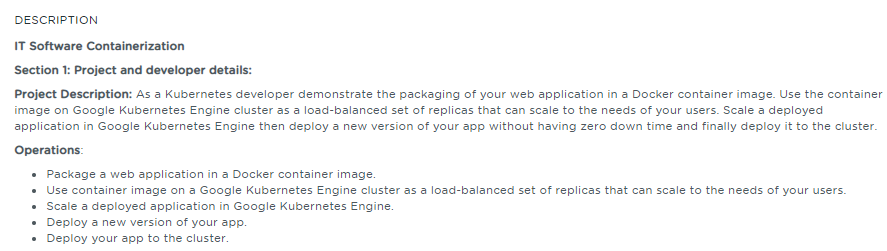
**Deploy Containerized web application in GKE - Assessment**



**Table of Contents**

[**1.** **Introduction** 2](#_Toc45287115)

[**2.** **Work Environment** 2](#_Toc45287116)

[**2.1.** **Prerequisites** 2](#_Toc45287117)

[**2.2.** **Steps to Enable Kubernetes Engine API** 2](#_Toc45287118)

[**2.3.** **Use Google Cloud Shell** 3](#_Toc45287119)

[**3.** **Deploying a Containerized Web application** 4](#_Toc45287120)

[**3.1.** **Download and Configure the Project in Google Cloud** 4](#_Toc45287121)

[**3.2.** **Build the web application as a Docker image** 5](#_Toc45287122)

[**3.3.** **Run a container application locally** 6](#_Toc45287123)

[**3.4.** **Upload the Docker image to Container registry** 6](#_Toc45287124)

[**3.5.** **Create a Google Kubernetes Engine (GKE) Cluster** 7](#_Toc45287125)

[**3.6.** **Deploy an application to Google Kubernetes Engine (GKE) Cluster** 9](#_Toc45287126)

[**3.7.** **Expose the web application to the internet** 11](#_Toc45287127)

[**3.8.** **Deploy a new version of the web application** 14](#_Toc45287128)

[**4.** **References** 18](#_Toc45287129)

# **Introduction**

The intent of this project is to perform the following activities

* Package a web application in a Docker container image,
* Run that container application locally.
* Upload the Docker image to Container Registry
* Create a Google Kubernetes Engine (GKE) cluster
* Deploy the web application as a load-balanced set of replicas that can scale to the needs of your users.
* Expose the web application to the internet
* Deploy a new version of the web application

# **Work Environment**

## **Prerequisites**

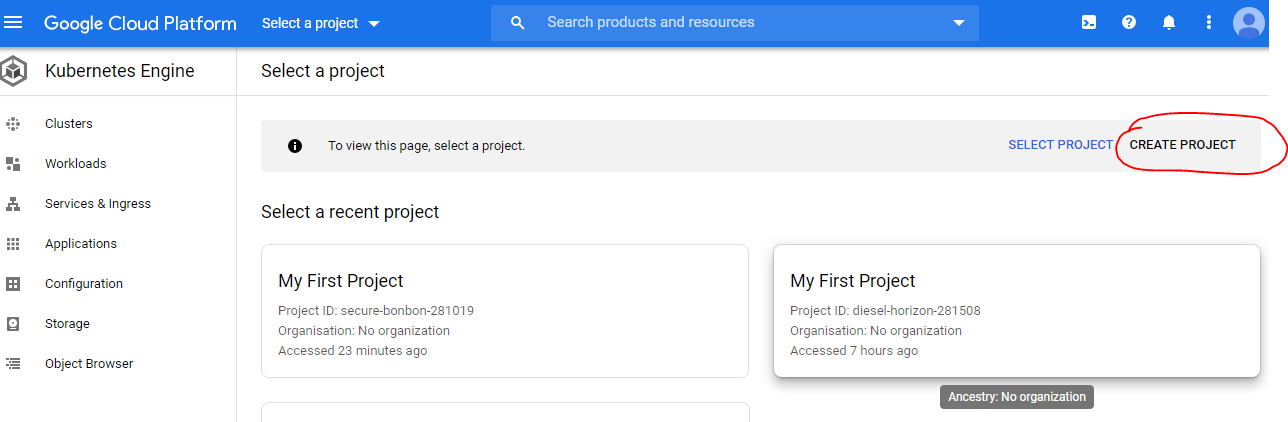
To build and deploy the container, the following software/tools have been used

* **Docker** –A lightweight Containerization platform
* **NodeJS** – JavaScript runtime built on Chrome's V8 JavaScript engine to build the sample web application
* [**Kubernetes**](https://kubernetes.io/)**-** An open-source platform for automating deployment, scaling, and management of containerized applications.
* **Google Cloud Platform Subscription or Trial account**

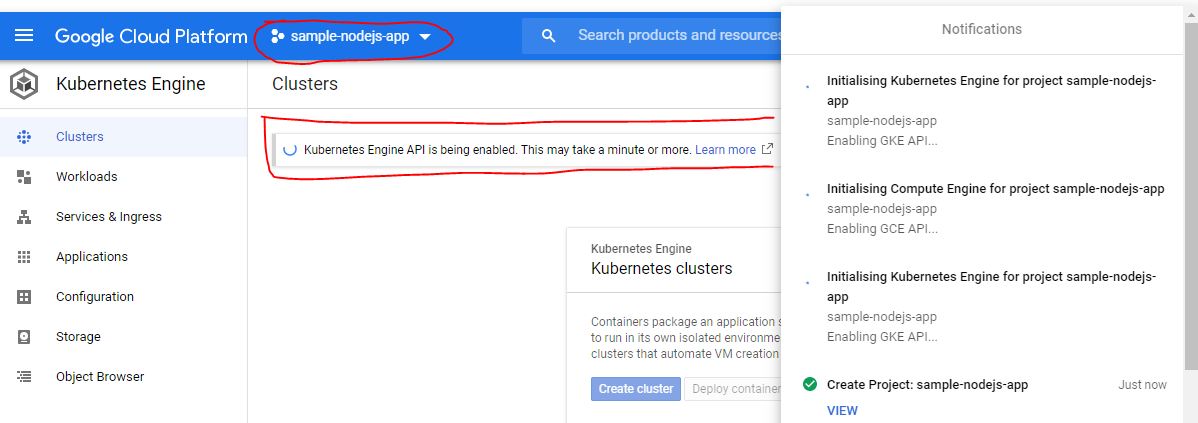
## **Steps to Enable Kubernetes Engine API**

Perform the below steps to enable the Kubernetes Engine API.

* Visit the [Kubernetes Engine page](https://console.cloud.google.com/projectselector/kubernetes?_ga=2.223049375.545394318.1593773550-1905346242.1592210676&_gac=1.123799032.1593778404.CjwKCAjwrvv3BRAJEiwAhwOdM1v2xX4LU1eEvpqiGkMDpI1HO0Xi3sz1vGbhbirfZdPkyj293Rgt0BoC8mwQAvD_BwE) in the Google Cloud Console.
* Create a new project as shown in the below figure



The new project named sample-nodejs-app is created and it may take few minutes to enable the Kubernetes Engine API for this project as shown below in the figure.

