Google Maps for Work

Serving raster layers on Google Cloud Platform

Last updated: 22 December 2014

Contents

Introduction

This document shows you how to serve custom raster layers using Google Cloud Storage, Google Compute Engine, and the Google Maps API. This approach is useful if you have a large quantity of satellite or aerial imagery that you need to serve at scale onto a Google map or other GIS tool.

Using this document, you'll complete the following steps:

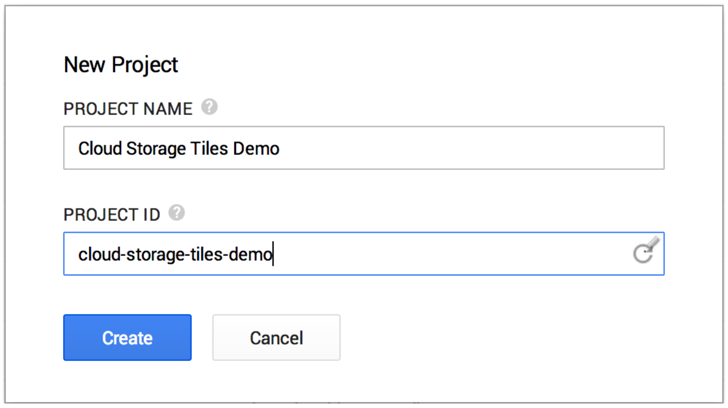
1. Configure a Google Cloud Platform project.
2. Upload imagery tiles to Google Cloud Storage.
3. Display tiles using the Google Maps API.
4. Serve map tiles using OGC standards.
5. Load balance your application.

Note: This document does not describe how to create raster tiles and assumes you already have raster tiles to upload and host in Google Cloud Storage. You can use one of many third-party tools to create raster tiles.

Before you begin, make sure you have a Google Account. Consider using a shared Google Account for your organization, rather than a personal Google Account.

1. Configure a Cloud Platform project for serving rasters

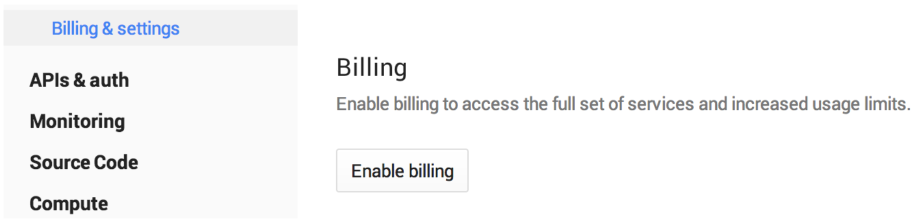
1. Sign in to the Google Developers Console and click **Create Project**.



PROJECT NAME: Cloud Storage Tiles Demo

PROJECT ID: cloud-storage-tiles-demo

2. Enable billing.



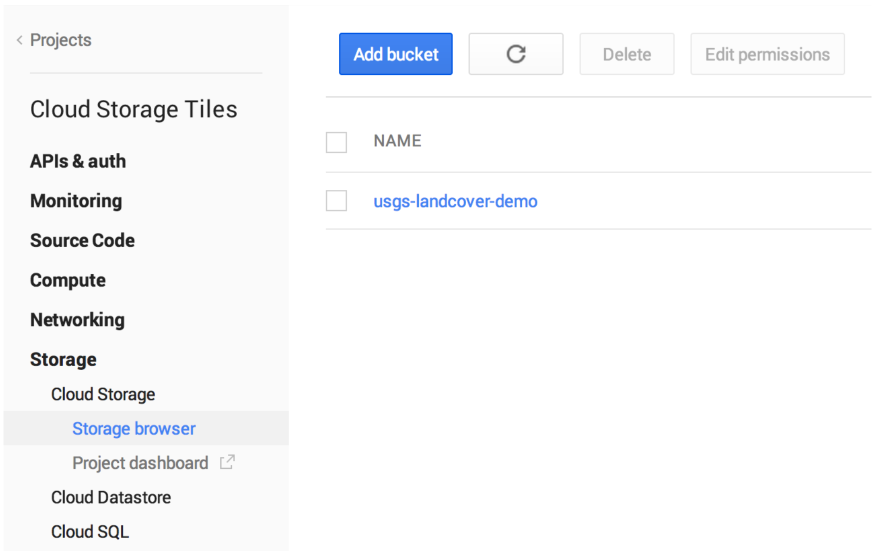
Enable billing to access the full set of services and increased usage limits.

For details about billing in Google Cloud Platform, see the Google APIs Console Help.

3. Add a credit card or bank account to the project for hosting (storage) and downloading (egress) the source imagery for the contract term.

   Google Cloud Platform provides a calculator for estimating monthly billing charges based on usage.

4. Create a Google Cloud storage bucket.



   Note: Cloud storage buckets are global, so if the name you choose is already in use, choose a different name.

5. Install and authenticate the Google Cloud SDK to easily conduct bulk downloads. Follow the installation instructions for your operating system.

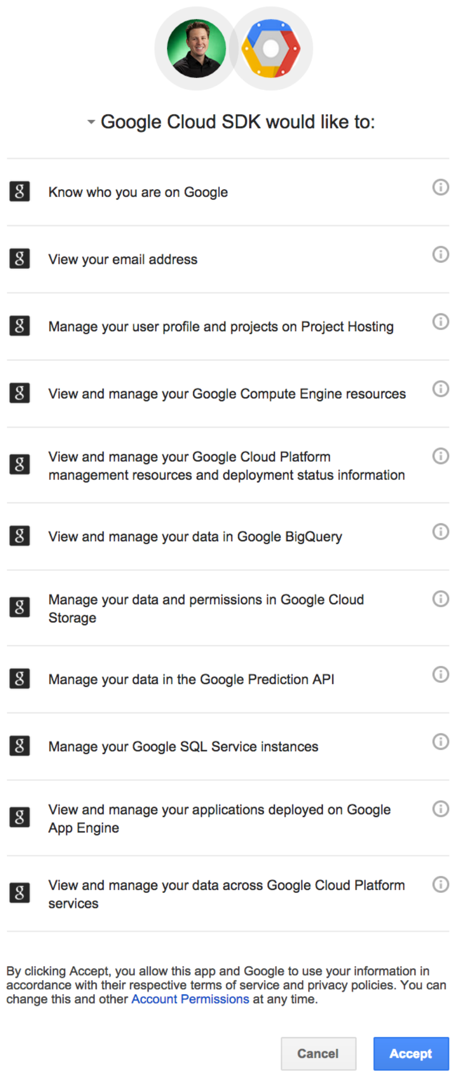
   Once you install the SDK, you'll have new system commands like `gcloud` and `gsutil`.

6. Use the following command to authorize the Cloud SDK to your Google Account. Use the same Google Account you used for the owner of the Cloud Platform Project and the Imagery Bucket.

```
$ gcloud auth login --project your-project-name
```

Details about this command

An OAuth 2 authorization screen will open a browser window, giving the Cloud SDK access to your project.



Google Cloud SDK would like to:

- Know who you are on Google
- View your email address
- Manage your user profile and projects on Project Hosting
- View and manage your Google Compute Engine resources
- View and manage your Google Cloud Platform management resources and deployment status information
- View and manage your data in Google BigQuery
- Manage your data and permissions in Google Cloud Storage
- Manage your data in the Google Prediction API
- Manage your Google SQL Service instances
- View and manage your applications deployed on Google App Engine
- View and manage your data across Google Cloud Platform services

By clicking Accept, you allow this app and Google to use your information in accordance with their respective terms of service and privacy policies. You can change this and other Account Permissions at any time.

2. Upload tiles to Google Cloud Storage

Access control decisions

You can find a detailed description of how Google Cloud Storage handles Authentication and Access Control in the Google Cloud Storage documentation. This document focuses on the use of access control lists (or ACLs) for public or private access.

With public access, the tiles you place on Google Cloud Storage are accessible to anyone on the Internet. With private access, only those to whom you explicitly grant access can view the tiles.

Private access is controlled using OAuth 2.0, and requires viewers to present credentials based one of the following:

- A Google Account (such as Gmail or Google Apps)
- A Google Account that is a member of an approved Google Group
- Access via a third-party service or proxy that handles authentication via server-side authentication (a web service that accepts a username and password and proxies data with a server-side access token).

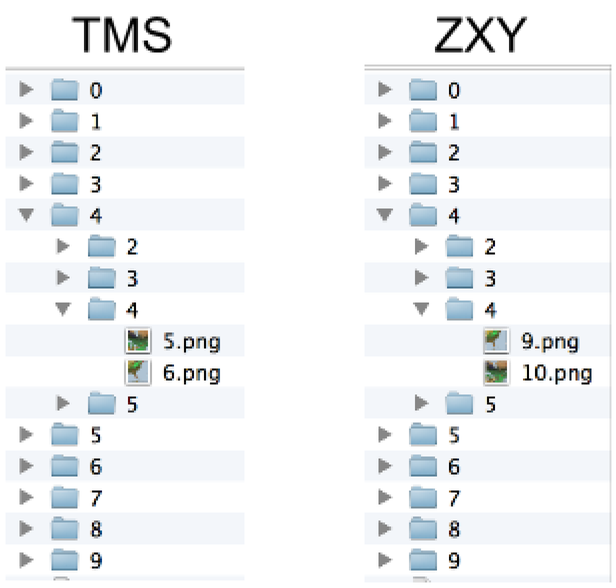
Map tiles

Map tiles can have various naming standards and can vary based on map projection. This document focuses on map tiles in a `z/x/y.png` naming standard, where

- **`Z`** is the current zoom level, and
- **`X`** is the X coordinate and **`Y`** is the Y coordinate, measured from the top left of the map for each zoom level

Google Maps, ESRI, Bing, Open Street Map, and others use this top left "standard." However, the OGC standard, called TMS, starts from the bottom left.

