GCP Certification series: 3.3 Deploying and implementing App Engine and Cloud Functions resources

Prashanta Paudel

Oct 26, 2018 · 14 min read

Google App Engine

Developer(s)	Google
Initial release	April 7, 2008; 10 years ago
Stable release	1.9.63 / 27 February 2018
Written in	Python, Java, Go, PHP, Node.js
Operating system	linux (glibc), Windows
Platform	little-endian 32bits
Type	Web framework, cloud computing platform
License	Proprietary, LGPL
Website	cloud.google.com /appengine/

Reference: https://en.wikipedia.org/wiki/Google_App_Engine

App Engine

Google App Engine is a *platform as a service(PaaS)* from Google. It means that Google will provide all the infrastructure and software necessary for you to write the code and Google will handle the rest. Google App Engine is Google's platform as a service offering that allows developers and businesses to build and run applications using Google's advanced infrastructure. It also requires the use of Google query language and that the database used is Google Big Table. Applications must abide by these standards, so applications either must be developed with GAE in mind or else modified to meet the requirements. Google will do scaling up and down automatically to meet the need and will ensure that your code will be serving the users.

You will only pay for the resources that you use and no more.

App engine is a perfect fit for companies that don't want to care about underlying infrastructure and only concern about making the service available all the time.

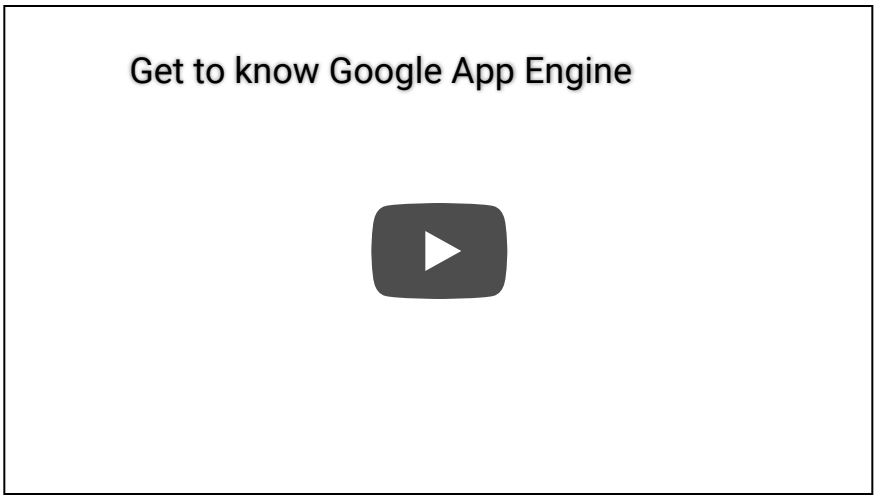
For a startup, this is a perfect solution.

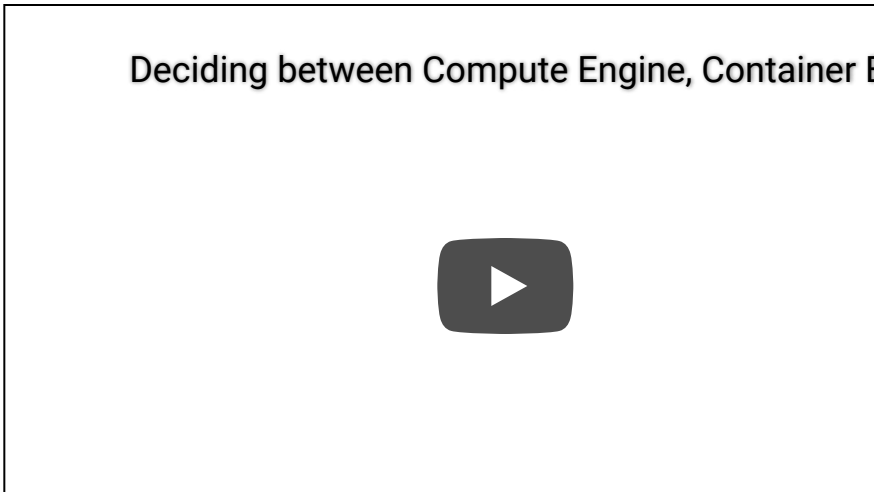
App Engine is generally used for web application rather than mobile applications.

App Engine has built-in services like load balancing, health checks and application logging which is very useful for application administration.

App Engine offers several storage options like cloud SQL and NoSQL through API's.

Applications are sandboxed and run across multiple servers. App Engine offers automatic scaling for web applications which means as the number of requests increases for an application, App Engine automatically allocates more resources for the web application to handle additional traffic.

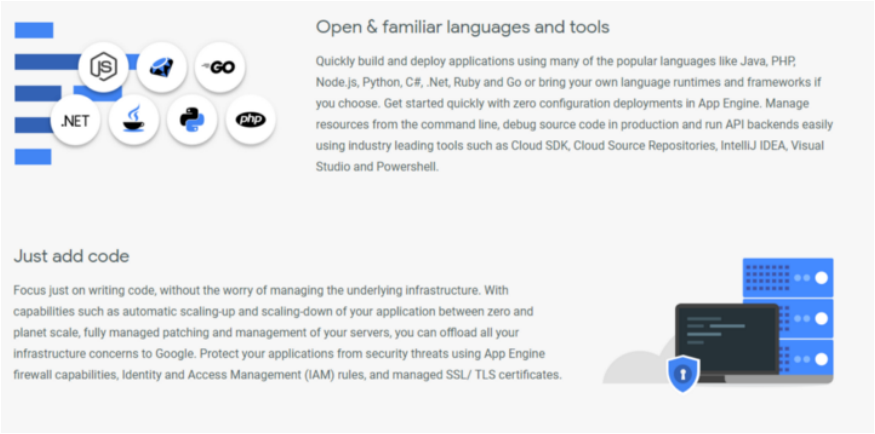




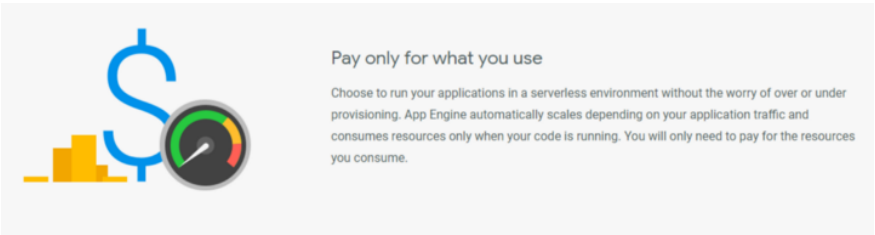
Fully managed serverless application platform

Build and deploy applications on a fully managed platform. Scale your applications seamlessly from zero to planet scale without having to worry about managing the underlying infrastructure. With zero server management and zero configuration deployments, developers can focus only on building great applications without the management overhead. App Engine enables developers to stay more productive and agile by supporting popular development languages and a wide range of developer tools.