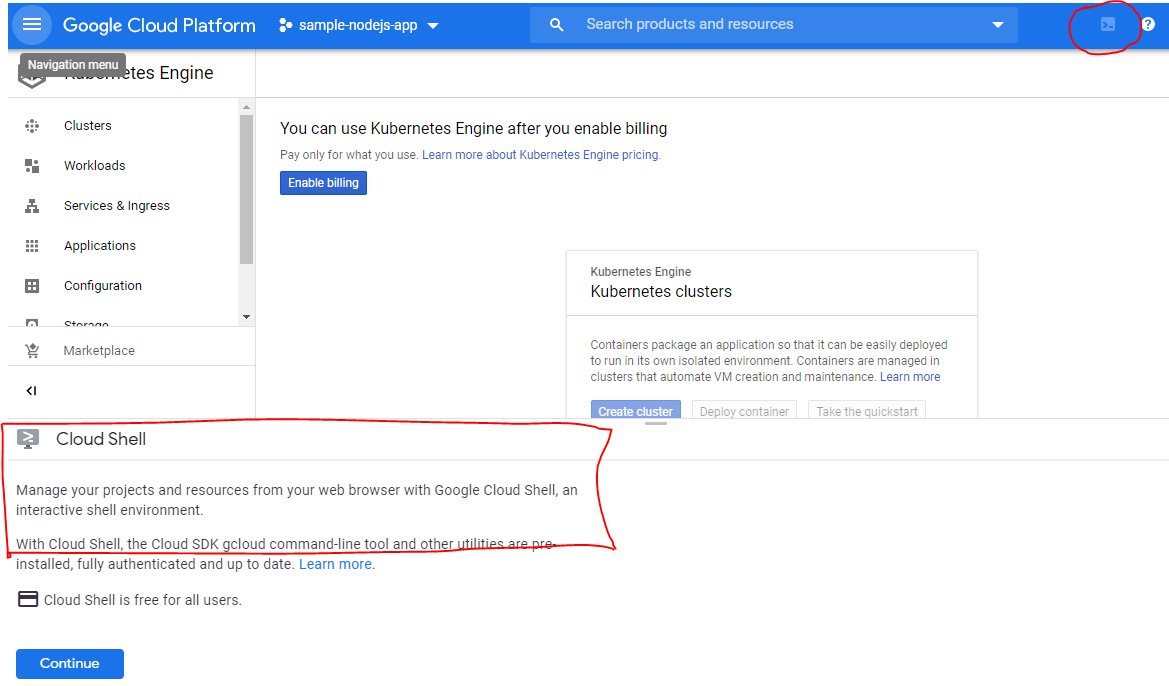


**Note**

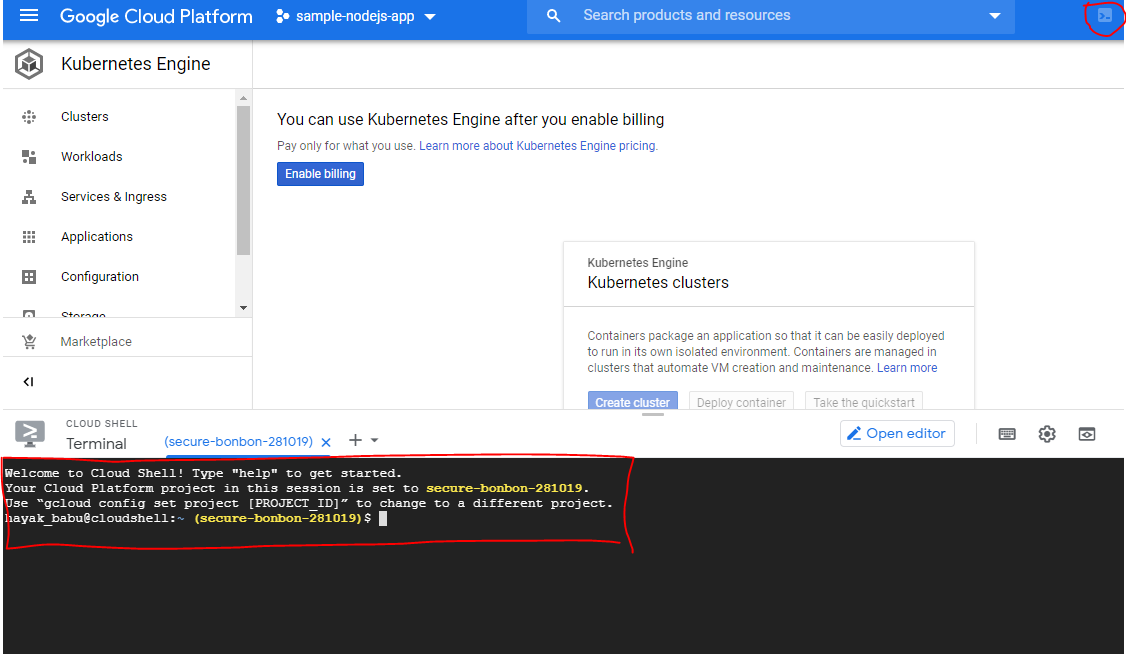
To perform the below steps, one should have the Google Cloud Platform Subscription or Trial account

## **Use Google Cloud Shell**

Google Cloud Shell comes preinstalled with gcloud, docker and kubectl command line tools. To use this option, Activate Cloud Shell button at the top of the Cloud Console Window as shown in the below figure



After clicking Continue, Google Cloud Shell gets provisioned and the Shell window appears at the bottom of the window as shown below in the figure:



**Note**:

Other option is to use command-line tools locally i.e Install the Google Cloud SDK, install the kubectl command line utility and Docker CE version to build a container image for the application

To build this project, I have selected Google Cloud Shell option.

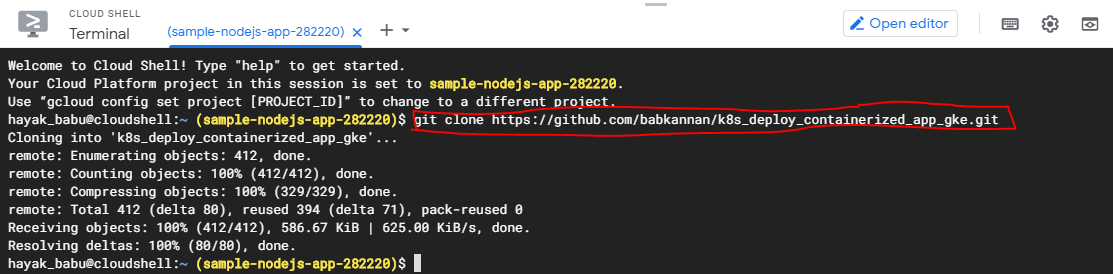
# **Deploying a Containerized Web application**

## **Download and Configure the Project in Google Cloud**

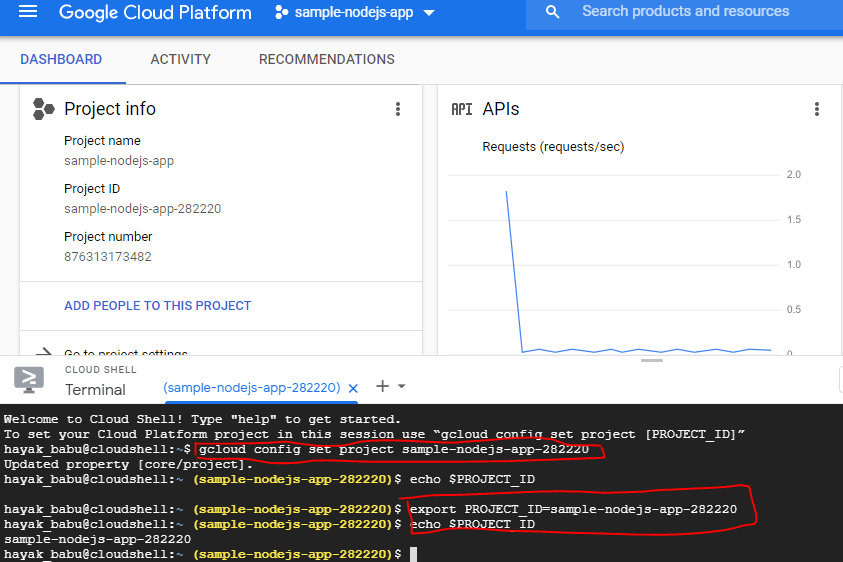
To deploy the sample web application, I have used the application called **simple-nodejs-app** which is written using NodeJS that responds to the requests with the message ***Hello World!*** on port 8080.

