

Hosting a Web App on Google Cloud Using Compute Engine

Task 1. Enable Compute Engine API

- Enable the [Compute Engine API](#) by executing the following:

```
gcloud services enable compute.googleapis.com
```

Task 2. Create Cloud Storage bucket

You will use a Cloud Storage bucket to house your built code as well as your startup scripts.

- From Cloud Shell, execute the following to create a new Cloud Storage bucket:

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gsutil mb gs://fancy-store-$DEVSHHELL_PROJECT_ID
Creating gs://fancy-store-qwiklabs-gcp-02-0ef921607e5c/...
```

Task 3. Clone source repository

Use the existing Fancy Store ecommerce website based on the monolith-to-microservices repository as the basis for your website.

Clone the source code so you can focus on the aspects of deploying to Compute Engine. Later on in this lab, you will perform a small update to the code to demonstrate the simplicity of updating on Compute Engine.

1. Clone the source code and then navigate to the monolith-to-microservices directory:

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

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ git clone https://github.com/googlecodelabs/monolith-to-microservices.git
Cloning into 'monolith-to-microservices'...
remote: Enumerating objects: 1226, done.
remote: Counting objects: 100% (268/268), done.
remote: Compressing objects: 100% (144/144), done.
remote: Total 1226 (delta 211), reused 137 (delta 119), pack-reused 958 (from 4)
Receiving objects: 100% (1226/1226), 4.37 MiB | 15.92 MiB/s, done.
Resolving deltas: 100% (590/590), done.
```

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

2. Run the initial build of the code to allow the application to run locally:

```
./setup.sh
```

```

student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices (qwiklabs-gcp-02-0ef921607e5c)$ ./setup.sh
Installing monolith dependencies...

added 68 packages, and audited 69 packages in 2s

14 packages are looking for funding
  run `npm fund` for details

5 vulnerabilities (3 low, 2 high)

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
Completed.

Installing microservices dependencies...

```

- Once completed, ensure Cloud Shell is running a compatible nodeJS version with the following command:

```

nvm install --lts

```

```

student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices (qwiklabs-gcp-02-0ef921607e5c)$ nvm install --lts
Installing latest LTS version.
v22.15.0 is already installed.
Now using node v22.15.0 (npm v11.3.0)

```

- Next, run the following to test the application, switch to the microservices directory, and start the web server:

```

cd microservices

```

```

npm start

```

```

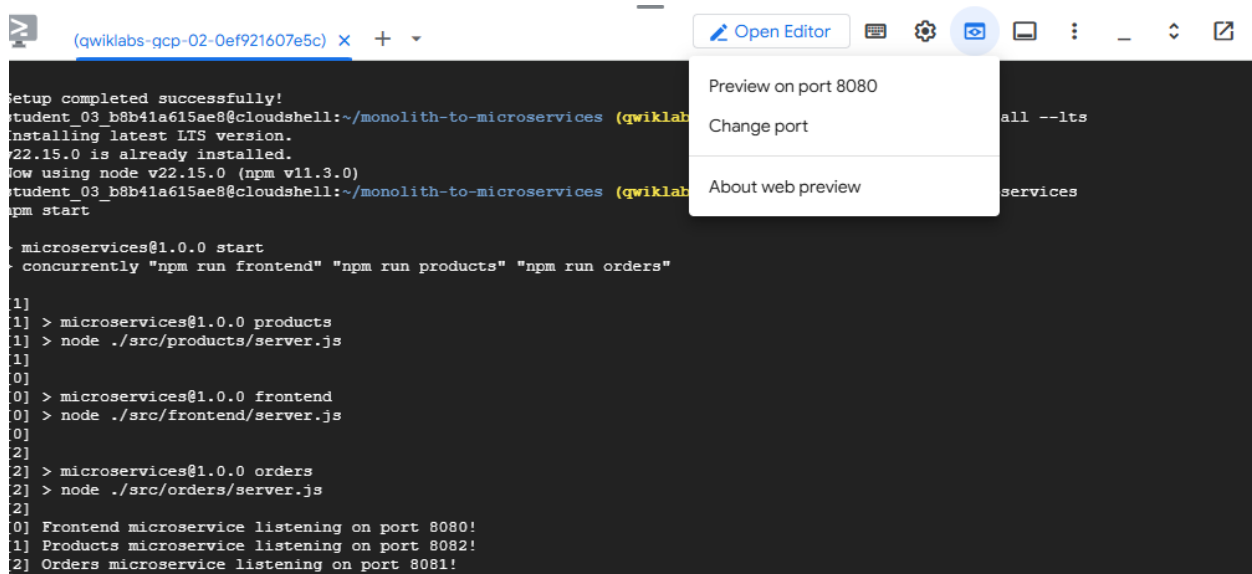
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices (qwiklabs-gcp-02-0ef921607e5c)$ cd microservices
npm start

> microservices@1.0.0 start
> concurrently "npm run frontend" "npm run products" "npm run orders"

[1]
[1] > microservices@1.0.0 products
[1] > node ./src/products/server.js
[1]
[0]
[0] > microservices@1.0.0 frontend
[0] > node ./src/frontend/server.js
[0]
[2]
[2] > microservices@1.0.0 orders
[2] > node ./src/orders/server.js
[2]
[0] Frontend microservice listening on port 8080!
[1] Products microservice listening on port 8082!
[2] Orders microservice listening on port 8081!

