

[Kubernetes Engine](https://cloud.google.com/kubernetes-engine/) (https://cloud.google.com/kubernetes-engine/)

[Documentation](https://cloud.google.com/kubernetes-engine/docs/) (https://cloud.google.com/kubernetes-engine/docs/) [Guides](#)

Quickstart: Deploying a language-specific app

This page shows you how to do the following:

1. Create a Hello World app.
2. Package the app into a container image using Cloud Build.
3. Create a cluster in Google Kubernetes Engine (GKE).
4. Deploy the container image to your cluster.

The sample is shown in several languages, but note that you can use other languages in addition to the ones shown.

Before you begin

1. [Sign in](https://accounts.google.com/Login) (https://accounts.google.com/Login) to your Google Account.

If you don't already have one, [sign up for a new account](https://accounts.google.com/SignUp) (https://accounts.google.com/SignUp).

2. In the Cloud Console, on the project selector page, select or create a Google Cloud project.

★ **Note:** If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

[GO TO THE PROJECT SELECTOR PAGE](https://console.cloud.google.com/projectselector) (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/PROJECTSELECTOR)

3. Make sure that billing is enabled for your Google Cloud project. [Learn how to confirm billing is enabled for your project](https://cloud.google.com/billing/docs/how-to/modify-project) (https://cloud.google.com/billing/docs/how-to/modify-project).
4. Enable the Cloud Build and Google Kubernetes Engine APIs.

[ENABLE THE APIS](https://console.cloud.google.com/flows/enableapi?APIID=CLOUDBUILD) (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/FLOWS/ENABLEAPI?APIID=CLOUDBUILD)

5. [Install and initialize the Cloud SDK](https://cloud.google.com/sdk/docs/) (https://cloud.google.com/sdk/docs/).

6. `kubectl` is used to manage Kubernetes, the cluster orchestration system used by GKE. You can install `kubectl` by using `gcloud`:

```
gcloud components install kubectl
```



Writing the sample app

For instructions on creating a Hello World app that runs on GKE, click your language:

GO

NODE.JS

PYTHON

MORE ▾

1. Create a new directory named `helloworld-gke` and change directory into it:

```
mkdir helloworld-gke
cd helloworld-gke
```



2. Create a new module named `example.com/helloworld`:

```
go mod init example.com/helloworld
```



3. Create a new file named `helloworld.go` and paste the following code into it:

```
quickstart/go/helloworld.go
(https://github.com/GoogleCloudPlatform/kubernetes-engine-
samples/blob/master/quickstart/go/helloworld.go)
```

LOUDPLATFORM/KUBERNETES-ENGINE-SAMPLES/BLOB/MASTER/QUICKSTART/GO/HELLOWORLD.GO)

```
package main

import (
    "fmt"
    "log"
    "net/http"
    "os"
)

func main() {
    http.HandleFunc("/", handler)
    port := os.Getenv("PORT")
    if port == "" {
```



```

        port = "8080"
    }
    log.Printf("Listening on localhost:%s", port)
    log.Fatal(http.ListenAndServe(fmt.Sprintf(":%s", port), nil))
}

func handler(w http.ResponseWriter, r *http.Request) {
    log.Print("Hello world received a request.")
    target := os.Getenv("TARGET")
    if target == "" {
        target = "World"
    }
    fmt.Fprintf(w, "Hello %s!\n", target)
}
}

```

This code creates a web server that listens on the port defined by the `PORT` environment variable.

Your app is finished and ready to be packaged in a Docker container, and then uploaded to Container Registry.