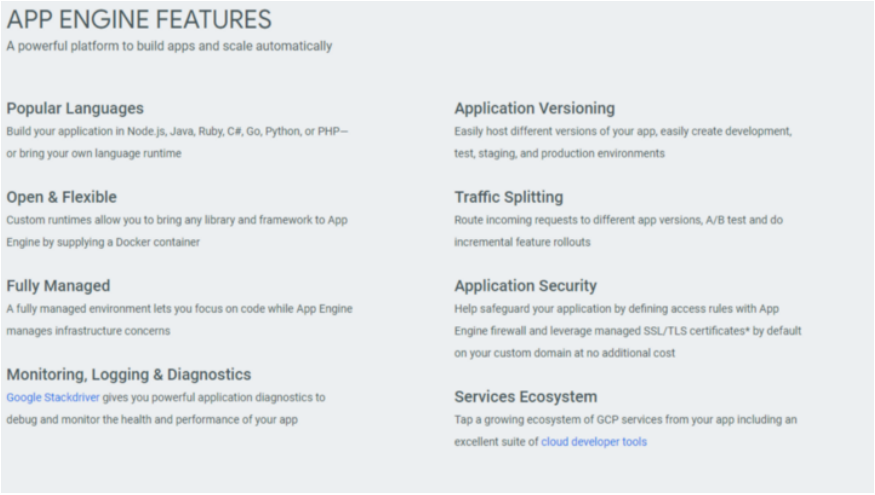
Reference: https://cloud.google.com/appengine/



Reference: https://cloud.google.com/appengine/



Reference: https://cloud.google.com/appengine/



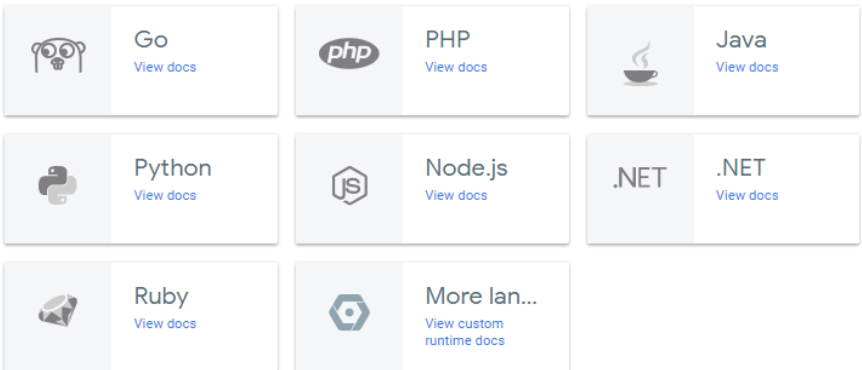
Reference: https://cloud.google.com/appengine/

Supported Languages

App Engine's standard Environment offers support for languages

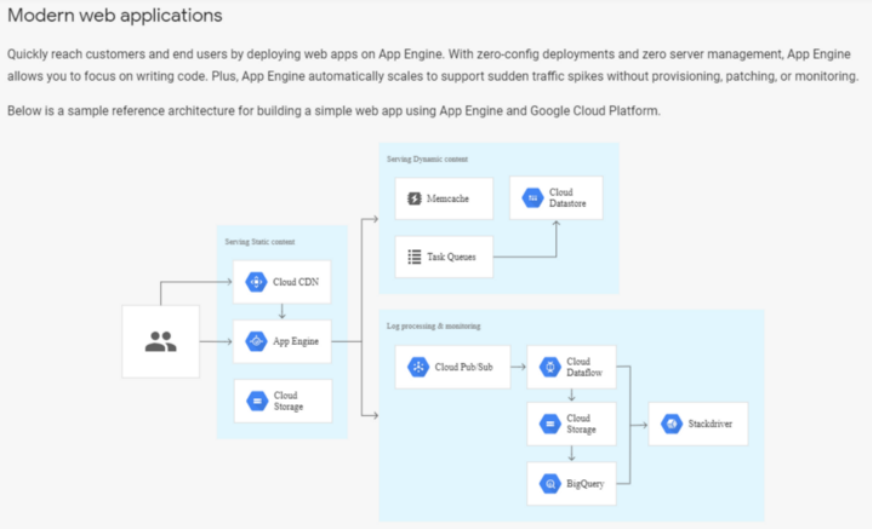
reference: https://cloud.google.com/appengine/

- Java
- Python
- PHP
- Go
- C#
- Ruby
- Node.js
- .Net



The flexible environment offers other language runtimes as well. Get started quickly with zero configuration deployments in App Engine.

Manage resources from the command line, debug source code in production and run API backends easily using industry-leading tools such as Cloud SDK, Cloud Source Repositories, IntelliJ IDEA, Visual Studio, and Powershell.



Reference: https://cloud.google.com/appengine/

The App Engine Standard Environment

The App Engine standard environment is based on container instances running on Google's infrastructure. Containers are preconfigured with one of several available runtimes.

The App Engine standard environment makes it easy to build and deploy an application that runs reliably even under heavy load and with large amounts of data.

Applications run in a secure, sandboxed environment, allowing App Engine standard environment to distribute requests across multiple servers, and scaling servers to meet traffic demands. Your application runs within its own secure, reliable environment that is independent of the hardware, operating system, or physical location of the server.

Instance classes

Each application running in the standard environment has an *instance class*, which determines its compute resources and pricing. This table

summarizes the memory and CPU limits of the various instance classes. See the pricing page for billing information.

Instance Class	Memory Limit	CPU Limit	Supported Scaling Types
F1 (default)	128 MB	600 MHz	automatic
F2	256 MB	1.2 GHz	automatic
F4	512 MB	2.4 GHz	automatic
F4_1G	1024 MB	2.4 GHz	automatic
B1	128 MB	600 MHz	manual, basic
B2 (default)	256 MB	1.2 GHz	manual, basic
B4	512 MB	2.4 GHz	manual, basic
B4_1G	1024 MB	2.4 GHz	manual, basic
B8	1024 MB	4.8 GHz	manual, basic

Reference: https://cloud.google.com/appengine/docs/standard/

Quotas and limits

The app Engine standard environment gives you 1 GB of data storage and traffic for free, which can be increased by enabling paid applications. However, some features impose limits unrelated to quotas to protect the stability of the system. For more details on quotas, including how you can edit them to suit your needs, see the Quotas page.