Download the **simple-nodejs-app** sources from my public GitHub repository

Below screenshot shows on getting the simple-nodejs-app project from the GitHub public repository using the Google Cloud Shell



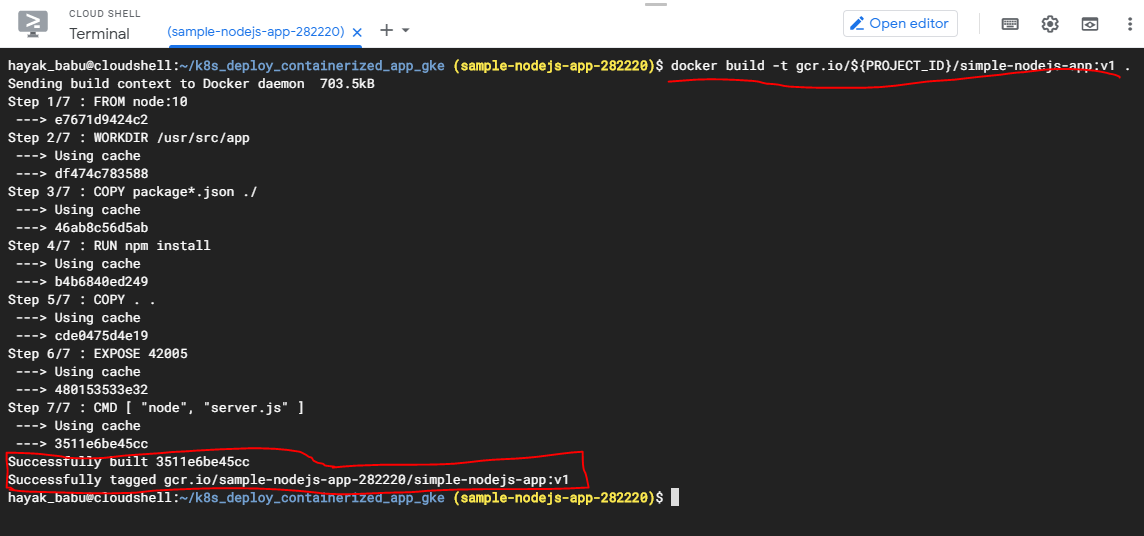
Below screenshot shows on how to set the PROJECT\_ID and configure the project in the Google Cloud Shell:



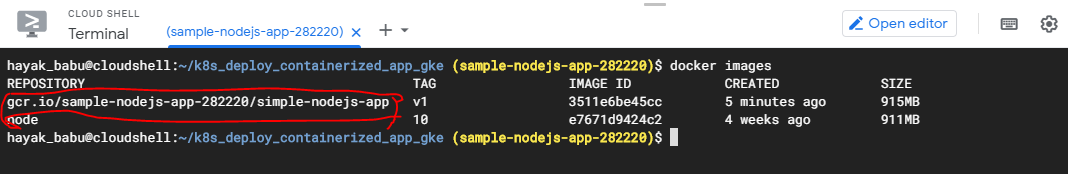
## **Build the web application as a Docker image**

First we need to package the **simple-nodejs-app** source code as a Docker image.

To build a Docker image, you need source code and a [Dockerfile](https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/tree/master/hello-app/Dockerfile). Build and tag the Docker image for **simple-nodejs-app** as shown in the below figure:



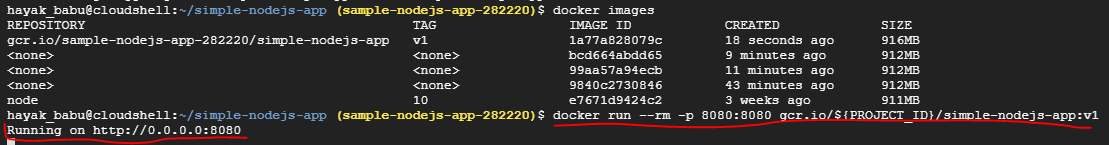
Below screenshot shows the application image after running the command *docker images* command to verify that the build was successful:



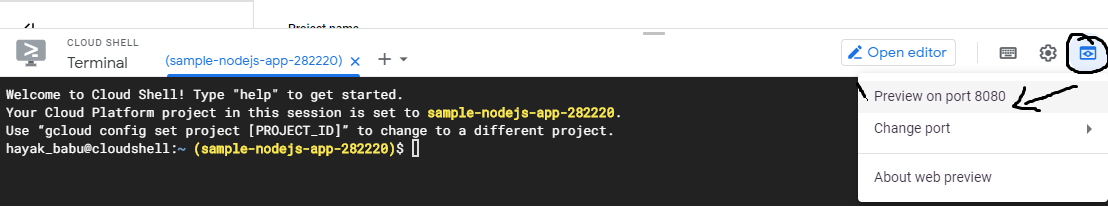
## **Run a container application locally**

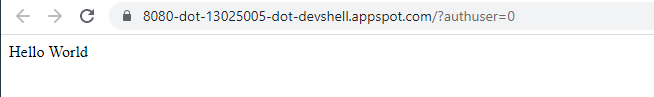
Now it’s time to test your container image using your local Docker engine:

Below screenshot shows the execution of simple-nodejs-app running container



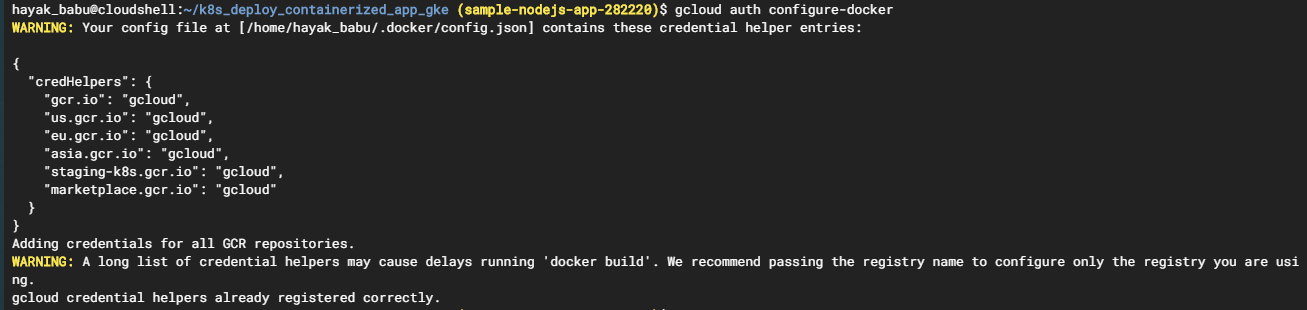
We can preview the running container on port 8080 as shown below. Using the Web Preview feature of Google Cloud Shell, we can see the output of the container in the browser.



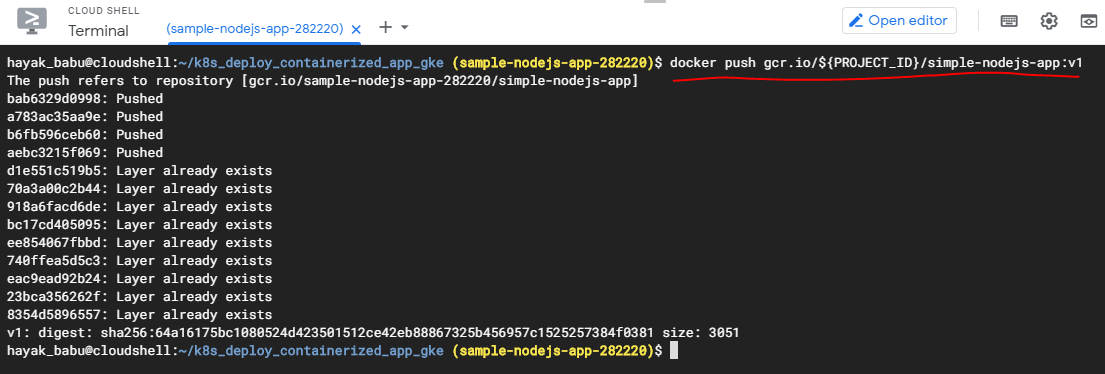


## **Upload the Docker image to Container registry**

Configure the Docker command-line tool to authenticate to [Container Registry](https://cloud.google.com/container-registry) as shown below in the screenshot:



Below Screenshot shows on how to upload the container image to a registry so that your GKE cluster can download and run it