Containerizing an app with Cloud Build

1. To containerize the sample app, create a new file named `Dockerfile` in the same directory as the source files, and copy the following content:

GO
NODE.JS
PYTHON
MORE ▾

[quickstart/go/Dockerfile](https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/blob/master/quickstart/go/Dockerfile)
 (https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/blob/master/quickstart/go/Dockerfile)

3LECLOUDPLATFORM/KUBERNETES-ENGINE-SAMPLES/BLOB/MASTER/QUICKSTART/GO/DOCKERFILE)

```

# Use the official Golang image to create a build artifact.
# This is based on Debian and sets the GOPATH to /go.
# https://hub.docker.com/_/golang
FROM golang:1.12 as builder

# Copy local code to the container image.
WORKDIR /app
COPY . .

```

```
# Build the command inside the container.
RUN CGO_ENABLED=0 GOOS=linux go build -v -o helloworld

# Use a Docker multi-stage build to create a lean production image.
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-
FROM alpine
RUN apk add --no-cache ca-certificates

# Copy the binary to the production image from the builder stage.
COPY --from=builder /app/helloworld /helloworld

# Run the web service on container startup.
CMD ["/helloworld"]
```

2. Get your Google Cloud project ID:

```
gcloud config get-value project
```



Build your container image using [Cloud Build](https://cloud.google.com/cloud-build) (<https://cloud.google.com/cloud-build>), which is similar to running `docker build` and `docker push`, but it happens on Google Cloud.

Replace ***PROJECT_ID*** with your Google Cloud ID:

```
gcloud builds submit --tag gcr.io/PROJECT_ID/helloworld-gke .
```



The image is stored in [Container Registry](https://cloud.google.com/container-registry) (<https://cloud.google.com/container-registry>).

Creating a Kubernetes Engine cluster

A GKE cluster is a managed set of Compute Engine virtual machines that operate as a single GKE cluster. This tutorial uses a single node.

1. Create the cluster. Replace ***YOUR_GCP_ZONE*** with the Google Cloud zone where you want to host your cluster. For a complete list, see [Geography and regions](https://cloud.google.com/docs/geography-and-regions) (<https://cloud.google.com/docs/geography-and-regions>).

```
gcloud container clusters create helloworld-gke \
  --num-nodes 1 \
  --enable-basic-auth \
  --issue-client-certificate \
  --zone YOUR_GCP_ZONE
```



★ **Note:** A region contains one or more zones. Using a region instead of a zone for the `--zone` option (for example, `us-central1` instead of `us-central1-b`) creates a node in each zone within the region.

★ **Note:** The default scope is set to `--scopes=gke-default`. For a full list of available scopes, see [OAuth 2.0 scopes for Google APIs](https://developers.google.com/identity/protocols/googlescopes) (<https://developers.google.com/identity/protocols/googlescopes>).

2. Verify that you have access to the cluster. The following command lists the nodes in your container cluster which are up and running and indicates that you have access to it.

```
kubectl get nodes
```



If you run into errors, refer to the [Kubernetes Troubleshooting guide](https://kubernetes.io/docs/tasks/debug-application-cluster/debug-cluster/) (<https://kubernetes.io/docs/tasks/debug-application-cluster/debug-cluster/>)

Deploying to GKE

To deploy your app to the GKE cluster you created, you need two Kubernetes objects.

1. A [Deployment](http://kubernetes.io/docs/user-guide/deployments/) (<http://kubernetes.io/docs/user-guide/deployments/>) to define your app.
2. A [Service](https://kubernetes.io/docs/concepts/services-networking/service/) (<https://kubernetes.io/docs/concepts/services-networking/service/>) to define how to access your app.

Deploy an app

The app has a frontend server that handles the web requests. You define the cluster resources needed to run the frontend in a new file called `deployment.yaml`. These resources are described as a Deployment. You use Deployments to create and update a [ReplicaSet](https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/) (<https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>) and its associated pods.

