

Hosting a Web App on Google Cloud Using Compute Engine

GSP662



Overview

There are many ways to deploy web sites within Google Cloud with each solution offering different features, capabilities, and levels of control. Compute Engine offers a deep level of control over the infrastructure used to run a web site, but also requires a little more operational management compared to solutions like Google Kubernetes Engines (GKE), App Engine, or others. With Compute Engine, you have fine-grained control of aspects of the infrastructure, including the virtual machines, load balancers, and more. In this lab you will deploy a sample application, the "Fancy Store" ecommerce website, to show how a website can be deployed and scaled easily with Compute Engine.

What you'll learn

- How to create [Compute Engine instances](#)
- How to create [instance templates](#) from source instances
- How to create [managed instance groups](#)
- How to create and test [managed instance group health checks](#)
- How to create HTTP(S) [Load Balancers](#)
- How to create [load balancer health checks](#)
- How to use a [Content Delivery Network \(CDN\)](#) for Caching

At the end of the lab, you will have instances inside managed instance groups to provide autohealing, load balancing, autoscaling, and rolling updates for our website.

Environment Setup

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

What you need

To complete this lab, you need:

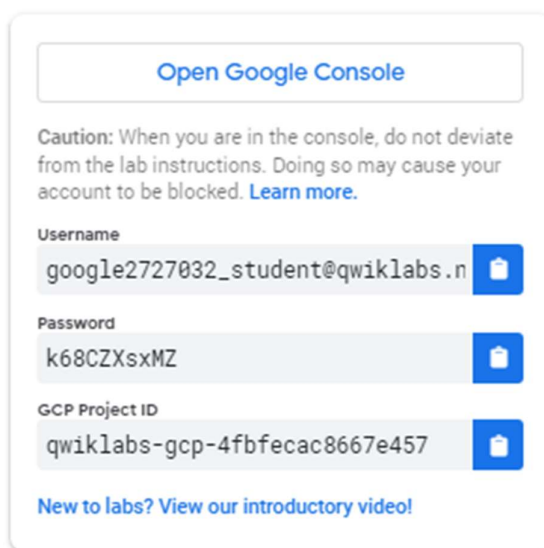
- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.

Note: If you already have your own personal Google Cloud account or project, do not use it for this lab.

Note: If you are using a Pixelbook, open an Incognito window to run this lab.

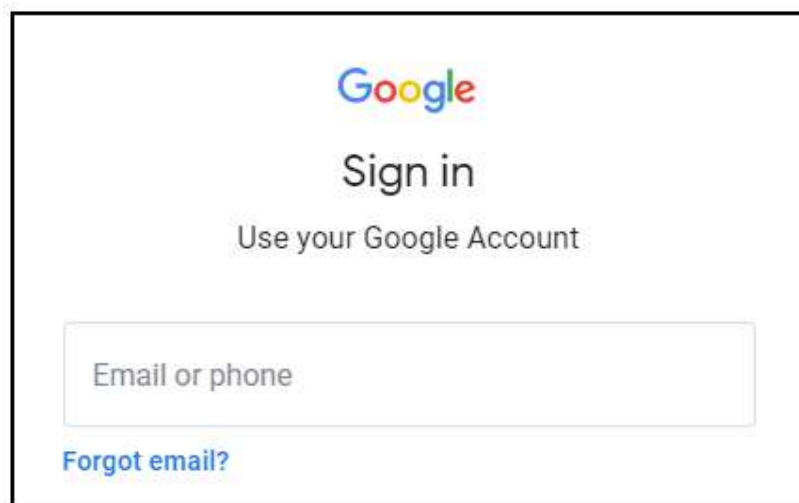
How to start your lab and sign in to the Google Cloud Console

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



A screenshot of a web interface for starting a lab. At the top is a button labeled "Open Google Console". Below it is a caution message: "Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. [Learn more.](#)". There are three input fields, each with a copy icon to its right: "Username" with the value "google2727032_student@qwiklabs.n", "Password" with the value "k68CZXsxMZ", and "GCP Project ID" with the value "qwiklabs-gcp-4fbfecac8667e457". At the bottom is a link: "New to labs? View our introductory video!"

2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



A screenshot of the Google Sign in page. It features the Google logo at the top, followed by the text "Sign in" and "Use your Google Account". Below this is a large input field labeled "Email or phone". At the bottom left of the input field is a link that says "Forgot email?"

Tip: Open the tabs in separate windows, side-by-side.

If you see the **Choose an account** page, click **Use Another**



Account.

3. In the **Sign in** page, paste the username that you copied from the Connection Details panel. Then copy and paste the password.

Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

4. Click through the subsequent pages:

- Accept the terms and conditions.
- Do not add recovery options or two-factor authentication (because this is a temporary account).
- Do not sign up for free trials.

After a few moments, the Cloud Console opens in this tab.

