

Migrating a Monolithic Website to Microservices on Google

Kubernetes Engine

Set the default zone and project configuration:

gcloud config set compute/zone (zone)

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-01-9c14e4a9b438) $ gcloud config set compute/zone us-central1-c
WARNING: Property validation for compute/zone was skipped.
Updated property [compute/zone].
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-01-9c14e4a9b438) $
```

Task 1. Clone the source repository

You will use an existing monolithic application of an imaginary ecommerce website, with a simple welcome page, a products page and an order history page. We will just need to clone the source from our git repo, so we can focus on breaking it down into microservices and deploying to Google Kubernetes Engine (GKE).

- Run the following commands to clone the git repo to your Cloud Shell instance and change to the appropriate directory. You will also install the NodeJS dependencies so you can test your monolith before deploying:

```
cd ~
```

```
git clone https://github.com/googlecodelabs/monolith-to-microservices.git
```

```
cd ~/monolith-to-microservices
```

```
./setup.sh
```

Task 2. Create a GKE cluster

Now that you have your working developer environment, you need a Kubernetes cluster to deploy your monolith, and eventually the microservices, to! Before you can create a cluster, make sure the proper API's are enabled.

1. Run the following command to enable the Containers API so you can use Google Kubernetes Engine:

gcloud services enable container.googleapis.com

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices (qwiklabs-gcp-01-9c14e4a9b438) $ gcloud services enable container.googleapis.com
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices (qwiklabs-gcp-01-9c14e4a9b438) $ gcloud container clusters create fancy-cluster --num-nodes 3 --machine-type e2-standard-4
Note: The Kubelet readonly port (10255) is now deprecated. Please update your workloads to use the recommended alternatives. See https://cloud.google.com/kubernetes-engine/docs/how-to/disable-kubelet-readonly-port for ways to check usage and for migration instructions.
Note: Your Pod address range ('--cluster-ip4-cidr') can accommodate at most 1008 node(s).
```

2. Run the command below to create a GKE cluster named **fancy-cluster** with **3** nodes:

gcloud container clusters create fancy-cluster --num-nodes 3 --machine-type=e2-standard-4

3. Once the command has completed, run the following to see the cluster's three worker VM instances:

gcloud compute instances list

```
ZONE: us-central1-c
MACHINE_TYPE: e2-standard-4
PREEMPTIBLE:
INTERNAL_IP: 10.128.0.3
EXTERNAL_IP: 104.197.180.127
STATUS: RUNNING

NAME: gke-fancy-cluster-default-pool-29d7c8de-dv8f
ZONE: us-central1-c
MACHINE_TYPE: e2-standard-4
PREEMPTIBLE:
INTERNAL_IP: 10.128.0.5
EXTERNAL_IP: 34.58.87.137
STATUS: RUNNING

NAME: gke-fancy-cluster-default-pool-29d7c8de-jtnq
ZONE: us-central1-c
MACHINE_TYPE: e2-standard-4
PREEMPTIBLE:
INTERNAL_IP: 10.128.0.4
EXTERNAL_IP: 34.9.26.215
STATUS: RUNNING
```

You can also view your Kubernetes cluster and related information in the Cloud Console. From the **Navigation menu**, scroll down to **Kubernetes Engine** and click **Clusters**.

You should see your cluster named *fancy-cluster*.

Task 3. Deploy the existing monolith

Since the focus of this lab is to break down a monolith into microservices, you need to get a monolith application up and running.

- Run the following script to deploy a monolith application to your GKE cluster:

```
cd ~/monolith-to-microservices
```

```
./deploy-monolith.sh
```

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices (qwiklabs-gcp-01-9c14e4a9b438)$ cd ~/monolith-to-microservices
./deploy-monolith.sh
Enabling Cloud Build APIs...
```

Accessing the monolith

1. To find the external IP address for the monolith application, run the following command:

kubectl get service monolith