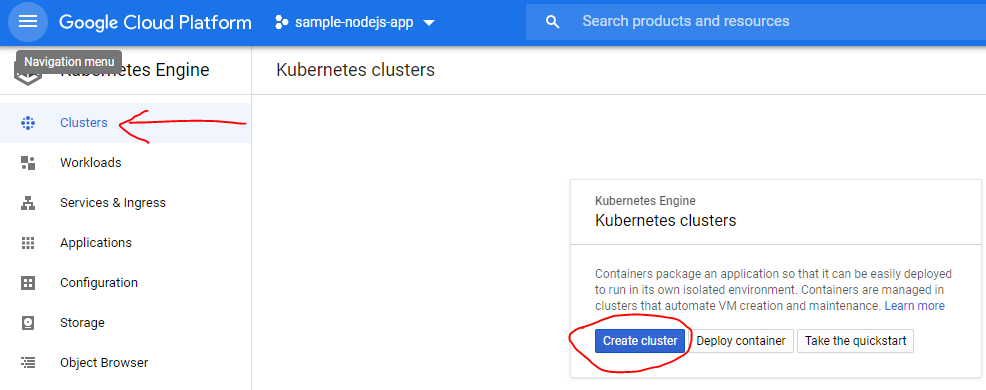


## **Create a Google Kubernetes Engine (GKE) Cluster**

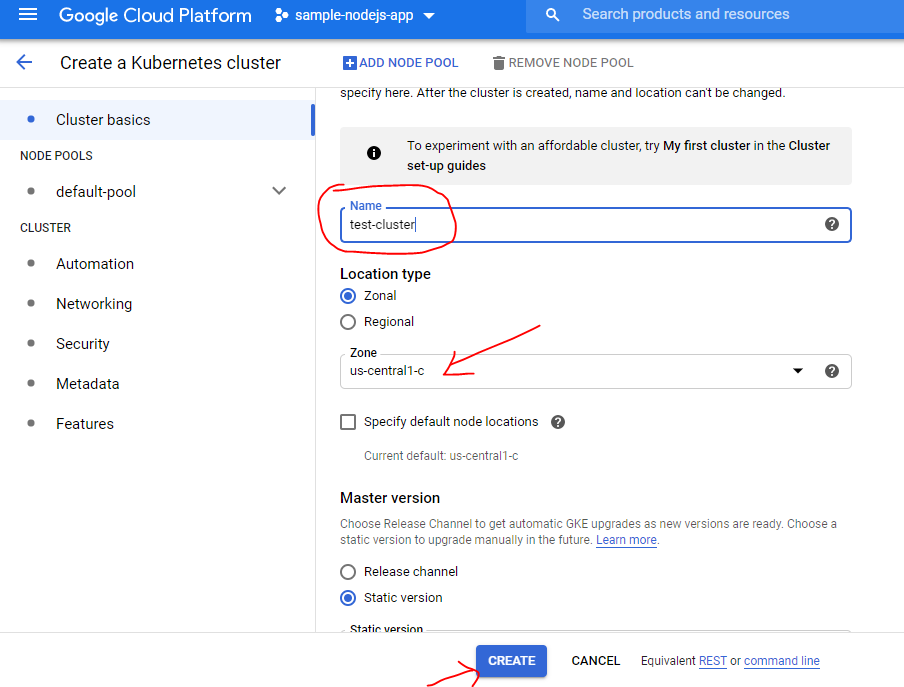
Now that the Docker image is stored in Container Registry, we need to create a GKE [cluster](https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture) to run simple-nodejs-app. A GKE cluster consists of a pool of [Compute Engine VM instances](https://cloud.google.com/compute) running [Kubernetes](https://kubernetes.io/), the open source cluster orchestration system that powers GKE.

Visit the Google Kubernetes Engine menu in Cloud Console and perform the below steps as shown below in the screenshots.

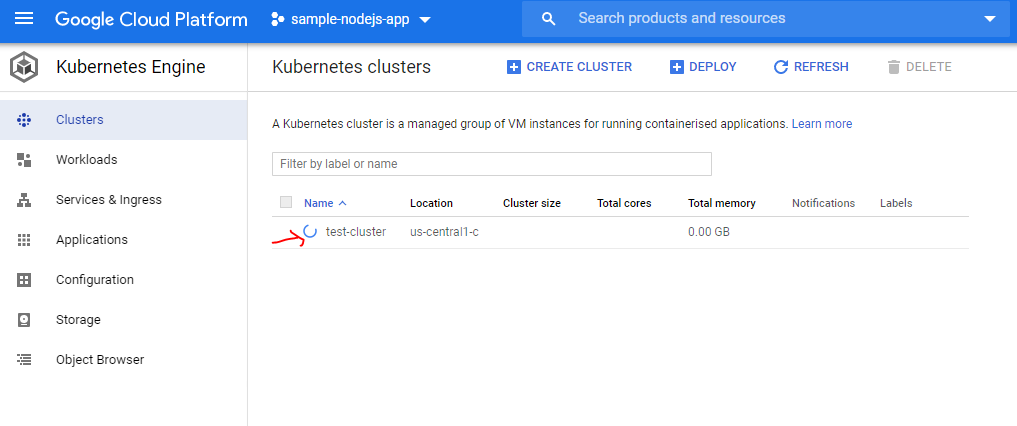
**Step 1 –** Click Create Cluster



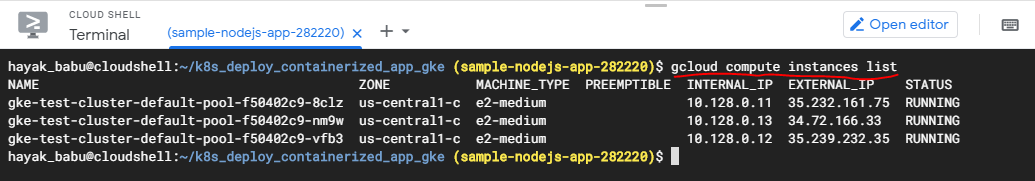
**Step 2 –** Configure Cluster. Provide name, Location Type



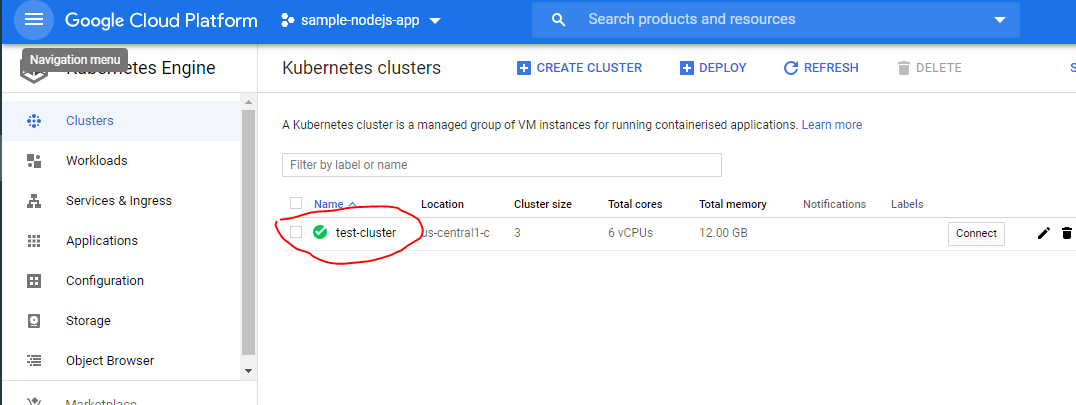
**Step 3 –** Below process shows the view of Cluster creation process. It may take few minutes to create the Cluster of the requested type



**Step 4 -** Below screenshot shows the command to view the the cluster's worker VM instances:



**Step 5 -** Below screenshot shows the **test-cluster** creation in the Google Cloud Platform. **-** It may take few minutes to create the Cluster of the requested type

