**Hosting a Web App on Google Cloud Using Compute Engine**

experimentLabschedule1 hour 30 minutesuniversal\_currency\_alt1 Creditshow\_chartIntroductory

Rate Lab

infoThis lab may incorporate AI tools to support your learning

**GSP662**



**Overview**

There are many ways to deploy web sites within Google Cloud. Each solution offers 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

In this lab you learn how to:

* Create [Compute Engine instances](https://cloud.google.com/compute/docs/instances/)
* Create [instance templates](https://cloud.google.com/compute/docs/instance-templates/) from source instances
* Create [managed instance groups](https://cloud.google.com/compute/docs/instance-groups/)
* Create and test [managed instance group health checks](https://cloud.google.com/compute/docs/instance-groups/autohealing-instances-in-migs)
* Create HTTP(S) [Load Balancers](https://cloud.google.com/load-balancing/)
* Create [load balancer health checks](https://cloud.google.com/load-balancing/docs/health-checks)
* Use a [Content Delivery Network (CDN)](https://cloud.google.com/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 your website.

**Setup and requirements**

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 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.

To complete this lab, you need:

* Access to a standard internet browser (Chrome browser recommended).

**Note:** Use an Incognito or private browser window to run this lab. This prevents any conflicts between your personal account and the Student account, which may cause extra charges incurred to your personal account.

* Time to complete the lab---remember, once you start, you cannot pause a lab.

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

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 the **Lab Details** panel with the following:
   * The **Open Google Cloud console** button
   * Time remaining
   * The temporary credentials that you must use for this lab
   * Other information, if needed, to step through this lab
2. Click **Open Google Cloud console** (or right-click and select **Open Link in Incognito Window** if you are running the Chrome browser).

The lab spins up resources, and then opens another tab that shows the **Sign in** page.

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

**Note:**If you see the **Choose an account** dialog, click **Use Another Account**.

1. If necessary, copy the **Username** below and paste it into the **Sign in** dialog.

student-00-72e4be373c4d@qwiklabs.net

Copied!

content\_copy

You can also find the **Username** in the **Lab Details** panel.

1. Click **Next**.
2. Copy the **Password** below and paste it into the **Welcome** dialog.

UNPt6IvTAIOH

Copied!

content\_copy

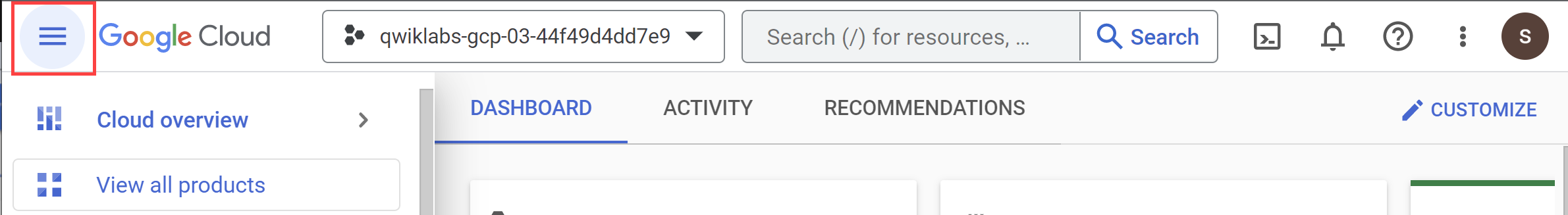
You can also find the **Password** in the **Lab Details** panel.

1. Click **Next**.

**Important:**You must use the credentials the lab provides you. Do not use your Google Cloud account credentials.**Note:**Using your own Google Cloud account for this lab may incur extra charges.

1. 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 Google Cloud console opens in this tab.

**Note:** To view a menu with a list of Google Cloud products and services, click 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.