Standard environment languages and runtimes

- Java 7 runtime
- Java 8 runtime
- Python runtime
- PHP runtime
- Go runtime

App Engine Flexible Environment

App Engine allows developers to focus on doing what they do best, writing code. Based on Google Compute Engine, the App Engine

flexible environment automatically scales your app up and down while balancing the load. Microservices, authorization, SQL and NoSQL databases, traffic splitting, logging, versioning, security scanning, and content delivery networks are all supported natively. In addition, the App Engine flexible environment allows you to customize the runtime and even the operating system of your virtual machine using Dockerfiles. Learn about the differences between the standard environment and the flexible environment.

- **Runtimes**—The flexible environment includes native support for Java 8 (with no web-serving framework), Eclipse Jetty 9, Python 2.7 and Python 3.6, Node.js, Ruby, PHP, .NET core, and Go. Developers can customize these runtimes or provide their own runtime by supplying a custom Docker image or Dockerfile from the open source community.
- **Infrastructure Customization**—Because VM instances in the flexible environment are Google Compute Engine virtual machines, you can take advantage of custom libraries, use SSH for debugging, and deploy your own Docker containers.
- **Performance**—Take advantage of a wide array of CPU and memory configurations. You can specify how much CPU and memory each instance of your application needs and the flexible environment will provide the necessary infrastructure for you.

App Engine manages your virtual machines, ensuring that:

- Instances are health-checked, healed as necessary, and co-located with other services within the project.
- Critical, backward compatible updates are automatically applied to the underlying operating system.
- VM instances are automatically located by geographical region according to the settings in your project. Google's management services ensure that all of a project's VM instances are co-located for optimal performance.
- VM instances are restarted on a weekly basis. During restarts, Google's management services will apply any necessary operating system and security updates.

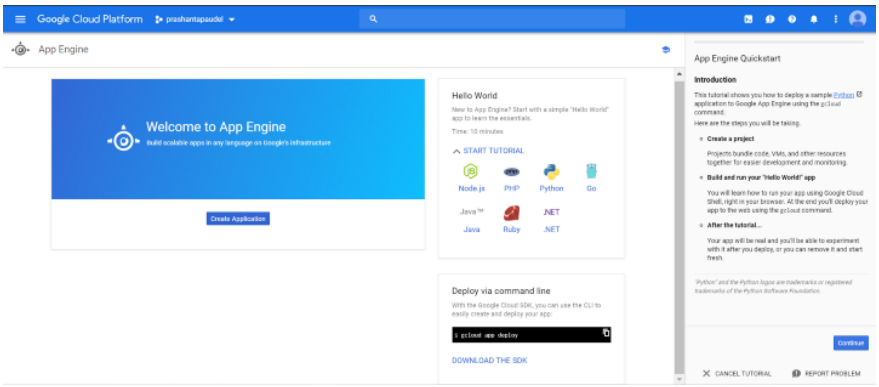
- You always have root access to Compute Engine VM instances. SSH access to VM instances in the flexible environment is disabled by default. If you choose, you can enable root access to your app's VM instances.

———————————————————————————————————————

Deploying an application to App Engine

We will see how to deploy a simple app in App Engine

First of all, go to Google cloud dashboard and then to compute engine. Inside compute engine you will see App Engine. Click on App engine



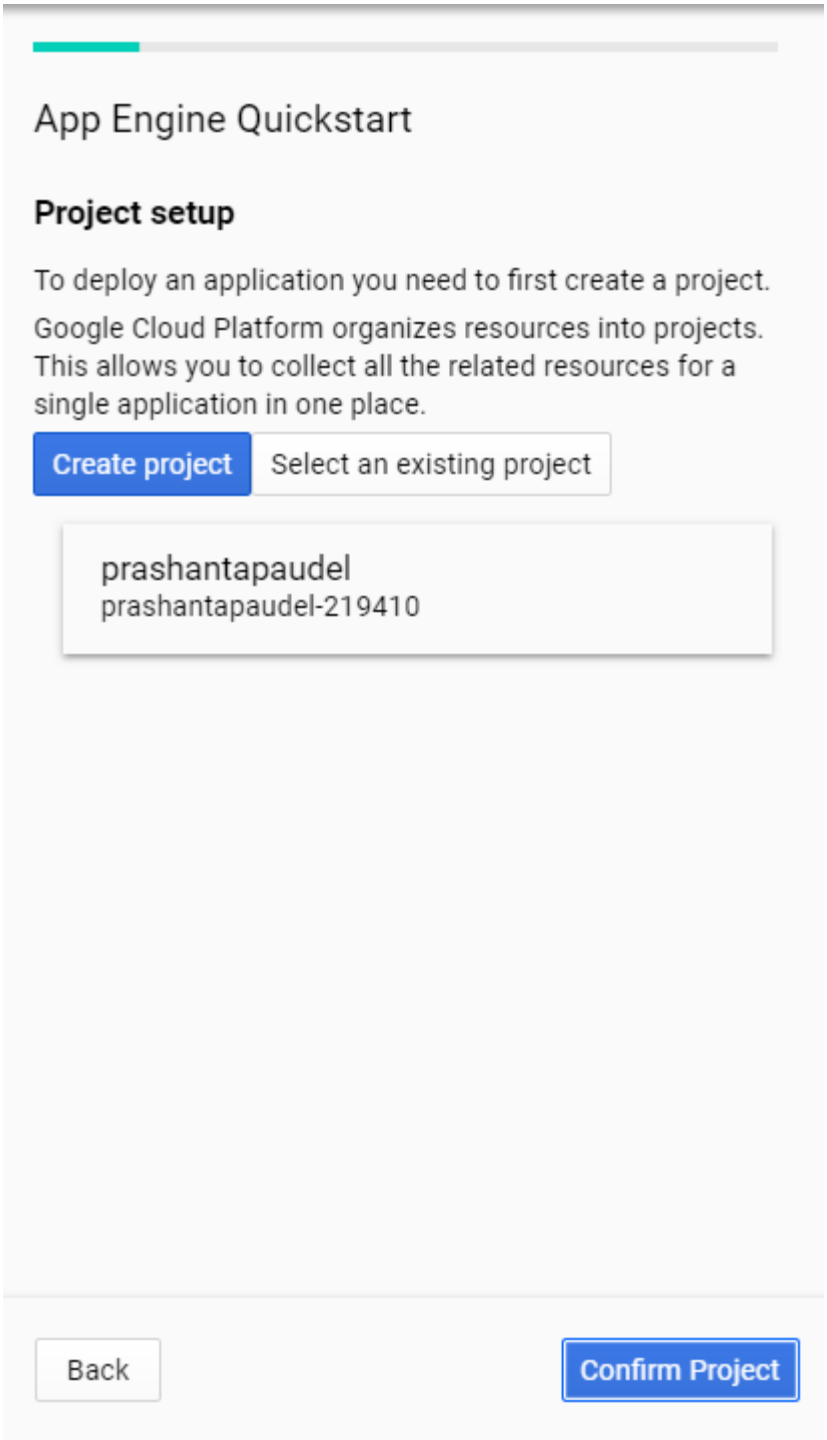
App Engine Dashboard

After clicking the App Engine you will see app engine Dashboard

Step 1: Create Project

In almost all GCP tasks, you must have at least one project to start with.