1. Create the `deployment.yaml` file in the same directory as your other files and copy the following content, replacing `$GCP_PROJECT` with your Google Cloud project ID:

```
quickstart/deployment.yaml
```

```
(https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/blob/master/quickstart/deployment.yaml)
```

E.CLOUDPLATFORM/KUBERNETES-ENGINE-SAMPLES/BLOB/MASTER/QUICKSTART/DEPLOYMENT.YAML)

```
# This file configures the hello-world app which serves public web traffic.
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: helloworld-gke
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
      - name: hello-app
        # Replace $GCLLOUD_PROJECT with your project ID
        image: gcr.io/$GCLLOUD_PROJECT/helloworld-gke:latest
        # This app listens on port 8080 for web traffic by default.
        ports:
        - containerPort: 8080
        env:
        - name: PORT
          value: "8080"
```

2. Deploy the resource to the cluster:

```
kubectl apply -f deployment.yaml
```

3. Track the status of the Deployment:

```
kubectl get deployments
```

After the Deployment has the same number of **AVAILABLE** pods as **DESIRED** pods, the Deployment is complete.

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
hello-deployment	1	1	1	1	20s

If the Deployment has a mistake, run `kubectl apply -f deployment.yaml` again to update the Deployment with any changes.

4. After the Deployment is complete, you can see the pods that the Deployment created:

```
kubectl get pods
```



Deploy a Service

[Services](https://kubernetes.io/docs/concepts/services-networking/service/) (https://kubernetes.io/docs/concepts/services-networking/service/) provide a single point of access to a set of pods. While it's possible to access a single pod, pods are ephemeral and can only be accessed reliably by using a Service address. In your Hello World app, the "hello" Service defines a [load balancer](https://kubernetes.io/docs/tasks/access-application-cluster/create-external-load-balancer/)

(https://kubernetes.io/docs/tasks/access-application-cluster/create-external-load-balancer/) to access the `hello-app` pods from a single IP address. This Service is defined in the `service.yaml` file.

1. Create the file `service.yaml` in the same directory as your other source files with the following content:

[quickstart/service.yaml](#)

(<https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/blob/master/quickstart/service.yaml>)

```
GOOGLECLOUDPLATFORM/KUBERNETES-ENGINE-SAMPLES/BLOB/MASTER/QUICKSTART/SERVICE.YAML)
```

```
# The hello service provides a load-balancing proxy over the hello-app
# pods. By specifying the type as a 'LoadBalancer', Kubernetes Engine will
# create an external HTTP load balancer.
apiVersion: v1
kind: Service
metadata:
  name: hello
spec:
  type: LoadBalancer
  selector:
    app: hello
  ports:
    - port: 80
      targetPort: 8080
```



The pods are defined separately from the Service that uses the pods. Kubernetes uses [labels](https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/) (https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/) to select the

Pods that a Service addresses. With labels, you can have a Service that addresses pods from different replica sets and have multiple Services that point to an individual pod.

2. Create the Hello World Service:

```
kubectl apply -f service.yaml
```



3. Get the Service's external IP address:

```
kubectl get services
```



It can take up to 60 seconds to allocate the IP address. The external IP address is listed under the column **EXTERNAL-IP** for the **hello** Service.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
hello	LoadBalancer	10.22.222.222	35.111.111.11	80:32341/TCP	1m
kubernetes	ClusterIP	10.22.222.1	<none>	443/TCP	20m



View a deployed app

You have now deployed all the resources needed to run the Hello World app on GKE. Use the external IP address from the previous step to load the app in your web browser, and see your running app!

```
# Example cURL call to your running application on GKE
$ kubectl get service hello \
  -o=custom-columns=NAME:.status.loadBalancer.ingress[*].ip --no-headers
35.111.111.11
$ curl 35.111.111.11
Hello World!
```



Clean up

To avoid incurring charges to your Google Cloud account for the resources used in this quickstart, follow these steps.

You are charged for [the Compute Engine instances](#)

(<https://cloud.google.com/kubernetes-engine/pricing>) running in your cluster, as well as for [the](#)

container image in Container Registry (<https://cloud.google.com/container-registry/pricing>).

