**Deploying, Scaling, and Updating an E-Commerce Website on Google Kubernetes Engine**

**We’ll also transition the existing Monolithic architecture to a Micro-services architecture.**

[[](https://medium.com/@rogernem?source=post_page-----0ff6d0847a3c--------------------------------)](https://medium.com/@rogernem?source=post_page-----0ff6d0847a3c--------------------------------)

[Roger Nem](https://medium.com/@rogernem?source=post_page-----0ff6d0847a3c--------------------------------)

12 min read

·

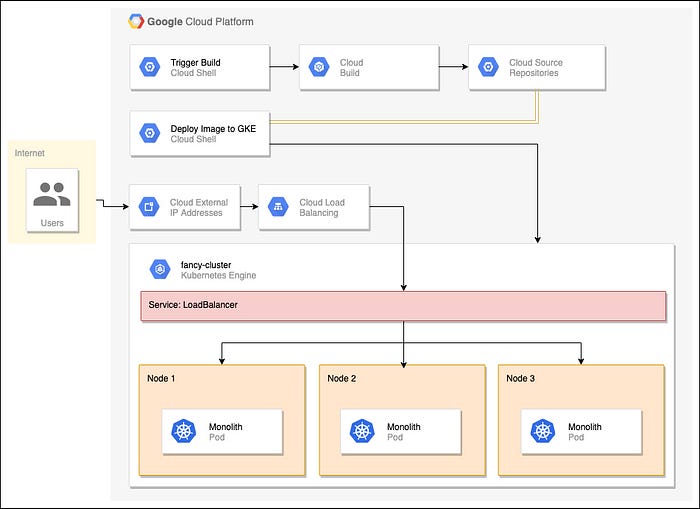
21 hours ago



— Google Kubernetes Engine (picture by author)

Operating websites and applications presents inherent challenges. Anomalies arise unexpectedly, server downtimes occur, surges in demand increase resource utilization, and implementing changes without service disruptions proves intricate and demanding.

Kubernetes facilitates the execution of these tasks and provides the capability for automation. In this article, I will guide you through the deployment of an e-commerce application onto the Google Kubernetes Engine (GKE) based on a real-world scenario.



— Architecture Diagram

**Table of Contents**

1. [K8s in short](https://medium.com/@rogernem/deploying-scaling-and-updating-an-e-commerce-website-on-google-kubernetes-engine-0ff6d0847a3c#28fb)
2. [Prerequisites](https://medium.com/@rogernem/deploying-scaling-and-updating-an-e-commerce-website-on-google-kubernetes-engine-0ff6d0847a3c#65da)
3. [Step 1: Create a GKE Cluster](https://medium.com/@rogernem/deploying-scaling-and-updating-an-e-commerce-website-on-google-kubernetes-engine-0ff6d0847a3c#9393)
4. [Step 2: Create a Docker Container](https://medium.com/@rogernem/deploying-scaling-and-updating-an-e-commerce-website-on-google-kubernetes-engine-0ff6d0847a3c#60a5)
5. [Step 3: Deploy, Expose and Scale the Container](https://medium.com/@rogernem/deploying-scaling-and-updating-an-e-commerce-website-on-google-kubernetes-engine-0ff6d0847a3c#6991)  
   - [Deploying the container to GKE](https://medium.com/@rogernem/deploying-scaling-and-updating-an-e-commerce-website-on-google-kubernetes-engine-0ff6d0847a3c#34d6)  
   - [Verifying the Deployment](https://medium.com/@rogernem/deploying-scaling-and-updating-an-e-commerce-website-on-google-kubernetes-engine-0ff6d0847a3c#996f)  
   - [Exposing the GKE deployment](https://medium.com/@rogernem/deploying-scaling-and-updating-an-e-commerce-website-on-google-kubernetes-engine-0ff6d0847a3c#087e)  
   - [Accessing the service](https://medium.com/@rogernem/deploying-scaling-and-updating-an-e-commerce-website-on-google-kubernetes-engine-0ff6d0847a3c#5f06)  
   - [Scaling the GKE deployment](https://medium.com/@rogernem/deploying-scaling-and-updating-an-e-commerce-website-on-google-kubernetes-engine-0ff6d0847a3c#e72b)
6. [Step 4: Modify our E-Commerce Application](https://medium.com/@rogernem/deploying-scaling-and-updating-an-e-commerce-website-on-google-kubernetes-engine-0ff6d0847a3c#46cb)
7. [Step 5: Rollout our New Version (Zero Downtime)](https://medium.com/@rogernem/deploying-scaling-and-updating-an-e-commerce-website-on-google-kubernetes-engine-0ff6d0847a3c#2f59)
8. [Conclusion](https://medium.com/@rogernem/deploying-scaling-and-updating-an-e-commerce-website-on-google-kubernetes-engine-0ff6d0847a3c#38e8)