So, first create a project, which I have already done.



select project

Step 2: Copy/Clone the code from outside to the App Engine

Remember, App Engine requires the use of one of the languages supported by App Engine.

Here, I select **Python** for my test project.

Using Google Cloud Shell

Cloud Shell is a built-in command line tool for the console. We're going to use Cloud Shell to deploy our app.

Open Google Cloud Shell

Open Cloud Shell by clicking shell icon in the navigation bar at the top of the console.

Clone the sample code

Use Cloud Shell to clone and navigate to the "Hello World" code. The sample code is cloned from your project repository to the Cloud Shell.

Note: If the directory already exists, remove the previous files before cloning.

In Cloud Shell enter:

```
git clone https://github.com/GoogleCloudPlatform/python-
docs-samples
```

Then, switch to the tutorial directory:

```
cd python-docs-
samples/appengine/standard_python37/hello_world
```

Have a look at the sample application's structure—the `website` folder contains your website content and `app.yaml` is your application configuration file.

- Your website content must go inside the `website` folder, and its landing page must be called `index.html`, but apart from that, it can take whatever form you like.

- The `app.yaml` file is a configuration file that tells App Engine how to map URLs to your static files. You don't need to edit it.

Step 3: Configuring your deployment

You are now in the main directory for the sample code. We'll look at the files that configure your application.

Exploring the application

Enter the following command to view your application code:

```
cat main.py
```

The application is a simple Python application that uses the Flask web framework. This Python app responds to a request with an HTTP header and the message `Hello World!`.

Exploring your configuration

Google App Engine uses YAML files to specify a deployment's configuration. `app.yaml` files contain information about your application, like the runtime environment, environment variables, and more.

Enter the following command to view your configuration file:

```
cat app.yaml
```

This file contains the minimal amount of configuration required for a Python 3 application: the `runtime` field specifies the `python37` runtime environment.

The syntax of this file is YAML. For a complete list of configuration options, see the `app.yaml` reference.

Step 4: Testing your app

Cloud Shell lets you test your app before deploying to make sure it's running as intended, just like debugging on your local machine.

To test your app, first, create an isolated virtual environment. This ensures that your app does not interfere with other Python applications that may be available on your system.

```
virtualenv --python python3 \
    ~/envs/hello_world
```

Activate your newly-created virtual environment:

```
source \
    ~/envs/hello_world/bin/activate
```

Use `pip` to install project dependencies. This "Hello World" app depends on the Flask microframework:

```
pip install -r requirements.txt
```

Finally, run your app in Cloud Shell using the Flask development server:

```
python main.py
```

Preview your app with "Web preview"

Your app is now running on Cloud Shell. You can access the app by using Web preview

to connect to port8080.

Terminating the preview instance

Terminate the instance of the application by pressing `Ctrl+C` in the Cloud Shell.

Step 5: Deploying to Google App Engine

In order to deploy your app, you need to create an app in a region:

```
gcloud app create
```

Note: If you already created an app, you can skip this step.

Deploying with Cloud Shell

You can use Cloud Shell to deploy your app. To deploy your app enter:

```
gcloud app deploy app.yaml --project \
    prashantapaudel-219410
```

Visit your app

Congratulations! Your app has been deployed. The default URL of your app is prashantapaudel-219410.appspot.com. Click the URL to visit it.

==========

See also : https://developer.mozilla.org/en-US/docs/Learn/Common_questions/How_do_you_host_your_website_on_Google_App_Engine

---

Deploying Your Web Service

Use the `gcloud` tool of the Cloud SDK to deploy your web service to App Engine.

Though this initial version of the web service doesn't have Cloud Datastore or Firebase authentication, you can deploy it to App Engine at this stage to test and ensure that it works as expected.

Before you begin

If you have completed all the previous steps in this guide, skip this section. Otherwise, complete one of the following:

- Start from Building a Python 3.7 App and complete all the steps leading up to this one.
- If you already have a GCP project, you can continue by downloading a copy of the web service:

1. Download the sample application repository using Git

- `git clone https://github.com/GoogleCloudPlatform/python-docs-samples`

1. Alternatively, you can download the sample as a zip file and then extract it.
2. Navigate to the directory that contains a copy of the files from the previous step

- `cd python-docs-samples/appengine/standard_python37/building-an-app/building-an-app-1`

Deploying your service

To deploy your web service, you run the `gcloud app deploy` command from the root directory of your project, where your `app.yaml` file is located:

```
gcloud app deploy
```

Each time that you deploy your web service, a new version of that app is created in App Engine. During deployment, a container image is created using the Cloud Build service, and then a copy is uploaded to Google Cloud Storage before it is run in App Engine.

For more information about deploying to App Engine, see Testing and Deploying Your App.

Note: Files listed in your `.gcloudignore` file are not uploaded to App Engine during deployment.

Viewing your service

To quickly launch your browser and access your web service at `https://GCP_PROJECT_ID.appspot.com`, enter the following command:

```
gcloud app browse
```

Tip: If you would like to change the URL of your web service to something other than the default `appspot.com` URL, you can add a custom domain.

Managing services and versions

You've just deployed a version of the web service to App Engine. Each time that you deploy a version of your code, that version is created in a service. The initial deployment to App Engine must be created in the `default` service, but for subsequent deployments, you can specify the name of your service in your `app.yaml` file.

You can update a service at any time by running the `gcloud app deploy` command and deploying new versions to that service. Each time that you update a service, traffic is automatically routed to the version last deployed. However, you can include `gcloud` flags to change the deploy command behavior.

