Hosting a Web App on Google Cloud Using Compute Engine

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

2. Click through the following windows:

    - Continue through the Cloud Shell information window.
    - Authorize Cloud Shell to use your credentials to make Google Cloud API calls.

When you are connected, you are already authenticated, and the project is set to your **Project_ID**, `PROJECT_ID`. The output contains a line that declares the **Project_ID** for this session:

```
Your Cloud Platform project in this session is set to "PROJECT_ID"
```

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

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

```
gcloud auth list
```

Copied!
content_copy

4. Click **Authorize**.

Output:

```
ACTIVE: *
ACCOUNT: "ACCOUNT"

To set the active account, run:
    $ gcloud config set account `ACCOUNT`
```

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

```
gcloud config list project
```

Copied!
content_copy
Output:

```
[core]
project = "PROJECT_ID"
```

Enable Compute Engine API

- Enable the Compute Engine API by executing the following:
```
gcloud services enable compute.googleapis.com
```

CREATE CLOUD STORAGE BUCKET

- From Cloud Shell, execute the following to create a new Cloud Storage bucket:
```
gsutil mb gs://fancy-store-$DEVSHELL_PROJECT_ID
```

CLONE SOURCE REPOSITORY

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
2. 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.

3. Once completed, ensure Cloud Shell is running a compatible nodeJS version with the following command:
```
nvm install --lts
```
Copied!
content_copy
4. 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!
```

5. Preview your application by clicking the **web preview icon** then selecting **Preview on port 8080**.
6. Close this window after viewing the website and then press CTRL+C in the terminal window to stop the web server process.

CREATE COMPUTE ENGINE ISTANCES

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

2. Click **Open Editor** in the Cloud Shell ribbon to open the Code Editor.
3. Navigate to the `monolith-to-microservices` folder.

4. Add the following code to the `startup-script.sh` file. You will edit some of the code after it's added:

```
#!/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

5. Find the text `[DEVSHELL_PROJECT_ID]` in the file and replace it with your Project ID: `Project ID`

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

```
gs://fancy-store-Project ID/monolith-to-microservices/microservices/*
/fancy-store/
```

6. **Save** the `startup-script.sh` file, but do not close it yet.

7. Look at the bottom right of Cloud Shell Code Editor, and ensure "End of Line Sequence" is set to "LF" and not "CRLF".

8. Close the `startup-script.sh` file.

9. Return to Cloud Shell Terminal and run the following to copy the `startup-script.sh` file into your bucket:

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

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

Copy code into the Cloud Storage bucket

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/
```

Deploy the backend instance

- 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
```

Configure a connection to the backend

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: zone
MACHINE_TYPE: e2-standard-2
PREEMPTIBLE:
INTERNAL_IP: 10.142.0.2
EXTERNAL_IP: 35.237.245.193
STATUS: RUNNING
```

2. **Copy the External IP** for the backend.

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

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

In the next step, you edit the `.env` file 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.

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

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

6. **Save** the file.

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

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

Copied!

content_copy

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

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

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
```

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.

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

```
gcloud compute instances list
```

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.

3. 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.

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

Create managed instance groups

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

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

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

Copied!

content_copy

```
gcloud compute instance-templates create fancy-be \
    --source-instance-zone=$ZONE \
    --source-instance=backend
```

Copied!

content_copy

3. 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
```

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

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

Copied!

content_copy

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

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

Create managed instance group

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.

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

```
gcloud compute instance-groups set-named-ports fancy-fe-mig \
    --zone=$ZONE \
    --named-ports frontend:8080
```

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

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

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

```
gcloud compute firewall-rules create allow-health-check \
    --allow tcp:8080-8081 \
    --source-ranges 130.211.0.0/22,35.191.0.0/16 \
    --network default
```

Copied!
content_copy

3. Apply the health checks to their respective services:

```
gcloud compute instance-groups managed update fancy-fe-mig \
    --zone=$ZONE \
    --health-check fancy-fe-hc \
    --initial-delay 300
```

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.

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

Create load balancers

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.

2. 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

3. Add the Load Balancer's backend services:

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

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

4. Create a URL map. The URL map defines which URLs are directed to which backend services:
```
gcloud compute url-maps create fancy-map \
  --default-service fancy-fe-frontend
```
Copied!
content_copy

5. Create a path matcher to allow the `/api/orders` and `/api/products` paths to route to their respective services:
```
gcloud compute url-maps add-path-matcher fancy-map \
   --default-service fancy-fe-frontend \
   --path-matcher-name orders \
   --path-rules "/api/orders=fancy-be-orders,/api/products=fancy-be-
products"
```
Copied!
content_copy

6. Create the proxy which ties to the URL map:
```
gcloud compute target-http-proxies create fancy-proxy \
  --url-map fancy-map
```
Copied!
content_copy

7. Create a global forwarding rule that ties a public IP address and port to the proxy:
```
  gcloud compute forwarding-rules create fancy-http-rule \
--global \
--target-http-proxy fancy-proxy \
--ports 80
```

Update the configuration

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

2. 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
```

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

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

6. **Save** the file.

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

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

Copied!
content_copy

8. Copy the application code into your bucket:

```
9. cd ~
10.  rm -rf monolith-to-microservices/*/node_modules
11.  gsutil -m cp -r monolith-to-microservices gs://fancy-store-
   $DEVSHELL_PROJECT_ID/
```

Update the frontend instances

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%
```

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

2. 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
```

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

Scaling Compute Engine

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.

Update the website

- 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

2. 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

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

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

Copied!
content_copy

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

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

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**)

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

6. CTRL+C to exit the `watch` process.

7. Run the following to see if the virtual machine is using the new machine type (e2-small), where [VM_NAME] is the newly created instance:

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

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

2. 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.

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

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

Copied!

content_copy

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

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

Push changes with rolling replacements

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

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

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

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

Copied!
content_copy

3. 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
```

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

5. Browse to the website via `http://[LB_IP]` 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

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

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

Copied!

content_copy

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

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

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

Copied!

content_copy

5. Exit the instance:

```
exit
```

Copied!

content_copy

6. Monitor the repair operations:

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

Copied!

content_copy

This will take a few minutes to complete.

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.

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