## Migrating a Monolithic Website to Microservices on Google Kubernetes Engine

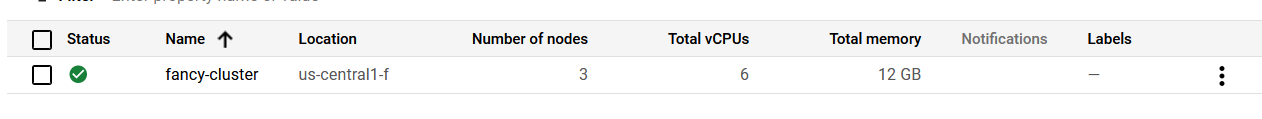
### Preface

I set the compute zone, cloned the repo, enabled the container API and created the three worker nodes.

### Deployment: Monolith

Text

Description automatically generated  
The instance we are running currently are displayed on this image. We cannot be sure if these are real machines or virtualized.



Text

Description automatically generated

After running the deployment script and going to the external IP address on the browser we can see the website is **live**.

Graphical user interface, text, application

Description automatically generated  
Doing some digging this website was created via frameworks: React and Express.

## Microservice Migration

Since we are not creating the Order Container and its already present we will practice creating a docker container from a image. However, the point of this command was to show how we can deploy the container and put it on the cloud registry.

Graphical user interface, text, application

Description automatically generated

The **Build History** shows the containers we have created.

Now we are set to deploy.

We create a deployment object. Recall that the default deployment object has the replicasets set to one. So, when we look at the k8s internals we notice that only one orders MS has been created:

Graphical user interface, text

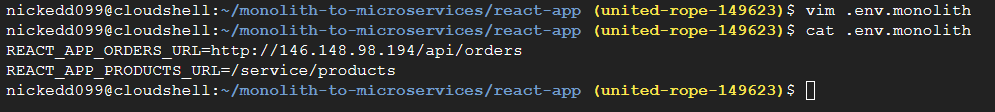
Description automatically generated

Then, we can expose the Order MS via a service to truly use the load balancing abstractions Kubernetes provides. Graphical user interface, text

Description automatically generated

Now that we have decomposed the Order service into its **standalone** microservice we just need to integrate it with the core monolith app.

We can alter the Express Paths to this:



The server now calls on the Order API and not the direct resource

Then we run the node build command to rebuild the app with the new API route. We are also asked to update our docker container and by extension the google container registry. The screenshot spans many pages so I have only included a snipper of the container update success.

A picture containing graphical user interface

Description automatically generated

## Migrating Products

We do the same thing we did to orders but to the products.

The steps are:

1. Create the container and deploy it.
2. Assign the container to a service to be picked up by a Kubernetes pod.
   1. This is done via the kubectl expose command

Text

Description automatically generated

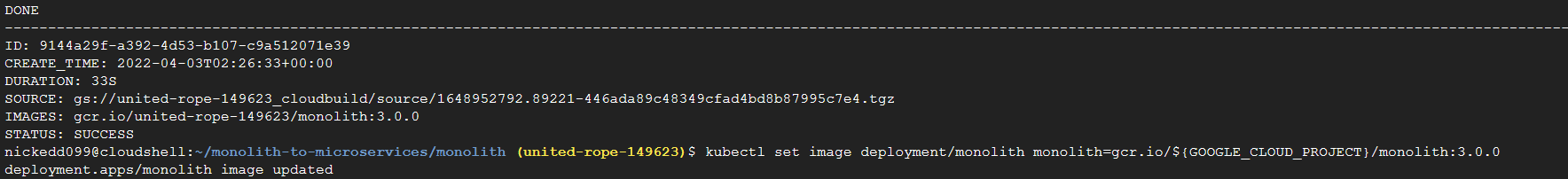
Here are the commands ran and their results. The product side has been migrated and now is ready to be integrated.

We first replace the API routes:

Graphical user interface, text, application

Description automatically generated

Then we update our image and version tag on the GCR:



At this point I can view the website and all the sections work as intended.

### Migrating the Frontend

The steps are **identical** to the other two migrations.

The only difference is that we run a predefined script instead of altering the routes.

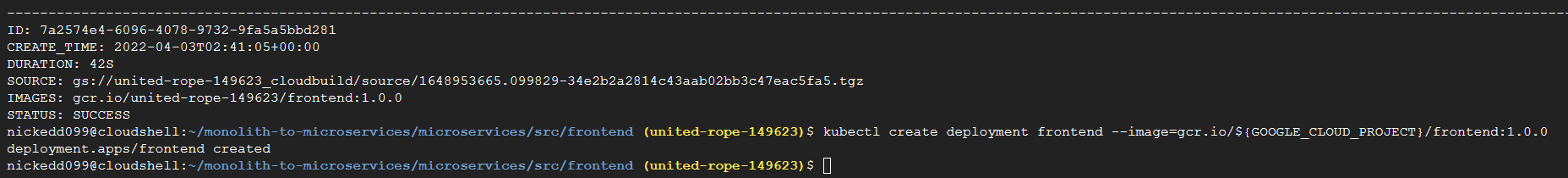
We run the new frontend script, deploy the new build and upload it to the GCR with a new version tag.

Text

Description automatically generated with medium confidence

This is an image of the npm build command which I had to play with to get it to work. (disable eslint)

Then we upload our container to the registry and create the service and load balancer.



A picture containing graphical user interface

Description automatically generated

A new *external* IP appears and we can call it to see to see if our application has been successfully converted.

Graphical user interface, website

Description automatically generated