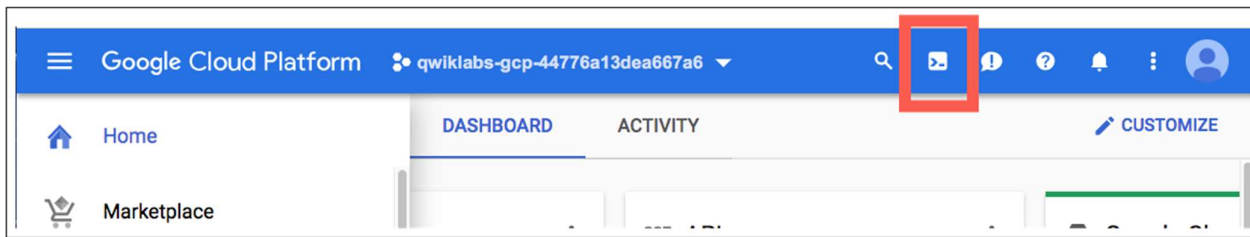
Note: You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.



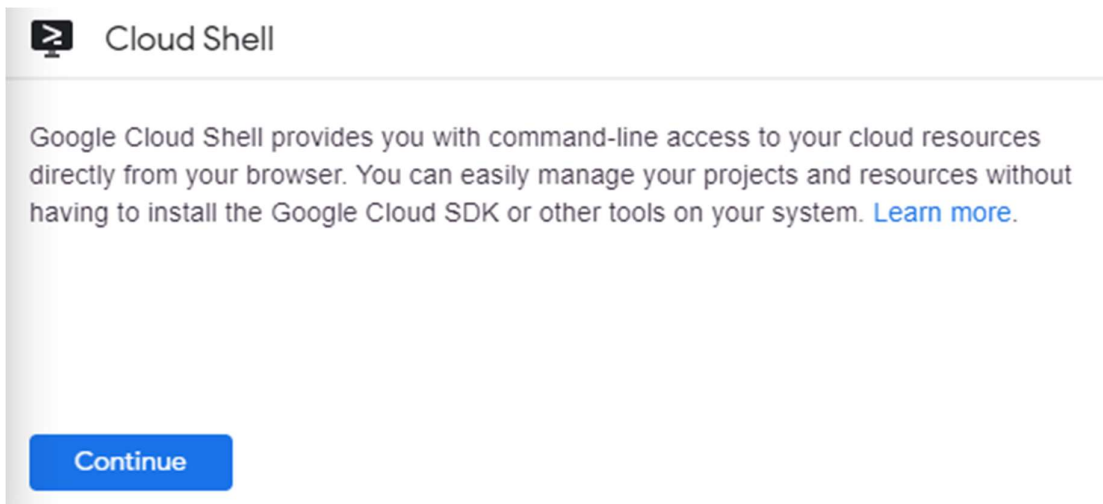
Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

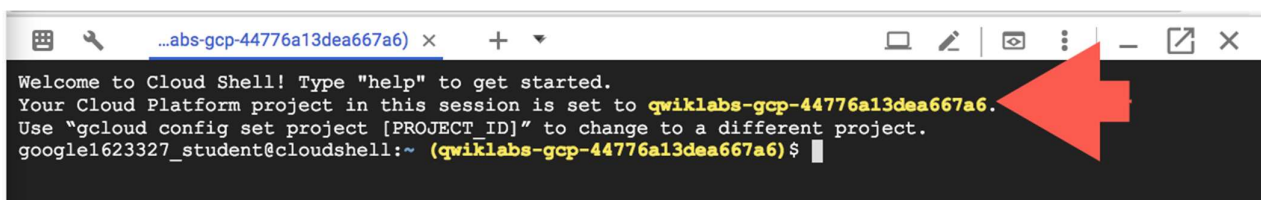
In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



Click **Continue**.



It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT_ID*. For example:



`gcloud` is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

```
gcloud auth list
```

(Output)

```
Credentialed accounts:
- <myaccount>@<mydomain>.com (active)
```

(Example output)

```
Credentialed accounts:  
- google1623327 student@gwiklabs.net
```

You can list the project ID with this command:

```
gcloud config list project
```

(Output)

```
[core]  
project = <project_ID>
```

(Example output)

```
[core]  
project = qwiklabs-gcp-44776a13dea667a6
```

For full documentation of `gcloud` see the [gcloud command-line tool overview](#).

Set the default zone and project configuration:

```
gcloud config set compute/zone us-central1-f
```

Learn more in the [Regions & Zones documentation](#).

When you run `gcloud` on your own machine, the config settings are persisted across sessions. But in Cloud Shell, you need to set this for every new session or reconnection.

Enable Compute Engine API

Next, enable the [Compute Engine API](#). Execute the following to enable the Compute Engine API:

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

Create Cloud Storage bucket

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

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

```
gsutil mb gs://fancy-store-$DEVSHELL_PROJECT_ID
```

Use of the \$DEVSHELL_PROJECT_ID environment variable within Cloud Shell is to help ensure the names of objects are unique. Since all Project IDs within Google Cloud must be unique, appending the Project ID should make other names unique as well.