```

- Preview your application by clicking the **web preview icon** then selecting **Preview on port 8080**.



```
Setup completed successfully!
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices (qwiklab)
Installing latest LTS version.
v22.15.0 is already installed.
Now using node v22.15.0 (npm v11.3.0)
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices (qwiklab)
npm start

microservices@1.0.0 start
concurrently "npm run frontend" "npm run products" "npm run orders"

[1]
[1] > microservices@1.0.0 products
[1] > node ./src/products/server.js
[1]
[0]
[0] > microservices@1.0.0 frontend
[0] > node ./src/frontend/server.js
[0]
[2]
[2] > microservices@1.0.0 orders
[2] > node ./src/orders/server.js
[2]
[0] Frontend microservice listening on port 8080!
[1] Products microservice listening on port 8082!
[2] Orders microservice listening on port 8081!
```

6. Close this window after viewing the website and then press CTRL+C in the terminal window to stop the web server process.

Task 4. Create Compute Engine instances

Now it's time to start deploying some Compute Engine instances!

In the following steps you will:

1. Create a startup script to configure instances.
2. Clone source code and upload to Cloud Storage.
3. Deploy a Compute Engine instance to host the backend microservices.
4. Reconfigure the frontend code to utilize the backend microservices instance.
5. Deploy a Compute Engine instance to host the frontend microservice.
6. Configure the network to allow communication.

Create the startup script

A startup script will be used to instruct the instance what to do each time it is started. This way the instances are automatically configured.

1. In Cloud Shell, run the following command to create a file called startup-script.sh:

```
touch ~/monolith-to-microservices/startup-script.sh
```

2. Click **Open Editor** in the Cloud Shell ribbon to open the Code Editor.

3. Navigate to the monolith-to-microservices folder.
4. Add the following code to the startup-script.sh file. You will edit some of the code after it's added:

Find the text [DEVSHHELL_PROJECT_ID] in the file and replace it with your Project ID: Project ID

6. **Save** the startup-script.sh file, but do not close it yet.
7. Look at the bottom right of Cloud Shell Code Editor, and ensure "End of Line Sequence" is set to "LF" and not "CRLF".
8. Close the startup-script.sh file.
9. Return to Cloud Shell Terminal and run the following to copy the startup-script.sh file into your bucket:

```
gsutil cp ~/monolith-to-microservices/startup-script.sh gs://fancy-store-  
$DEVSHHELL_PROJECT_ID
```

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/microservices (qwiklabs-gcp-02-0ef921607e5c)$ gsutil cp ~/monolith-to-microservices/startup-script.sh gs://fancy-store-$DEVSHHELL_PROJECT_ID  
Copying file:///home/student_03_b8b41a615ae8/monolith-to-microservices/startup-script.sh [Content-Type=text/x-sh]...  
/ [1 files] [ 1.3 KiB/ 1.3 KiB]  
Operation completed over 1 objects/1.3 KiB.
```

It will now be accessible at: [https://storage.googleapis.com/\[BUCKET_NAME\]/startup-script.sh](https://storage.googleapis.com/[BUCKET_NAME]/startup-script.sh).

[BUCKET_NAME] represents the name of the Cloud Storage bucket. This will only be viewable by authorized users and service accounts by default, therefore inaccessible through a web browser. Compute Engine instances will automatically be able to access this through their service account.

The startup script performs the following tasks:

- Installs the Logging agent. The agent automatically collects logs from syslog.
- Installs Node.js and Supervisor. Supervisor runs the app as a daemon.
- Clones the app's source code from Cloud Storage Bucket and installs dependencies.

- Configures Supervisor to run the app. Supervisor makes sure the app is restarted if it exits unexpectedly or is stopped by an admin or process. It also sends the app's stdout and stderr to syslog for the Logging agent to collect.

Copy code into the Cloud Storage bucket

When instances launch, they pull code from the Cloud Storage bucket, so you can store some configuration variables within the .env file of the code.

1. Copy the cloned code into your bucket:

```
cd ~ rm -rf monolith-to-microservices/*/node_modules gsutil -m cp -r monolith-to-microservices gs://fancy-store-$DEVSHHELL_PROJECT_ID/
```

Deploy the backend instance

The first instance to be deployed will be the backend instance which will house the Orders and Products microservices.

- Execute the following command to create an e2-standard-2 instance that is configured to use the startup script. It is tagged as a backend instance so you can apply specific firewall rules to it later:

```
gcloud compute instances create backend
```

```
--zone=$ZONE
```

```
--machine-type=e2-standard-2
```

```
--tags=backend
```

```
--metadata=startup-script-url=https://storage.googleapis.com/fancy-store-\$DEVSHHELL\_PROJECT\_ID/startup-script.sh
```

```
Operation completed over 101 objects/110 MB.
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c) $ gcloud compute instances create backend \
--zone=$ZONE \
--machine-type=e2-standard-2 \
--tags=backend \
--metadata=startup-script-url=https://storage.googleapis.com/fancy-store-$DEVSHHELL_PROJECT_ID/startup-script.sh
Created [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/zones/us-west1-a/instances/backend].
NAME: backend
ZONE: us-west1-a
MACHINE_TYPE: e2-standard-2
PREEMPTIBLE:
INTERNAL_IP: 10.138.0.2
EXTERNAL_IP: 34.168.174.180
STATUS: RUNNING
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c) $
```

Configure a connection to the backend

Before you deploy the frontend of the application, you need to update the configuration to point to the backend you just deployed.

1. Retrieve the external IP address of the backend with the following command, look under the EXTERNAL_IP tab for the backend instance:

```
gcloud compute instances list
```

2. **Copy the External IP** for the backend.
3. In the Cloud Shell Explorer, navigate to monolith-to-microservices > react-app.
4. In the Code Editor, select **View > Toggle Hidden Files** in order to see the .env file.

In the next step, you edit the .env file

5. In the .env file, replace localhost with your [BACKEND_ADDRESS]:

```
REACT_APP_ORDERS_URL=http://[BACKEND_ADDRESS]:8081/api/orders  
REACT_APP_PRODUCTS_URL=http://[BACKEND_ADDRESS]:8082/api/products
```

6. **Save** the file.
7. In Cloud Shell, run the following to rebuild react-app, which will update the frontend code:

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

```
npm install && npm run-script build
```

```

student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ cd ~/monolith-to-microservices/react-app
npm install && npm run-script build
npm warn deprecated source-map-url@0.4.1: See https://github.com/lydell/source-map-url#deprecated
npm warn deprecated svgo@1.3.2: This SVGO version is no longer supported. Upgrade to v2.x.x.

added 1481 packages, and audited 1482 packages in 34s

205 packages are looking for funding
  run `npm fund` for details

28 vulnerabilities (1 low, 8 moderate, 18 high, 1 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

> 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

```

8. Then copy the application code into the Cloud Storage bucket:

```
cd ~ && rm -rf monolith-to-microservices/* && node_modules/gsutil -m cp -r monolith-to-microservices gs://fancy-store-$DEVSHHELL_PROJECT_ID/
```

```

student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/react-app (qwiklabs-gcp-02-0ef921607e5c)$ cd ~
rm -rf monolith-to-microservices/* && node_modules/gsutil -m cp -r monolith-to-microservices gs://fancy-store-$DEVSHHELL_PROJECT_ID/
Copying file://monolith-to-microservices/CONTRIBUTING.md [Content-Type=text/markdown]...
Copying file://monolith-to-microservices/README.md [Content-Type=text/markdown]...
Copying file://monolith-to-microservices/setup.sh [Content-Type=text/x-sh]...
Copying file://monolith-to-microservices/.gitignore [Content-Type=application/octet-stream]...
Copying file://monolith-to-microservices/package-lock.json [Content-Type=application/json]...
Copying file://monolith-to-microservices/deploy-monolith.sh [Content-Type=text/x-sh]...
Copying file://monolith-to-microservices/startup-script.sh [Content-Type=text/x-sh]...
Copying file://monolith-to-microservices/monolith/.gitignore [Content-Type=application/octet-stream]...
Copying file://monolith-to-microservices/monolith/package.json [Content-Type=application/json]...
Copying file://monolith-to-microservices/LICENSE [Content-Type=application/octet-stream]...
Copying file://monolith-to-microservices/monolith/package-lock.json [Content-Type=application/json]...
Copying file://monolith-to-microservices/monolith/.gcloudignore [Content-Type=application/octet-stream]...
Copying file://monolith-to-microservices/monolith/Dockerfile [Content-Type=application/octet-stream]...
Copying file://monolith-to-microservices/monolith/.dockerignore [Content-Type=application/octet-stream]...
Copying file://monolith-to-microservices/monolith/data/orders.json [Content-Type=application/json]...
Copying file://monolith-to-microservices/monolith/data/products.json [Content-Type=application/json]...
Copying file://monolith-to-microservices/monolith/k8s/service.yml [Content-Type=application/yaml]...
Copying file://monolith-to-microservices/monolith/public/index.html [Content-Type=text/html]...
Copying file://monolith-to-microservices/monolith/k8s/deployment.yml [Content-Type=application/yaml]...
Copying file://monolith-to-microservices/monolith/public/asset-manifest.json [Content-Type=application/json]...
Copying file://monolith-to-microservices/monolith/public/manifest.json [Content-Type=application/json]...
Copying file://monolith-to-microservices/monolith/public/robots.txt [Content-Type=text/plain]...
Copying file://monolith-to-microservices/monolith/public/static/img/products/record-player.jpg [Content-Type=image/jpeg]...
Copying file://monolith-to-microservices/monolith/public/static/img/products/film-camera.jpg [Content-Type=image/jpeg]...
Copying file://monolith-to-microservices/monolith/public/static/img/products/air-plant.jpg [Content-Type=image/jpeg]...
Copying file://monolith-to-microservices/monolith/public/static/img/products/credits.txt [Content-Type=text/plain]...
Copying file://monolith-to-microservices/monolith/public/static/img/products/barista-kit.jpg [Content-Type=image/jpeg]...
Copying file://monolith-to-microservices/monolith/public/static/img/products/typewriter.jpg [Content-Type=image/jpeg]...
Copying file://monolith-to-microservices/monolith/public/static/img/products/terrarium.jpg [Content-Type=image/jpeg]...
Copying file://monolith-to-microservices/monolith/public/static/img/products/camp-mug.jpg [Content-Type=image/jpeg]...
Copying file://monolith-to-microservices/monolith/public/static/img/products/camera-lens.jpg [Content-Type=image/jpeg]...

```

Deploy the frontend instance

Now that the code is configured, deploy the frontend instance.

- Execute the following to deploy the frontend instance with a similar command as before. This instance is tagged as frontend for firewall purposes:

gcloud compute instances create frontend

--zone=\$ZONE

--machine-type=e2-standard-2

--tags=frontend

--metadata=startup-script-url=[https://storage.googleapis.com/fancy-store-\\$DEVSHHELL_PROJECT_ID/startup-script.sh](https://storage.googleapis.com/fancy-store-$DEVSHHELL_PROJECT_ID/startup-script.sh)

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gcloud compute instances create frontend \
  --zone=$ZONE \
  --machine-type=e2-standard-2 \
  --tags=frontend \
  --metadata=startup-script-url=https://storage.googleapis.com/fancy-store-$DEVSHHELL_PROJECT_ID/startup-script.sh
Created [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/zones/us-west1-a/instances/frontend].
NAME: frontend
ZONE: us-west1-a
MACHINE TYPE: e2-standard-2
PREEMPTIBLE:
INTERNAL IP: 10.138.0.3
EXTERNAL IP: 34.53.48.36
STATUS: RUNNING
```