Use the GCP Console to manage and view the services and versions that you deploy to App Engine:

- Use the GCP Console to view your App Engine services:
- GO TO THE SERVICES PAGE
- Use the GCP Console to view your versions:
- GO TO THE VERSIONS PAGE

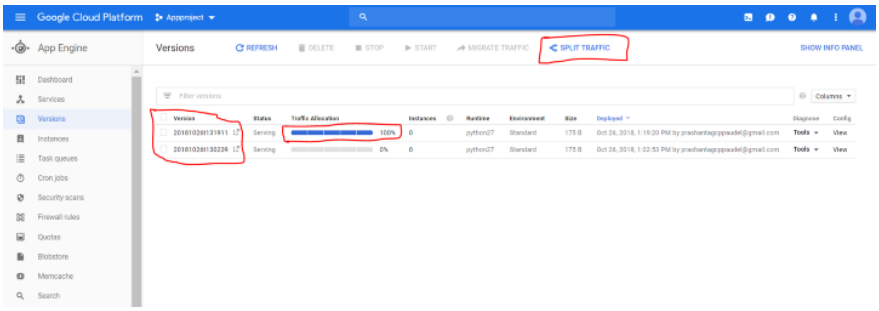
For more information about the multi-service design pattern, see An Overview of App Engine. To learn how to send requests to specific services and versions, see Splitting Traffic.

Splitting traffic across multiple versions

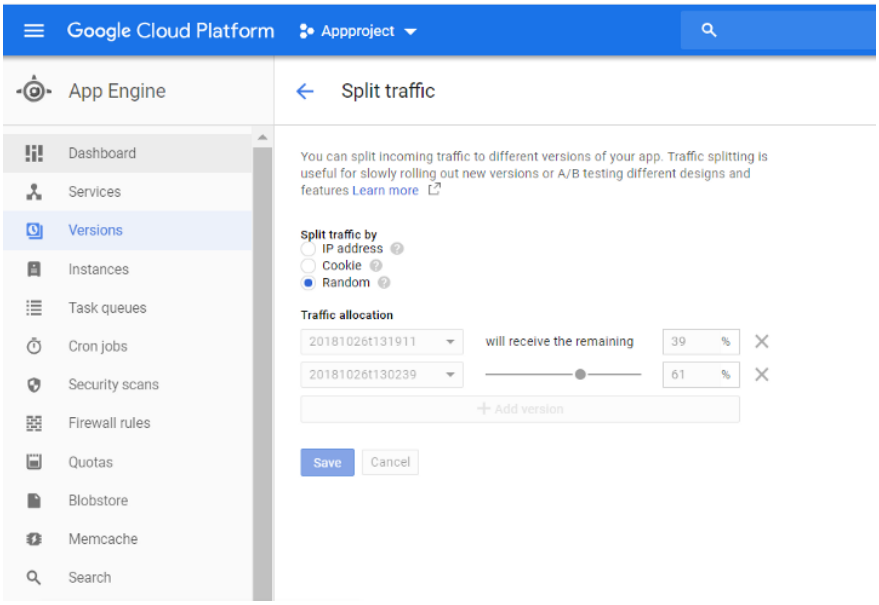
When you have specified two or more versions for splitting, you must choose whether to split traffic by using either an IP address or HTTP cookie. It's easier to set up an IP address split, but a cookie split is more precise. For more information, see IP address splitting and Cookie splitting.

When you update or add another version of the app in App Engine, you will see 2 versions in Version tab of the App Engine

Now select these two versions and click SPLIT TRAFFIC at top of the Version page

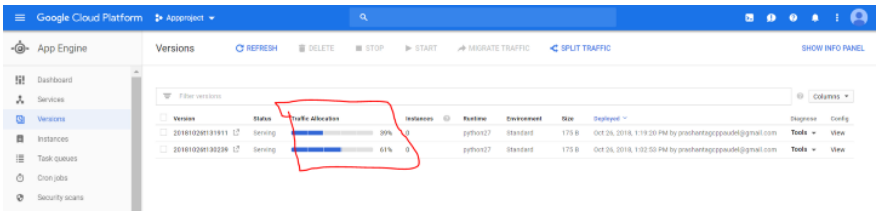


Version page



Splitting traffic by Random

Now you can see the different versions serving



two versions

———————————————————————————————————————

Google Cloud Functions

Cloud Functions

Event-driven serverless compute platform

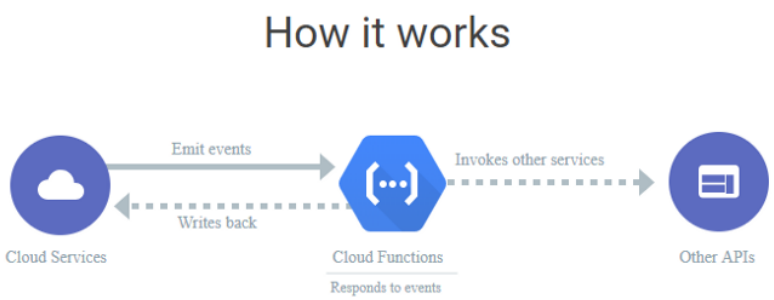
- The simplest way to run your code in the cloud
- Automatically scales, highly available and fault tolerant
- No servers to provision, manage, patch or update
- Pay only while your code runs
- Connects and extends cloud services

Google cloud functions(GCF) is a new offering that enables developers to run a particular function based on trigger events. This is, even more, higher in level than App Engine in terms of infrastructure independent.

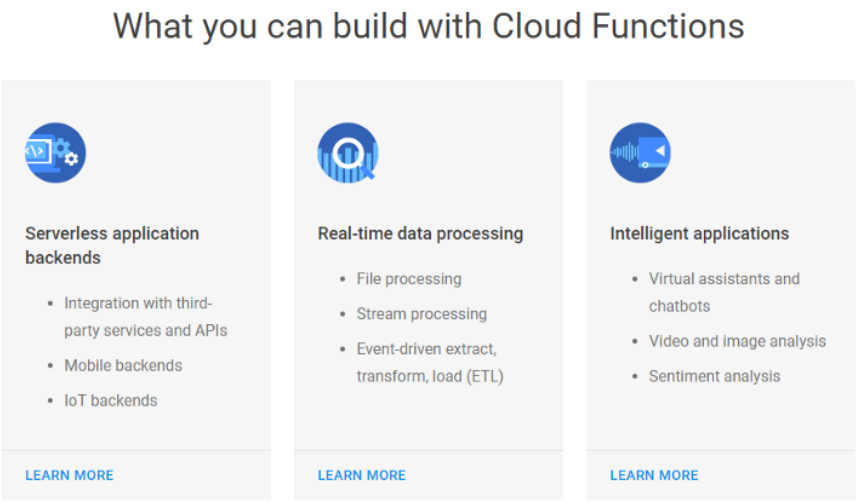
GCP is the main product that represents the serverless concept in Google Cloud Platform.