Click *Check my progress* to verify the objective.

Create Cloud Storage bucket

Check my progress

Clone source repository

You will be using the existing Fancy Store ecommerce website based on the `monolith-to-microservices` repository as the basis for your website. You will 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.

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

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

```
./setup.sh
```

It will take a few minutes for this script to finish.

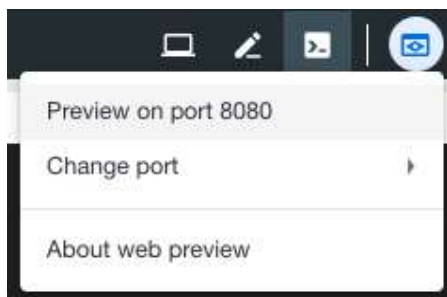
Once completed, run the following to test the application, switch to the `microservices` directory, and start the web server:

```
cd microservices  
npm start
```

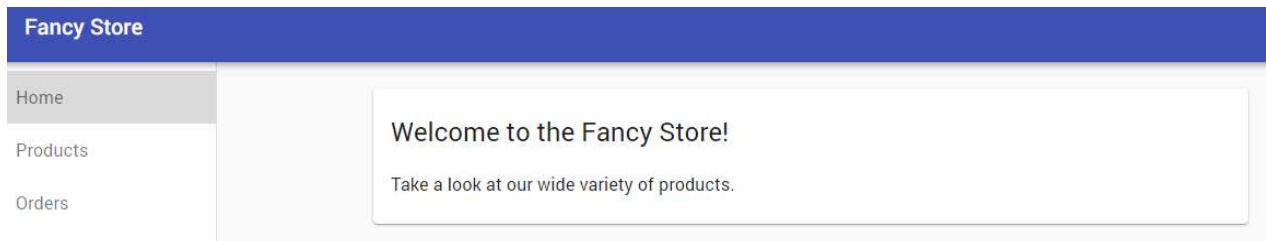
You should see the following output:

```
Products microservice listening on port 8082!  
Frontend microservice listening on port 8080!  
Orders microservice listening on port 8081!
```

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



This will open a new window where you can see the frontend of Fancy Store in action!



Within the Preview option, you will be able to see the Frontend; however, the Products and Orders functions will not work, as those services are not yet exposed.

You can close this window after viewing the website and to stop the web server process, press `CTRL+C` in the terminal window.

Create Compute Engine instances

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

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 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.

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



Navigate to the `monolith-to-microservices` folder.

Click on **File > New File** and create a file called `startup-script.sh`



Add the following code to the file. You will edit some of the code after it's added:

```
#!/bin/bash

# Install logging monitor. The monitor will automatically pick up logs sent to
# syslog.
curl -s "https://storage.googleapis.com/signals-agents/logging/google-fluentd-
install.sh" | bash
service google-fluentd restart &

# Install dependencies from apt
apt-get update
apt-get install -yq ca-certificates git build-essential supervisor psmisc

# Install nodejs
mkdir /opt/nodejs
curl https://nodejs.org/dist/v8.12.0/node-v8.12.0-linux-x64.tar.gz | tar xvzf - -C
/opt/nodejs --strip-components=1
ln -s /opt/nodejs/bin/node /usr/bin/node
ln -s /opt/nodejs/bin/npm /usr/bin/npm

# Get the application source code from the Google Cloud Storage bucket.
mkdir /fancy-store
gsutil -m cp -r gs://fancy-store-[DEVSHHELL_PROJECT_ID]/monolith-to-
microservices/microservices/* /fancy-store/

# Install app dependencies.
cd /fancy-store/
npm install

# Create a nodeapp user. The application will run as this user.
useradd -m -d /home/nodeapp nodeapp
chown -R nodeapp:nodeapp /opt/app

# Configure supervisor to run the node app.
cat >/etc/supervisor/conf.d/node-app.conf << EOF
[program:nodeapp]
directory=/fancy-store
command=npm start
autostart=true
autorestart=true
user=nodeapp
environment=HOME="/home/nodeapp",USER="nodeapp",NODE_ENV="production"
stdout logfile=syslog
stderr logfile=syslog
EOF

supervisorctl reread
supervisorctl update
```

Find the text [DEVSHHELL_PROJECT_ID] in the file and replace it with the output from the following command:

```
echo $DEVSHHELL_PROJECT_ID
```

Example output:

```
qwiklabs-gcp-123456789xyz
```

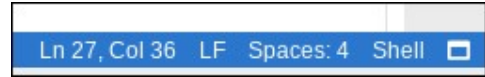
The line of code within `startup-script.sh` should now be similar to the following:

```
gs://fancy-store-qwiklabs-gcp-123456789xyz/monolith-to-microservices/microservices/*
/fancy-store/
```

Save the file, then close it.