1. Click **Activate Cloud Shell** Activate Cloud Shell icon at the top of the Google Cloud console.

When you are connected, you are already authenticated, and the project is set to your **Project\_ID**, qwiklabs-gcp-02-40616d0794db. The output contains a line that declares the **Project\_ID** for this session:

Your Cloud Platform project in this session is set to qwiklabs-gcp-02-40616d0794db

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

1. (Optional) You can list the active account name with this command:

gcloud auth list

Copied!

content\_copy

1. Click **Authorize**.

**Output:**

ACTIVE: \*

ACCOUNT: student-00-72e4be373c4d@qwiklabs.net

To set the active account, run:

$ gcloud config set account `ACCOUNT`

1. (Optional) You can list the project ID with this command:

gcloud config list project

Copied!

content\_copy

**Output:**

[core]

project = qwiklabs-gcp-02-40616d0794db

**Note:**For full documentation of gcloud, in Google Cloud, refer to [the gcloud CLI overview guide](https://cloud.google.com/sdk/gcloud).

Set your region and zone

Certain Compute Engine resources live in regions and zones. A region is a specific geographical location where you can run your resources. Each region has one or more zones.

Learn more about regions and zones and see a complete list in [Regions & Zones documentation](https://cloud.google.com/compute/docs/regions-zones/).

Run the following gcloud commands in Cloud Console to set the default region and zone for your lab:

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

export ZONE=$(gcloud config get compute/zone)

gcloud config set compute/region "us-central1"

export REGION=$(gcloud config get compute/region)

Copied!

content\_copy

**Task 1. Enable Compute Engine API**

* Enable the [Compute Engine API](https://console.cloud.google.com/flows/enableapi?apiid=compute) by executing the following:

gcloud services enable compute.googleapis.com

Copied!

content\_copy

**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:

gsutil mb gs://fancy-store-$DEVSHELL\_PROJECT\_ID

Copied!

content\_copy

**Note:**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

**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

Copied!

content\_copy

cd ~/monolith-to-microservices

Copied!

content\_copy

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

./setup.sh

Copied!

content\_copy

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

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

nvm install --lts

Copied!

content\_copy

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

cd microservices

npm start

Copied!

content\_copy

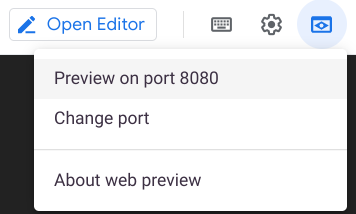
You should see the following output:

Products microservice listening on port 8082!

Frontend microservice listening on port 8080!

Orders microservice listening on port 8081!

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



This opens a new window where you can see the frontend of Fancy Store.

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

1. 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.

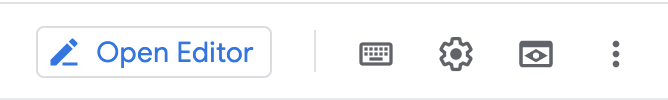
1. In Cloud Shell, run the following command to create a file called startup-script.sh:

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

Copied!

content\_copy

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



1. Navigate to the monolith-to-microservices folder.
2. Add the following code to the startup-script.sh 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/v16.14.0/node-v16.14.0-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-[DEVSHELL\_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

Copied!

content\_copy

1. Find the text [DEVSHELL\_PROJECT\_ID] in the file and replace it with your Project ID: qwiklabs-gcp-02-40616d0794db

The line of code within startup-script.sh should now resemble:

gs://fancy-store-qwiklabs-gcp-02-40616d0794db/monolith-to-microservices/microservices/\* /fancy-store/

1. **Save** the startup-script.sh file, but do not close it yet.
2. Look at the bottom right of Cloud Shell Code Editor, and ensure "End of Line Sequence" is set to "LF" and not "CRLF".

"End of Line Sequence"

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

1. Close the startup-script.sh file.
2. 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-$DEVSHELL\_PROJECT\_ID

Copied!

content\_copy

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, therefor 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.

**Note:**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.

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-$DEVSHELL\_PROJECT\_ID/

Copied!

content\_copy

**Note:**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.

Copy startup script and code to Cloud Storage bucket

Check my progress

Deploy the backend instance

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

**Note:**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 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-$DEVSHELL\_PROJECT\_ID/startup-script.sh

Copied!

content\_copy

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

Copied!

content\_copy

Example output:

NAME: backend

ZONE: us-central1-f

MACHINE\_TYPE: e2-standard-2

PREEMPTIBLE:

INTERNAL\_IP: 10.142.0.2

EXTERNAL\_IP: 35.237.245.193

STATUS: RUNNING

1. **Copy the External IP** for the backend.
2. In the Cloud Shell Explorer, navigate to monolith-to-microservices > react-app.
3. 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 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.