You can see the differences in the two examples below, which show a typical tiles file structure (`Z/X/Y.png`), where, in the 4th X column of zoom level 4, there are two images for the Y coordinates. Note that their names are different based on whether counting began at the top left of bottom left.



The file structure in which your software exported the tiles will determine the logic you need to use in your application when you want to load a specific tile. However, the file structure doesn't impact the actual upload process.

For example purposes, this document uses the USGS National Land Cover Database.

Uploading tiles in bulk for public access

To make the tiles publicly accessible to anyone on the Internet, set the default ACL for them as `-public read`. In the directory where you have your tiles in a subfolder called `zxy`, run:

```
$ gsutil -m cp -a public-read -R zxy/ gs://usgs-landcover-demo/
```

Details about this command

The `-m` parameter will make the `cp` process run in parallel threads, which will greatly speed up your upload.

Uploading tiles in bulk for private access

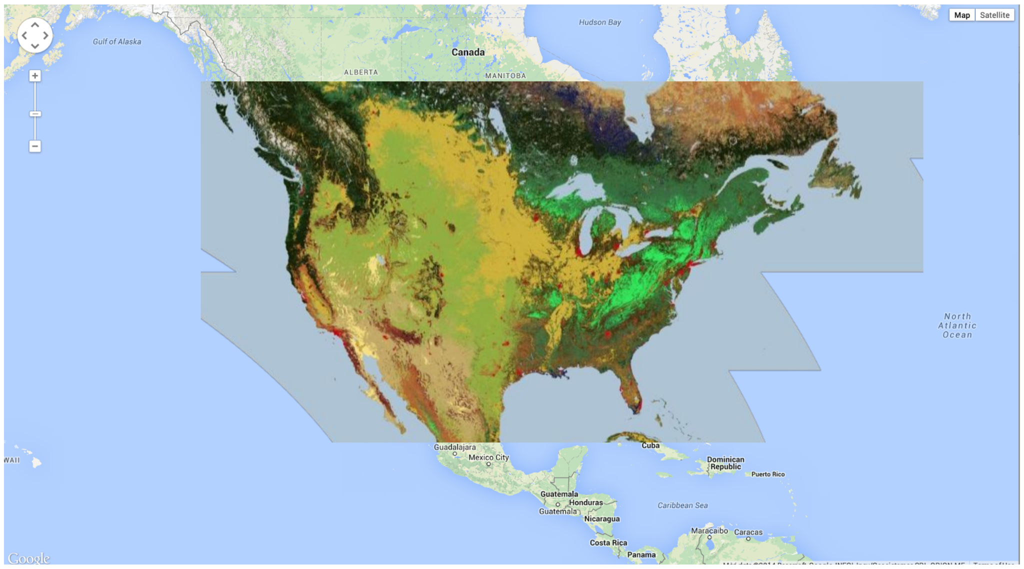
To manage access to private tiles, you can use Google Groups. With Groups, it's easy to grant read permission to a Google Cloud Storage bucket to members of a group, and then manage write access for individual users using the standard Google Groups administration tools. For this example, we'll use a group named Cloud-Storage-Tiles-Private. To join this group, click **Join** on this page:

https://groups.google.com/forum/#!forum/cloud-storage-tiles-private

```
$ gsutil mb -p cloud-storage-tiles-demo gs://usgs-landcover-demo-private/
$ gsutil defacl ch -g cloud-storage-tiles-private@googlegroups.com:R
gs://usgs-landcover-demo-private/
```

3. Use tiles from Google Cloud Storage with the Google Maps API

Maps API Public tiles example



See the demo

The following code snippet shows the Google Maps API JavaScript you need to call the tiles as a Google Maps API ImageMapType. These files reside in the `/z/x/y.png` structure in the same bucket in which the HTML that loads the Maps API resides.

```
var imageMapType = new google.maps.ImageMapType({
            getTileUrl: function(coord, zoom) {
              var proj = map.getProjection();
              var z2 = Math.pow(2, zoom);
              var tileXSize = 256 / z2;
              var tileYSize = 256 / z2;
              var tileBounds = new google.maps.LatLngBounds(
                proj.fromPointToLatLng(new google.maps.Point(coord.x * tileXSize,
(coord.y + 1) * tileYSize)),
                proj.fromPointToLatLng(new google.maps.Point((coord.x + 1) *
tileXSize, coord.y * tileYSize))
              );
              return
"{z}/{x}/{y}.png".replace('{z}',zoom).replace('{x}',coord.x).replace('{y}',coord.y)
;
            },
            tileSize: new google.maps.Size(256, 256),
            minZoom: mapMinZoom,
            maxZoom: mapMaxZoom,
            name: 'Tiles'
        });
```

You can also host the HTML page on Google App Engine, Google Compute Engine, or your own web server and still point to tiles in a Google Cloud storage bucket. To do this, just modify the URL from a local reference to a remote reference, as follows:

```
return
"{z}/{x}/{y}.png".replace('{z}',zoom).replace('{x}',coord.x).replace(
'{y}',coord.y); to
return
"http://storage.googleapis.com/usgs-landcover-demo/zxy/{z}/{x}/{y}.pn
g".replace('{z}',zoom).replace('{x}',coord.x).replace('{y}',coord.y);
```

For TMS tiles, you need to flip the Y coordinate, as shown in the following code sample:

```
var imageMapType = new google.maps.ImageMapType({
            getTileUrl: function(coord, zoom) {
              var proj = map.getProjection();
              var z2 = Math.pow(2, zoom);
              var tileXSize = 256 / z2;
              var tileYSize = 256 / z2;
              var tileBounds = new google.maps.LatLngBounds(
                proj.fromPointToLatLng(new google.maps.Point(coord.x * tileXSize,
(coord.y + 1) * tileYSize)),
                proj.fromPointToLatLng(new google.maps.Point((coord.x + 1) *
tileXSize, coord.y * tileYSize))
              );
              // Flip the Y value
              var ymax = 1 << zoom;
              var Y = ymax - coord.y - 1;
              return
"{z}/{x}/{y}.png".replace('{z}',zoom).replace('{x}',coord.x).replace('{y}',Y);
```

```
            },
            tileSize: new google.maps.Size(256, 256),
            minZoom: mapMinZoom,
            maxZoom: mapMaxZoom,
            name: 'Tiles'
        });
```

See a demo

Maps API Google Cloud Storage cookie auth example

For this example, we'll use the Google group named Cloud-Storage-Tiles-Private. To join this group, click **Join** on this page:

https://groups.google.com/forum/#!forum/cloud-storage-tiles-private

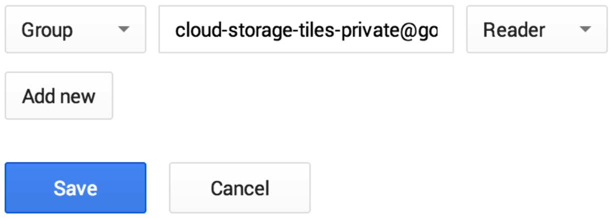
You must be a member of this group to view the tiles in the following demo:

https://storage.cloud.google.com/usgs-landcover-demo/zxy/cookie-zxy.html

From the Google Cloud Storage Documentation:

> Google Cloud Storage lets you provide browser-based authenticated downloads to users who do not have Google Cloud Storage accounts. To do this, apply Google Account-based ACLs to the object and then provide users with a URL that is scoped to the object. The URL for browser-based authenticated downloads is:
>
> https://storage.cloud.google.com/bucket/object
>
> Note: This URL base is slightly different than the `http://storage.googleapis.com/` base this document uses for the public map examples.

Once you've uploaded your HTML file, you can add or modify the permissions. Make sure you've applied the group you're using for the bucket's default reader ACL.



When a user visits the private web page URL in their browser, the user is automatically prompted to sign in to their Google Account (if not already signed in). After the user is authenticated and the browser has acquired a cookie with an encapsulated identity token, the user is redirected to the page in the Google Cloud Storage repository. Google Cloud Storage then verifies that the user is allowed to read the page, and then loads the page into the browser.

In the code sample below, note that it is a requirement that the absolute, and not localized URL pattern, be given for the tiles:

```
var imageMapType = new google.maps.ImageMapType({
            getTileUrl: function(coord, zoom) {
              var proj = map.getProjection();
              var z2 = Math.pow(2, zoom);
              var tileXSize = 256 / z2;
              var tileYSize = 256 / z2;
              var tileBounds = new google.maps.LatLngBounds(
                proj.fromPointToLatLng(new google.maps.Point(coord.x * tileXSize,
(coord.y + 1) * tileYSize)),
                proj.fromPointToLatLng(new google.maps.Point((coord.x + 1) *
tileXSize, coord.y * tileYSize))
              );
              return
"https://storage.cloud.google.com/usgs-landcover-demo/zxy/{z}/{x}/{y}.png".replace('
{z}',zoom).replace('{x}',coord.x).replace('{y}',coord.y);
            },
            tileSize: new google.maps.Size(256, 256),
            minZoom: mapMinZoom,
            maxZoom: mapMaxZoom,
            name: 'Tiles'
        });
```

Maps API Google Cloud Storage OAuth 2.0 Example

OAuth 2.0 is an open authentication protocol used frequently in Google projects. It offers authentication for both client- and server-based applications. It's useful for managing quota usage for APIs that have daily limits or costs associated with them, and for restricting quota usage and data consumption to specific domains.

