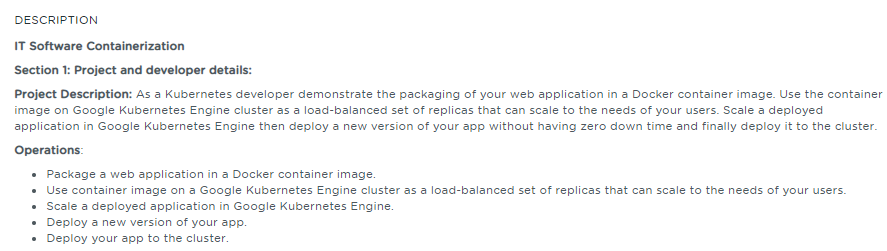
**Deploy Containerized web application in GKE - Assessment**



**Table of Contents**

[**1.** **Introduction** 2](#_Toc45286842)

[**2.** **Work Environment** 2](#_Toc45286843)

[**2.1.** **Prerequisites** 2](#_Toc45286844)

[**2.2.** **Steps to Enable Kubernetes Engine API** 2](#_Toc45286845)

[**2.3.** **Use Google Cloud Shell** 3](#_Toc45286846)

[**3.** **Deploying a Containerized Web application** 4](#_Toc45286847)

[**3.1.** **Download and Configure the Project in Google Cloud** 4](#_Toc45286848)

[**3.2.** **Build the web application as a Docker image** 5](#_Toc45286849)

[**3.3.** **Run a container application locally** 6](#_Toc45286850)

[**3.4.** **Upload the Docker image to Container registry** 6](#_Toc45286851)

[**3.5.** **Create a Google Kubernetes Engine (GKE) Cluster** 7](#_Toc45286852)

[**3.6.** **Deploy an application to Google Kubernetes Engine (GKE) Cluster** 7](#_Toc45286853)

[**3.7.** **Expose the web application to the internet** 8](#_Toc45286854)

[**3.8.** **Deploy a new version of the web application** 8](#_Toc45286855)

[**4.** **References** 10](#_Toc45286856)

# **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**

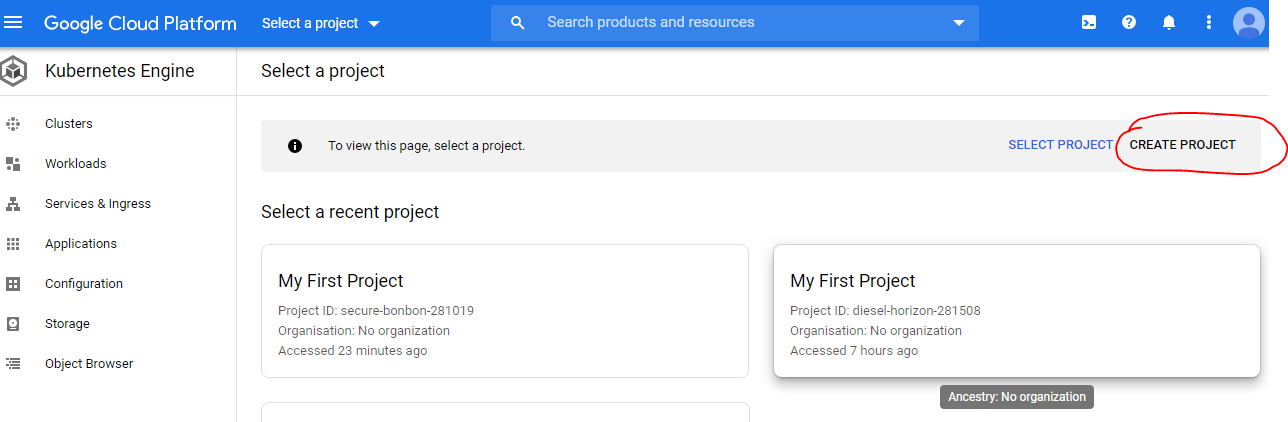
To build and deploy the container, the following software/tools have been used