Delete the project


Deleting your Google Cloud project stops billing for all the resources used within that project.

! **Caution:** Deleting a project has the following effects:

- **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.
- **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an **appspot.com** URL, delete selected resources inside the project instead of deleting the whole project.

1. In the Cloud Console, go to the **Manage resources** page.

GO TO THE MANAGE RESOURCES PAGE ([HTTPS://CONSOLE.CLOUD.GOOGLE.COM/IAM-ADMIN/PROJ](https://console.cloud.google.com/iam-admin/PROJ))

2. In the project list, select the project you want to delete and click **Delete** .

3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

Delete your cluster and container

If you want to keep your project but only delete the resources used in this tutorial, delete your cluster and image.

To delete a cluster using the gcloud command-line tool, run the following command:

```
gcloud container clusters delete helloworld-gke
```



Note: For more information, refer to the documentation on [Deleting a cluster](https://cloud.google.com/kubernetes-engine/docs/how-to/deleting-a-cluster) (<https://cloud.google.com/kubernetes-engine/docs/how-to/deleting-a-cluster>).

To delete an image from one of your Container Registry repositories, run the following command:

```
gcloud container images delete gcr.io/[PROJECT-ID]/helloworld-gke
```



Note: For more information, refer to the documentation on [Managing images](https://cloud.google.com/container-registry/docs/managing#deleting_images) (https://cloud.google.com/container-registry/docs/managing#deleting_images).

What's next

For more information on Kubernetes, see the following:

- [Learn more about creating clusters](https://cloud.google.com/kubernetes-engine/docs/how-to/creating-a-container-cluster) (<https://cloud.google.com/kubernetes-engine/docs/how-to/creating-a-container-cluster>).
- [Learn more about Kubernetes](http://kubernetes.io/) (<http://kubernetes.io/>).
- [Read the kubectl reference documentation](http://kubernetes.io/docs/user-guide/kubectl-overview/) (<http://kubernetes.io/docs/user-guide/kubectl-overview/>).

For more information on deploying to GKE, see the following:

- [Learn how to package, host, and deploy a simple web server application](https://cloud.google.com/kubernetes-engine/docs/tutorials/hello-app) (<https://cloud.google.com/kubernetes-engine/docs/tutorials/hello-app>).
- [Deploy a Guestbook application with Redis and PHP](https://cloud.google.com/kubernetes-engine/docs/tutorials/guestbook) (<https://cloud.google.com/kubernetes-engine/docs/tutorials/guestbook>).
- [Deploy a stateful WordPress application with persistent storage and MySQL](https://cloud.google.com/kubernetes-engine/docs/tutorials/persistent-disk) (<https://cloud.google.com/kubernetes-engine/docs/tutorials/persistent-disk>).
- [Setting up Cloud Run on GKE](https://cloud.google.com/run/docs/gke/setup) (<https://cloud.google.com/run/docs/gke/setup>).

For more information on deploying to GKE directly from your IDE with Cloud Code, see the following:

- [Cloud Code for VS Code](https://cloud.google.com/code/docs/vscode/deploying-an-application) (<https://cloud.google.com/code/docs/vscode/deploying-an-application>)
- [Cloud Code for IntelliJ](https://cloud.google.com/code/docs/intellij/deploying-a-k8-app) (<https://cloud.google.com/code/docs/intellij/deploying-a-k8-app>)

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 4.0 License](https://creativecommons.org/licenses/by/4.0/) (<https://creativecommons.org/licenses/by/4.0/>), and code samples are licensed under the [Apache 2.0 License](https://www.apache.org/licenses/LICENSE-2.0) (<https://www.apache.org/licenses/LICENSE-2.0>). For details, see our [Site Policies](https://developers.google.com/terms/site-policies) (<https://developers.google.com/terms/site-policies>). Java is a registered trademark of Oracle and/or its affiliates.

Last updated December 18, 2019.