1. 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

1. **Save** the file.
2. 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

Copied!

content\_copy

1. 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/

Copied!

content\_copy

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-$DEVSHELL\_PROJECT\_ID/startup-script.sh

Copied!

content\_copy

**Note:**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 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

Copied!

content\_copy

gcloud compute firewall-rules create fw-be \

--allow tcp:8081-8082 \

--target-tags=backend

Copied!

content\_copy

The website should now be fully functional.

1. 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

Copied!

content\_copy

Example output:

NAME: backend

ZONE: us-central1-f

MACHINE\_TYPE: e2-standard-2

PREEMPTIBLE:

INTERNAL\_IP: 10.128.0.2

EXTERNAL\_IP: 34.27.178.79

STATUS: RUNNING

NAME: frontend

ZONE: us-central1-f

MACHINE\_TYPE: e2-standard-2

PREEMPTIBLE:

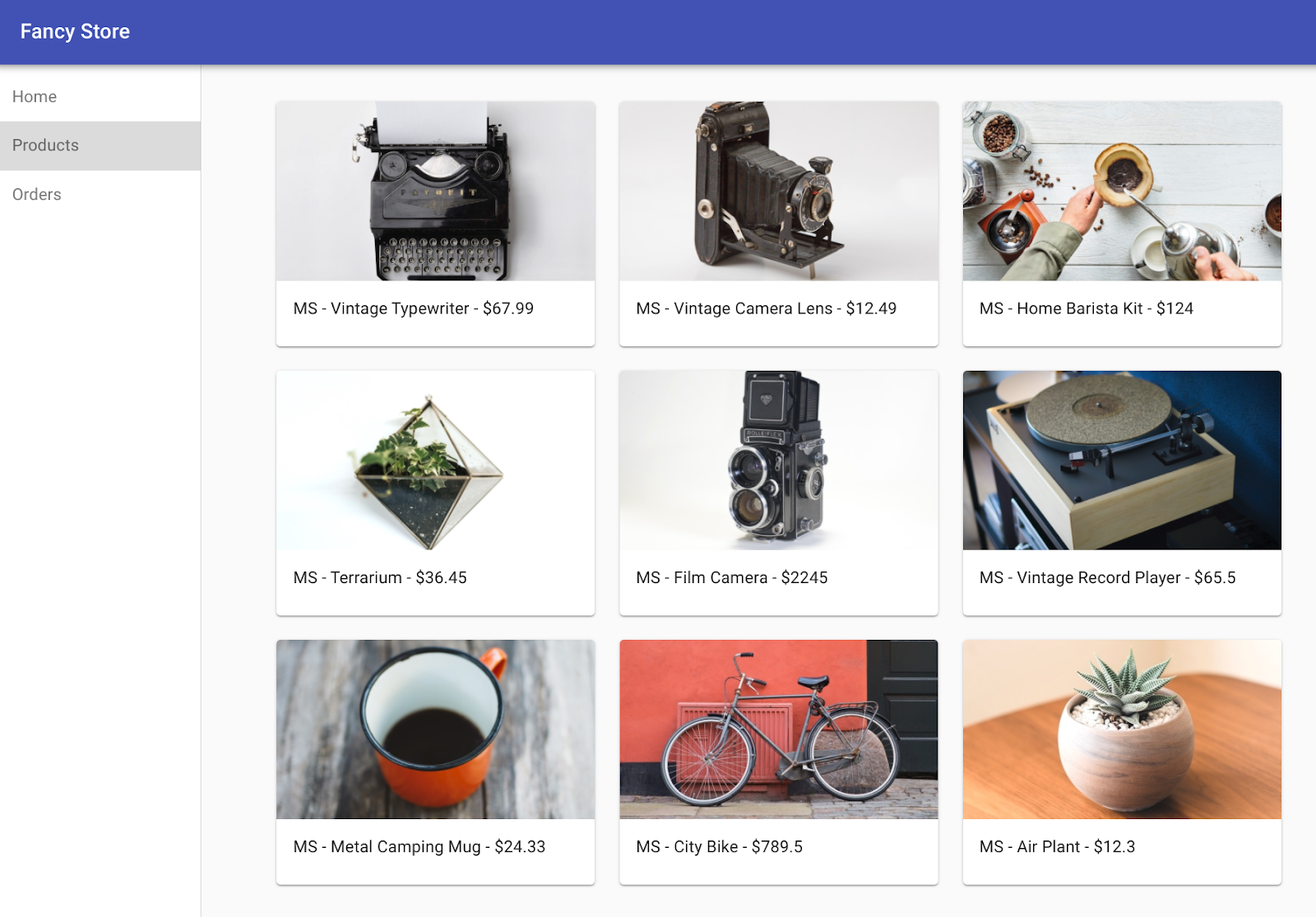
INTERNAL\_IP: 10.128.0.3

EXTERNAL\_IP: 34.172.241.242

STATUS: RUNNING

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

1. Wait 3 minutes and then 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.
2. Try navigating to the **Products** and **Orders** pages; these should now work.



Click **Check my progress** to verify the objective.

Deploy instances and configure network

Check my progress

**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

Copied!

content\_copy

gcloud compute instances stop backend --zone=$ZONE

Copied!

content\_copy

1. 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

Copied!

content\_copy

gcloud compute instance-templates create fancy-be \

--source-instance-zone=$ZONE \

--source-instance=backend

Copied!

content\_copy

1. Confirm the instance templates were created:

gcloud compute instance-templates list

Copied!

content\_copy

Example output:

NAME: fancy-be

MACHINE\_TYPE: e2-standard-2

PREEMPTIBLE:

CREATION\_TIMESTAMP: 2023-07-25T14:52:21.933-07:00

NAME: fancy-fe

MACHINE\_TYPE: e2-standard-2

PREEMPTIBLE:

CREATION\_TIMESTAMP: 2023-07-25T14:52:15.442-07:00

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

gcloud compute instances delete backend --zone=$ZONE

Copied!

content\_copy

1. 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

1. 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

Copied!

content\_copy

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

--zone=$ZONE \

--base-instance-name fancy-be \

--size 2 \

--template fancy-be

Copied!

content\_copy

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.

1. 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

Copied!

content\_copy

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

--zone=$ZONE \

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

Copied!