Cloud functions allow specific functions that the developer has written to be triggered by some event.

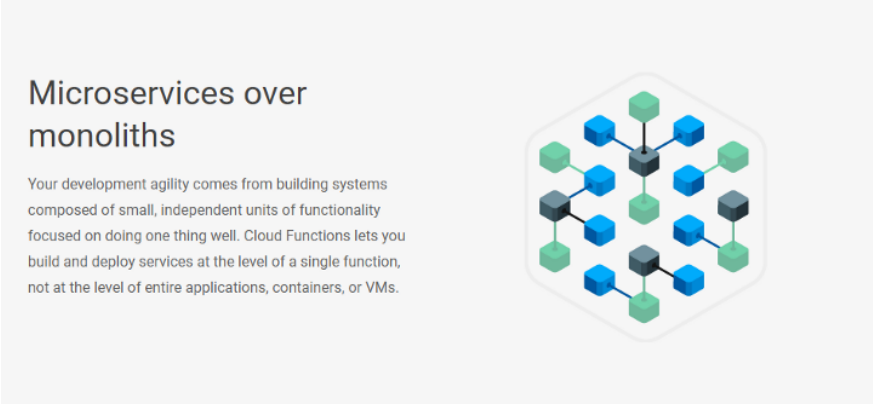
For example, when a sensor receives a signal then some functions can be called to do some task. When someone signup for the even email will be sent to someone.



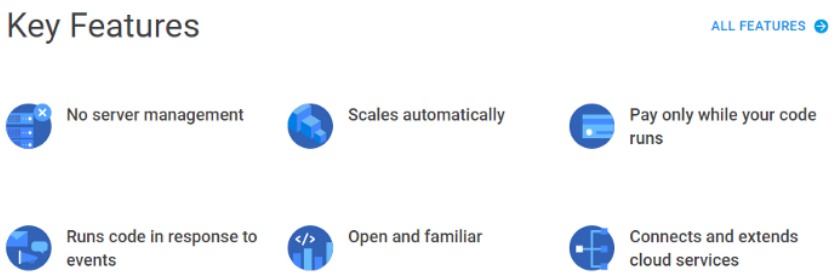
Reference: https://cloud.google.com/functions/



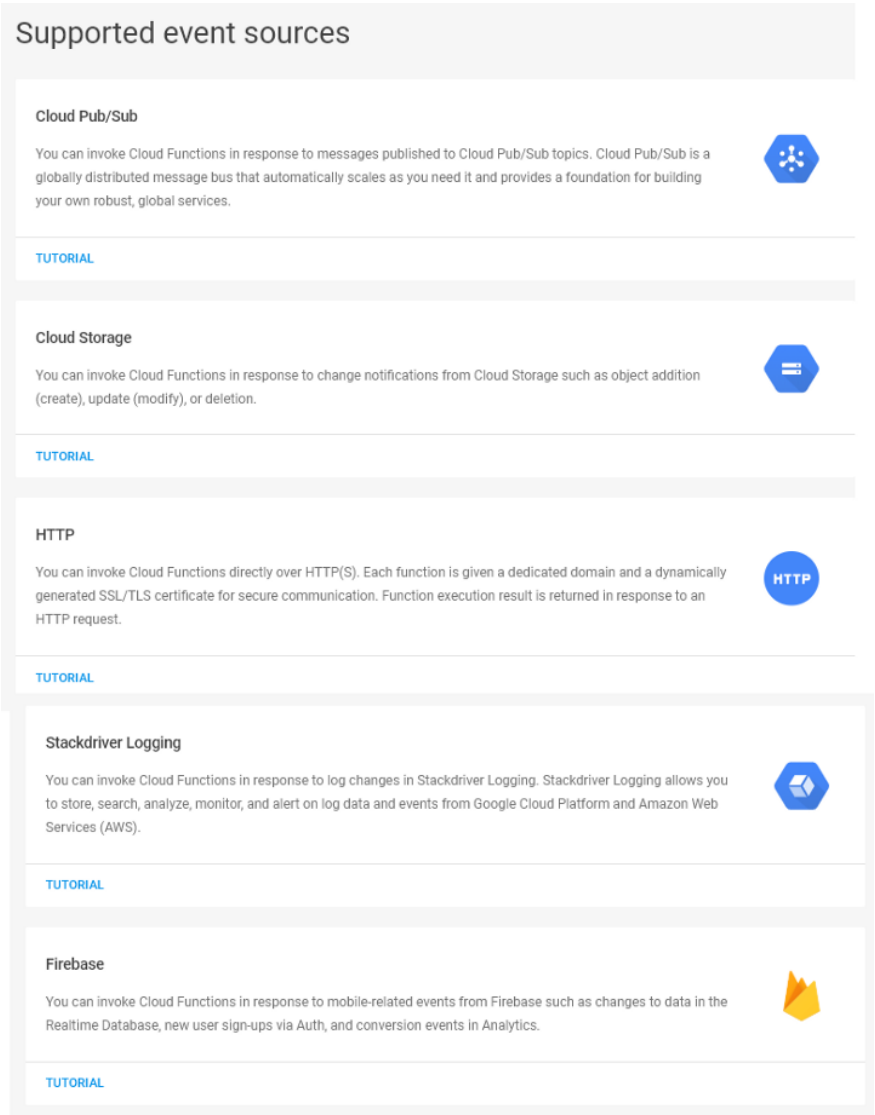
Reference: https://cloud.google.com/functions/



Reference: https://cloud.google.com/functions/



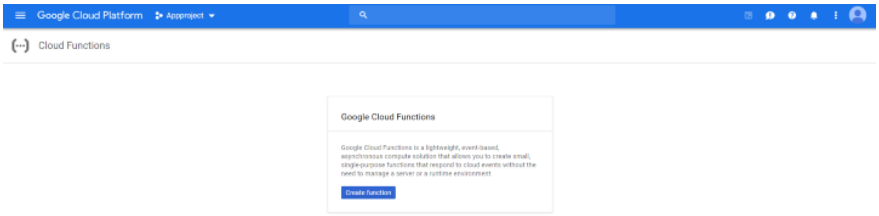
Reference: https://cloud.google.com/functions/