```
Deployment completed successfully!
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices (qwiklabs-gcp-01-9c14e4a9b438)$ kubectl get service monolith
NAME      TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
monolith  LoadBalancer  34.118.239.242   <pending>        80:31262/TCP     4s
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices (qwiklabs-gcp-01-9c14e4a9b438)$
```

2. If your output lists the external IP as <pending> give it a minute and run the command again.
3. Once you've determined the external IP address for your monolith, copy the IP address. Point your browser to this URL (such as <http://203.0.113.0>) to check if your monolith is accessible.

You should see the welcome page for the monolithic website. The welcome page is a static page that will be served up by the Frontend microservice later on. You now have your monolith fully running on Kubernetes!

Task 4. Migrate orders to a microservice

Now that you have a monolith website running on GKE, start breaking each service into a microservice. Typically, a planning effort should take place to determine which services to break into smaller chunks, usually around specific parts of the application like business domain.

For this lab you will create an example and break out each service around the business domain: Orders, Products, and Frontend. The code has already been migrated for you so you can focus on building and deploying the services on Google Kubernetes Engine (GKE).

Create Orders microservice

The first service to break out is the Orders service. Make use of the separate codebase provided and create a separate Docker container for this service.

Create a Docker container with Cloud Build

Since the codebase is already available, your first step will be to create a Docker container of your Order service using Cloud Build.

Normally this is done in a two step process that entails building a Docker container and pushing it to a registry to store the image for GKE to pull from. Cloud Build can be

used to build the Docker container **and** put the image in the Artifact Registry with a single command!

Google Cloud Build will compress the files from the directory and move them to a Cloud Storage bucket. The build process will then take all the files from the bucket and use the Dockerfile to run the Docker build process. The --tag flag is specified with the host as gcr.io for the Docker image, the resulting Docker image will be pushed to the Artifact Registry.

1. Run the following commands to build your Docker container and push it to the Artifact Registry:

```
cd ~/monolith-to-microservices/microservices/src/orders gcloud builds submit --tag gcr.io/${GOOGLE_CLOUD_PROJECT}/orders:1.0.0 .
```

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices (qwklabs-gcp-01-9c14e4a9b438)$ cd ~/monolith-to-microservices/microservices/src/orders
gcloud builds submit --tag gcr.io/${GOOGLE_CLOUD_PROJECT}/orders:1.0.0 .
Creating temporary archive of 8 file(s) totalling 24.3 KiB before compression.
Some files were not included in the source upload.

Check the gcloud log [/tmp/tmp.uYTehZsN2t/logs/2025.05.06/09.48.15.244534.log] to see which files and the contents of the
default gcloudignore file used (see '$ gcloud topic gcloudignore' to learn
more).

Uploading tarball of [.] to [gs://qwklabs-gcp-01-9c14e4a9b438_cloudbuild/source/1746524895.520254-efb76ed708fd4c739ff49dd44fb3c627.tgz]
Created [https://cloudbuild.googleapis.com/v1/projects/qwklabs-gcp-01-9c14e4a9b438/locations/global/builds/b038c93a-62fd-4218-bb29-f2bc3f2d19d1].
Logs are available at [ https://console.cloud.google.com/cloud-build/builds/b038c93a-62fd-4218-bb29-f2bc3f2d19d1?project=277308962514 ].
Waiting for build to complete. Polling interval: 1 second(s).

----- REMOTE BUILD OUTPUT -----
starting build "b038c93a-62fd-4218-bb29-f2bc3f2d19d1"

FETCHSOURCE
Fetching storage object: gs://qwklabs-gcp-01-9c14e4a9b438_cloudbuild/source/1746524895.520254-efb76ed708fd4c739ff49dd44fb3c627.tgz#1746524896704692
█

DONE

-----
INFO: The service account running this build projects/qwklabs-gcp-01-9c14e4a9b438/serviceAccounts/277308962514-compute@developer.gserviceaccount.com does not have permission to write logs to Cloud Logging. To fix this, grant the Logs Writer (roles/logging.logWriter) role to the service account.

1 message(s) issued.
ID: b038c93a-62fd-4218-bb29-f2bc3f2d19d1
CREATE_TIME: 2025-05-06T09:48:18+00:00
DURATION: 38S
SOURCE: gs://qwklabs-gcp-01-9c14e4a9b438_cloudbuild/source/1746524895.520254-efb76ed708fd4c739ff49dd44fb3c627.tgz
IMAGES: gcr.io/qwklabs-gcp-01-9c14e4a9b438/orders:1.0.0
STATUS: SUCCESS
```

2. To view your build history, or watch the process in real time, in the console, search for **Cloud Build** then click on the **Cloud Build** result.
3. On the **History** page you can see a list of all your builds; there should only be 1 that you just created. If you click on the build ID, you can see all the details for that build including the log output.
4. From the build details page, to view the container image that was created, in the right section click the **Execution Details** tab and see Image.