content\_copy

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.

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

Copied!

content\_copy

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

Copied!

content\_copy

1. 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

Copied!

content\_copy

1. 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

Copied!

content\_copy

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

--zone=$ZONE \

--health-check fancy-be-hc \

--initial-delay 300

Copied!

content\_copy

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

1. 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 managed instance groups

Check my progress

**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.

You can learn more about the Load Balancing options on Google Cloud: [Overview of Load Balancing](https://cloud.google.com/load-balancing/docs/load-balancing-overview).

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.

**Note:**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.

1. 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

Copied!

content\_copy

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

--request-path /api/orders \

--port 8081

Copied!

content\_copy

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

--request-path /api/products \

--port 8082

Copied!

content\_copy

**Note:**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.

1. 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

Copied!

content\_copy

gcloud compute backend-services create fancy-be-orders \

--http-health-checks fancy-be-orders-hc \

--port-name orders \

--global

Copied!

content\_copy

gcloud compute backend-services create fancy-be-products \

--http-health-checks fancy-be-products-hc \

--port-name products \

--global

Copied!

content\_copy

1. Add the Load Balancer's [backend services](https://cloud.google.com/load-balancing/docs/backend-service):

gcloud compute backend-services add-backend fancy-fe-frontend \

--instance-group-zone=$ZONE \

--instance-group fancy-fe-mig \

--global

Copied!

content\_copy

gcloud compute backend-services add-backend fancy-be-orders \

--instance-group-zone=$ZONE \

--instance-group fancy-be-mig \

--global

Copied!

content\_copy

gcloud compute backend-services add-backend fancy-be-products \

--instance-group-zone=$ZONE \

--instance-group fancy-be-mig \

--global

Copied!

content\_copy

1. 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

Copied!

content\_copy

1. 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"

Copied!

content\_copy

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

gcloud compute target-http-proxies create fancy-proxy \

--url-map fancy-map

Copied!

content\_copy

1. 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

Copied!

content\_copy

Click **Check my progress** to verify the objective.