Sources of trigger

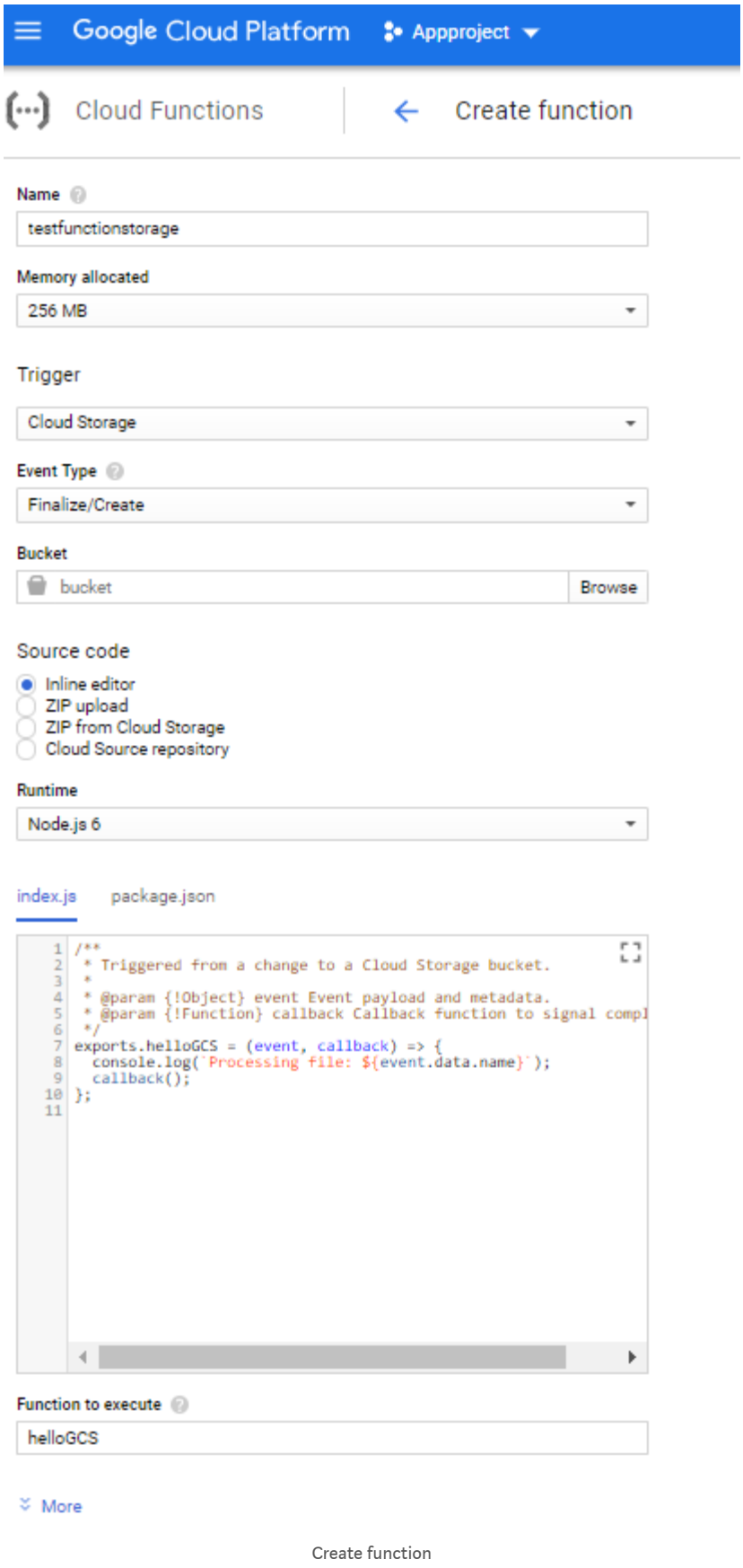
Deploying a Cloud Function that receives Google Cloud events

First Goto Cloud Functions Dashboard

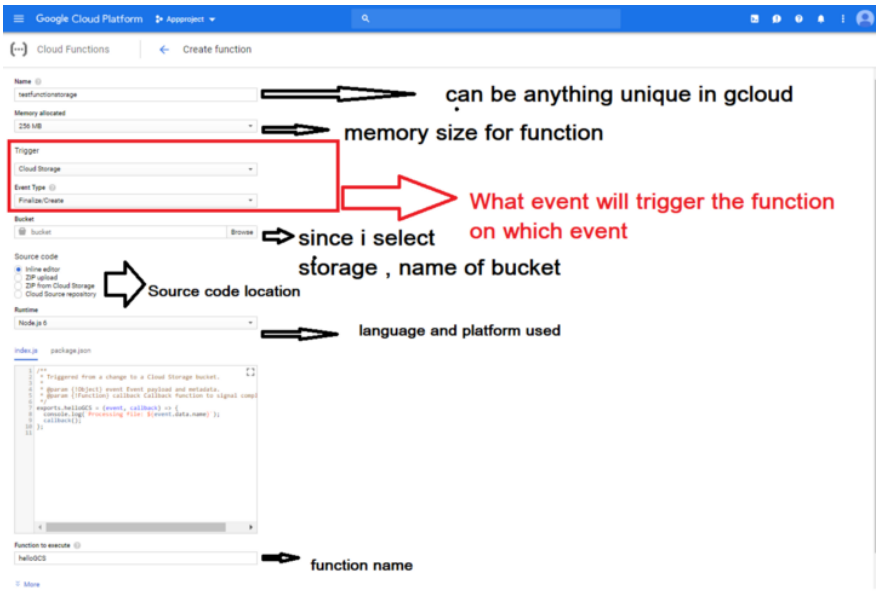


Cloud Function Dashboard

Click on Create Functions

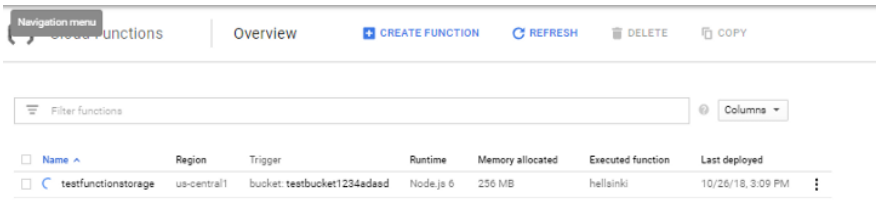

Google Cloud Platform
Appproject
Cloud Functions
Create function
Name
testfunctionstorage
Memory allocated
256 MB
Trigger
Cloud Storage
Event Type
Finalize/Create
Bucket
bucket
Browse
Source code
Inline editor
ZIP upload
ZIP from Cloud Storage
Cloud Source repository
Runtime
Node.js 6
index.js
package.json
/**
* Triggered from a change to a Cloud Storage bucket.
*
* @param {!Object} event Event payload and metadata.
* @param {!Function} callback Callback function to signal compl
*/
exports.helloGCS = (event, callback) => {
console.log(`Processing file: ${event.data.name}`);
callback();
};
Function to execute
helloGCS
More
Create function