Deploy container to GKE

Now that you have containerized the website and pushed the container to the Artifact Registry, it is time to deploy to Kubernetes!

Kubernetes represents applications as **Pods**, which are units that represent a container (or group of tightly-coupled containers). The Pod is the smallest deployable unit in Kubernetes. In this tutorial, each Pod contains only your microservices container.

To deploy and manage applications on a GKE cluster, you must communicate with the Kubernetes cluster management system. You typically do this by using the **kubectl** command-line tool from within Cloud Shell.

First, create a **Deployment** resource. The Deployment manages multiple copies of your application, called replicas, and schedules them to run on the individual nodes in your cluster. In this case, the Deployment will be running only one pod of your application. Deployments ensure this by creating a **ReplicaSet**. The ReplicaSet is responsible for making sure the number of replicas specified are always running.

The kubectl create deployment command below causes Kubernetes to create a Deployment named **Orders** on your cluster with **1** replica.

- Run the following command to deploy your application:

```
kubectl create deployment orders --image=gcr.io/${GOOGLE_CLOUD_PROJECT}/orders:1.0.0
```

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/microservices/src/orders (qwklabs-gcp-01-9c14e4a9b438)$ kubectl create deployment orders --image=gcr.io/${GOOGLE_CLOUD_PROJECT}/orders:1.0.0
deployment.apps/orders created
```

Verify the deployment

- To verify the Deployment was created successfully, run the following command:

```
kubectl get all
```

```
deployment.apps/orders created
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/microservices/src/orders (qwklabs-gcp-01-9c14e4a9b438)$ kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/monolith-7cf97878ff-n5fwk	1/1	Running	0	3m31s
pod/orders-767599c49d-xwgnl	1/1	Running	0	29s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	34.118.224.1	<none>	443/TCP	8m37s
service/monolith	LoadBalancer	34.118.239.242	34.122.18.70	80:31262/TCP	3m28s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/monolith	1/1	1	1	3m32s
deployment.apps/orders	1/1	1	1	30s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/monolith-7cf97878ff	1	1	1	3m32s
replicaset.apps/orders-767599c49d	1	1	1	30s

You can see your Deployment which is current, the replicaset with the desired pod count of 1, and the pod which is running. Looks like everything was created successfully!

You can also view your Kubernetes deployments in the Cloud Console from the **Navigation menu**, go to **Kubernetes Engine > Workloads**.

Expose GKE container

You have deployed our application on GKE, but don't have a way of accessing it outside of the cluster. By default, the containers you run on GKE are not accessible from the Internet, because they do not have external IP addresses. You must explicitly expose your application to traffic from the Internet via a [Service](#) resource. A Service provides networking and IP support to your application's Pods. GKE creates an external IP and a Load Balancer.

For purposes of this lab, the exposure of the service has been simplified. Typically, you would use an API gateway to secure your public endpoints.

When you deployed the Orders service, you exposed it on port 8081 internally via a Kubernetes deployment. In order to expose this service externally, you need to create a Kubernetes service of type LoadBalancer to route traffic from port 80 externally to internal port 8081.

- Run the following command to expose your website to the Internet:

```
kubectl expose deployment orders --type=LoadBalancer --port 80 --target-port 8081
```

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/microservices/src/orders (qwiklabs-gcp-01-9c14e4a9b438)$ kubectl expose deployment orders --type=LoadBalancer --port 80 --target-port 8081
service/orders exposed
```

Accessing the service

GKE assigns the external IP address to the Service resource, not the Deployment.

- To find out the external IP that GKE provisioned for your application, inspect the Service with the `kubectl get service` command:

```
kubectl get service orders
```

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/microservices/src/orders (qwiklabs-gcp-01-9c14e4a9b438)$ kubectl get service orders
NAME      TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
orders    LoadBalancer 34.118.231.219 <pending>     80:31969/TCP    26s
```

Once you've determined the external IP address for your application, copy the IP address. Save it for the next step when you change your monolith to point to the new Orders service

Reconfigure the monolith

Since you removed the Orders service from the monolith, you will have to modify the monolith to point to the new external Orders microservice.