* **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**

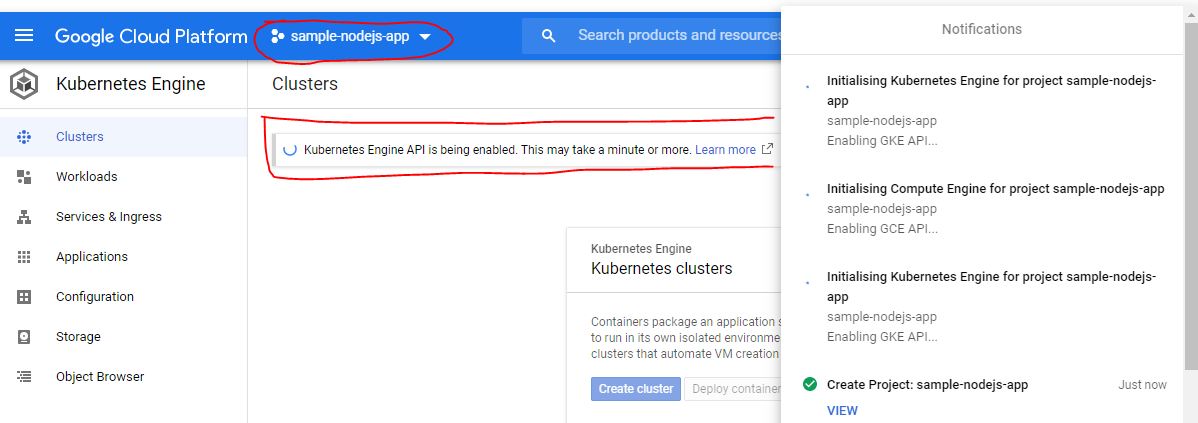
## **Steps to Enable Kubernetes Engine API**

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.