Configure the network

1. Create firewall rules to allow access to port 8080 for the frontend, and ports 8081-8082 for the backend. These firewall commands use the tags assigned during instance creation for application:

gcloud compute firewall-rules create fw-fe \

--allow tcp:8080 \

--target-tags=frontend

gcloud compute firewall-rules create fw-be \

--allow tcp:8081-8082 \

--target-tags=backend

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gcloud compute firewall-rules create fw-fe \
  --allow tcp:8080 \
  --target-tags=frontend
Creating firewall...working..Created [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/global/firewalls/fw-fe].
Creating firewall...done.
NAME: fw-fe
NETWORK: default
DIRECTION: INGRESS
PRIORITY: 1000
ALLOW: tcp:8080
DENY:
DISABLED: False
```



```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gcloud compute firewall-rules create fw-be \
--allow tcp:8081-8082 \
--target-tags=backend
Creating firewall...working..Created [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/global/firewalls/fw-be].
Creating firewall...done.
NAME: fw-be
NETWORK: default
DIRECTION: INGRESS
PRIORITY: 1000
ALLOW: tcp:8081-8082
DENY:
DISABLED: False
```

The website should now be fully functional.

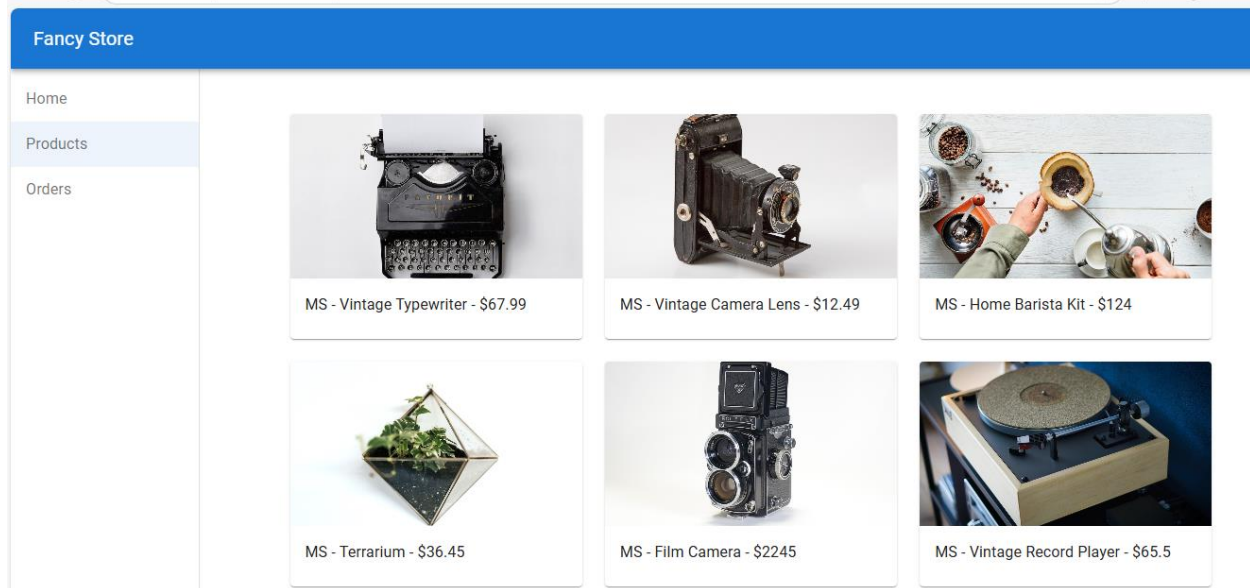
2. In order to navigate to the external IP of the frontend, you need to know the address. Run the following and look for the EXTERNAL_IP of the frontend instance:

gcloud compute instances list

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gcloud compute instances list
NAME: backend
ZONE: us-west1-a
MACHINE_TYPE: e2-standard-2
PREEMPTIBLE:
INTERNAL_IP: 10.138.0.2
EXTERNAL_IP: 34.168.174.180
STATUS: RUNNING

NAME: frontend
ZONE: us-west1-a
MACHINE_TYPE: e2-standard-2
PREEMPTIBLE:
INTERNAL_IP: 10.138.0.3
EXTERNAL_IP: 34.53.48.36
STATUS: RUNNING
```

3. Wait 3 minutes and then open a new browser tab and browse to [http://\[FRONTEND_ADDRESS\]:8080](http://[FRONTEND_ADDRESS]:8080) to access the website, where [FRONTEND_ADDRESS] is the frontend EXTERNAL_IP determined above.
4. Try navigating to the **Products** and **Orders** pages; these should now work.



Task 5. Create managed instance groups

To allow the application to scale, managed instance groups will be created and will use the frontend and backend instances as Instance Templates.

A managed instance group (MIG) contains identical instances that you can manage as a single entity in a single zone. Managed instance groups maintain high availability of your apps by proactively keeping your instances available, that is, in the RUNNING state. You will be using managed instance groups for your frontend and backend instances to provide autohealing, load balancing, autoscaling, and rolling updates.