Cloud Shell Code Editor: Ensure "End of Line Sequence" is set to "LF" and not "CRLF".

Check by looking at the bottom right of the Code Editor:



If this is set to CRLF, click **CRLF** and then select **LF** in the drop down.

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.

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
```

It will now be accessible

at: `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, so inaccessible through a web browser. Compute Engine instances will automatically be able to access this through their service account.

Copy code into 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.

You could also code this to pull environment variables from elsewhere, but for demonstration purposes this is a simple method to handle configuration. In production, environment variables would likely be stored outside of the code.

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/
```

The `node_modules` dependencies directories are deleted to ensure the copy is as fast and efficient as possible. These are recreated on the instances when they start up.

Click *Check my progress* to verify the objective.

Deploy backend instance

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

In a production environment, you may want to separate each microservice into their own instance and instance group to allow them to scale independently. For demonstration purposes, both backend microservices (Orders & Products) will reside on the same instance and instance group.

Execute the following command to create an `n1-standard-1` 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 \
  --machine-type=n1-standard-1 \
  --tags=backend \
  --metadata=startup-script-url=https://storage.googleapis.com/fancy-store-
$DEVSHHELL PROJECT ID/startup-script.sh
```

If you are asked to specify a zone, ensure a default zone was configured within the Set Up portion of this lab.

Configure connection to backend

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

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
```

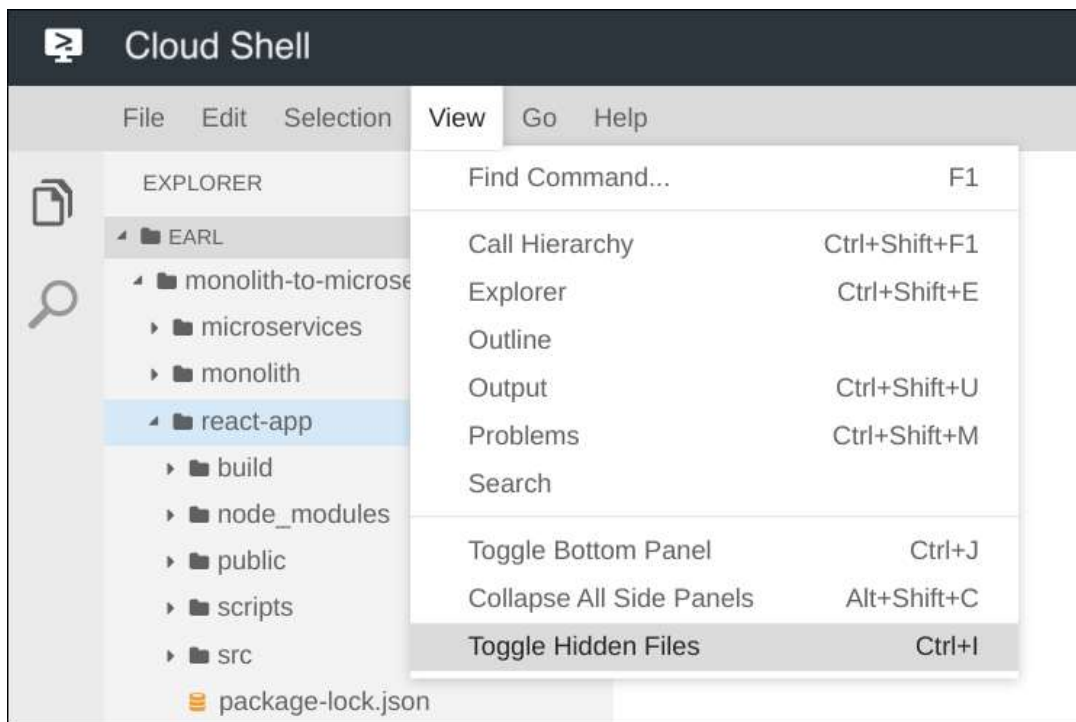
Example output:

NAME	ZONE	MACHINE_TYPE	PREEMPTIBLE	INTERNAL_IP	EXTERNAL_IP	STATUS
backend	us-central1-f	n1-standard-1		10.128.0.2	34.68.223.88	RUNNING

Copy the External IP for the backend.

In the Cloud Shell Explorer, navigate to `monolith-to-microservices > react-app`.

In the Code Editor, select **View > Toggle Hidden Files** in order to see the `.env` file.



Edit the `.env` file to point to the External IP of the backend. **[BACKEND_ADDRESS]** represents the External IP address of the backend instance determined from the above `gcloud` command.