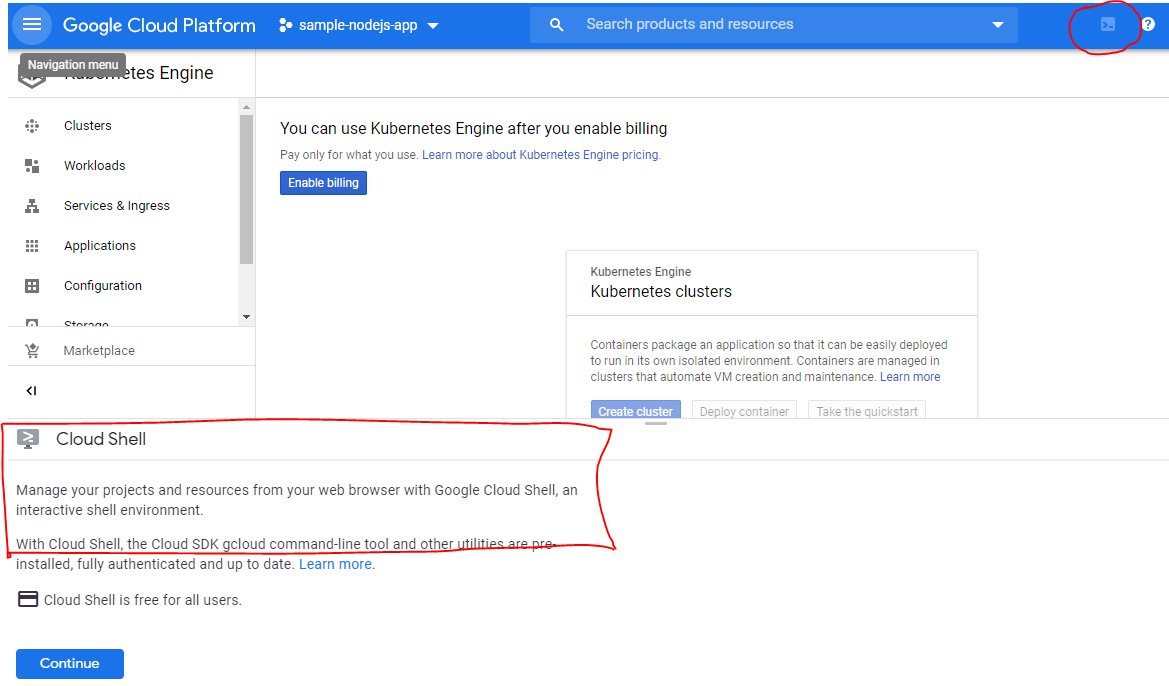


**Note**

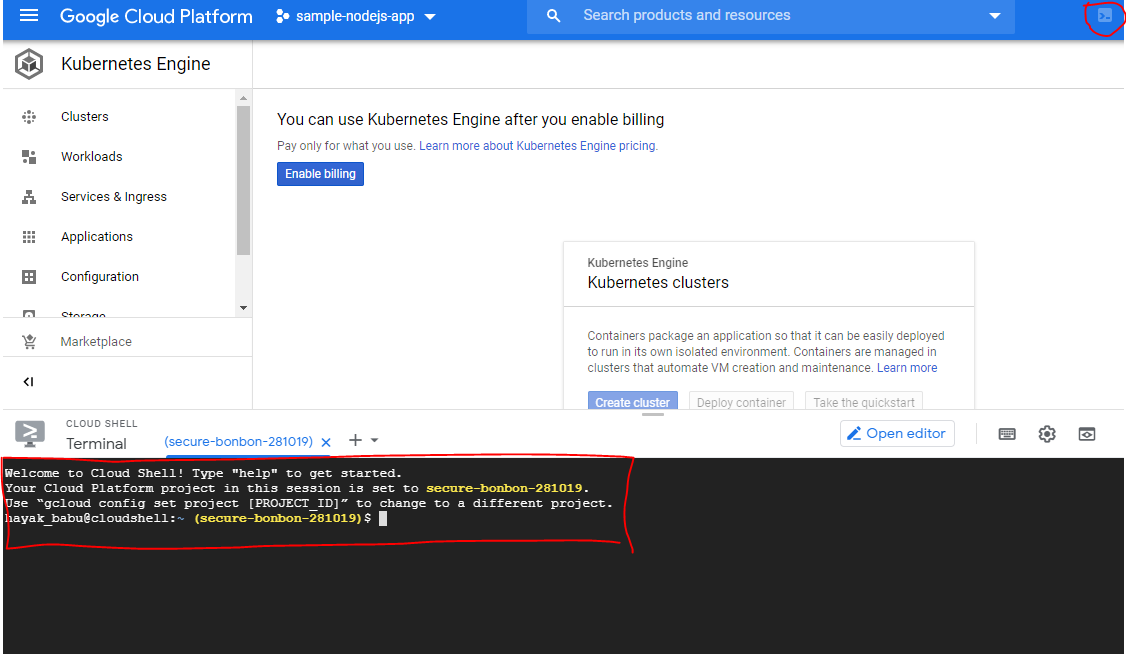
To perform the below steps, one should have the Google Cloud Platform Subscription or Trial account

## **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

|  |
| --- |
| $ git clone https://github.com/babkannan/k8s\_deploy\_containerized\_app\_gke.git |

Configure and set your project in the Google Cloud Shell

|  |
| --- |
| $ gcloud config set project sample-nodejs-app-282220 |

Set the PROJECT\_ID environment variable to your [Google Cloud project ID](https://cloud.google.com/resource-manager/docs/creating-managing-projects#identifying_projects) (*project-id*). The PROJECT\_ID variable will be used to associate the container image with your project's [Container Registry](https://cloud.google.com/container-registry).

|  |
| --- |
| $ export PROJECT\_ID=sample-nodejs-app-282220 |

For more details on the user workflow, please refer the **Deploy\_Containerized\_Application\_in\_GKE\_screenshots.docx**

## **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**

|  |
| --- |
| $ docker build –t gcr.io/${PROJECT\_ID}/simple-nodejs-app:v1 . |

The output is similar to this:

**Sending build context to Docker daemon 127.5kB**

**Step 1/7 : FROM node:10**

**---> e7671d9424c2**

**Step 2/7 : WORKDIR /usr/src/app**

**---> Using cache**

**---> 0b43ec2560e8**

**Step 3/7 : COPY package\*.json ./**

**---> Using cache**

**---> 582f1cff39a6**

**Step 4/7 : RUN npm install**

**---> Using cache**

**---> 6d0e2137e6fa**

**Step 5/7 : COPY . .**

**---> Using cache**

**---> dd9be80c336b**

**Step 6/7 : EXPOSE 32000**

**---> Using cache**

**---> 11899fcf26c6**

**Step 7/7 : CMD [ "node", "server.js" ]**

**---> Using cache**

**---> 9840c2730846**

**Successfully built 9840c2730846**

**Successfully tagged gcr.io/sample-nodejs-app-282220/simple-nodejs-app:v1**

The above command instructs Docker to build the image using the Dockerfile in the current directory and tag it with a name, such as ***gcr.io/sample-nodejs-app-282220/simple-nodejs-app:v1***. The gcr.io prefix refers to [Container Registry](https://cloud.google.com/container-registry), where the image will be hosted. Running this command does not upload the image yet.