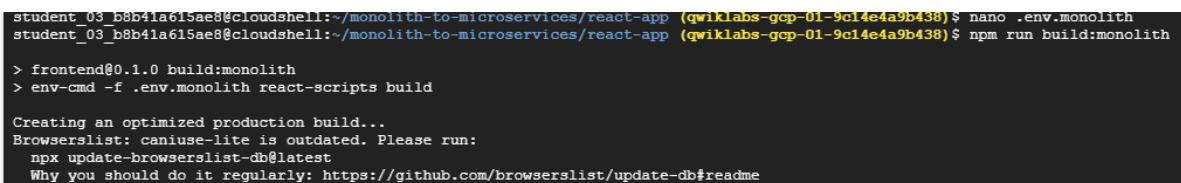
When breaking down a monolith, you are removing pieces of code from a single codebase to multiple microservices and deploying them separately. Since the microservices are running on a different server, you can no longer reference your service URLs as absolute paths - you need to route to the Order microservice server address. This will require some downtime to the monolith service to update the URL for each service that has been broken out. This should be accounted for when planning on moving your microservices and monolith to production during the microservices migration process.

You need to update your config file in the monolith to point to the new Orders microservices IP address.

1. Use the nano editor to replace the local URL with the IP address of the Orders microservice:

```
cd ~/monolith-to-microservices/react-app
```

```
nano .env.monolith
```



```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/react-app (qwiklabs-gcp-01-9c14e4a9b438)$ nano .env.monolith
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/react-app (qwiklabs-gcp-01-9c14e4a9b438)$ npm run build:monolith

> frontend@0.1.0 build:monolith
> env-cmd -f .env.monolith react-scripts build

Creating an optimized production build...
Browserslist: caniuse-lite is outdated. Please run:
  npx update-browserslist-db@latest
  Why you should do it regularly: https://github.com/browserslist/update-db#readme
```

When the editor opens, your file should look like this:

```
REACT_APP_ORDERS_URL=/service/orders
REACT_APP_PRODUCTS_URL=/service/products
```

2. Replace the REACT_APP_ORDERS_URL to the new format while replacing with your Orders microservice IP address so it matches below:

```
REACT_APP_ORDERS_URL=http://<ORDERS_IP_ADDRESS>/api/orders
REACT_APP_PRODUCTS_URL=/service/products
```

Copied!

```
content_copy
```

3. Press CTRL+O, press ENTER, then CTRL+X to save the file in the nano editor.
4. Test the new microservice by navigating the URL you just set in the file. The webpage should return a JSON response from your Orders microservice.

- Next, rebuild the monolith frontend and repeat the build process to build the container for the monolith and redeploy to the GKE cluster:

```
npm run build:monolith
```

- Create Docker container with Cloud Build:

```
cd ~/monolith-to-microservices/monolith
gcloud builds submit --tag gcr.io/${GOOGLE_CLOUD_PROJECT}/monolith:2.0.0 .
```

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/react-app (qwiklabs-gcp-01-9c14e4a9b438)$ cd ~/monolith-to-microservices/monolith
gcloud builds submit --tag gcr.io/${GOOGLE_CLOUD_PROJECT}/monolith:2.0.0 .
Creating temporary archive of 27 file(s) totalling 2.4 MiB before compression.
Uploading tarball of [.] to [gs://qwiklabs-gcp-01-9c14e4a9b438.cloudbuild/source/1746525338.165366-8fcc84c946ce42d6b5835f3caa2bb544.tgz]
Created [https://cloudbuild.googleapis.com/v1/projects/qwiklabs-gcp-01-9c14e4a9b438/locations/global/builds/49c7377f-7846-43b8-9372-f9b3401d2b93].
Logs are available at [ https://console.cloud.google.com/cloud-build/builds/49c7377f-7846-43b8-9372-f9b3401d2b93?project=277308962514 ].
Waiting for build to complete. Polling interval: 1 second(s).
```

- Deploy container to GKE:

```
kubectl set image deployment/monolith
monolith=gcr.io/${GOOGLE_CLOUD_PROJECT}/monolith:2.0
```

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/monolith (qwiklabs-gcp-01-9c14e4a9b438)$ kubectl set image deployment/monolith monolith=gcr.io/${GOOGLE_CLOUD_PROJECT}/monolith:2.0
deployment.apps/monolith image updated
```

- Verify the application is now hitting the Orders microservice by going to the monolith application in your browser and navigating to the Orders page.