For this example, we'll again use the Google group named Cloud-Storage-Tiles-Private. To join this group, click **Join** on this page:

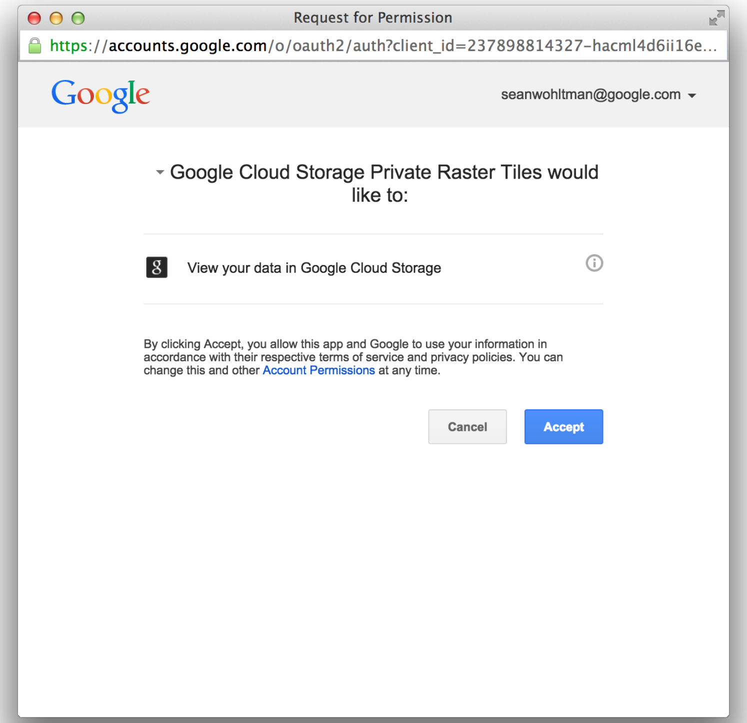
https://groups.google.com/forum/#!forum/cloud-storage-tiles-private

You must be a member of this group to view the tiles in the following demo:

http://storage.googleapis.com/usgs-landcover-demo-private/zxy/private-zxy.html

As a member of the group, you'll first see a page that asks you to authorize. Click the **Authorize** button to provide the required client-side OAuth 2.0 credentials to access the tiles.

Next, you'll see a screen that asks you to grant the application permission to access files in Google Cloud Storage on your behalf.

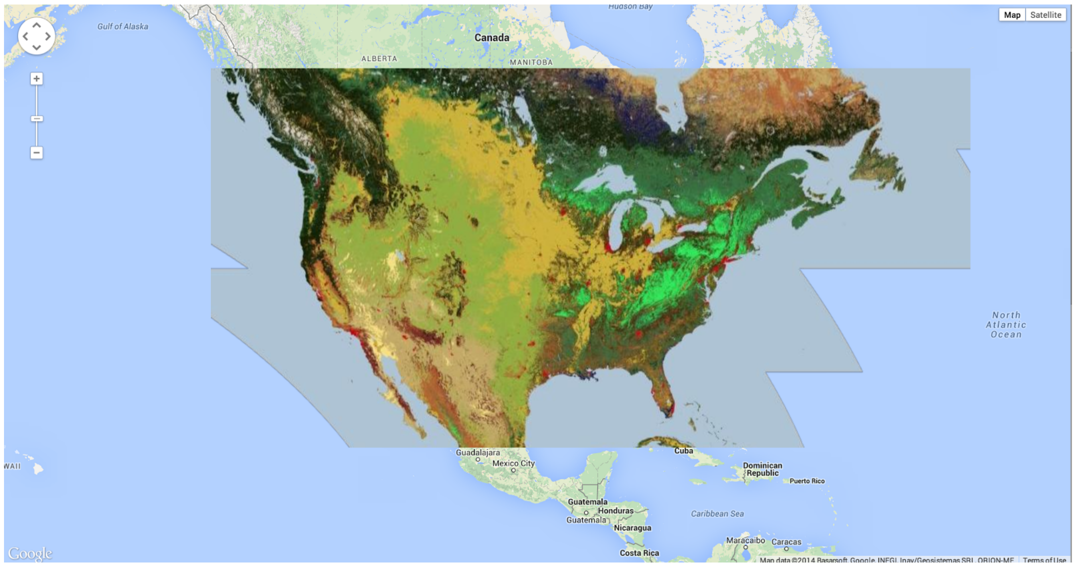

Request for Permission
https://accounts.google.com/o/oauth2/auth?client_id=237898814327-hacml4d6ii16e...
Google
seanwohltman@google.com
Cancel
Accept

Google Cloud Storage Private Raster Tiles would like to:

View your data in Google Cloud Storage

By clicking Accept, you allow this app and Google to use your information in accordance with their respective terms of service and privacy policies. You can change this and other Account Permissions at any time.

Click **Accept**. A map will appear and load the privately ACL'd tiles.



How this demo works

The top of the source code for the demo above is a block of JavaScript:

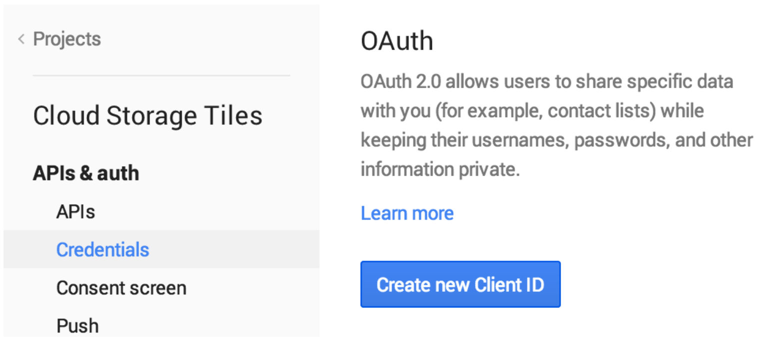
```
<script>
      // The Client ID for your application, as configured on the Google APIs
console.
      var clientId =
'237898814327-hacml4d6ii16ekeopho52vepp5cq6e5o.apps.googleusercontent.com';

      // The oauth scope for displaying Google Cloud Storage data.
      var scopes = 'https://www.googleapis.com/auth/devstorage.read_only';

      var access_token = '';

      function initialize() {
        authorizationFlow(authorizationComplete, refreshComplete);
      }

      function authorizationComplete(authResult) {
        access_token = authResult.access_token;

        initMap();
      }
```

This code configures the OAuth 2.0 Workflow parameters. The first defined variable is an OAuth 2.0 ClientID.

To create a Client ID:

1. From the **API's & Auth** list in the Developers Console project, click **Credentials.**

2. Click **Create new Client ID**:



< Projects

Cloud Storage Tiles

APIs & auth

APIs

Credentials

Consent screen

Push

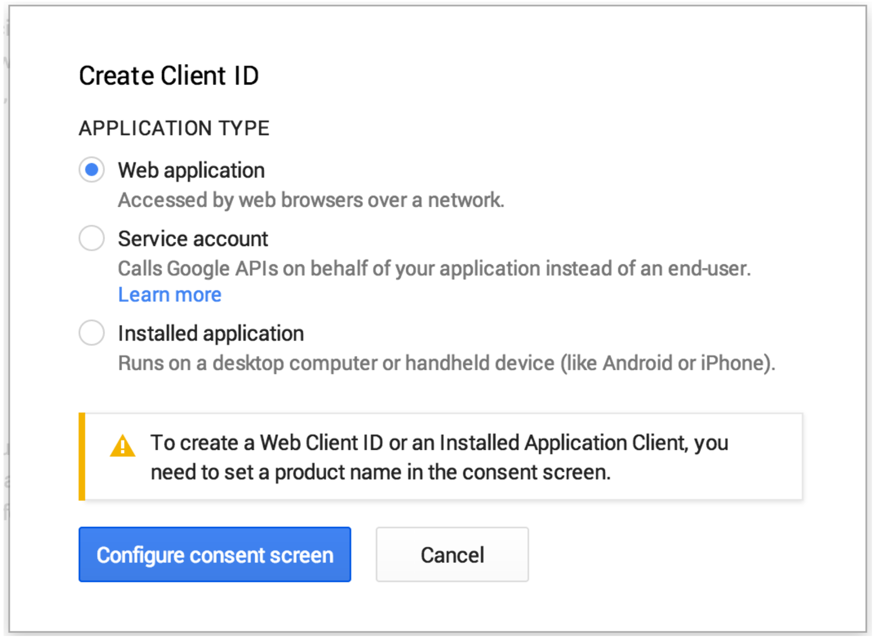
OAuth

OAuth 2.0 allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private.