Run the *docker images* command to verify that the build was successful:

|  |
| --- |
| $ docker images |

The output is similar to this:

**REPOSITORY TAG IMAGE ID CREATED SIZE**

**gcr.io/sample-nodejs-app-282220/simple-nodejs-app v1 9840c2730846 17 minutes ago 912MB**

**node**

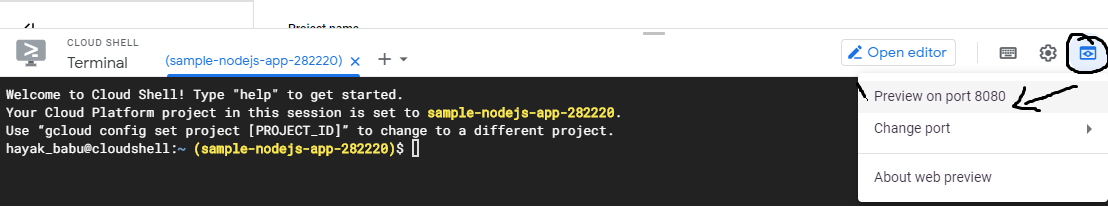
## **Run a container application locally**

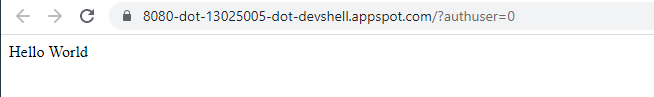
Now it’s time to test your container image using your local Docker engine:

Run the below command to check the container is running

|  |
| --- |
| $ docker run --rm -p 8080:8080 gcr.io/${PROJECT\_ID}/simple-nodejs-app:v1 |

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):

Run the below command for the readiness of Container registry

|  |
| --- |
| $ gcloud auth configure-docker |

Upload the container image to a registry so that your GKE cluster can download and run it

|  |
| --- |
| $ docker push gcr.io/${PROJECT\_ID}/simple-nodejs-app:v1 |

The output is similar to this:

**The push refers to repository [gcr.io/sample-nodejs-app-282220/simple-nodejs-app]**

**cba46ecb3493: Pushed**

**11f42552d22e: Pushed**

**107cb5d158c2: Pushed**

**e4b702b6dff8: Pushed**

**d1e551c519b5: Layer already exists**

**70a3a00c2b44: Layer already exists**

**918a6facd6de: Layer already exists**

**bc17cd405095: Layer already exists**

**ee854067fbbd: Layer already exists**

**740ffea5d5c3: Layer already exists**

**eac9ead92b24: Layer already exists**

**23bca356262f: Layer already exists**

**8354d5896557: Layer already exists**

**v1: digest: sha256:a4e519a200b74badee3cc73b9bc82c8264cfe3f88946836daec50c1c469853d4 size: 3051**

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

Now that the Docker image is stored in Container Registry, you 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.
* Click the **Create cluster** button.
* In the **Cluster basics** section, enter the name **test-cluster**.
* Set the [**zone**](https://cloud.google.com/compute/docs/zones#available) by choosing a Compute Engine zone from the drop-down menu.
* Click **Create**. This will create a GKE cluster with 3 nodes.
* Wait for the cluster to be created. When the cluster is ready, a green check mark will appear next to the cluster name.

It will take a few minutes for your GKE cluster to be created and health-checked.

After the command completes, run the following command to see the cluster's two worker VM instances:

|  |
| --- |
| $ gcloud compute instances list |

The output is similar to this:

**NAME ZONE MACHINE\_TYPE PREEMPTIBLE INTERNAL\_IP EXTERNAL\_IP STATUS**

**gke-test-cluster-default-pool-f50402c9-8clz us-central1-c e2-medium 10.128.0.11 35.232.161.75 RUNNING**

**gke-test-cluster-default-pool-f50402c9-nm9w us-central1-c e2-medium 10.128.0.13 34.72.166.33 RUNNING**

**gke-test-cluster-default-pool-f50402c9-vfb3 us-central1-c e2-medium 10.128.0.12 35.239.232.35 RUNNING**

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

You are now ready to deploy the Docker image you built to your GKE cluster.