Task 5. Migrate Products to microservice

Create new Products microservice

Continue breaking out the services by migrating the Products service next. Follow the same process as before. Run the following commands to build a Docker container, deploy your container, and expose it via a Kubernetes service.

- Create Docker container with Cloud Build:

```
cd ~/monolith-to-microservices/microservices/src/products
gcloud builds submit --tag gcr.io/${GOOGLE_CLOUD_PROJECT}/products:1.0.0 .
```

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/monolith (qwiklabs-gcp-01-9c14e4a9b438)$ cd ~/monolith-to-microservices/microservices/src/products
gcloud builds submit --tag gcr.io/${GOOGLE_CLOUD_PROJECT}/products:1.0.0 .
Creating temporary archive of 8 file(s) totalling 26.6 KiB before compression.
Some files were not included in the source upload.

Check the gcloud log [/tmp/tmp.uYtehZmN2t/logs/2025.05.06/09.57.04.222663.log] to see which files and the contents of the
default gcloudignore file used (see '$ gcloud topic gcloudignore' to learn
more).

Uploading tarball of [.] to [gs://qwiklabs-gcp-01-9c14e4a9b438.cloudbuild/source/1746525424.508464-b1f7a2379e5b4881a7b3fec21e546775.tgz]
Created [https://cloudbuild.googleapis.com/v1/projects/qwiklabs-gcp-01-9c14e4a9b438/locations/global/builds/09e18fff-659e-425a-88c0-b451bf9568c4].
Logs are available at [ https://console.cloud.google.com/cloud-build/builds/09e18fff-659e-425a-88c0-b451bf9568c4?project=277308962514 ].
Waiting for build to complete. Polling interval: 1 second(s).
```

- Deploy container to GKE:


```
kubectl create deployment products --image=gcr.io/${GOOGLE_CLOUD_PROJECT}/products:1.0.
```

```
INFO: The service account running this build projects/qwiklabs-gcp-01-9c14e4a9b438/serviceAccounts/277308962514-compute@developer.gserviceaccount.com does not have permission to write logs to Cloud Logging. To fix this, grant the Logs Writer (roles/logging.logWriter) role to the service account.
1 message(s) issued.
ID: 09e18frrf-659e-425a-88c0-b451bf9568c4
CREATE TIME: 2025-05-06T09:57:06+00:00
DURATION: 44s
SOURCE: gs://qwiklabs-gcp-01-9c14e4a9b438_cloudbuild/source/1746525424.508464-b1f7a2379e5b4881a7b3fec21e546775.tgz
IMAGES: gcr.io/qwiklabs-gcp-01-9c14e4a9b438/products:1.0.0
STATUS: SUCCESS
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/microservices/src/products (qwiklabs-gcp-01-9c14e4a9b438)$ kubectl create deployment products --image=gcr.io/${GOOGLE_CLOUD_PROJECT}/products:1.0.0
deployment.apps/products created
```

3. Expose the GKE container:

```
kubectl expose deployment products --type=LoadBalancer --port 80 --target-port 8082
```

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/microservices/src/products (qwiklabs-gcp-01-9c14e4a9b438)$ kubectl expose deployment products --type=LoadBalancer --port 80 --target-port 8082
service/products exposed
```

4. Find the public IP of the Products services the same way you did for the Orders service:

```
kubectl get service products
```

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/microservices/src/products (qwiklabs-gcp-01-9c14e4a9b438)$ kubectl get service products
NAME      TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
products  LoadBalancer  34.118.231.114 <pending>      80:30679/TCP     22s
```

You will use the IP address in the next step when you reconfigure the monolith to point to your new Products microservice.

Reconfigure the monolith

1. Use the nano editor to replace the local URL with the IP address of the new Products microservices:

```
cd ~/monolith-to-microservices/react-app
```

```
nano .env.monolith
```

When the editor opens, your file should look like this:

```
REACT_APP_ORDERS_URL=http://<ORDERS_IP_ADDRESS>/api/orders
```

```
REACT_APP_PRODUCTS_URL=/service/products
```

2. Replace the REACT_APP_PRODUCTS_URL to the new format while replacing with your Product microservice IP address so it matches below:

```
REACT_APP_ORDERS_URL=http://<ORDERS_IP_ADDRESS>/api/orders
```

```
REACT_APP_PRODUCTS_URL=http://<PRODUCTS_IP_ADDRESS>/api/products
```

Copied!