Learn more

Create new Client ID

3. On the “Create Client ID” screen, select **Web application** (default) for the **Application Type**. (This application is accessed by a web browser over a network, and is not a server-side or installed application.)



Create Client ID

APPLICATION TYPE

● Web application
Accessed by web browsers over a network.

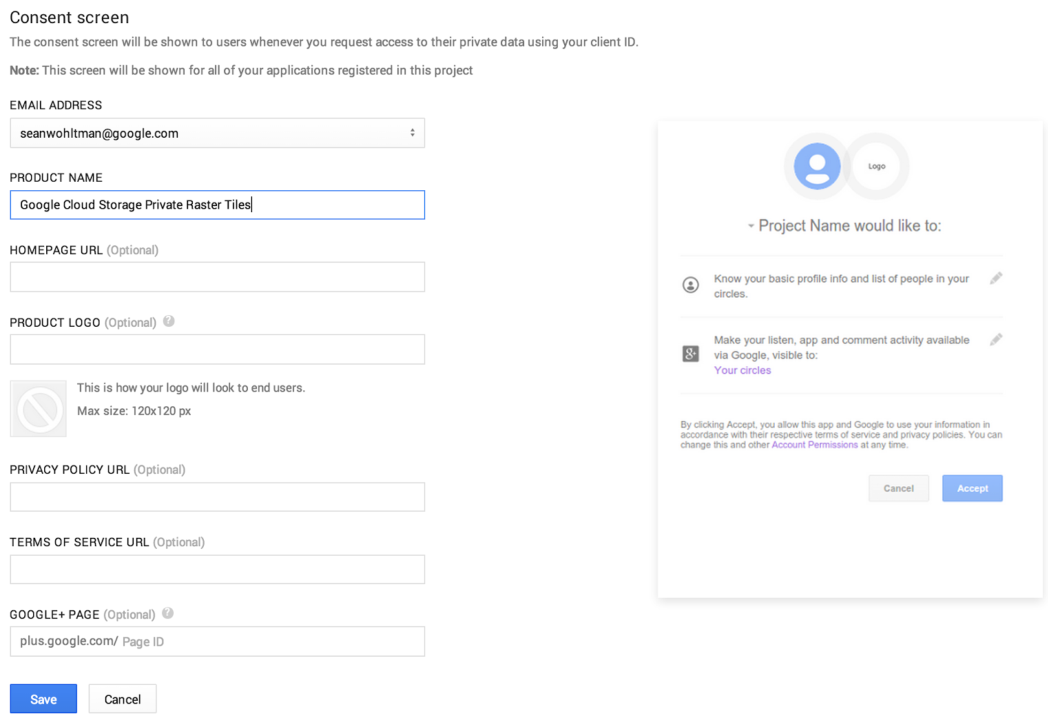
○ Service account
Calls Google APIs on behalf of your application instead of an end-user.
Learn more

○ Installed application
Runs on a desktop computer or handheld device (like Android or iPhone).

To create a Web Client ID or an Installed Application Client, you need to set a product name in the consent screen.

Configure consent screen | Cancel

4. Click **Configure consent screen**.



Consent screen

The consent screen will be shown to users whenever you request access to their private data using your client ID.

Note: This screen will be shown for all of your applications registered in this project

EMAIL ADDRESS
seanwohltman@google.com

PRODUCT NAME
Google Cloud Storage Private Raster Tiles

HOMEPAGE URL (Optional)

PRODUCT LOGO (Optional)

This is how your logo will look to end users.
Max size: 120x120 px

PRIVACY POLICY URL (Optional)

TERMS OF SERVICE URL (Optional)

GOOGLE+ PAGE (Optional)
plus.google.com/ Page ID

Save Cancel

Logo

Project Name would like to:

Know your basic profile info and list of people in your circles.

Make your listen, app and comment activity available via Google, visible to:
Your circles

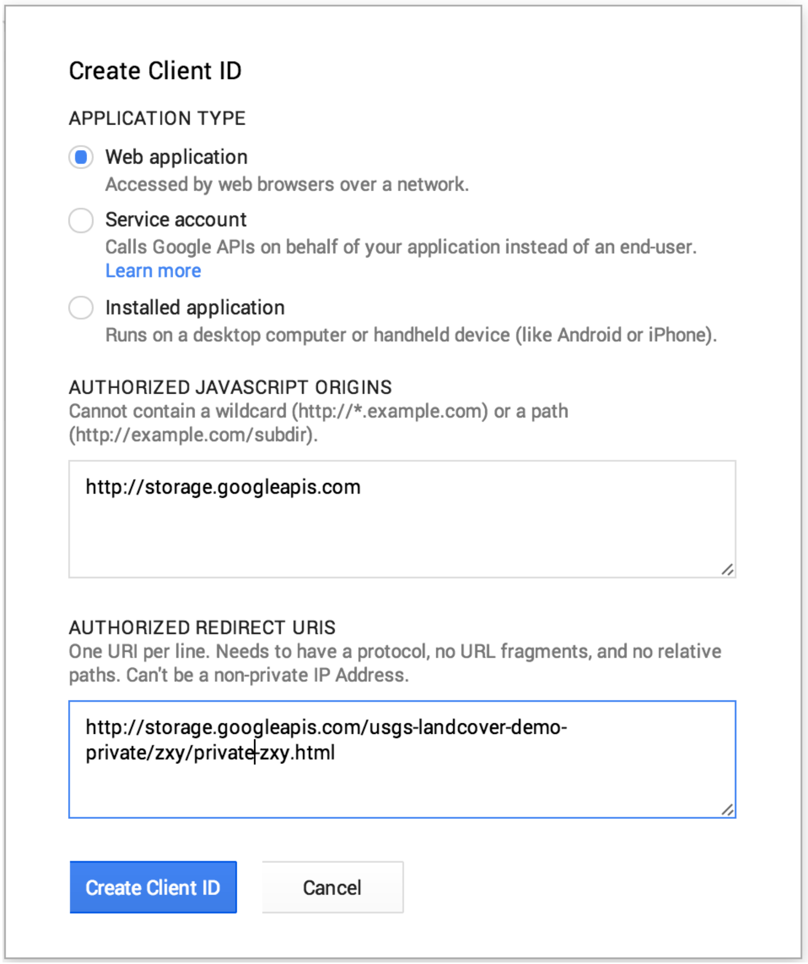
By clicking Accept, you allow this app and Google to use your information in accordance with their respective terms of service and privacy policies. You can change this and other Account Permissions at any time.

Cancel Accept

The user sees the consent screen after invoking the function to authorize the application. We recommend that you provide logos and links for your application, to encourage users to trust the application to which they are granting permissions to access their data.

5. At a minimum, enter your contact email address and a project name. Then click **Save**.

6. On the “Create Client ID” screen, provide the **Authorized JavaScript Origins** and the **Authorized Redirect URIs**.



Create Client ID

APPLICATION TYPE

[✓ Web application]
Accessed by web browsers over a network.

Service account
Calls Google APIs on behalf of your application instead of an end-user.
Learn more

Installed application
Runs on a desktop computer or handheld device (like Android or iPhone).

AUTHORIZED JAVASCRIPT ORIGINS
Cannot contain a wildcard (http://*.example.com) or a path (http://example.com/subdir).

http://storage.googleapis.com

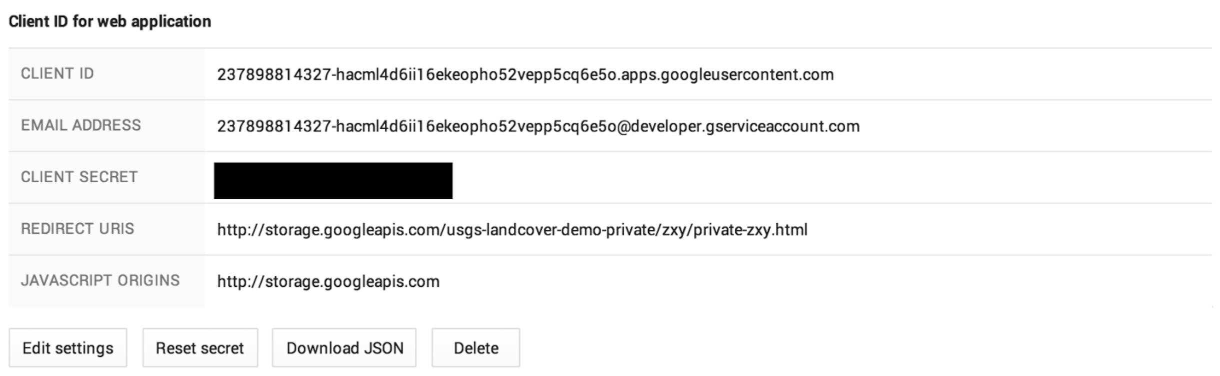
AUTHORIZED REDIRECT URIS
One URI per line. Needs to have a protocol, no URL fragments, and no relative paths. Can't be a non-private IP Address.

http://storage.googleapis.com/usgs-landcover-demo-private/zxy/private-zxy.html

This information prevents someone from copying your HTML/JavaScript and hosting it on an insecure or third-party server, to intercept user data or use your project's quotas.