You will get a number of options to select

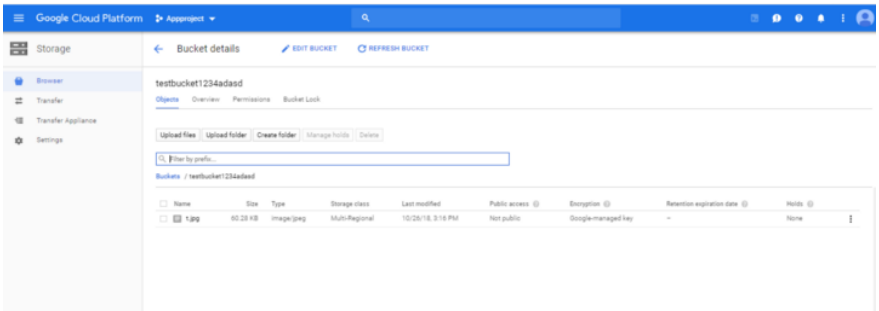


Options in creating the function

It will take some time to create the function



Now load the file in the bucket



After upload use the cloud shell or SDK to check the triggered function by

```
$ gcloud functions logs read --limit 50

D      function-1           278457471527247  2018-10-26
12:16:45.259  Function execution started
I      function-1           278457471527247  2018-10-26
12:16:45.269  Processing file: t.jpg.
D      function-1           278457471527247  2018-10-26
12:16:45.355  Function execution took 97 ms, finished with
status: 'ok'
```

——-

Cloud Storage with Function Tutorial

This simple tutorial demonstrates writing, deploying, and triggering a Background Cloud Function with a Cloud Storage trigger.

Objectives

- Write and deploy a Background Cloud Function.
- Trigger the function by uploading a file to Cloud Storage.

Costs

This tutorial uses billable components of Cloud Platform, including:

- Google Cloud Functions
- Google Cloud Storage

Use the Pricing Calculator to generate a cost estimate based on your projected usage.

New Cloud Platform users might be eligible for a free trial.

Preparing the application

1. Create a Cloud Storage bucket to upload a test file, where `YOUR_TRIGGER_BUCKET_NAME` is a globally unique bucket name:

- gsutil mb gs://**YOUR_TRIGGER_BUCKET_NAME**

1. Clone the sample app repository to your local machine:

- NODE.JS 6
- NODE.JS 8 (BETA)
- PYTHON (BETA)
- git clone https://github.com/GoogleCloudPlatform/python-docs-samples.git

1. Alternatively, you can download the sample as a zip file and extract it.
2. Change to the directory that contains the Cloud Functions sample code:

- NODE.JS 6
- NODE.JS 8 (BETA)
- PYTHON (BETA)

cd python-docs-samples/functions/gcs/

Deploying and triggering the function

Currently, Cloud Storage functions are based on Pub/Sub notifications from Cloud Storage and support similar event types:

- finalize
- delete
- archive
- metadata update

The following sections describe how to deploy and trigger a function for each of these event types.

Object Finalize

Object finalize events trigger when a “write” of a Cloud Storage Object is successfully finalized. In particular, this means that creating a new object or overwriting an existing object triggers this event. Archive and metadata update operations are ignored by this trigger.

Object Finalize: deploying the function

Take a look at the sample function, which handles Cloud Storage events:

functions/gcs/main.py

VIEW ON GITHUB

```
def hello_gcs_generic(data, context):
    """Background Cloud Function to be triggered by Cloud
Storage.
       This generic function logs relevant data when a file
is changed.

    Args:
        data (dict): The Cloud Functions event payload.
        context (google.cloud.functions.Context): Metadata
of triggering event.
    Returns:
        None; the output is written to Stackdriver Logging
    """

    print('Event ID: {}'.format(context.event_id))
    print('Event type: {}'.format(context.event_type))
    print('Bucket: {}'.format(data['bucket']))
    print('File: {}'.format(data['name']))
    print('Metageneration:
{}'.format(data['metageneration']))
    print('Created: {}'.format(data['timeCreated']))
    print('Updated: {}'.format(data['updated']))
```

Cloud Functions looks for your code in a file named `main.py` .

To deploy the function, run the following command in the directory where the sample code is located:

PYTHON (BETA)

```
gcloud functions deploy hello_gcs_generic --runtime python37
--trigger-resource YOUR_TRIGGER_BUCKET_NAME --trigger-event
google.storage.object.finalize
```

where `YOUR_TRIGGER_BUCKET_NAME` is the name of the Cloud Storage bucket that triggers the function.

Object Finalize: triggering the function

To trigger the function:

1. Create an empty `gcf-test.txt` file in the directory where the sample code is located.
2. Upload the file to Cloud Storage in order to trigger the function:

- gsutil cp gcf-test.txt gs://**YOUR_TRIGGER_BUCKET_NAME**

1. where `YOUR_TRIGGER_BUCKET_NAME` is the name of your Cloud Storage bucket where you will upload a test file.
2. Check the logs to make sure the executions have completed:

- `gcloud functions logs read --limit 50`

```
ctions_v1beta2/worker.py", line 183, in call_user_function
                                                    event_context.Context(**request_or_event.context))
                                                  File "/user_code/main.py", line 8, in hello_gcs
                                                    print(f"Processing file: {file['name']}.")
                                                KeyError: 'name'
D       function-1          a50sh81gbidt    2018-10-26 11:21:52.663  Function execution took 73 ms, finished with status: 'crash'
D       hello_gcs_generic   278319035660757  2018-10-26 11:56:41.747  Function execution started
I       hello_gcs_generic   278319035660757  2018-10-26 11:56:41.780  Event ID: 278319035660757
I       hello_gcs_generic   278319035660757  2018-10-26 11:56:41.780  Event type: google.storage.object.finalize
I       hello_gcs_generic   278319035660757  2018-10-26 11:56:41.780  Bucket: testmybucket1234321
I       hello_gcs_generic   278319035660757  2018-10-26 11:56:41.780  File: gcf-test.txt
I       hello_gcs_generic   278319035660757  2018-10-26 11:56:41.780  Metageneration: 1
I       hello_gcs_generic   278319035660757  2018-10-26 11:56:41.780  Created: 2018-10-26T11:56:40.824Z
I       hello_gcs_generic   278319035660757  2018-10-26 11:56:41.780  Updated: 2018-10-26T11:56:40.824Z
D       hello_gcs_generic   278319035660757  2018-10-26 11:56:41.782  Function execution took 36 ms, finished with status: 'ok'
```

In this way, we can use App Engine and Cloud Functions.