Create instance template from source instance

Before you can create a managed instance group, you have to first create an instance template that will be the foundation for the group. Instance templates allow you to define the machine type, boot disk image or container image, network, and other instance properties to use when creating new VM instances. You can use instance templates to create instances in a managed instance group or even to create individual instances.

To create the instance template, use the existing instances you created previously.

1. First, stop both instances:

```
gcloud compute instances stop frontend --zone=$ZONE
```

```
gcloud compute instances stop backend --zone=$ZONE
```

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gcloud compute instances stop frontend --zone=$ZONE
Stopping instance(s) frontend...done.
Updated [https://compute.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/zones/us-west1-a/instances/frontend].

student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gcloud compute instances stop backend --zone=$ZONE
Stopping instance(s) backend...done.
Updated [https://compute.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/zones/us-west1-a/instances/backend].
```

2. Then, create the instance template from each of the source instances:

gcloud compute instance-templates create fancy-fe \

--source-instance-zone=\$ZONE \

--source-instance=frontend

gcloud compute instance-templates create fancy-be \

--source-instance-zone=\$ZONE \

--source-instance=backend

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gcloud compute instance-templates create fancy-fe \
--source-instance-zone=$ZONE \
--source-instance=frontend
Created [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/global/instanceTemplates/fancy-fe].
NAME: fancy-fe
MACHINE_TYPE: e2-standard-2
PREEMPTIBLE:
CREATION_TIMESTAMP: 2025-05-05T04:20:52.908-07:00
```

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gcloud compute instance-templates create fancy-be \
--source-instance-zone=$ZONE \
--source-instance=backend
Created [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/global/instanceTemplates/fancy-be].
NAME: fancy-be
MACHINE_TYPE: e2-standard-2
PREEMPTIBLE:
CREATION_TIMESTAMP: 2025-05-05T04:21:17.140-07:00
```

Confirm the instance templates were created

gcloud compute instance-templates list

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gcloud compute instance-templates list
NAME: fancy-be
MACHINE_TYPE: e2-standard-2
PREEMPTIBLE:
CREATION_TIMESTAMP: 2025-05-05T04:21:17.140-07:00

NAME: fancy-fe
MACHINE_TYPE: e2-standard-2
PREEMPTIBLE:
CREATION_TIMESTAMP: 2025-05-05T04:20:52.908-07:00
```

4. With the instance templates created, delete the backend vm to save resource space:

gcloud compute instances delete backend --zone=\$ZONE

```
student_03 b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c) $ gcloud compute instances delete backend --zone=$ZONE
The following instances will be deleted. Any attached disks configured to be auto-deleted will be deleted unless they are attached
to any other instances or the '--keep-disks' flag is given and specifies them for keeping. Deleting a disk is irreversible and any
data on the disk will be lost.
- [backend] in [us-west1-a]

Do you want to continue (Y/n)? y

Deleted [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/zones/us-west1-a/instances/backend].
```

5. Type and enter **y** when prompted.

Normally, you could delete the frontend vm as well, but you will use it to update the instance template later in the lab.

Create managed instance group

Next, create two managed instance groups, one for the frontend and one for the backend:

```
gcloud compute instance-groups managed create fancy-fe-mig \
```

```
--zone=$ZONE \
```

```
--base-instance-name fancy-fe \
```

```
--size 2 \
```

```
--template fancy-fe
```

```
student_03 b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c) $ gcloud compute instance-groups managed create fancy-fe-mig \
--zone=$ZONE \
--base-instance-name fancy-fe \
--size 2 \
--template fancy-fe
Created [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/zones/us-west1-a/instanceGroupManagers/fancy-f
e-mig].
NAME: fancy-fe-mig
LOCATION: us-west1-a
SCOPE: zone
BASE_INSTANCE_NAME: fancy-fe
SIZE: 0
TARGET_SIZE: 2
INSTANCE_TEMPLATE: fancy-fe
AUTOSCALED: no
```

```
gcloud compute instance-groups managed create fancy-be-mig \
```

```
--zone=$ZONE \
```

```
--base-instance-name fancy-be \
```

```
--size 2 \
```

```
--template fancy-be
```

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gcloud compute instance-groups managed create fancy-be-mig \
--zone=$ZONE \
--base-instance-name fancy-be \
--size 2 \
--template fancy-be
Created [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/zones/us-west1-a/instanceGroupManagers/fancy-be-mig].
NAME: fancy-be-mig
LOCATION: us-west1-a
SCOPE: zone
BASE_INSTANCE_NAME: fancy-be
SIZE: 0
TARGET_SIZE: 2
INSTANCE_TEMPLATE: fancy-be
AUTOSCALED: no
```

These managed instance groups will use the instance templates and are configured for two instances each within each group to start. The instances are automatically named based on the base-instance-name specified with random characters appended.

2. For your application, the frontend microservice runs on port 8080, and the backend microservice runs on port 8081 for orders and port 8082 for products:

```
gcloud compute instance-groups set-named-ports fancy-fe-mig \
```

```
--zone=$ZONE \
```

```
--named-ports frontend:8080
```

```
gcloud compute instance-groups set-named-ports fancy-be-mig \
```

```
--zone=$ZONE \
```

```
--named-ports orders:8081,products:8082
```

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gcloud compute instance-groups set-named-ports fancy-fe-mig \
--zone=$ZONE \
--named-ports frontend:8080
Updated [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/zones/us-west1-a/instanceGroups/fancy-fe-mig].
```