**K8s in short**

First developed by a team at Google and later donated to the [Cloud Native Computing Foundation](https://www.cncf.io), Kubernetes, **commonly referred to as K8s**, stands as an open-source container orchestration system for automating software deployment, scaling, and management of containerized applications.

In other words, you can cluster together groups of hosts running Linux containers, and Kubernetes helps you easily and efficiently manage those clusters.

**Prerequisites**

To maximize your understanding and follow along this guide, please make sure the following requirements are met:

* **Google Cloud Account:** You can sign up [here](https://cloud.google.com/free).
* Basic understanding of [Docker](https://docs.docker.com/) and [Kubernetes](https://kubernetes.io/docs/home/) (recommended).

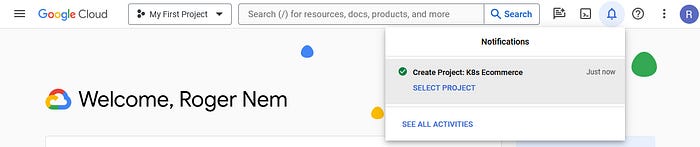
**Step 1: Create a GKE Cluster**

To initiate the deployment of our e-commerce application, a Kubernetes cluster is required.

1. Sign in to the [Google Cloud console](https://console.cloud.google.com).
2. Before doing anything, you must [create a **Google Cloud project**](https://developers.google.com/workspace/guides/create-project)**.**

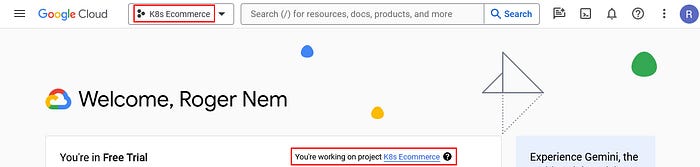
Your project name must be unique and it cannot be changed after created. I named my project “K8s Ecommerce”.

You should see the following notification.



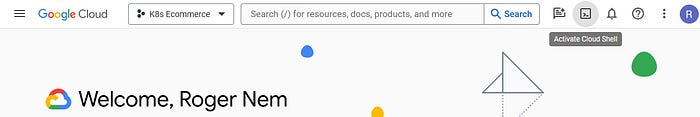
— K8s E-commerce Google Cloud project created (picture by author)

3. Click on “**SELECT PROJECT**”. Next, click on the Notifications icon to close the notification.



— K8s E-commerce Project Selected (picture by author)

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



— Activate Cloud Shell (picture by author)

When you do, you should see the console opened at the bottom of the screen:



(picture by author)

5. Run the following command to enable the Container Registry API. Click on “**AUTHORIZE**” when prompted:

$ gcloud services enable container.googleapis.com

6. Run the following command to create a GKE cluster named k8s-ecommerce-cluster with **3** nodes:

$ gcloud container clusters create k8s-ecommerce-cluster --num-nodes 3

It will take a few minutes for the cluster to be created.