Create HTTP(S) load balancers

Check my progress

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/

Copied!

content\_copy

1. Find the IP address for the Load Balancer:

gcloud compute forwarding-rules list --global

Copied!

content\_copy

Example output:

NAME: fancy-http-rule

REGION:

IP\_ADDRESS: 34.111.203.235

IP\_PROTOCOL: TCP

TARGET: fancy-proxy

1. 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

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

1. **Save** the file.
2. Rebuild react-app, which will update the frontend code:

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

npm install && npm run-script build

Copied!

content\_copy

1. 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-$DEVSHELL\_PROJECT\_ID/

Copied!

content\_copy

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 \

--zone=$ZONE \

--max-unavailable 100%

Copied!

content\_copy

**Note:**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

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

Copied!

content\_copy

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

Example output:

backend: https://www.googleapis.com/compute/v1/projects/my-gce-codelab/zones/us-central1-a/instanceGroups/fancy-fe-mig

status:

healthStatus:

- healthState: HEALTHY

instance: https://www.googleapis.com/compute/v1/projects/my-gce-codelab/zones/us-central1-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-central1-a/instances/fancy-fe-cgrt

ipAddress: 10.128.0.11

port: 8080

kind: compute#backendServiceGroupHealth

**Note:**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.

1. Once both items appear as HEALTHY on the list, exit the watch command by pressing CTRL+C.

**Note:**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.

**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

Copied!

content\_copy

gcloud compute instance-groups managed set-autoscaling \

fancy-be-mig \

--zone=$ZONE \

--max-num-replicas 2 \

--target-load-balancing-utilization 0.60

Copied!

content\_copy

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

Copied!

content\_copy

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.

Assessment completed!

Scaling Compute Engine

Check my progress

*Assessment completed!*

**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.

1. 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

Copied!

content\_copy

1. Create the new Instance Template:

gcloud compute instance-templates create fancy-fe-new \

--region=$REGION \

--source-instance=frontend \

--source-instance-zone=$ZONE

Copied!

content\_copy

1. 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

Copied!

content\_copy

1. 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

Copied!

content\_copy

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**)

1. **Copy** the name of one of the machines listed for use in the next command.
2. CTRL+C to exit the watch process.
3. 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

Copied!

content\_copy

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

Copied!

content\_copy

1. Print the file contents to verify the changes:

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

Copied!

content\_copy

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 { Box, Paper, Typography } from "@mui/material";

export default function Home() {

return (

<Box sx={{ flexGrow: 1 }}>

<Paper

elevation={3}

sx={{

width: "800px",

margin: "0 auto",

padding: (theme) => theme.spacing(3, 2),

}}

>

<Typography variant="h5">Welcome to the Fancy Store!</Typography>

<br />

<Typography variant="body1">

Take a look at our wide variety of products.

</Typography>

</Paper>

</Box>

);

}

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

1. 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

Copied!

content\_copy

1. 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-$DEVSHELL\_PROJECT\_ID/

Copied!

content\_copy

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%

Copied!

content\_copy

**Note:**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.

Assessment completed!

Update the website

Check my progress

*Assessment completed!*

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

Copied!

content\_copy

1. 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-central1-a/instanceGroups/fancy-fe-mig

status:

healthStatus:

- healthState: HEALTHY

instance: https://www.googleapis.com/compute/v1/projects/my-gce-codelab/zones/us-central1-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-central1-a/instances/fancy-fe-cgrt

ipAddress: 10.128.0.11

port: 8080

kind: compute#backendServiceGroupHealth

1. Once items appear in the list with HEALTHY status, exit the watch command by pressing CTRL+C.
2. 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

Copied!

content\_copy

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.

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

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

Copied!

content\_copy

1. 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

Copied!

content\_copy

1. Type in "y" to confirm, and press **Enter** twice to not use a password.
2. Within the instance, use supervisorctl to stop the application:

sudo supervisorctl stop nodeapp; sudo killall node

Copied!

content\_copy

1. Exit the instance:

exit

Copied!

content\_copy

1. 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.

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.

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

**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!