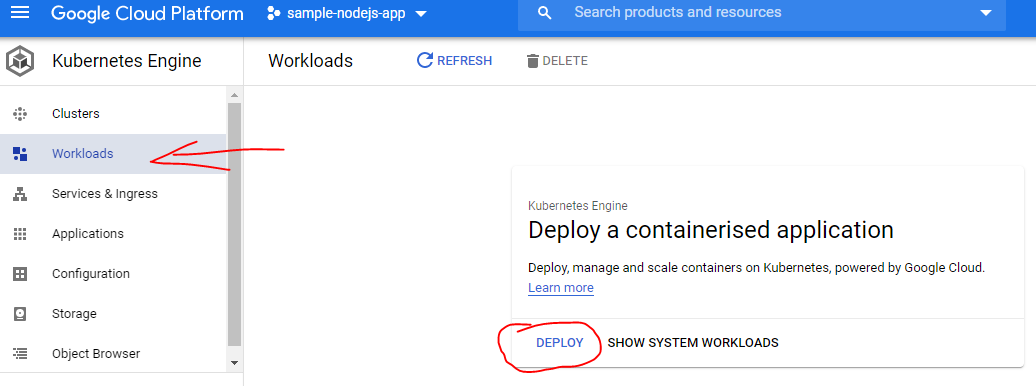


## **Deploy an application to Google Kubernetes Engine (GKE) Cluster**

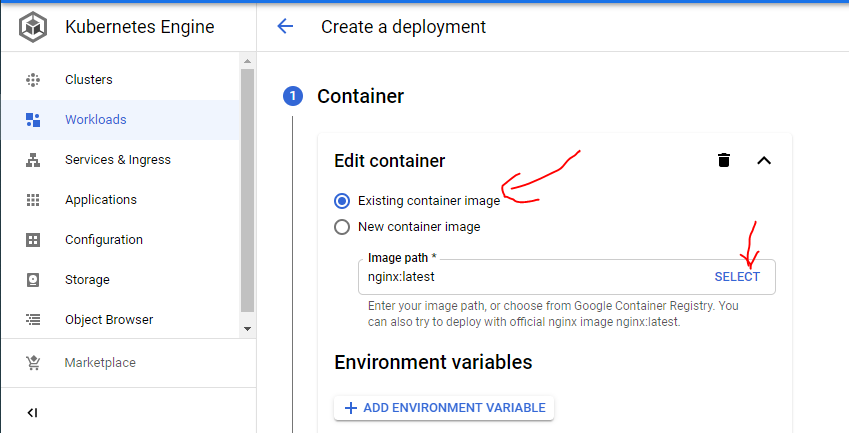
We are now ready to deploy the Docker image we built to your GKE cluster.

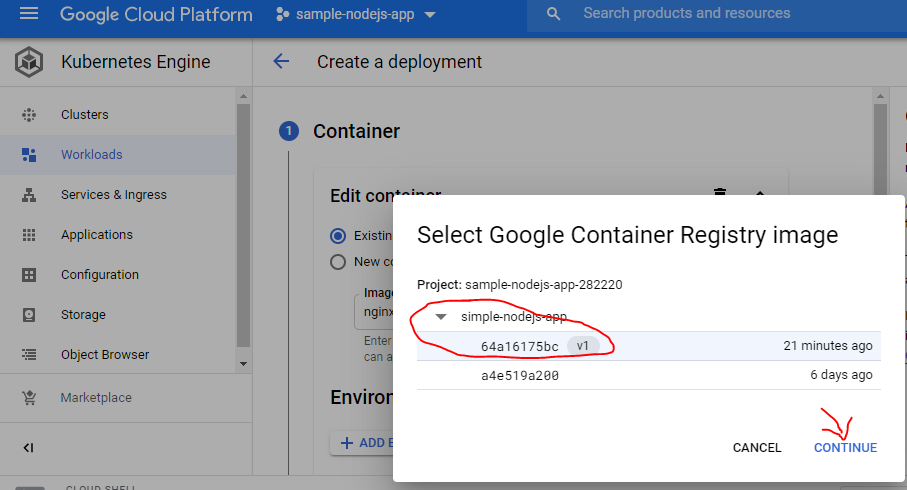
Visit the Google Kubernetes Engine **Workloads** menu in Cloud Console to perform the below steps for deploying the application into GKE cluster

**Step 1 -** Below screenshot shows the first step in Deploying the application in the “Workloads” section menu as shown below:

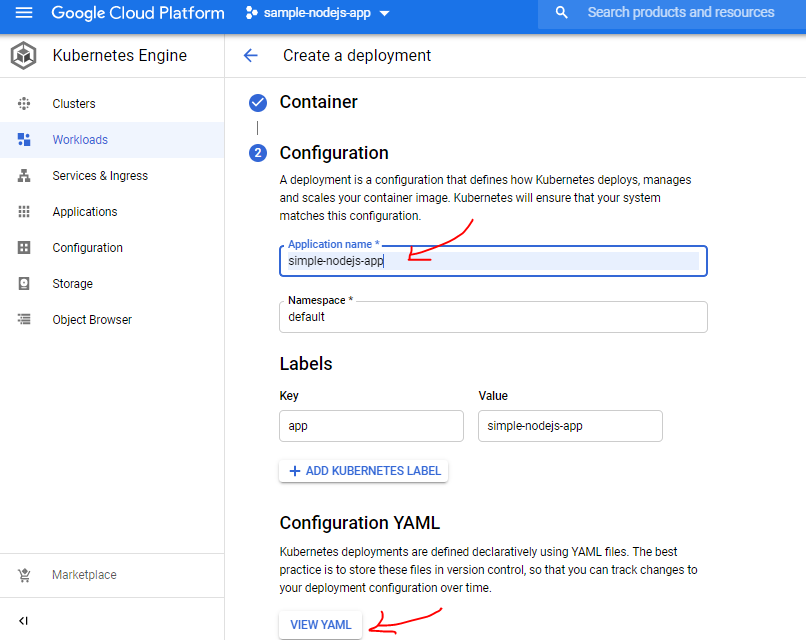


**Step 2 -** Below screenshot shows the second step in Deploying the application in the “Workloads” section menu. Here we can select an existing image as shown below:

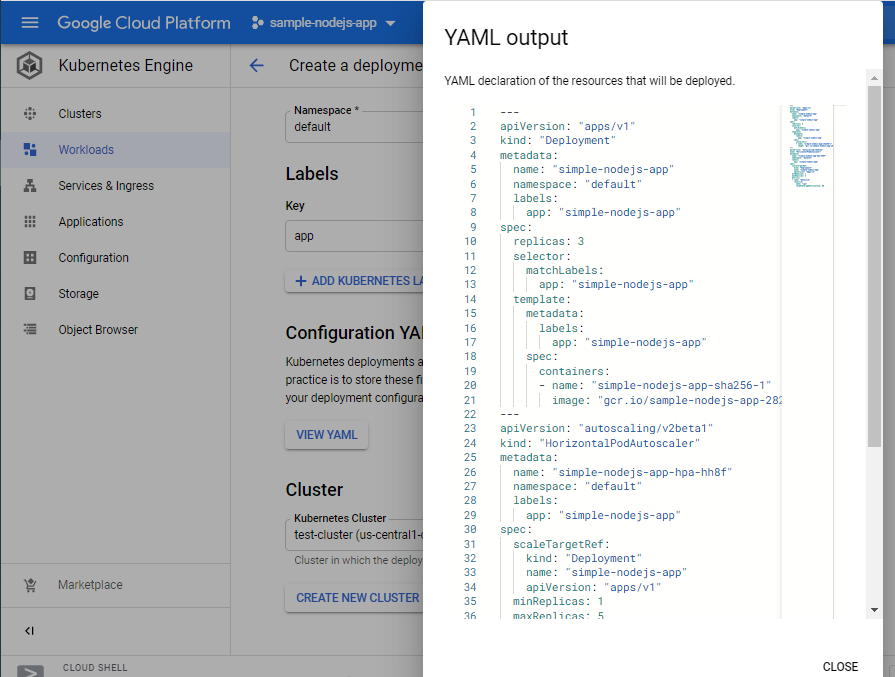




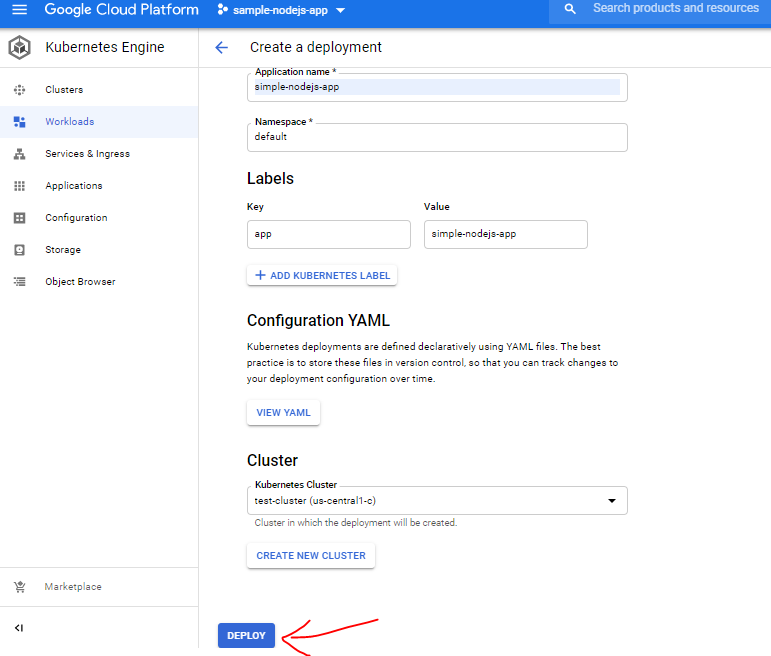
**Step 3 -** Below screenshot shows the third step of Deploying the application in the “Workloads” section menu. Here we can name our deployment like “simple-nodejs-app” and also we can view the YAML file for the deployment as shown below:



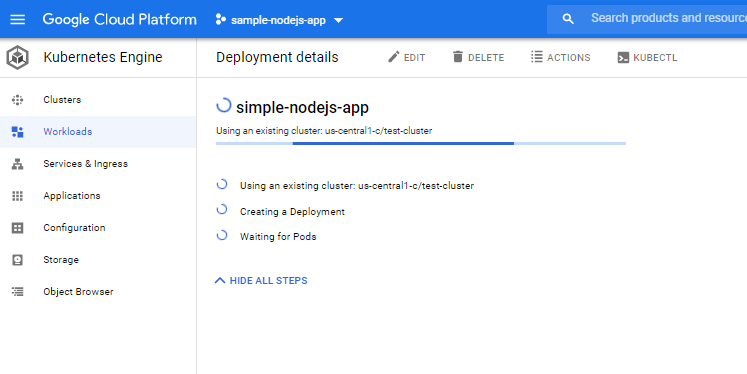
**Step 4 –** To view the YAML file



**Step 5 -** Below screenshot shows the fifth step of Deploying the application in the “Workloads” section menu using the “Deploy” button as shown below:



**Step 6 -** Below screenshot shows the sixth step of Deployment creation process and the pods were created after the successful deployment as shown below:

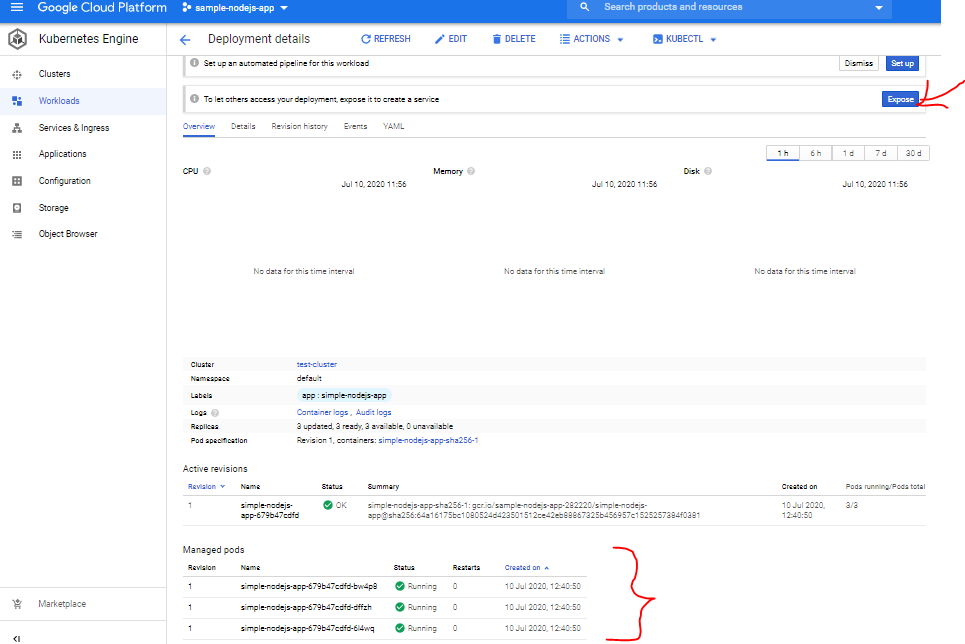


## **Expose the web application to the internet**

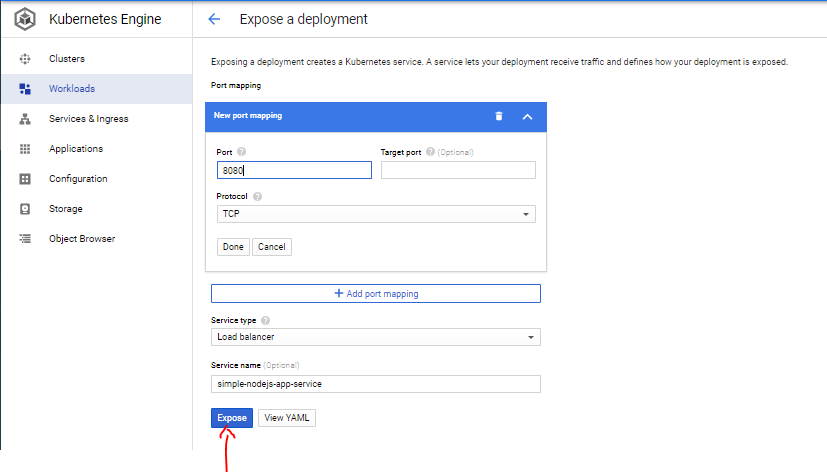
Next step is to expose the web application to the internet i.e it can be accessible outside of the cluster.

Perform the below steps to expose the deployment of web application.

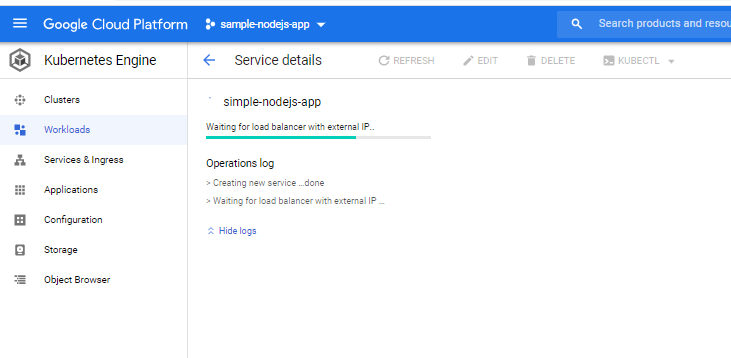
**Step 1** – Click on **simple-nodejs-app.** From the Deployment details page, click **Expose** as shown below in the screenshot. This will allow your deployment to receive traffic and defines how your deployment is exposed