**Note:** If you get an error about region/zone not being specified, please make sure to set the default compute zone: gcloud config set compute/region us-central1

7. Run the following command and see the cluster’s three worker VM instances:

$ gcloud compute instances list

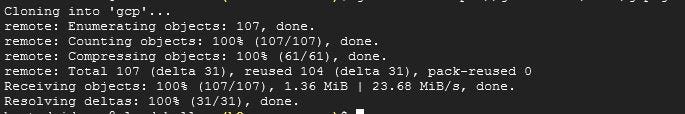
NAME: gke-k8s-ecommerce-cluster-default-pool-fb932da6-4sk6  
ZONE: us-central1-f  
MACHINE\_TYPE: e2-medium  
PREEMPTIBLE:  
INTERNAL\_IP: 10.128.0.3  
EXTERNAL\_IP: 34.172.106.173  
STATUS: RUNNING  
  
NAME: gke-k8s-ecommerce-cluster-default-pool-fb932da6-d6qc  
ZONE: us-central1-f  
MACHINE\_TYPE: e2-medium  
PREEMPTIBLE:  
INTERNAL\_IP: 10.128.0.4  
EXTERNAL\_IP: 34.133.99.176  
STATUS: RUNNING  
  
NAME: gke-k8s-ecommerce-cluster-default-pool-fb932da6-ztnh  
ZONE: us-central1-f  
MACHINE\_TYPE: e2-medium  
PREEMPTIBLE:  
INTERNAL\_IP: 10.128.0.5  
EXTERNAL\_IP: 34.136.180.45  
STATUS: RUNNING

8. Click the **Navigation menu > Kubernetes Engine > Clusters**. You should see your cluster named *k8s-ecommerce-cluster*.

**Step 2: Create a Docker Container**

1. Run the following commands to clone my git repo to your Cloud Shell instance:

$ cd ~  
$ git clone https://github.com/rnem/gcp.git

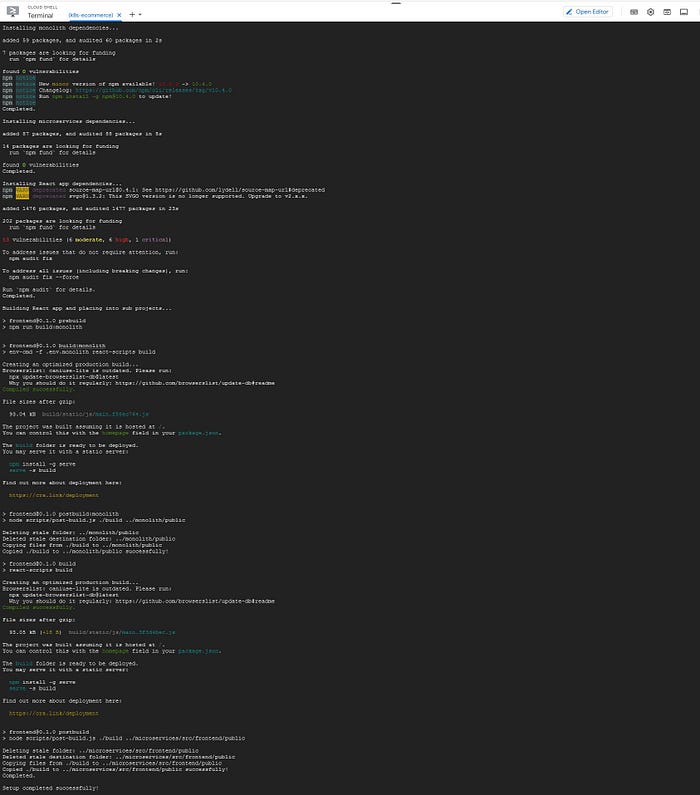


— Expected Git Clone Results (picture by author)

2. Change to the appropriate directory and install the NodeJS dependencies so you can test the application before deploying it.

$ cd ~/gcp/monolith-to-microservices/  
$ chmod +x setup.sh  
$ ./setup.sh

3. Wait a few minutes for this script to finish running.



— Setup Results (picture by author)

4. Run the following command to ensure you are running Cloud Shell with the latest version of npm:

nvm install --lts

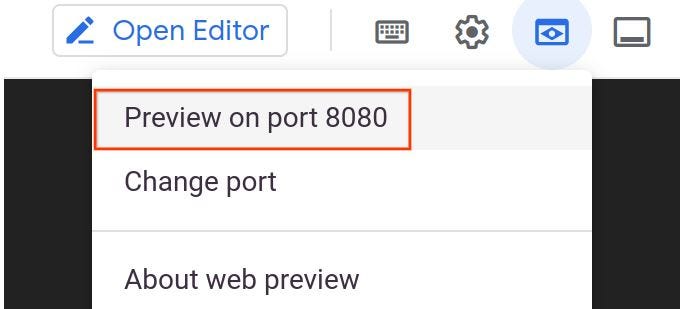
5. Test the Monolith application by running the following command to start the web server.