In this example, we want to host a private map on Google Cloud Storage, so we need to set the **Authorized JavaScript Origins** to `http://storage.gooogleapis.com`. We also need to set the **Authorized Redirect URIs** to the final URL at which the HTML page will be hosted. Once the user is authorized, the OAuth worfklow redirects the user back to the original page, this time with a URL parameter that is their access token. When the application sees that there's an access token, it will then replace the blank page with the **Authorize** button by initializing the Google Maps API.

Finally, a unique **Client ID** entry will be created. You'll add this Client ID to the JavaScript for the variable `clientID` in the example.



Client ID for web application

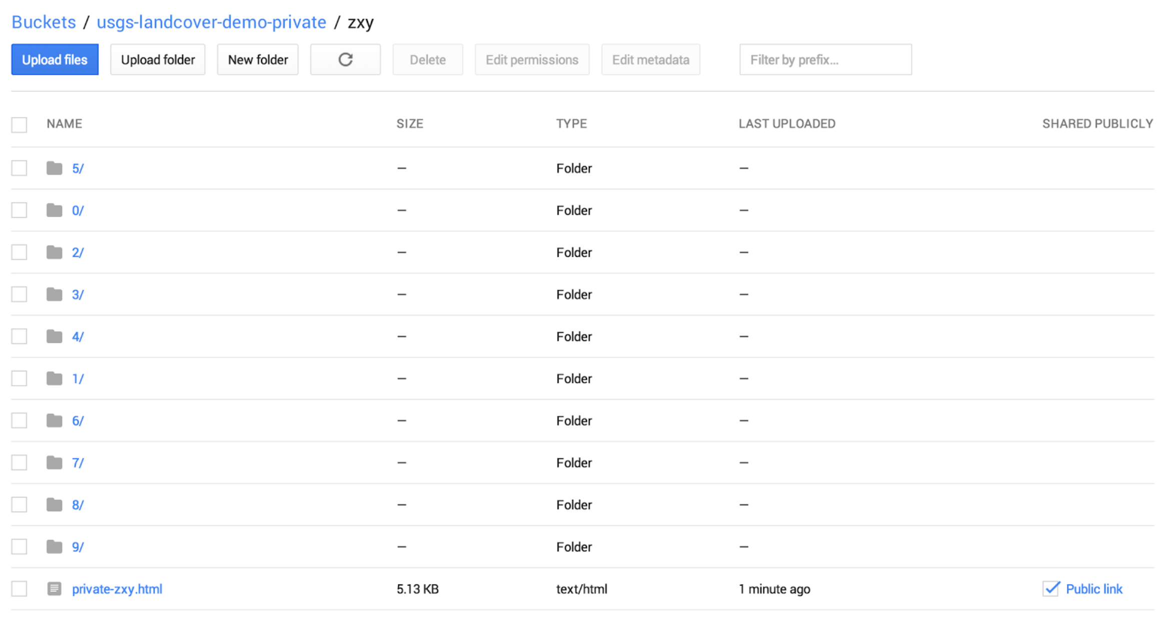
CLIENT ID	237898814327-hacml4d6ii16ekeopho52vepp5cq6e5o.apps.googleusercontent.com
EMAIL ADDRESS	237898814327-hacml4d6ii16ekeopho52vepp5cq6e5o@developer.gserviceaccount.com
CLIENT SECRET	[redacted]
REDIRECT URIS	http://storage.googleapis.com/usgs-landcover-demo-private/zxy/private-zxy.html
JAVASCRIPT ORIGINS	http://storage.googleapis.com

The following line defines the OAuth 2.0 Scope, which provides the data access rights that the application needs the user to grant to the application:

```
var scopes = 'https://www.googleapis.com/auth/devstorage.read_only';
```

At Google, these scopes are tied to individual services. In this case, we are calling the `devstorage` scope (Google Cloud Storage), and because the user just needs to access (not write) tiles, the application only needs `read_only` access. It's best to keep scopes as narrow and conservative as possible, both for security and user trust. For example, you wouldn't want to grant a web page that is supposed to show your map tiles the ability to post to your Google+ profile or read your Gmail contacts list.

Because we want to allow users to access the web page before the users provide their credentials to initiate the OAuth 2.0 workflow, we need to make the actual web page publically accessible. Don't worry, users will not be able to access the privately ACL'd tiles in the bucket, because we are not going to modify their ACLs; however, we'll make the HTML page in the bucket publically accessible.



Buckets / usgs-landcover-demo-private / zxy

	NAME	SIZE	TYPE	LAST UPLOADED	SHARED PUBLICLY
☐	5/	–	Folder	–	
☐	0/	–	Folder	–	
☐	2/	–	Folder	–	
☐	3/	–	Folder	–	
☐	4/	–	Folder	–	
☐	1/	–	Folder	–	
☐	6/	–	Folder	–	
☐	7/	–	Folder	–	
☐	8/	–	Folder	–	
☐	9/	–	Folder	–	
☐	private-zxy.html	5.13 KB	text/html	1 minute ago	✓ Public link

Note that OAuth 2.0 credentials expire after a given amount of time. At Google, the default timeout is 60 minutes. The timeout is a security feature that prevents someone who may have nefariously intercepted an access token from using that token for a prolonged period of time.

This application refreshes every hour to retrieve a fresh access token:

```
function handleAuthResult(authResult) {
          var authorizeButton = document.getElementById('authorize_button');

          // Has the user authorized this application?
          if (authResult && !authResult.error) {
            // The application is authorized. Hide the 'Authorization' button.
            authorizeButton.style.display = 'none';
            authorization_complete(authResult);

            // We must refresh the token after it expires.
            window.setTimeout(refreshToken, authResult.expires_in * 1000);
```

Finally, unlike the public examples, the private examples append a query parameter to the tile URL, `?access_token={access_token}`. When requesting data that's protected by private ACLs from Google Cloud Storage, you must pass the `access_token` query parameter with a valid and properly scoped access token.

```
return
"{z}/{x}/{y}.png?access_token={access_token}".replace('{z}',zoom).replace('{x}',coor
d.x).replace('{y}',coord.y).replace('{access_token}', access_token);
```

4. Wrap OGC endpoints around tiles from Google Cloud Storage with Google Compute Engine

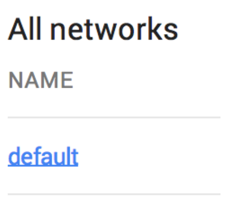
Creating a Google Compute Engine VM

To enable OGC-compliant endpoints, you can host access to the imagery tiles from the open-source application MapProxy. In this section, you'll create a Linux virtual machine within Google Compute Engine to host MapProxy, and also create a custom firewall configuration to permit incoming traffic to MapProxy.

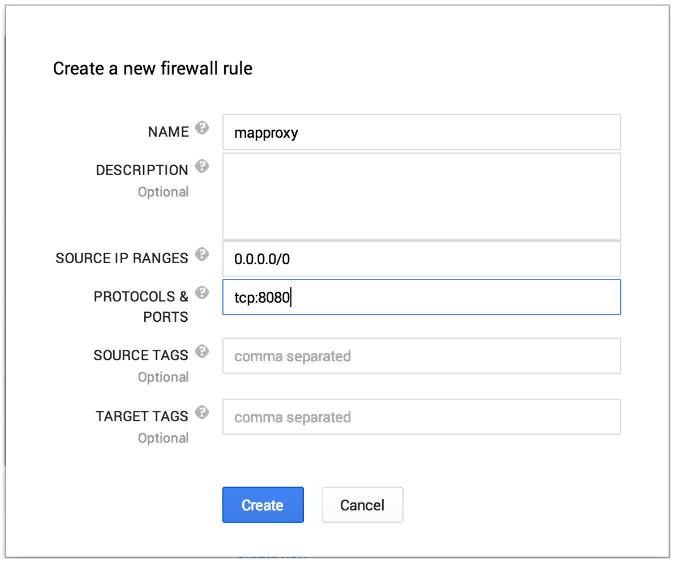
The Google Compute Engine Developers Center has several resources to help you get started running virtual machines, and in-depth information for configuring firewall rules.

1. From the **Compute Engine** list in the Developers Console project, click **Networks**.

2. Under **All networks**, click **default** to add a new firewall rule. This rule lets you test the development server on port 8080.



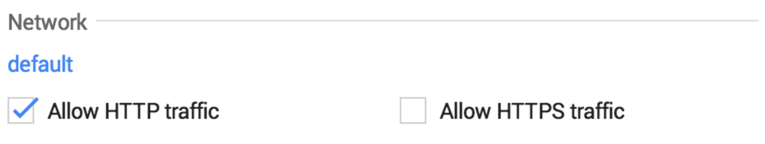
3. Create a firewall rule.



Create a new firewall rule

NAME	mapproxy
DESCRIPTION Optional	
SOURCE IP RANGES	0.0.0.0/0
PROTOCOLS & PORTS	tcp:8080
SOURCE TAGS Optional	comma separated
TARGET TAGS Optional	comma separated

4. Enable port 80 by clicking on **Allow HTTP traffic**.



Network
default

[✓] Allow HTTP traffic
[ ] Allow HTTPS traffic

Installing and configuring MapProxy

1. Create a new Ubuntu 14.04 Compute Engine instance in your project:

```
gcloud compute instances create mapproxy --image
ubuntu-1404-trusty-v20141031a  --image-project ubuntu-os-cloud --zone
us-central1-a
```

2. Connect via SSH, using a terminal or the Developers Console browser terminal.

```
$ sudo apt-get update
$ sudo apt-get install python-virtualenv
$ sudo mkdir /mapproxy
$ sudo chmod 777 /mapproxy
$ cd /mapproxy
$ virtualenv --system-site-packages mapproxy
$ sudo apt-get install python-imaging python-yaml libproj0
$ sudo apt-get install libgeos-dev python-lxml libgdal-dev python-shapely
$ sudo apt-get install build-essential python-dev libjpeg-dev zlib1g-dev
libfreetype6-dev
$ source mapproxy/bin/activate
$ pip install MapProxy
$ nano gcs.yaml
```

3. Copy and paste the following MapProxy configuration file into the nano editor. Note that you'll need to change the URL parameter to match your GCS bucket.