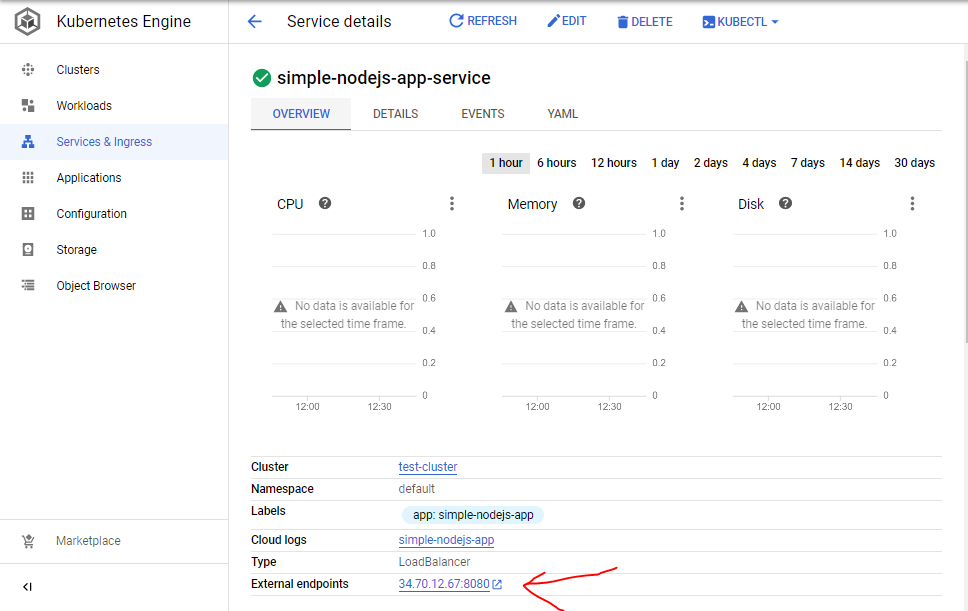
**Step 2** - From the Expose a Deployment menu, set the **Target Port** to 8080. This is the port the **simple-nodejs-app** container listens on as shown below in the screenshot

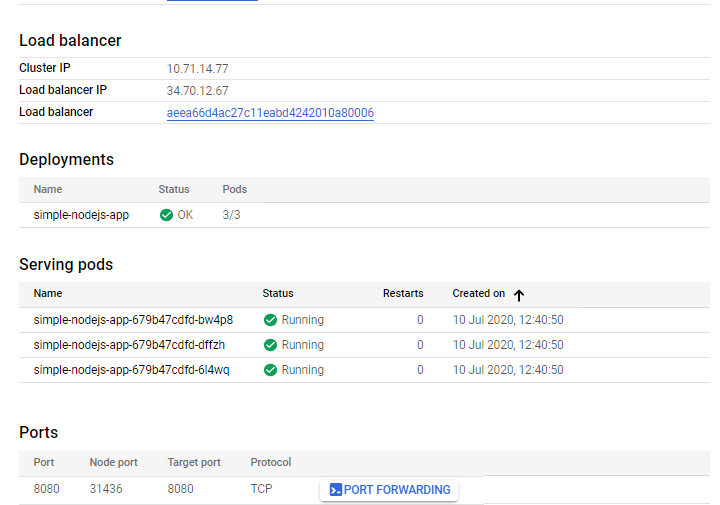


**Step 3 -** Below screenshots shows the service creation process. Once the Load Balancer is ready, you can view the External endpoints along with the port as shown in the below screenshots

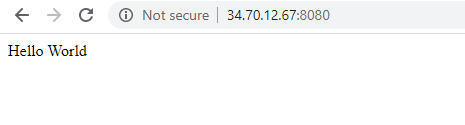


**Step 4 -** Scroll down to the **External endpoints** field, and copy the address to the clipboard





**Step 5 -** Now that the **simple-nodejs-app** pods are exposed to the internet via a Kubernetes Service, you can open a new browser tab, and navigate to the Service IP address you copied to the clipboard. You should see a Hello, World! message.

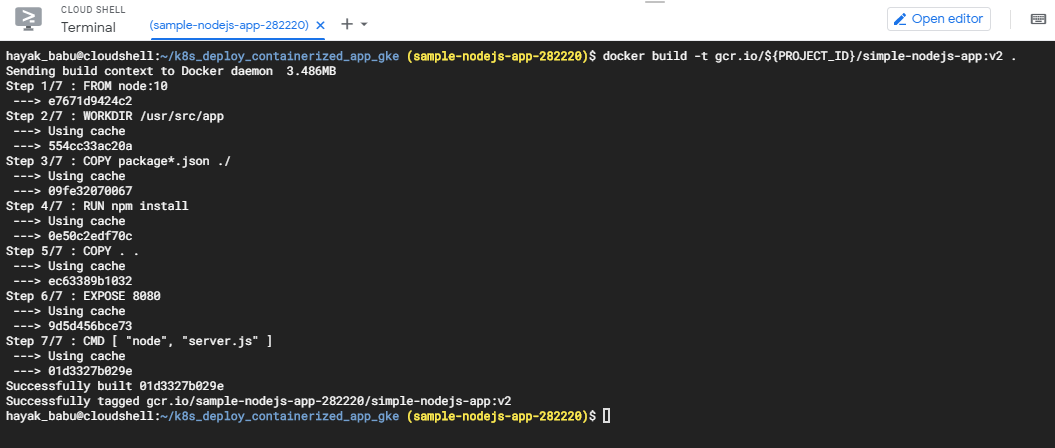


## **Deploy a new version of the web application**

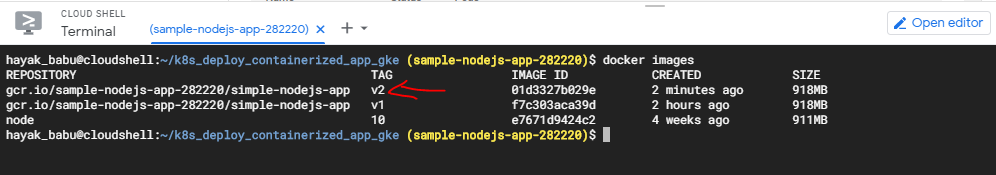
In this section, we will upgrade **simple-nodejs-app** to a new version by building and deploying a new Docker image to your GKE cluster.

GKE's [rolling update](https://cloud.google.com/kubernetes-engine/docs/how-to/updating-apps) feature allows to update our Deployments without downtime. During a rolling update, your GKE cluster will incrementally replace the existing simple-nodejs-app Pods with Pods containing the Docker image for the new version. During the update, your load balancer service will route traffic only into available Pods.