```
content_copy
```

3. Press CTRL+O, press ENTER, then CTRL+X to save the file.
4. Test the new microservice by navigating the URL you just set in the file. The webpage should return a JSON response from the Products microservice.
5. Next, rebuild the monolith frontend and repeat the build process to build the container for the monolith and redeploy to the GKE cluster. Run the following commands complete these steps:
6. Rebuild monolith config files:

```
npm run build:monolith
```

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/react-app (qwiklabs-gcp-01-9c14e4a9b438)$ npm run build:monolith
> frontend@0.1.0 build:monolith
> env-cmd -f .env.monolith react-scripts build

Creating an optimized production build...
Browserslist: caniuse-lite is outdated. Please run:
  npx update-browserslist-db@latest
  Why you should do it regularly: https://github.com/browserslist/update-db#readme
```

7. Create Docker container with Cloud Build:

```
cd ~/monolith-to-microservices/monolith
gcloud builds submit --tag gcr.io/${GOOGLE_CLOUD_PROJECT}/monolith:3.0.0 .
```

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/react-app (qwiklabs-gcp-01-9c14e4a9b438)$ cd ~/monolith-to-microservices/monolith
gcloud builds submit --tag gcr.io/${GOOGLE_CLOUD_PROJECT}/monolith:3.0.0 .
Creating temporary archive of 27 file(s) totalling 2.4 MiB before compression.
Uploading tarball of [.] to [gs://qwiklabs-gcp-01-9c14e4a9b438_cloudbuild/source/1746525824.304249-0983b49a8ee84594a2be421d91be7fa8.tgz]
Created [https://cloudbuild.googleapis.com/v1/projects/qwiklabs-gcp-01-9c14e4a9b438/locations/global/builds/d35c8c11-bf8f-4cae-90a7-addf73d1f2bc].
Logs are available at [ https://console.cloud.google.com/cloud-build/builds/d35c8c11-bf8f-4cae-90a7-addf73d1f2bc?project=277308962514 ].
Waiting for build to complete. Polling interval: 1 second(s).

----- REMOTE BUILD OUTPUT -----
starting build "d35c8c11-bf8f-4cae-90a7-addf73d1f2bc"

FETCHSOURCE
Fetching storage object: gs://qwiklabs-gcp-01-9c14e4a9b438_cloudbuild/source/1746525824.304249-0983b49a8ee84594a2be421d91be7fa8.tgz#1746525826959585
```

8. Deploy container to GKE:

```
kubectl set image deployment/monolith
monolith=gcr.io/${GOOGLE_CLOUD_PROJECT}/monolith:3.0.0
```

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/monolith (qwiklabs-gcp-01-9c14e4a9b438)$ kubectl set image deployment/monolith monolith=gcr.io/${GOOGLE_CLOUD_PROJECT}/monolith:3.0.0
deployment.apps/monolith image updated
```

Verify your application is now hitting the new Products microservice by going to the monolith application in your browser and navigating to the Products page



MS - Vintage Typewriter - \$67.99



MS - Vintage Camera Lens - \$12.49



MS - Home Barista Kit - \$124



MS - Terrarium - \$36.45



MS - Film Camera - \$2245



MS - Vintage Record Player - \$65.5



MS - Metal Camping Mug - \$24.33



MS - City Bike - \$789.5



MS - Air Plant - \$12.3

Task 6. Migrate Frontend to microservice

The last step in the migration process is to move the Frontend code to a microservice and shut down the monolith! After this step is completed, you will have successfully migrated the monolith to a microservices architecture!

Create a new frontend microservice

Follow the same procedure as the last two steps to create a new frontend microservice.

Previously when you rebuilt the monolith you updated the config to point to the monolith. Now you need to use the same config for the frontend microservice.