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gcloud compute instance-groups set-named-ports fancy-be-mig \
--zone=$ZONE \
--named-ports orders:8081,products:8082
Updated [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/zones/us-west1-a/instanceGroups/fancy-be-mig].
```

Since these are non-standard ports, you specify named ports to identify these. Named ports are key:value pair metadata representing the service name and the port that it's running on. Named ports can be assigned to an instance group, which indicates that the service is available on all instances in the group. This information is used by the HTTP Load Balancing service that will be configured later.

Configure autohealing

To improve the availability of the application itself and to verify it is responding, configure an autohealing policy for the managed instance groups.

An autohealing policy relies on an application-based health check to verify that an app is responding as expected. Checking that an app responds is more precise than simply verifying that an instance is in a RUNNING state, which is the default behavior.

1. Create a health check that repairs the instance if it returns "unhealthy" 3 consecutive times for the frontend and backend:

```
gcloud compute health-checks create http fancy-fe-hc \
```

```
--port 8080 \
```

```
--check-interval 30s \
```

```
--healthy-threshold 1 \
```

```
--timeout 10s \
```

```
--unhealthy-threshold 3
```

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gcloud compute health-checks create http fancy-fe-hc \
--port 8080 \
--check-interval 30s \
--healthy-threshold 1 \
--timeout 10s \
--unhealthy-threshold 3
Created [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/global/healthChecks/fancy-fe-hc].
NAME: fancy-fe-hc
PROTOCOL: HTTP
```

```
gcloud compute health-checks create http fancy-be-hc \
```

```
--port 8081 \
```

```
--request-path=/api/orders \
```

```
--check-interval 30s \
```

```
--healthy-threshold 1 \
```

```
--timeout 10s \
```

```
--unhealthy-threshold 3
```

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gcloud compute health-checks create http fancy-be-hc \
--port 8081 \
--request-path=/api/orders \
--check-interval 30s \
--healthy-threshold 1 \
--timeout 10s \
--unhealthy-threshold 3
Created [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/global/healthChecks/fancy-be-hc].
NAME: fancy-be-hc
PROTOCOL: HTTP
```

2. Create a firewall rule to allow the health check probes to connect to the microservices on ports 8080-8081:

```
gcloud compute firewall-rules create allow-health-check \
```

```
--allow tcp:8080-8081 \

--source-ranges 130.211.0.0/22,35.191.0.0/16 \

--network default
```

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gcloud compute firewall-rules create allow-health-check \
--allow tcp:8080-8081 \
--source-ranges 130.211.0.0/22,35.191.0.0/16 \
--network default
Creating firewall...working..Created [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/global/firewalls/allow-health-check].
Creating firewall...done.
NAME: allow-health-check
NETWORK: default
DIRECTION: INGRESS
PRIORITY: 1000
ALLOW: tcp:8080-8081
DENY:
DISABLED: False
```

3. Apply the health checks to their respective services:

```
gcloud compute instance-groups managed update fancy-fe-mig \
```

```
--zone=$ZONE \

--health-check fancy-fe-hc \

--initial-delay 300
```

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c)$ gcloud compute instance-groups managed update fancy-fe-mig \
--zone=$ZONE \
--health-check fancy-fe-hc \
--initial-delay 300
Updated [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-02-0ef921607e5c/zones/us-west1-a/instanceGroupManagers/fancy-fe-mig].
```

```
gcloud compute instance-groups managed update fancy-be-mig \
```

```
--zone=$ZONE \

--health-check fancy-be-hc \

--initial-delay 300
```

4. Continue with the lab to allow some time for autohealing to monitor the instances in the group. You will simulate a failure to test the autohealing at the end of the lab.

Task 6. Create load balancers

To complement your managed instance groups, use HTTP(S) Load Balancers to serve traffic to the frontend and backend microservices, and use mappings to send traffic to the proper backend services based on pathing rules. This exposes a single load balanced IP for all services.

Create HTTP(S) load balancer

Google Cloud offers many different types of load balancers. For this lab you use an HTTP(S) Load Balancer for your traffic. An HTTP load balancer is structured as follows:

1. A forwarding rule directs incoming requests to a target HTTP proxy.
2. The target HTTP proxy checks each request against a URL map to determine the appropriate backend service for the request.
3. The backend service directs each request to an appropriate backend based on serving capacity, zone, and instance health of its attached backends. The health of each backend instance is verified using an HTTP health check. If the backend service is configured to use an HTTPS or HTTP/2 health check, the request will be encrypted on its way to the backend instance.
4. Sessions between the load balancer and the instance can use the HTTP, HTTPS, or HTTP/2 protocol. If you use HTTPS or HTTP/2, each instance in the backend services must have an SSL certificate.
5. Create health checks that will be used to determine which instances are capable of serving traffic for each service:

```
gcloud compute http-health-checks create fancy-fe-frontend-hc \
    --request-path /\
    --port 8080
```

```
gcloud compute http-health-checks create fancy-be-orders-hc \
    --request-path /api/orders \
    --port 8081
```

```
gcloud compute http-health-checks create fancy-be-products-hc \
    --request-path /api/products \
    --port 8082
```

```
gcloud compute backend-services create fancy-be-products \
    --http-health-checks fancy-be-products-hc \
    --port-name products \
    --global
```

3. Add the Load Balancer's [backend services](#):


```
gcloud compute backend-services add-backend fancy-fe-frontend \  
    --instance-group-zone=$ZONE \  
    --instance-group fancy-fe-mig \  
    --global
```

```
gcloud compute backend-services add-backend fancy-be-orders \  
    --instance-group-zone=$ZONE \  
    --instance-group fancy-be-mig \  
    --global
```