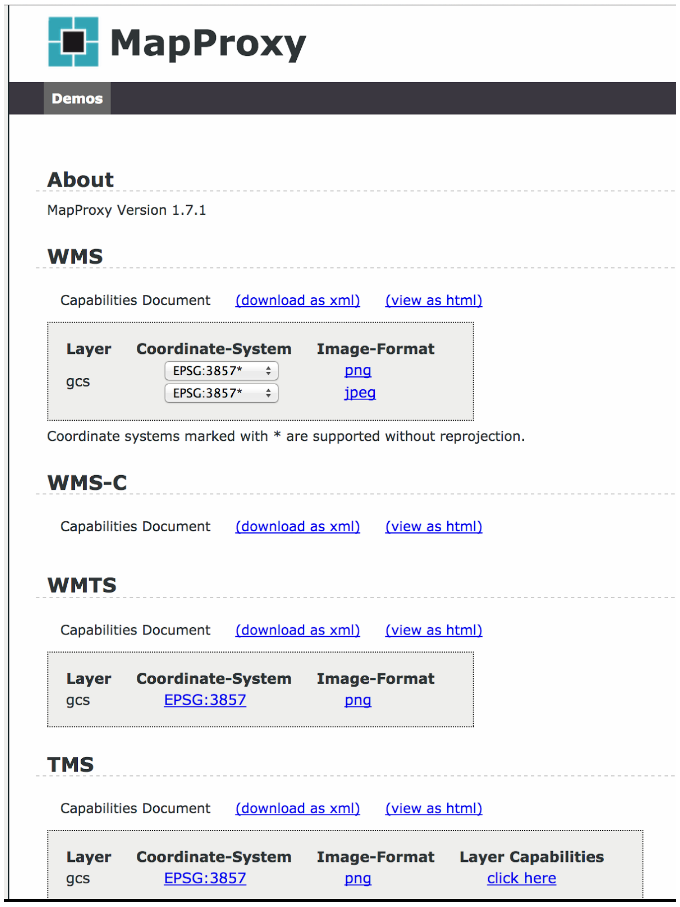
```
services:
  demo:
  tms:
    use_grid_names: true
    # origin for /tiles service
    origin: 'nw'
  kml:
      use_grid_names: true
  wmts:
  wms:
    md:
      title: MapProxy Google Cloud Storage
      abstract: This is a minimal MapProxy example.
layers:
 - name: gcs
   title: Google Cloud Storage
   sources: [gcs_cache]
caches:
 gcs_cache:
   grids: [webmercator]
   sources: [gcs]
sources:
 gcs:
   type: tile
   grid: webmercator
   url:
http://storage.googleapis.com/usgs-landcover-demo/zxy/%(z)s/%(x)s/%(y)s
.png
   coverage:
    srs: 'EPSG:4326'
    bbox: [-125.850,24.530,-66.800,50.289]
grids:
 webmercator:
  base: GLOBAL_WEBMERCATOR
```

4. Save the file (**Ctrl+O**) and exit (**Ctrl+X**) You can now test the configuration.

```
$ mapproxy-util serve-develop -b 0.0.0.0:8080 gcs.yaml
```

5. Visit the development server at:

```
http://YourIP:8080/demo/
```



MapProxy

Demos

About

MapProxy Version 1.7.1

WMS

Capabilities Document (download as xml) (view as html)

Layer	Coordinate-System	Image-Format
gcs	EPSG:3857*	png
	EPSG:3857*	jpeg

Coordinate systems marked with * are supported without reprojection.

WMS-C

Capabilities Document (download as xml) (view as html)

WMTS

Capabilities Document (download as xml) (view as html)

Layer	Coordinate-System	Image-Format
gcs	EPSG:3857	png

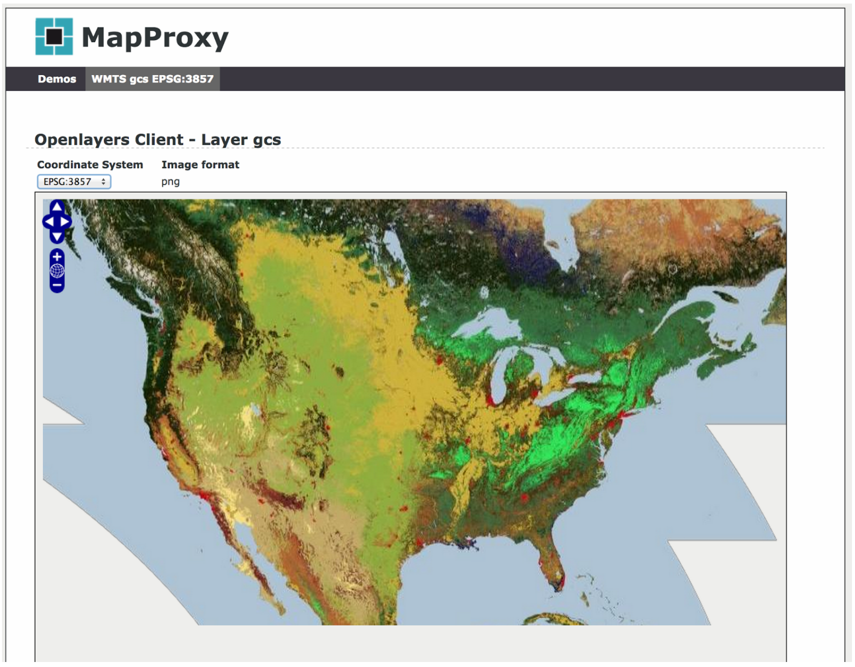
TMS

Capabilities Document (download as xml) (view as html)

Layer	Coordinate-System	Image-Format	Layer Capabilities
gcs	EPSG:3857	png	click here

There will be several different services, based on the configuration file that you used.

6. Click any of the PNG or JPEG links to try out the OGC services.



7. Once everything appears to be working, configure MapProxy to work with Apache via WSGI.

```
$ Ctrl+C to kill the development server

$ rm -rf cache_data

$ mapproxy-util create -t wsgi-app -f gcs.yaml config.py

$ deactivate

$ sudo apt-get install apache2 apache2-mpm-prefork apache2-utils
libexpat1 ssl-cert

$ sudo aptitude install libapache2-mod-wsgi

$ sudo nano /etc/apache2/apache2.conf
```

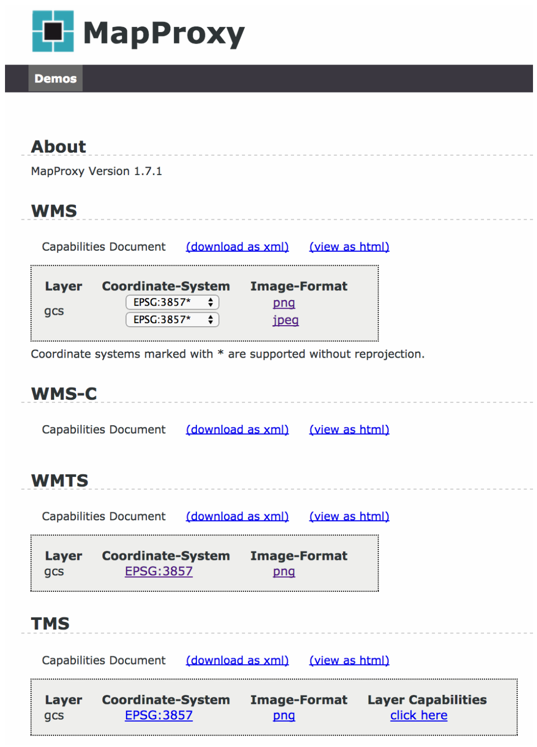
8. Scroll down to the end of the file and add the following code block. Then save the file and exit.

```
WSGIScriptAlias /mapproxy /mapproxy/config.py
WSGIDaemonProcess mapproxy-wsgi-daemon processes=4 threads=8
WSGIProcessGroup mapproxy-wsgi-daemon
WSGIPythonHome /mapproxy/mapproxy
WSGIApplicationGroup %{GLOBAL}
<Directory /mapproxy>
        Options All
        AllowOverride All
        Require all granted
</Directory>
```

```
$ sudo service apache2 restart
```

9. Access MapProxy via the Apache webserver listening on port 80:

   `http://YourIP/mapproxy/demo`



About

MapProxy Version 1.7.1

WMS

Capabilities Document (download as xml) (view as html)

Layer	Coordinate-System	Image-Format
gcs	EPSG:3857*	png
	EPSG:3857*	jpeg

Coordinate systems marked with * are supported without reprojection.