1. Run the following commands to copy the microservices URL config files to the frontend microservice codebase:

```
cd ~/monolith-to-microservices/react-app
```

```
cp .env.monolith .env
```

npm run build

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/monolith (qwklabs-gcp-01-9c14e4a9b438)$ cd ~/monolith-to-microservices/react-app
cp .env.monolith .env
npm run build

> frontend@0.1.0 prebuild
> npm run build:monolith

> frontend@0.1.0 build:monolith
> env-cmd -f .env.monolith react-scripts build

Creating an optimized production build...
Browserslist: caniuse-lite is outdated. Please run:
  npx update-browserslist-db@latest
  Why you should do it regularly: https://github.com/browserslist/update-db#readme
```

2. Once that is completed, follow the same process as the previous steps. Run the following commands to build a Docker container, deploy your container, and expose it to via a Kubernetes service.
3. Create Docker container with Google Cloud Build:

```
cd ~/monolith-to-microservices/microservices/src/frontend
gcloud builds submit --tag gcr.io/${GOOGLE_CLOUD_PROJECT}/frontend:1.0.0 .
```

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/react-app (qwklabs-gcp-01-9c14e4a9b438)$ cd ~/monolith-to-microservices/microservices/src/frontend
gcloud builds submit --tag gcr.io/${GOOGLE_CLOUD_PROJECT}/frontend:1.0.0 .
Creating temporary archive of 25 file(s) totalling 2.4 MiB before compression.
Uploading tarball of [...] to [gs://qwklabs-gcp-01-9c14e4a9b438_cloudbuild/source/1746525948.762397-cec70505cd0845929b840dd275245e9f.tgz]
```

4. Deploy container to GKE:

```
kubectl create deployment frontend --image=gcr.io/${GOOGLE_CLOUD_PROJECT}/frontend:1.0.0
```

```
INFO: The service account running this build projects/qwklabs-gcp-01-9c14e4a9b438/serviceAccounts/277308962514-compute@developer.gserviceaccount.com does not have permission to write logs to Cloud Logging. To fix this, grant the Logs Writer (roles/logging.logWriter) role to the service account.

1 message(s) issued.
ID: dc5f271b-df47-4d1b-a1d0-8da59dc689fe
CREATE_TIME: 2025-05-06T10:05:52+00:00
DURATION: 44S
SOURCE: gs://qwklabs-gcp-01-9c14e4a9b438_cloudbuild/source/1746525948.762397-cec70505cd0845929b840dd275245e9f.tgz
IMAGES: gcr.io/qwklabs-gcp-01-9c14e4a9b438/frontend:1.0.0
STATUS: SUCCESS
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/microservices/src/frontend (qwklabs-gcp-01-9c14e4a9b438)$ kubectl create deployment frontend --image=gcr.io/${GOOGLE_CLOUD_PROJECT}/frontend:1.0.0
deployment.apps/frontend created
```

5. Expose GKE container:

```
kubectl expose deployment frontend --type=LoadBalancer --port 80 --target-port 8080
```

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/microservices/src/frontend (qwklabs-gcp-01-9c14e4a9b438)$ kubectl expose deployment frontend --type=LoadBalancer --port 80 --target-port 8080
service/frontend exposed
```

Delete the monolith

Now that all of the services are running as microservices, delete the monolith application! In an actual migration, this would also entail DNS changes, etc., to get the existing domain names to point to the new frontend microservices for the application.

- Run the following commands to delete the monolith:

kubectl delete deployment monolith
kubectl delete service monolith

Copied!

content_copy

Test your work

To verify everything is working, your old IP address from your monolith service should not work now, and your new IP address from your frontend service should host the new application.

- To see a list of all the services and IP addresses, run the following command:

kubectl get services

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/microservices/src/frontend (qwiklabs-gcp-01-9c14e4a9b438)$ kubectl delete deployment monolith
kubectl delete service monolith
deployment.apps "monolith" deleted
service "monolith" deleted
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/microservices/src/frontend (qwiklabs-gcp-01-9c14e4a9b438)$ kubectl get services
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
frontend	LoadBalancer	34.118.239.86	34.71.114.255	80:30709/TCP	3m11s
kubernetes	ClusterIP	34.118.224.1	<none>	443/TCP	28m
orders	LoadBalancer	34.118.231.219	104.154.202.235	80:31969/TCP	19m
products	LoadBalancer	34.118.231.114	34.55.34.140	80:30679/TCP	11m

Once you've determined the external IP address for your frontend microservice, copy the IP address. Point your browser to this URL (such as <http://203.0.113.0>) to check if your frontend is accessible.