```
gcloud compute backend-services add-backend fancy-be-products \  
    --instance-group-zone=$ZONE \  
    --instance-group fancy-be-mig \  
    --global
```

4. Create a URL map. The URL map defines which URLs are directed to which backend services:

```
gcloud compute url-maps create fancy-map \  
    --default-service fancy-fe-frontend
```

5. Create a path matcher to allow the /api/orders and /api/products paths to route to their respective services:

```
gcloud compute url-maps add-path-matcher fancy-map \  
    --default-service fancy-fe-frontend \  
    --path-matcher-name orders \  
    --path-rules "/api/orders=fancy-be-orders,/api/products=fancy-be-products"
```

```
gcloud compute url-maps add-path-matcher fancy-map \  
    --default-service fancy-fe-frontend \  
    --path-matcher-name orders \  
    --path-rules "/api/orders=fancy-be-orders,/api/products=fancy-be-products"
```

6. Create the proxy which ties to the URL map:

```
gcloud compute target-http-proxies create fancy-proxy \
--url-map fancy-map
```

7. Create a global forwarding rule that ties a public IP address and port to the proxy:

```
gcloud compute forwarding-rules create fancy-http-rule \
--global \
--target-http-proxy fancy-proxy \
--ports 80
```

Update the configuration

Now that you have a new static IP address, update the code on the frontend to point to this new address instead of the ephemeral address used earlier that pointed to the backend instance.

1. In Cloud Shell, change to the react-app folder which houses the .env file that holds the configuration:

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

```
student_03_b8b41a615ae8@cloudshell:~/monolith-to-microservices/react-app (qwklabs-gcp-02-0ef921607e5c)$ cd ~/monolith-to-microservices/react-app
npm install && npm run-script build
npm warn deprecated source-map-url@0.4.1: See https://github.com/lydell/source-map-url#deprecated
npm warn deprecated svgo@1.3.2: This SVGO version is no longer supported. Upgrade to v2.x.x.
added 1481 packages, and audited 1482 packages in 31s
205 packages are looking for funding
  run `npm fund` for details
28 vulnerabilities (1 low, 8 moderate, 18 high, 1 critical)
To address issues that do not require attention, run:
  npm audit fix
To address all issues (including breaking changes), run:
  npm audit fix --force
Run `npm audit` for details.
> 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. Find the IP address for the Load Balancer:

```
gcloud compute forwarding-rules list --global
```

```
student_03_b8b41a615ae8@cloudshell:~ (qwklabs-gcp-02-0ef921607e5c)$ gcloud compute backend-services update fancy-fe-frontend \
--enable-cdn --global
Updated [https://www.googleapis.com/compute/v1/projects/qwklabs-gcp-02-0ef921607e5c/global/backendServices/fancy-fe-frontend].
student_03_b8b41a615ae8@cloudshell:~ (qwklabs-gcp-02-0ef921607e5c)$
```

3. Return to the Cloud Shell Editor and edit the .env file again to point to Public IP of Load Balancer. [LB_IP] represents the External IP address of the backend instance determined above.

```
REACT_APP_ORDERS_URL=http://[LB_IP]/api/orders  
REACT_APP_PRODUCTS_URL=http://[LB_IP]/api/products
```

4. **Save** the file.
5. Rebuild react-app, which will update the frontend code

```
cd ~/monolith-to-microservices/react-app  
npm install && npm run-script build
```

6. Copy the application code into your bucket:

```
cd ~ rm -rf monolith-to-microservices/*/node_modules gsutil -m cp -r monolith-to-microservices gs://fancy-store-$DEVSHHELL_PROJECT_ID/
```

Update the frontend instances

Now that there is new code and configuration, you want the frontend instances within the managed instance group to pull the new code.

Since your instances pull the code at startup, you can issue a rolling restart command:

Test the website

1. Wait 3 minutes after issuing the rolling-action replace command in order to give the instances time to be processed, and then check the status of the managed instance group. Run the following to confirm the service is listed as **HEALTHY**:

```
watch -n 2 gcloud compute backend-services get-health fancy-fe-frontend --global
```
2. Wait until the 2 services are listed as **HEALTHY**.
3. Once both items appear as HEALTHY on the list, exit the watch command by pressing CTRL+C.

Task 7. Scaling Compute Engine

So far, you have created two managed instance groups with two instances each. This configuration is fully functional, but a static configuration regardless of load. Next, you create an autoscaling policy based on utilization to automatically scale each managed instance group.

Automatically resize by utilization

- To create the autoscaling policy, execute the following:

```
gcloud compute instance-groups managed set-autoscaling \
  fancy-fe-mig \
  --zone=$ZONE \
  --max-num-replicas 2 \
  --target-load-balancing-utilization 0.60

gcloud compute instance-groups managed set-autoscaling \
  fancy-be-mig \
  --zone=$ZONE \
  --max-num-replicas 2 \
  --target-load-balancing-utilization 0.60
```

These commands create an autoscaler on the managed instance groups that automatically adds instances when utilization is above 60% utilization, and removes instances when the load balancer is below 60% utilization.

Enable content delivery network

Another feature that can help with scaling is to enable a Content Delivery Network service, to provide caching for the frontend.

- Execute the following command on the frontend service:

```
gcloud compute backend-services update fancy-fe-frontend \
  --enable-cdn --global
```

When a user requests content from the HTTP(S) load balancer, the request arrives at a Google Front End (GFE) which first looks in the Cloud CDN cache for a response to the user's request. If the GFE finds a cached response, the GFE sends the cached response to the user. This is called a cache hit.