* Visit the Google Kubernetes Engine **Workloads** menu in Cloud Console.
* From the Workloads menu, click **Deploy**.
* In the Create Deployment window that appears, click **Existing Container Image.**
* Using the drop-down menu, click on the **simple-nodejs-app** image you pushed to Container Registry.
* Click **View YAML**. This will open a YAML configuration file representing the two Kubernetes API resources about to be deployed into your cluster: one Deployment, and one HorizontalPodAutoscaler for that Deployment.
* Click **Deploy**.
* Wait for the Deployment Pods to be ready. You will see blue spinning wheels as your GKE cluster deploys the 3 simple-nodejs-app Pods, then green check marks when the Pods have been successfully deployed.
* Wait for the Pods to be ready by navigating to Kubernetes Engine, then Workloads. Click on simple-nodejs-app and scroll down to the **Managed Pods** section. You should see 3 simple-nodejs-app pods.

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

* Visit the Google Kubernetes Engine Workloads menu in Cloud Console.
* Click on **simple-nodejs-app**.
* From the Deployment details page, click **Expose**.
* From the Expose a Deployment menu, set the **Target Port** to 8080. This is the port the **simple-nodejs-app** container listens on.
* Click **Expose** to generate a Kubernetes Service for simple-nodejs-app, called **simple-nodejs-app-service.**
* Wait for the External Load Balancer to be created. You will see a blue spinning wheel in the Google Cloud Console, then once the Load Balancer is ready, you will be redirected to the Service Details page for **simple-nodejs-app-service**.
* Scroll down to the **External endpoints** field, and copy the address to the clipboard.

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.

Build and tag a new **simple-nodejs-app**  docker image

Below command builds the updated sources and created the updated image of simple-nodejs-app

|  |
| --- |
| $ docker build -t gcr.io/${PROJECT\_ID}/simple-nodejs-app:v2 . |

The output is similar to this:

**Sending build context to Docker daemon 3.486MB**

**Step 1/7 : FROM node:10**

**---> e7671d9424c2**

**Step 2/7 : WORKDIR /usr/src/app**

**---> Using cache**

**---> 554cc33ac20a**

**Step 3/7 : COPY package\*.json ./**

**---> 09fe32070067**

Below command push the image to Container Registry

|  |
| --- |
| $ docker push gcr.io/${PROJECT\_ID}/simple-nodejs-app:v2 |

The output is similar to this:

**The push refers to repository [gcr.io/sample-nodejs-app-282220/simple-nodejs-app]**

**938cd88aa281: Pushed**

**a83588d27266: Pushed**

**0c00f62e7484: Pushed**

**aebc3215f069: Layer already exists**

**d1e551c519b5: Layer already exists**

**70a3a00c2b44: Layer already exists**

**918a6facd6de: Layer already exists**

**bc17cd405095: Layer already exists**

**ee854067fbbd: Layer already exists**

**740ffea5d5c3: Layer already exists**

**eac9ead92b24: Layer already exists**

**23bca356262f: Layer already exists**

**8354d5896557: Layer already exists**

**v2: digest: sha256:4c825d076cb1301cf34c65102e973a5b8a7df956e528582a8877c976738a6aff size: 3051**

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

Follow the below steps to use the new image of simple-nodejs-app in the Google Kubernetes Engine

* Visit the Google Kubernetes Engine Workloads menu in Cloud Console.
* Click **simple-nodejs-app**.
* Under **Actions**, click **Rolling update**.
* In the window that appears, set the **Image** field to **gcr.io/sample-nodejs-app-282220/simple-nodejs-app:v2**
* Click **Update** to begin the rolling update.
* Return to the Deployment view for **simple-nodejs-app**, then scroll down to **Active Revisions**. You should now see two Revisions, 1 and 2. Revision 1 corresponds to the initial Deployment you created earlier. Revision 2 is the rolling update you just started.
* After a few moments, refresh the page. Under **Managed Pods**, you should see that now, all the replicas of **simple-nodejs-app** correspond to Revision 2.
* In a separate tab, navigate again to the **simple-nodejs-app-service** External IP. You should now see the Version set to 2.0.0.

# **References**

For more details on the user workflow, Refer the Screenshots document - **Deploy\_Containerized\_Application\_in\_GKE\_screenshots.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 | Screenshots document | Deploy\_Containerized\_Application\_in\_GKE\_screenshots.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> |