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
```

Save the file.

Rebuild `react-app`, which will update the frontend code:

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

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-$DEVSHELL_PROJECT_ID/
```

Deploy 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 \
  --machine-type=n1-standard-1 \
  --tags=frontend \
  --metadata=startup-script-url=https://storage.googleapis.com/fancy-store-
$DEVSHHELL_PROJECT_ID/startup-script.sh
```

The deployment command and startup script is used with both the frontend and backend instances for simplicity, and because the code is configured to launch all microservices by default. As a result, all microservices run on both the frontend and backend in this sample. In a production environment you'd only run the microservices you need on each component.

Configure Network

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
```

The website should now be fully functional.

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
```

Example output:

NAME	ZONE	MACHINE_TYPE	PREEMPTIBLE	INTERNAL_IP	EXTERNAL_IP
STATUS					
backend	us-central1-f	n1-standard-1		10.128.0.2	35.184.46.126
RUNNING					
frontend	us-central1-f	n1-standard-1		10.128.0.3	35.223.110.167
RUNNING					

It may take a couple minutes for the instance to start and be configured.

Wait 30 seconds, then execute the following to monitor for the application becoming ready, replacing `FRONTEND_ADDRESS` with the External IP for the frontend instance:

```
watch -n 2 curl http://[FRONTEND_ADDRESS]:8080
```

Once you see output similar to the following, the website should be ready..

```
Every 5.0s: curl http://35.225.14.72:8090

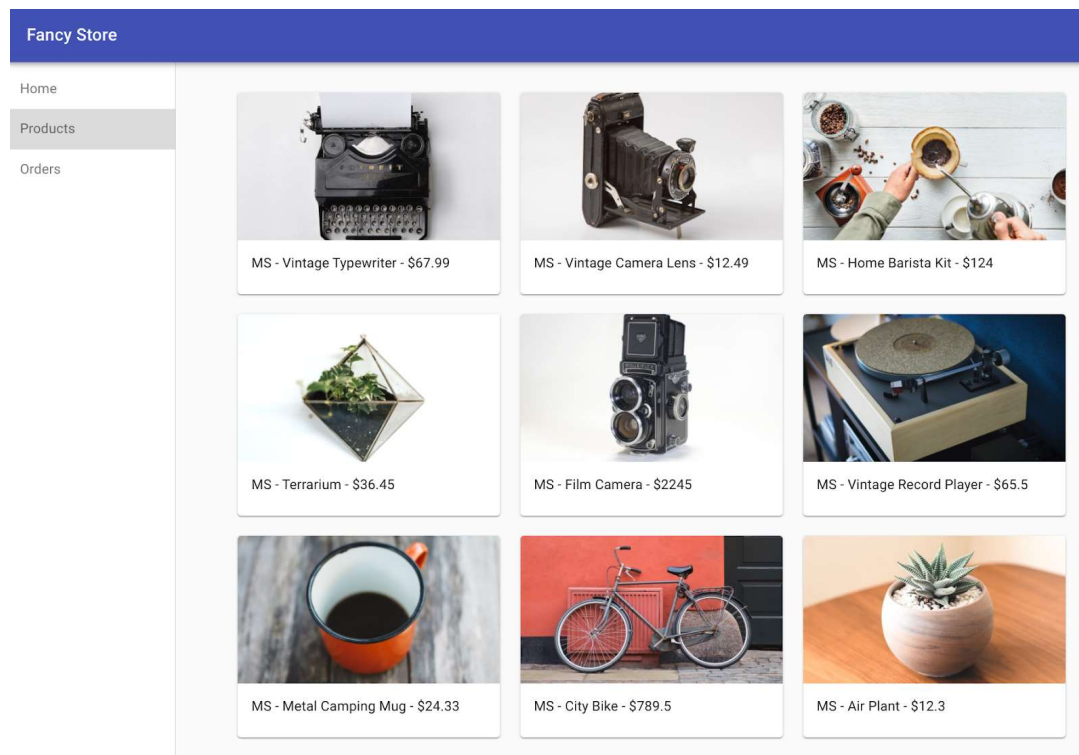
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed

  0   0   0    0    0    0     0      0      0     0  100  2165   100  2165
<!doctype html><html lang="en"><head><meta charset="utf-8"/><link rel="shortcut icon" href="/favicon.
/><meta name="description" content="Web site created using create-react-app"/><link rel="apple-touch-
pt>You need to enable JavaScript to run this app.</noscript><div id="root"></div><script>!function(i)
p,t)&&p[t]&&l.push(p[t][0]),p[t]=0;for(r in o)Object.prototype.hasOwnProperty.call(o,r)&&(i[r]=o[r]);
r t=c[r],n=!0,o=1;o<t.length;o++){var u=t[o];0!==p[u]&&(n=!1)}n&&(c.splice(r--,1),e=f(f.s=t[0]))}retu
ll(r.exports,r,r.exports,f),r.l=!0,r.exports}f.m=i,f.c=t,f.d=function(e,r,t){f.o(e,r)||Object.defineP
perty(e,Symbol.toStringTag,{value:"Module"}),Object.defineProperty(e,"__esModule",{value:!0})),f.t=fu
ull);if(f.r(t),Object.defineProperty(t,"default",{enumerable:!0,value:r}),2&&"string"!==typeof r)for
){return e.default}:function(){return e};return f.d(r,"a",r),r},f.o=function(e,r){return Object.proto
nd(r);r.push=e,r=r.slice();for(var o=0;o<r.length;o++)e(r[o]);var s=n;a()}(())</script><script src="/
```

Press **Ctrl+C** to cancel the `watch` command

Open a new browser tab and browse to `http://[FRONTEND_ADDRESS]:8080` to access the website, where `[FRONTEND_ADDRESS]` is the frontend `EXTERNAL_IP` determined above.

Try navigating to the **Products** and **Orders** pages; these should now work.



Click *Check my progress* to verify the objective.

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. We will be using managed instance groups for our `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.

First, stop both instances:

```
gcloud compute instances stop frontend
gcloud compute instances stop backend
```

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

```
gcloud compute instance-templates create fancy-fe \
  --source-instance=frontend
gcloud compute instance-templates create fancy-be \
  --source-instance=backend
```

Confirm the instance templates were created:

```
gcloud compute instance-templates list
```

Example output:

NAME	MACHINE_TYPE	PREEMPTIBLE	CREATION_TIMESTAMP
fancy-be	n1-standard-1		2020-02-03T10:34:12.966-08:00
fancy-fe	n1-standard-1		2020-02-03T10:34:01.082-08:00

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

```
gcloud compute instances delete backend
```

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 \
  --base-instance-name fancy-fe \
  --size 2 \
  --template fancy-fe

gcloud compute instance-groups managed create fancy-be-mig \
  --base-instance-name fancy-be \
  --size 2 \
  --template fancy-be
```

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.

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 \
  --named-ports frontend:8080

gcloud compute instance-groups set-named-ports fancy-be-mig \
  --named-ports orders:8081,products:8082
```

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.

Note: Separate health checks for load balancing and for autohealing will be used. Health checks for load balancing can and should be more aggressive because these health checks determine whether an instance receives user traffic. You want to catch non-

responsive instances quickly so you can redirect traffic if necessary. In contrast, health checking for autohealing causes Compute Engine to proactively replace failing instances, so this health check should be more conservative than a load balancing health check. 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
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
```

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
```

Apply the health checks to their respective services:

```
gcloud compute instance-groups managed update fancy-fe-mig \
  --health-check fancy-fe-hc \
  --initial-delay 300
gcloud compute instance-groups managed update fancy-be-mig \
  --health-check fancy-be-hc \
  --initial-delay 300
```

It can take 15 minutes before autohealing begins monitoring instances in the group. 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.

Click *Check my progress* to verify the objective.

Create Load Balancers

To complement our managed instance groups, you will be using an HTTP(S) Load Balancers to serve traffic to the frontend and backend microservices, and using mappings to send traffic to the proper backend services based on pathing rules. This will expose a single load balanced IP for all services.

You can learn more about the Load Balancing options on Google Cloud: [Overview of Load Balancing](#)..

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.

For demonstration purposes in order to avoid SSL certificate complexity, use HTTP instead of HTTPS. For production, it is recommended to use HTTPS for encryption wherever possible.

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
```

These health checks are for the load balancer, and only handle directing traffic from the load balancer; they do not cause the managed instance groups to recreate instances. Create backend services that are the target for load-balanced traffic. The backend services will use the health checks and named ports you created:

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

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

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

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

```
gcloud compute backend-services add-backend fancy-fe-frontend \
  --instance-group fancy-fe-mig \
  --instance-group-zone us-central1-f \
  --global

gcloud compute backend-services add-backend fancy-be-orders \
  --instance-group fancy-be-mig \
  --instance-group-zone us-central1-f \
  --global

gcloud compute backend-services add-backend fancy-be-products \
  --instance-group fancy-be-mig \
  --instance-group-zone us-central1-f \
  --global
```

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
```

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"
```

Create the proxy which ties to the URL map:

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

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
```

Click *Check my progress* to verify the objective.

Update 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.

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

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

Find the IP address for the Load Balancer:

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

Example output:

NAME	REGION	IP ADDRESS	IP PROTOCOL	TARGET
fancy-http-rule	34.102.237.51	TCP		fancy-proxy

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
```

The ports are removed in the new address because the load balancer is configured to handle this forwarding for you.

Save the file.

Rebuild `react-app`, which will update the frontend code:

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

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:

```
gcloud compute instance-groups managed rolling-action replace fancy-fe-mig \
--max-unavailable 100%
```

In this example of a rolling replace, you specifically state that all machines can be replaced immediately through the `--max-unavailable` parameter. Without this parameter, the command would keep an instance alive while restarting others to ensure availability. For testing purposes, you specify to replace all immediately for speed. Click *Check my progress* to verify the objective.