WMS-C

Capabilities Document (download as xml) (view as html)

WMTS

Capabilities Document (download as xml) (view as html)

Layer	Coordinate-System	Image-Format
gcs	EPSG:3857	png

TMS

Capabilities Document (download as xml) (view as html)

Layer	Coordinate-System	Image-Format	Layer Capabilities
gcs	EPSG:3857	png	click here

Load balancing

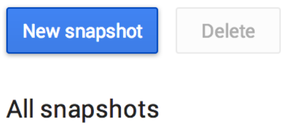
Wrapping services such as WMS and WMTS to tiles stored on Google Cloud storage introduces overhead. When a WMS request comes in, MapProxy has to download the tiles from Cloud Storage that intersect with the requested bounding box, composite them, and then cut them into a single new image. Unlike Cloud Storage, MapProxy running on a Compute Engine VM is, by default, not designed to handle very high QPS or support numerous users simultaneously. For this reason, we recommend that you incorporate load balancing.

For a few users who occasionally use your OGC services, you might be able to have just one VM running; however, you'll need the ability to scale to handle higher demand. Compute Engine offers many ways to configure load balancing; this document shows a network load balanced approach with autoscaling.

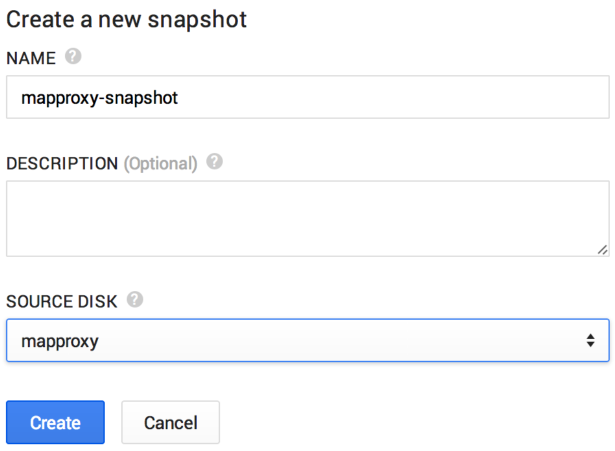
Create a snapshot of your working MapProxy VM

1. From the **Compute Engine** list in the Developers Console project, click **Snapshots**.

2. Click **New snapshot**.



3. Give the snapshot a name, and chose your working VM as the source disk. Then click **Create**.


Create a new snapshot

NAME
mapproxy-snapshot

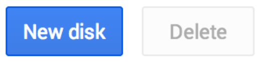
DESCRIPTION (Optional)

SOURCE DISK
mapproxy

Create a disk from the snapshot

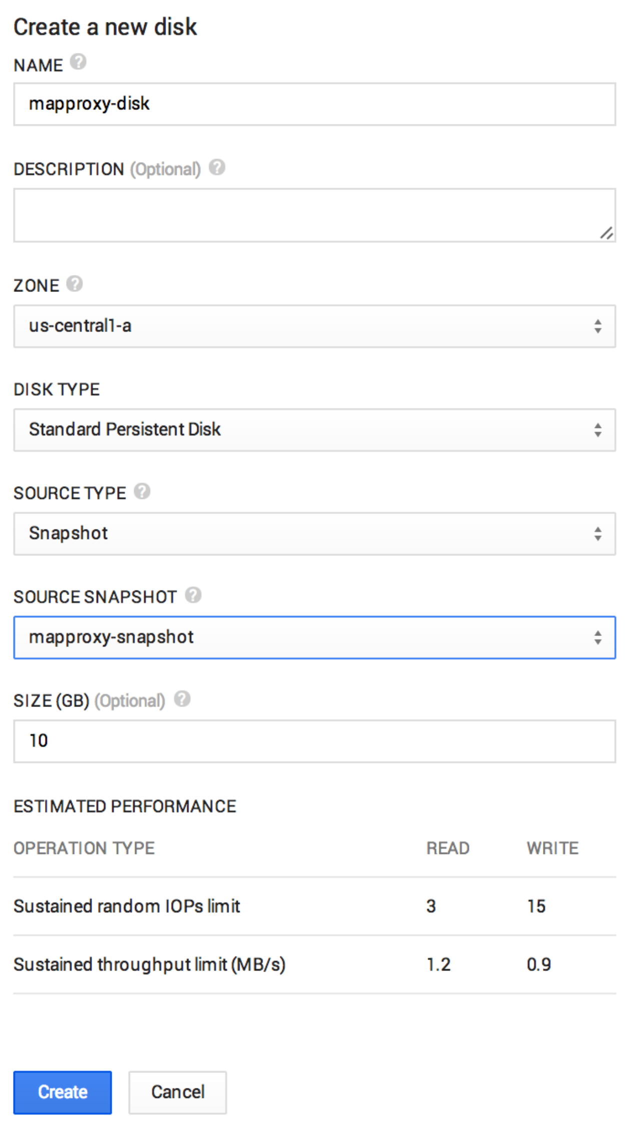
1. From the **Compute Engine** list in the Developers Console project, click **Disks**.

2. Click **New disk**.



3. Give the new disk a name, and chose the zone in which you'd like the disk to reside. This should be the same as the zone where you'd like to run your VMs.

   For the **Source Type**, chose **Snapshot**, and then chose the snapshot you created in the previous step as the source. Then click **Create**.



Create a new disk

NAME: mapproxy-disk

DESCRIPTION (Optional):

ZONE: us-central1-a

DISK TYPE: Standard Persistent Disk

SOURCE TYPE: Snapshot

SOURCE SNAPSHOT: mapproxy-snapshot

SIZE (GB) (Optional): 10

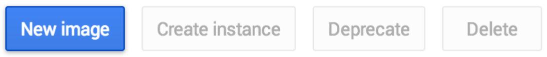
ESTIMATED PERFORMANCE

OPERATION TYPE	READ	WRITE
Sustained random IOPs limit	3	15
Sustained throughput limit (MB/s)	1.2	0.9

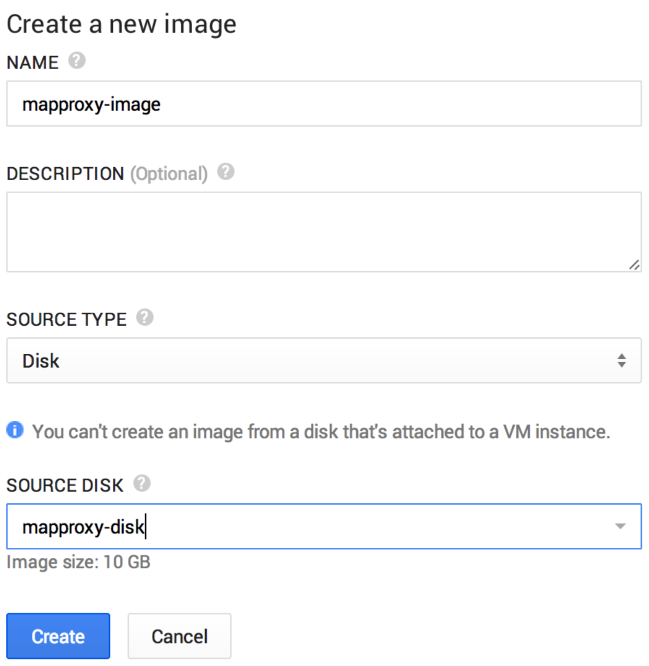
Create an image from the disk

1. From the **Compute Engine** list in the Developers Console project, click **Images**.

2. Click **New image**.



3. Give the image a name, and chose **Disk** as the **Source Type**. Select the disk you created in the previous step as the source disk. Then click **Create**.



Create a new image

NAME

mapproxy-image

DESCRIPTION (Optional)

SOURCE TYPE

Disk

You can't create an image from a disk that's attached to a VM instance.