**Step 1 -** Below screenshot shows the simple-nodejs-app image creation process for the updated sources



**Step 2 -** Below screenshot shows the simple-nodejs-app images list

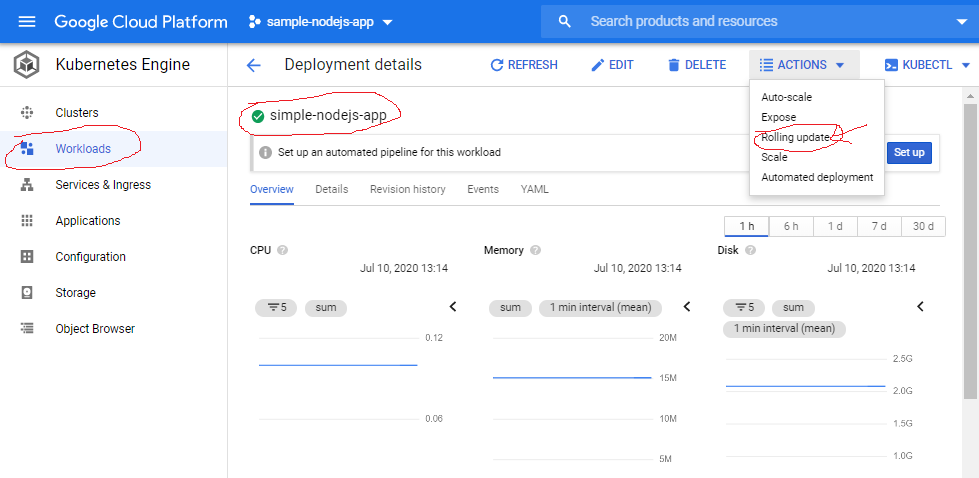


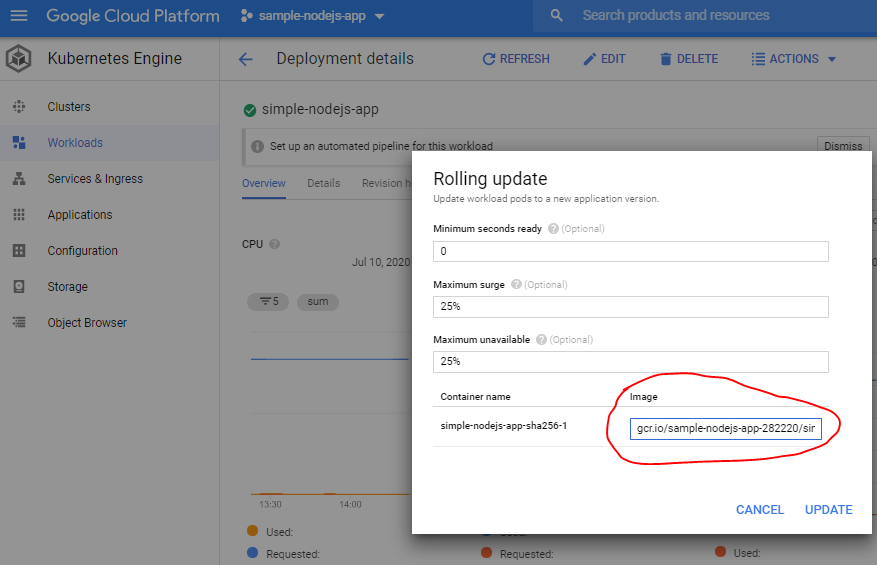
**Step 3 -** Below screenshot shows the updated version (v2) of simple-nodejs-app image is pushed to the Container registry



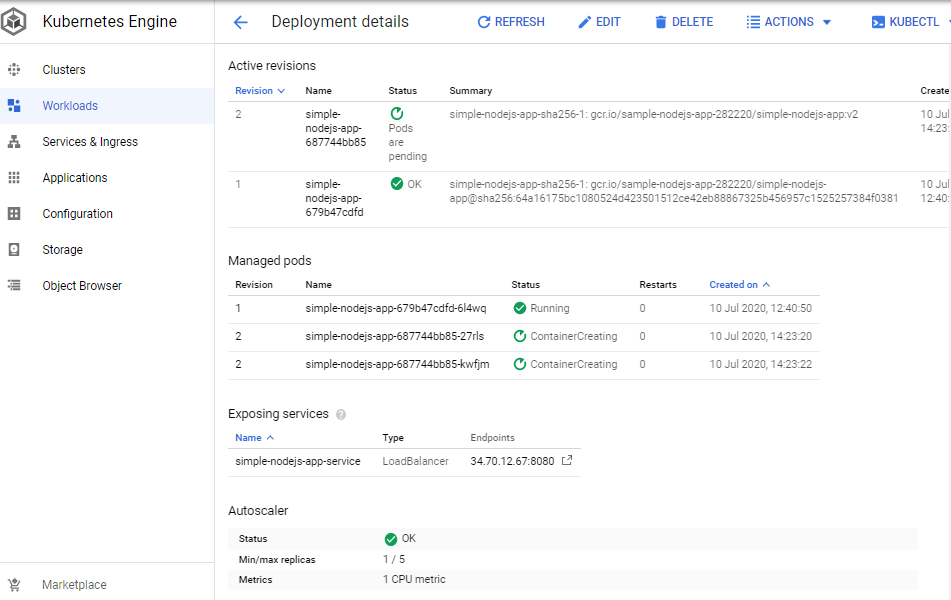
Now we are ready to update your simple-nodejs-app Kubernetes Deployment to use a new Docker image.

**Step 4 -** Below screenshot shows on the usage of “Rolling update” under the Actions menu. Here we need to provide the image path corresponds to the Revision 2 in the image field as shown below **gcr.io/sample-nodejs-app-282220/simple-nodejs-app:v2**

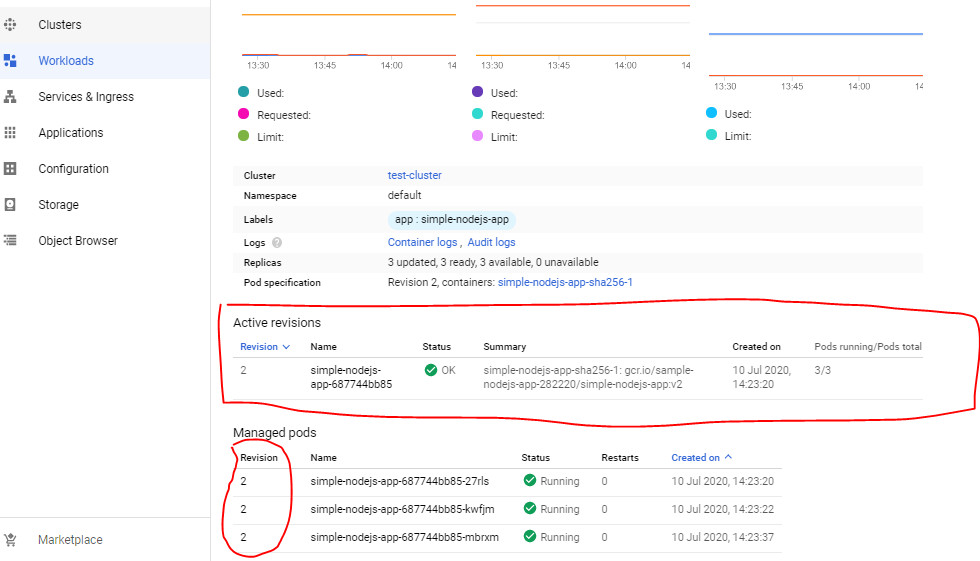




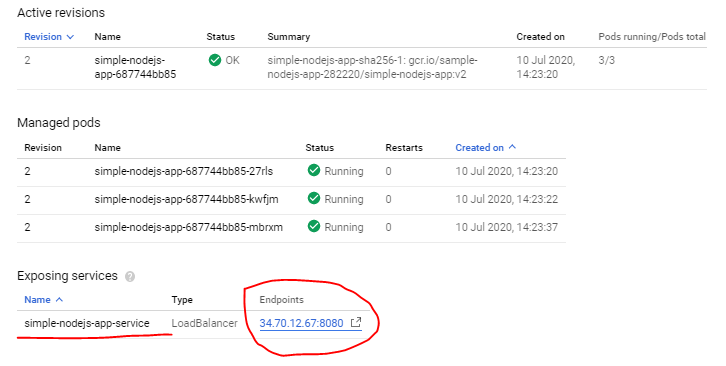
**Step 5 -** Below screenshot shows the Rolling update process i.e two Revisions, 1 and 2. Revision 1 corresponds to the initial Deployment we created earlier. Revision 2 is the rolling update you just started.

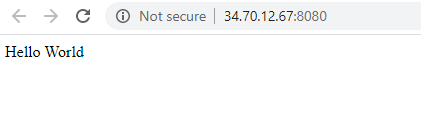


**Step 6 -** Below screenshot shows the Active revision version of **simple-nodejs-app** pointing to v2 under **Managed Pods**



**Step 7 -** Below screenshot shows the **simple-nodejs-app** service corresponds to Revision 2





# **References**

For more details on the details of the workflow, Refer the Writeup document - **Deploy\_Containerized\_Application\_in\_GKE\_writeup.docx**

For the sample web application sources and its link, Refer the Source document **Deploy\_Containerized\_Application\_in\_GKE\_sources.docx**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Components** | **Reference** |
| 1 | Writeup document | Deploy\_Containerized\_Application\_in\_GKE\_writeup.docx |
| 2 | Sources document | Deploy\_Containerized\_Application\_in\_GKE\_sources.docx |
| 3 | Google k8s document | <https://cloud.google.com/kubernetes-engine/docs/tutorials/hello-app> |