cd ~ monolith/  
npm start

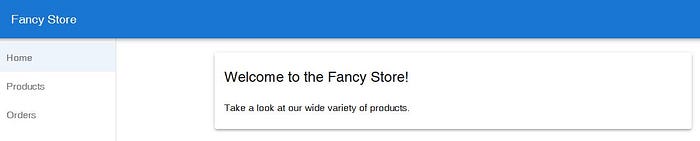
You should see the following:

> monolith@1.0.0 start  
> node ./src/server.js  
  
Monolith listening on port 8080!

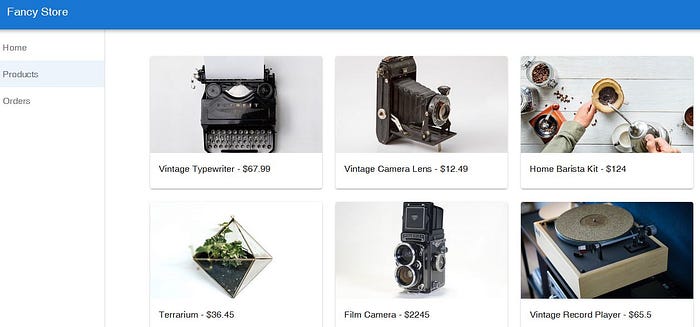
To preview your application, click the web preview icon and select “**Preview on port 8080”**:



— GCP | Web Preview (picture by author)



— Monolith E-Commerce Store running (picture by author)



— Monolith E-Commerce Store running | Products page (picture by author)

Press CTRL+C in Cloud Shell to stop the web server process.

**Leave this tab open, you’ll return to it later.**

6. Now it is time to Dockerize our application. 👍

Typically, you would follow a two-step process, involving the construction of a Docker container followed by pushing it to a registry to store the image, allowing Google Kubernetes Engine (GKE) to pull from.

But to make our lives easier we’ll use Google Cloud Build to build the Docker container and put the image in the Container Registry with a single command. Google Cloud Build will compress the files from the directory and move them to a Google Cloud Storage bucket. The build process will then take all the files from the bucket and use the Dockerfile to run the Docker build process.

6.1. Run the following command to **enable the Cloud Build API**.

gcloud services enable cloudbuild.googleapis.com

6.2. Change to the desired directory and **start the build process**:

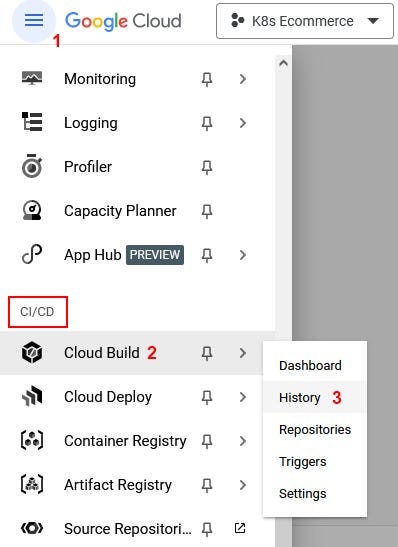
cd ~/gcp/monolith-to-microservices/monolith  
gcloud builds submit --tag gcr.io/${GOOGLE\_CLOUD\_PROJECT}/monolith:1.0.0 .

This process will take a few minutes to complete. Since we specified the --tag flag with the host as gcr.io for the Docker image, the resulting Docker image will be pushed to the Google Cloud Container Registry.