Update the frontend instances

Check my progress

Test the website

Wait approximately 30 seconds after issues 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 until instances appear in the list:

```
watch -n 2 gcloud compute instance-groups list-instances fancy-fe-mig
```

Once items appear in the list, exit the `watch` command by pressing **Ctrl+C**.

Run the following to confirm the service is listed as **HEALTHY**:

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

Wait until the 2 services are listed as **HEALTHY**.

Example output:

```
---
backend: https://www.googleapis.com/compute/v1/projects/my-gce-codelab/zones/us-
centrall1-a/instanceGroups/fancy-fe-mig
status:
  healthStatus:
    - healthState: HEALTHY
      instance: https://www.googleapis.com/compute/v1/projects/my-gce-codelab/zones/us-
centrall1-a/instances/fancy-fe-x151
      ipAddress: 10.128.0.7
      port: 8080
    - healthState: HEALTHY
      instance: https://www.googleapis.com/compute/v1/projects/my-gce-codelab/zones/us-
centrall1-a/instances/fancy-fe-cgrt
      ipAddress: 10.128.0.11
      port: 8080
  kind: compute#backendServiceGroupHealth
```

If one instance encounters an issue and is UNHEALTHY it should automatically be repaired. Wait for this to happen. If neither instance enters a HEALTHY state after waiting a little while, something is wrong with the setup of the frontend instances that accessing them on port 8080 doesn't work. Test this by browsing to the instances directly on port 8080.

Once both items appear as HEALTHY on the list, exit the `watch` command by pressing **Ctrl+C**.

The application will be accessible via `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
```

You'll be checking the application later in the lab.

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 will 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 \
  --max-num-replicas 2 \
  --target-load-balancing-utilization 0.60

gcloud compute instance-groups managed set-autoscaling \
  fancy-be-mig \
  --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.

Click *Check my progress* to verify the objective.

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.

Update the `frontend` instance, which acts as the basis for the instance template. During the update, you will 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.

Now modify the machine type of your instance template, by switching from the `n1-standard-1` machine type into a custom machine type with 4 vCPU and 3840MiB RAM.

Run the following command to modify the machine type of the frontend instance:

```
gcloud compute instances set-machine-type frontend --machine-type custom-4-3840
```

Create the new Instance Template:

```
gcloud compute instance-templates create fancy-fe-new \
  --source-instance=frontend \
  --source-instance-zone=us-central1-f
```

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


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

Wait 30 seconds then run the following to monitor the status of the update:

```
watch -n 2 gcloud compute instance-groups managed list-instances fancy-fe-mig
```

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**)
Copy the name of one of the machines listed for use in the next command.

Ctrl+C to exit the `watch` process.

Run the following to see if the virtual machine is using the new machine type (custom-4-3840), where [VM_NAME] is the newly created instance:

```
gcloud compute instances describe [VM_NAME] | grep machineType
```

Expected example output:

```
machineType: https://www.googleapis.com/compute/v1/projects/project-name/zones/us-
central1-f/machineTypes/custom-4-3840
```

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.

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
```

Print the file contents to verify the changes:

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

The resulting code should look like this:

```
/*
Copyright 2019 Google LLC
Licensed under the Apache License, Version 2.0 (the "License");
```

you may not use this file except in compliance with the License.
You may obtain a copy of the License at

<https://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

```
*/

import React from "react";
import { makeStyles } from "@material-ui/core/styles";
import Paper from "@material-ui/core/Paper";
import Typography from "@material-ui/core/Typography";
const useStyles = makeStyles(theme => ({
  root: {
    flexGrow: 1
  },
  paper: {
    width: "800px",
    margin: "0 auto",
    padding: theme.spacing(3, 2)
  }
}));
export default function Home() {
  const classes = useStyles();
  return (
    <div className={classes.root}>
      <Paper className={classes.paper}>
        <Typography variant="h5">
          Fancy Fashion & Style Online
        </Typography>
        <br />
        <Typography variant="body1">
          Tired of mainstream fashion ideas, popular trends and societal norms?
          This line of lifestyle products will help you catch up with the Fancy trend
and express your personal style.
          Start shopping Fancy items now!
        </Typography>
      </Paper>
    </div>
  );
}
```

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

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
```

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

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

```
gcloud compute instance-groups managed rolling-action replace fancy-fe-mig \
  --max-unavailable=100%
```

In this example of a rolling replace, you specifically state that all machines can be replaced immediately through the `--max-unavailable` parameter. Without this parameter, the command would keep an instance alive while replacing others. For testing purposes, you specify to replace all immediately for speed. In production, leaving a buffer would allow the website to continue serving the website while updating.

Click *Check my progress* to verify the objective.

Update the website

Check my progress

Wait approximately 30 seconds 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 until instances appear in the list:

```
watch -n 2 gcloud compute instance-groups list-instances fancy-fe-mig
```

Once items appear in the list, exit the `watch` command by pressing **Ctrl+C**.

Run the following to confirm the service is listed as **HEALTHY**:

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

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

Example output:

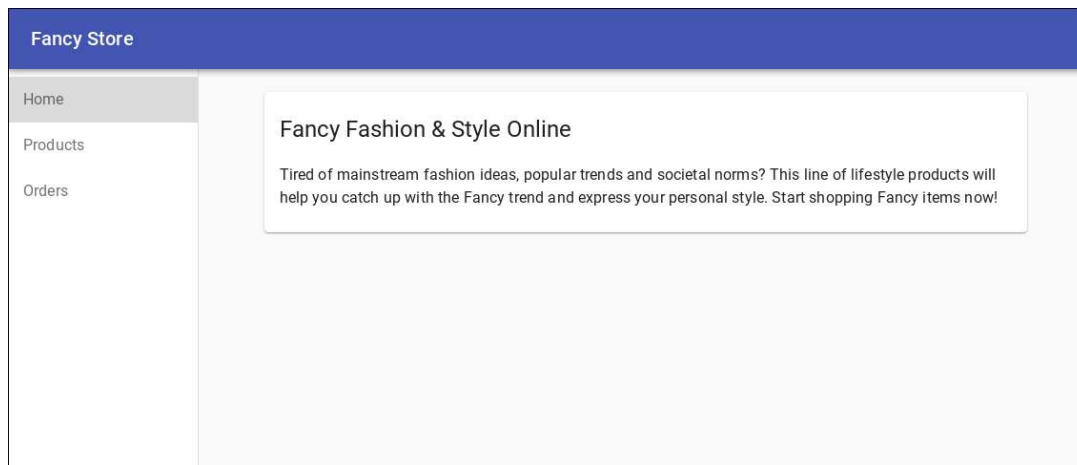
```
---
backend: https://www.googleapis.com/compute/v1/projects/my-gce-codelab/zones/us-
centrall1-a/instanceGroups/fancy-fe-mig
status:
  healthStatus:
    - healthState: HEALTHY
      instance: https://www.googleapis.com/compute/v1/projects/my-gce-codelab/zones/us-
centrall1-a/instances/fancy-fe-x151
      ipAddress: 10.128.0.7
      port: 8080
    - healthState: HEALTHY
      instance: https://www.googleapis.com/compute/v1/projects/my-gce-codelab/zones/us-
centrall1-a/instances/fancy-fe-cgrt
      ipAddress: 10.128.0.11
      port: 8080
  kind: compute#backendServiceGroupHealth
```

Once items appear in the list, exit the `watch` command by pressing **Ctrl+C**.

Browse to the website via `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
```

The new website changes should now be visible.



Simulate Failure

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

To find an instance name, execute the following:

```
gcloud compute instance-groups list-instances fancy-fe-mig
```

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]
```

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

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

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

Exit the instance:

```
exit
```

Monitor the repair operations:

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

This will take a few minutes to complete.

Look for the following example output:

NAME	TYPE
TARGET	HTTP_STATUS STATUS TIMESTAMP
repair-1568314034627-5925f90ee238d-fe645bf0-7becce15	
compute.instances.repair.recreateInstance	us-central1-a/instances/fancy-fe-1vqq 200
DONE	2019-09-12T11:47:14.627-07:00

The managed instance group recreated the instance to repair it.

You can also monitor through the Console - go to **Navigation menu > Compute Engine > VM instances**.

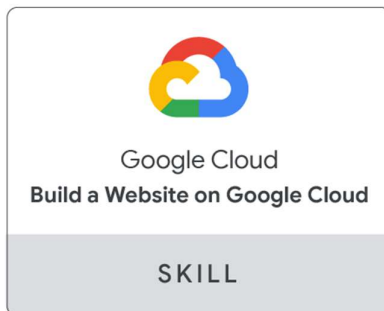
Congratulations!

You successfully deployed, scaled, and updated your website on Compute Engine. You are now experienced with Compute Engine, Managed Instance Groups, Load Balancers, and Health Checks!



Finish Your Quest

This self-paced lab is part of the Qwiklabs [Website on Google Cloud](#) Quest. A Quest is a series of related labs that form a learning path. Enroll in this Quest and get immediate completion credit if you've taken this lab. [See other available Qwiklabs Quests](#). Looking for a hands-on challenge lab to demonstrate your skills and validate your knowledge? On completing this quest, finish this additional [challenge lab](#) to receive an exclusive Google Cloud digital badge.



Take your next lab

Continue your learning with [Deploy, Scale, and Update your Website on Google Kubernetes Engine](#), or check out these suggestions:

- [Migrating a Monolithic Website to Microservices on Google Kubernetes Engine](#)
- Watch this video case study on [Hosting Scalable Web Applications on Google Cloud](#)

Next Steps / Learn More

- [Autohealing and health checks in Managed Instance Groups](#)