy results in zero downtime, and old containers are decommissioned when the new ones are healthy.

Here are some features that we would like to implement in the future:

- Run application changes through a Snyk container to check for vulnerabilities
- Set up a HTTPS target group, SSL certs, and redirect HTTP traffic to HTTPS
- Add Terraform to the pipeline along with IaC unit tests using OPA
- Increase reusability by utilising arguments, variables, and modules
- Introduce Terraform state locks through DynamoDb

If you would like to know more about the work carried out for the team, you think this work may be of benefit to your team, or if you have any questions, please reach out.