There will be output in the terminal similar to the following:

ID CREATE\_TIME DURATION SOURCE IMAGES STATUS  
1ae295d9-63cb-482c-959b-bc52e9644d53 2024-01-10T01:56:35+00:00 33S gs://\_cloudbuild/source/1567043793.94-abfd382011724422bf49af1558b894aa.tgz gcr.io//monolith:1.0.0 SUCCESS

To view your build history or watch the process in real time, click the **Navigation menu** and scroll down to the **CI/CD section**. Then click “**Cloud Build”** > “**History”**. Here you can see a list of all your previous builds.



— Cloud Build > History (picture by author)

Click on the build name to see all the details for that build including the log output.

**Step 3: Deploy, Expose and Scale the Container**

**Deploying the container to GKE**

Once you’ve containerized your website and uploaded the container to the Google Container Registry, the next step is deploying it onto Kubernetes.

To effectively deploy and administer applications on a GKE cluster, you must communicate with the Kubernetes cluster management system, commonly achieved by using the kubectl command-line tool.

Kubernetes represents applications as [Pods](https://kubernetes.io/docs/concepts/workloads/pods/pod/), which are units that represent a container (or group of tightly-coupled containers). The Pod is the smallest deployable unit in Kubernetes.

To deploy your application, create a [Deployment](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/) resource. The Deployment manages multiple copies of your application, called replicas, and schedules them to run on the individual nodes in your cluster. Deployments ensure this by creating a [ReplicaSet](https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/). The ReplicaSet is responsible for making sure the number of replicas specified are always running.

**Note**: In our example, each Pod contains only our monolith container and the Deployment will be running only one Pod of our application.

Run the following command to deploy your application:

kubectl create deployment monolith --image=gcr.io/${GOOGLE\_CLOUD\_PROJECT}/monolith:1.0.0

The kubectl create deployment command you'll use causes Kubernetes to create a Deployment named monolith on your cluster with **1** replica.

**Note:** As a best practice, using YAML file and a source control system such as GitHub or Cloud Source Repositories is recommended to store those changes. Learn more about these resources from the [Deployments documentation](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/).

**Verifying the Deployment**

kubectl get all

Rerun the command until the pod status is **Running**. We should see something similar to the following output:

NAME READY STATUS RESTARTS AGE  
pod/monolith-7d8bc7bf68-htm7z 1/1 Running 0 6m21s  
  
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
service/kubernetes ClusterIP 10.27.240.1 443/TCP 24h  
  
NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE  
deployment.apps/monolith 1 1 1 1 20m  
  
NAME DESIRED CURRENT READY AGE  
replicaset.apps/monolith-7d8bc7bf68 1 1 1 20m

This output shows:

* The **Deployment**, which is **current**.
* The **ReplicaSet** with desired pod count of **1**.
* The **Pod**, which is **running**.

**Note:** You can also view your Kubernetes deployments via the Console by opening the **Navigation menu** > **Kubernetes Engine** > **Workloads**.

If you are seeing errors or statuses you do not expect, you can debug your resources with the following commands to see detailed information about them:

kubectl describe pod monolith  
  
kubectl describe pod/monolith-7d8bc7bf68-2bxts  
  
kubectl describe deployment monolith  
  
kubectl describe deployment.apps/monolith

At the very end of the output, you will see a list of events that give errors and detailed information about your resources.

**Optional:** You can run commands to your deployments separately as well:

# Show pods  
kubectl get pods  
  
# Show deployments  
kubectl get deployments  
  
# Show replica sets  
kubectl get rs  
  
#You can also combine them  
kubectl get pods,deployments

To fully see the advantages of Kubernetes, we intend to simulate a server crash by deliberately removing a pod, thereby observing the ensuing consequences.

kubectl delete pod/<POD\_NAME>

Note: Replace **<POD\_NAME>** with the name of your pod (e.g. pod/monolith-7d8bc7bf68-htm7z)

If you act fast enough running the ‘***kubectl get all***’ command again, you will be able to observe the presence of two pods: one undergoing termination while the other is either in the creation or running state.

This occurrence is attributed to the ReplicaSet detecting the termination of a pod and consequently initiating the creation of a new pod to maintain the desired replica count.