If the GFE can't find a cached response for the request, the GFE makes a request directly to the backend. If the response to this request is cacheable, the GFE stores the response in the Cloud CDN cache so that the cache can be used for subsequent requests.

Task 8. Update the website

Updating instance template

Existing instance templates are not editable; however, since your instances are stateless and all configuration is done through the startup script, you only need to change the instance template if you want to change the template settings . Now you're going to make a simple change to use a larger machine type and push that out.

Complete the following steps to:

- Update the frontend instance, which acts as the basis for the instance template. During the update, put a file on the updated version of the instance template's image, then update the instance template, roll out the new template, and then confirm the file exists on the managed instance group instances.
- Modify the machine type of your instance template, by switching from the e2-standard-2 machine type to e2-small.
- Run the following command to modify the machine type of the frontend instance:

```
gcloud compute instances set-machine-type frontend \  
  --zone=$ZONE \  
  --machine-type e2-small
```

2. Create the new Instance Template:

```
gcloud compute instance-templates create fancy-fe-new \  
  --region=$REGION \  
  --source-instance=frontend \  
  --source-instance-zone=$ZONE
```

3. Roll out the updated instance template to the Managed Instance Group:

```
gcloud compute instance-groups managed rolling-action start-update fancy-fe-mig \  
  --zone=$ZONE \  
  --version template=fancy-fe-new
```

4. Wait 3 minutes, and then run the following to monitor the status of the update:

```
watch -n 2 gcloud compute instance-groups managed list-instances fancy-fe-mig \
--zone=$ZONE
```

This will take a few moments.

Once you have at least 1 instance in the following condition:

- STATUS: **RUNNING**
 - ACTION set to **None**
 - INSTANCE_TEMPLATE: the new template name (**fancy-fe-new**)
5. **Copy** the name of one of the machines listed for use in the next command.
 6. CTRL+C to exit the watch process.
 7. Run the following to see if the virtual machine is using the new machine type (e2-small), where [VM_NAME] is the newly created instance:

```
gcloud compute instances describe [VM_NAME] --zone=$ZONE | grep machineType
```

Expected example output:

machineType:<https://www.googleapis.com/compute/v1/projects/project-name/zones/us-central1-f/machineTypes/e2-small>

Make changes to the website

Scenario: Your marketing team has asked you to change the homepage for your site. They think it should be more informative of who your company is and what you actually sell.

Task: Add some text to the homepage to make the marketing team happy! It looks like one of the developers has already created the changes with the file name index.js.new. You can just copy this file to index.js and the changes should be reflected. Follow the instructions below to make the appropriate changes.

1. Run the following commands to copy the updated file to the correct file name:

```
cd ~/monolith-to-microservices/react-app/src/pages/Home
mv index.js.new index.js
```

2. Print the file contents to verify the changes:

```
cat ~/monolith-to-microservices/react-app/src/pages/Home/index.js
```

You updated the React components, but you need to build the React app to generate the static files.

3. Run the following command to build the React app and copy it into the monolith public directory:

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

```
npm install && npm run-script build
```

4. Then re-push this code to the bucket:

```
cd ~ rm -rf monolith-to-microservices/*/node_modules gsutil -m cp -r monolith-to-microservices gs://fancy-store-$DEVSHHELL_PROJECT_ID/
```

Push changes with rolling replacements

1. Now force all instances to be replaced to pull the update:

```
gcloud compute instance-groups managed rolling-action replace fancy-fe-mig \
```

```
--zone=$ZONE \
```

```
--max-unavailable=100%
```

2. Wait 3 minutes after issuing the rolling-action replace command in order to give the instances time to be processed, and then check the status of the managed instance group. Run the following to confirm the service is listed as **HEALTHY**:

```
watch -n 2 gcloud compute backend-services get-health fancy-fe-frontend --global
```

3. Wait a few moments for both services to appear and become HEALTHY.

```
student_03_b8b41a615ae8@cloudshell:~ (qwiklabs-gcp-02-0ef921607e5c) $ gcloud compute instance-groups list-instances fancy-fe-mig --zone=$ZONE
NAME: fancy-fe-d2zj
ZONE: us-west1-a
STATUS: RUNNING

NAME: fancy-fe-h6xn
ZONE: us-west1-a
STATUS: RUNNING
```

4. Once items appear in the list with HEALTHY status, exit the watch command by pressing CTRL+C.
5. Browse to the website via [http://\[LB_IP\]](http://[LB_IP]) where [LB_IP] is the IP_ADDRESS specified for the Load Balancer, which can be found with the following command:

```
gcloud compute forwarding-rules list --global
```

Simulate failure

In order to confirm the health check works, log in to an instance and stop the services.

1. To find an instance name, execute the following:

```
gcloud compute instance-groups list-instances fancy-fe-mig --zone=$ZONE
```

2. Copy an instance name, then run the following to secure shell into the instance, where `INSTANCE_NAME` is one of the instances from the list:

```
gcloud compute ssh [INSTANCE_NAME] --zone=$ZONE
```

3. Type in "y" to confirm, and press **Enter** twice to not use a password.

4. Within the instance, use `supervisorctl` to stop the application:

```
sudo supervisorctl stop nodeapp; sudo killall node
```

5. Exit the instance:

```
exit
```

Copied!

content_copy

6. Monitor the repair operations:

```
watch -n 2 gcloud compute operations list \
--filter='operationType~compute.instances.repair.*'
```

Copied!

content_copy

This will take a few minutes to complete.