NAME READY STATUS RESTARTS AGE  
pod/monolith-7d8bc7bf68-2bxts 1/1 Running 0 4s  
pod/monolith-7d8bc7bf68-htm7z 1/1 Terminating 0 9m35s  
  
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
service/kubernetes ClusterIP 10.27.240.1 443/TCP 24h  
  
NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE  
deployment.apps/monolith 1 1 1 1 24m  
  
NAME DESIRED CURRENT READY AGE  
replicaset.apps/monolith-7d8bc7bf68 1 1 1 24m

Later on you will see how to scale out, ensuring multiple instances are operational. This preemptive measure mitigates downtime for users in the event of a pod failure.

**Exposing the GKE deployment**

After deploying our application on GKE, external accessibility is not inherently granted as the containers lack external IP addresses by default. Consequently, accessing the application from outside the cluster is not feasible without additional configuration.

To enable external access to our application, it is imperative to explicitly expose it to Internet traffic through a Service resource. This Service facilitates networking and IP management for our application’s Pods. GKE creates an external IP and a Load Balancer for our application.

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

kubectl expose deployment monolith --type=LoadBalancer --port 80 --target-port 8080

**Accessing the service**

An external IP address is assigned by GKE to the Service resource, not the Deployment.

To obtain the external IP provisioned by GKE for your application, you can examine the Service using the kubectl get service command.

kubectl get service

Re-run the command until your service has an external IP address.

You should see something similar to the following output:

NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
monolith 10.3.251.122 205.0.116.0 80:30877/TCP 3d

After identifying the external IP address assigned to your application, copy the IP address and navigate your browser to the corresponding URL (e.g., “*http://205.0.116.0*") to verify the accessibility of your application. You should see the same website you tested earlier. You now have your website fully running on Kubernetes!

**Scaling the GKE deployment**

With your application now operational on GKE and accessible via the internet, envision its surge in popularity. To accommodate the increasing traffic, scaling your application to multiple instances becomes imperative. In our case, we will scale our deployment up to 3 replicas, by running the following command:

kubectl scale deployment monolith --replicas=3

We can verify if the deployment was scaled successfully using the kubectl get all command. We should see something like the following output:

NAME READY STATUS RESTARTS AGE  
pod/monolith-7d8bc7bf68-2bxts 1/1 Running 0 36m  
pod/monolith-7d8bc7bf68-7ds7q 1/1 Running 0 45s  
pod/monolith-7d8bc7bf68-c5kxk 1/1 Running 0 45s  
  
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
service/kubernetes ClusterIP 10.27.240.1 443/TCP 25h  
service/monolith LoadBalancer 10.27.253.64 XX.XX.XX.XX 80:32050/TCP 6m7s  
  
NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE  
deployment.apps/monolith 3 3 3 3 61m  
  
NAME DESIRED CURRENT READY AGE  
replicaset.apps/monolith-7d8bc7bf68 3 3 3 61m

At this point, you should observe the presence of three instances of your pod in operation. Take note that both your deployment and replica set reflect a desired count of three.

**Step 4: Modify our E-Commerce Application**

**Scenario**: Your marketing team has requested a revision to the homepage of your website. They advocate for a more comprehensive depiction of your company’s identity and the products or services it offers.

**Task**: To meet the marketing team’s expectations, incorporate additional text on the homepage by utilizing the provided file named index.js.new. Duplicate this file and save it as index.js to implement the desired modifications. Follow the instructions below for the necessary steps in effecting these changes.

diff ~/gcp/monolith-to-microservices/react-app/src/pages/Home/index.js ~/gcp/monolith-to-microservices/react-app/src/pages/Home/index.js.new  
30c30  
< <Typography variant="h5">Welcome to the Fancy Store!</Typography>  
---  
> <Typography variant="h5">Fancy Fashion &amp; Style Online</Typography>  
33c33,35  
< Take a look at our wide variety of products.  
---  
> 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!

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

The React components have been modified; however, it’s essential to build the React app to generate the requisite static files.

Run the following command to initiate the build process for the React app and subsequently transfer the generated files into the monolith public directory:

cd ~/gcp/monolith-to-microservices/react-app  
npm run build:monolith

With the code modifications completed, the next step involves rebuilding the Docker container and publishing it to the Google Cloud Container Registry. Utilize the same command as before, but ensure to update the version label accordingly.

cd ~/gcp/monolith-to-microservices/monolith  
gcloud builds submit --tag gcr.io/${GOOGLE\_CLOUD\_PROJECT}/monolith:2.0.0 .

**Step 5: Rollout our New Version (Zero Downtime)**

With the changes finalized and the marketing team satisfied with the updates, the next step is to seamlessly update the website without causing any disruption to the users.

GKE’s rolling update mechanism guarantees the continuous availability of your application throughout the process of replacing instances of the old container image with the new one across all active replicas.

Inform Kubernetes of your intent to update the image for your deployment to a new version by executing the following command:

kubectl set image deployment/monolith monolith=gcr.io/${GOOGLE\_CLOUD\_PROJECT}/monolith:2.0.0

You can validate your deployment update using the kubectl get pods command. The output should be something like the following:

NAME READY STATUS RESTARTS AGE  
monolith-584fbc994b-4hj68 1/1 Terminating 0 60m  
monolith-584fbc994b-fpwdw 1/1 Running 0 60m  
monolith-584fbc994b-xsk8s 1/1 Terminating 0 60m  
monolith-75f4cf58d5-24cq8 1/1 Running 0 3s  
monolith-75f4cf58d5-rfj8r 1/1 Running 0 5s  
monolith-75f4cf58d5-xm44v 0/1 ContainerCreating 0 1s

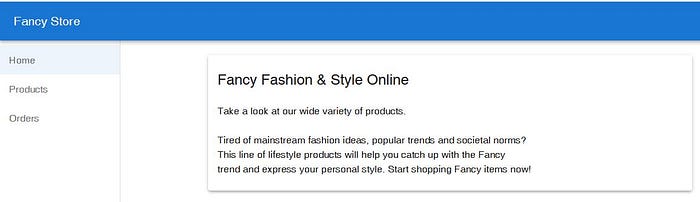
Here, you will observe the creation of three new pods alongside the shutdown of the old ones. You can differentiate them by their age, distinguishing between the new and old pods. Eventually, only three pods will remain, representing your updated instances.

Test the application by running the following command to start the web server:

npm start

To verify our changes, return to the app web page tab you left opened previously and refresh the page. Notice that your application has been updated.

Your website should now be displaying the text you recently incorporated into the homepage, reflecting the updates made to enhance its content!



— E-commerce Application Updated with Zero Downtime (picture by author)

To stop the web server process, press CTRL+C in Cloud Shell.

**Conclusion**

In this article you learned how to successfully deploy, scale, and update without any downtime a monolith e-commerce application onto the Google Kubernetes Engine (GKE). Congratulations on completing these tasks successfully. You are now experienced with Docker and Kubernetes!

In my next article I will show you how to migrate a Monolithic Website to Microservices on Google Kubernetes Engine.

**If you enjoyed this article and found it helpful, please don’t forget to leave a heart** ❤**, comment** 💬**, clap** 👏🏻**, and share** ➦ **it to show your support.**

**Also, don’t forget to** [**connect**](https://www.linkedin.com/in/rogertn/) **and** [**follow me**](https://medium.com/@rogernem) **for more articles. Thank you!**

References:  
- <https://cloud.google.com/learn/training/?hl=en>  
- <https://en.wikipedia.org/wiki/Kubernetes>  
- <https://kubernetes.io/docs/concepts/workloads/pods/pod/>  
- <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>  
- <https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>  
- <https://kubernetes.io/docs/concepts/services-networking/service/>

[Gcp](https://medium.com/tag/gcp?source=post_page-----0ff6d0847a3c---------------gcp-----------------)

[Kubernetes](https://medium.com/tag/kubernetes?source=post_page-----0ff6d0847a3c---------------kubernetes-----------------)

[Docker](https://medium.com/tag/docker?source=post_page-----0ff6d0847a3c---------------docker-----------------)

[React](https://medium.com/tag/react?source=post_page-----0ff6d0847a3c---------------react-----------------)

[Microservices](https://medium.com/tag/microservices?source=post_page-----0ff6d0847a3c---------------microservices-----------------)