SOURCE DISK

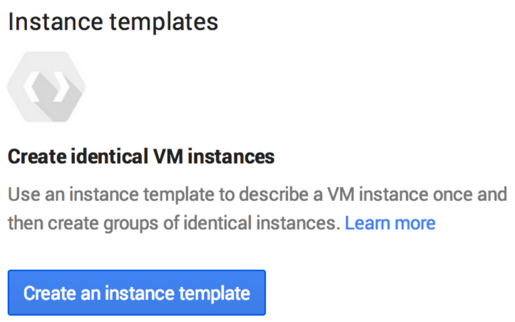
mapproxy-disk

Image size: 10 GB

Create an instance template

1. From the **Compute Engine** list in the Developers Console project, click **Instance templates**.

2. Click **Create an instance template**.



Instance templates

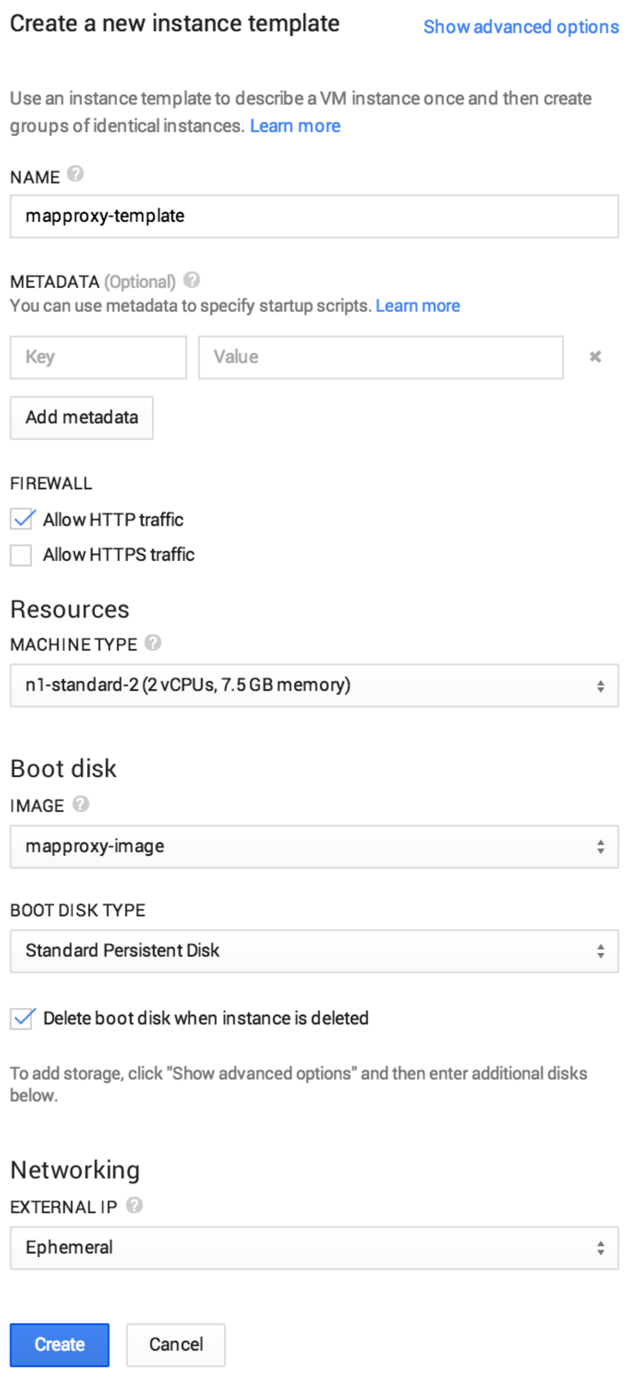
Create identical VM instances

Use an instance template to describe a VM instance once and then create groups of identical instances. Learn more

3. Give the instance template a name, and set the firewall to allow HTTP traffic.

4. Chose the machine type. A 2 CPU machine is a good balance between price and performance. It should allow your load balanced VMs to handle several simultaneous connections per machine, while also being less expensive than a high CPU machine.

5. For **Image for the template**, choose the image you created in the previous step. For the **Disk type**, chose a standard persistent disk.



Create a new instance template

Use an instance template to describe a VM instance once and then create groups of identical instances. Learn more

NAME: mapproxy-template

METADATA (Optional)
You can use metadata to specify startup scripts. Learn more

Key | Value

FIREWALL
- [✓] Allow HTTP traffic
- [ ] Allow HTTPS traffic

Resources
MACHINE TYPE: n1-standard-2 (2 vCPUs, 7.5 GB memory)

Boot disk
IMAGE: mapproxy-image

BOOT DISK TYPE: Standard Persistent Disk

- [✓] Delete boot disk when instance is deleted

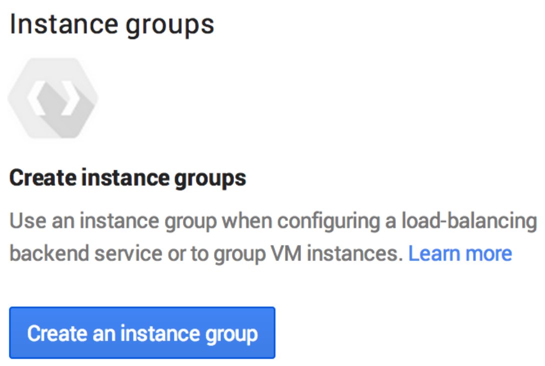
To add storage, click "Show advanced options" and then enter additional disks below.

Networking
EXTERNAL IP: Ephemeral

Create an Instance Group

1. From the **Compute Engine** list in the Developers Console, click **Instance groups**.

2. Click **Create a new instance group**.



Instance groups

Create instance groups

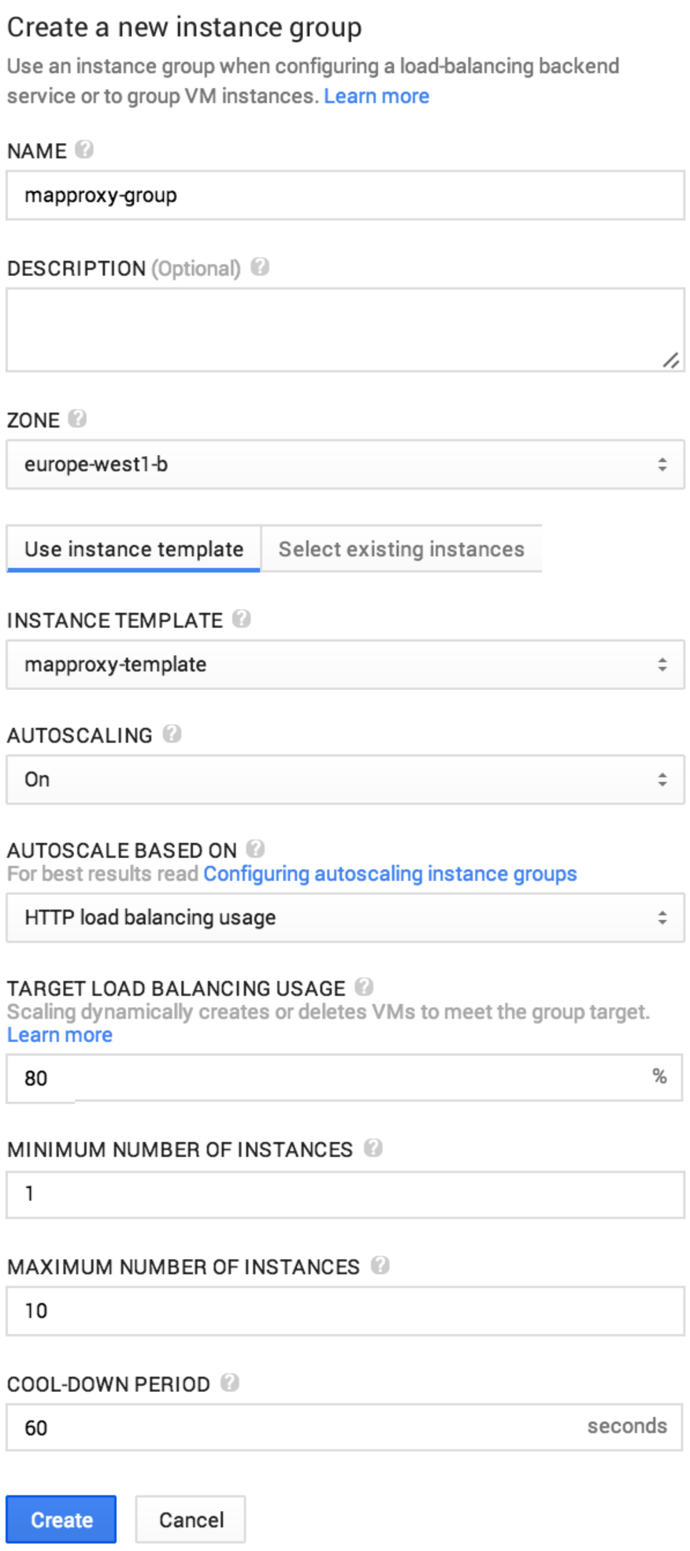
Use an instance group when configuring a load-balancing backend service or to group VM instances. Learn more

3. Give the instance group a name, and chose the instance group template you previously configured.

4. Turn on **Autoscaling**, and chose to autoscale based on **HTTP load balancing usage**. Accept the default value of 80%.

   Later, we'll configure the Rate of Requests per second (RPS/QPS), which says, "When my load balancer is receiving 80% of the RPS that the current number of instances can support, spin up a new instance."

5. Configure a minimum of 1 instance, and set a limit for the maximum amount of instances you're willing to allow to be created. For most cases, 10 to 20 is sufficient.



Create a new instance group

Use an instance group when configuring a load-balancing backend service or to group VM instances. Learn more

NAME: mapproxy-group

DESCRIPTION (Optional):

ZONE: europe-west1-b

Use instance template | Select existing instances

INSTANCE TEMPLATE: mapproxy-template

AUTOSCALING: On

AUTOSCALE BASED ON
For best results read Configuring autoscaling instance groups
HTTP load balancing usage

TARGET LOAD BALANCING USAGE
Scaling dynamically creates or deletes VMs to meet the group target. Learn more
80 %

MINIMUM NUMBER OF INSTANCES: 1

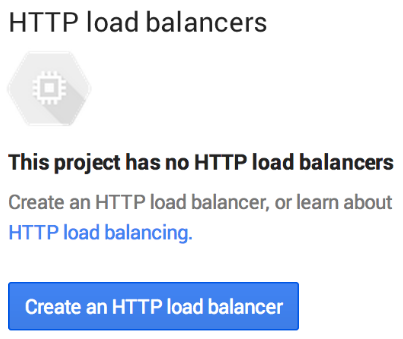
MAXIMUM NUMBER OF INSTANCES: 10

COOL-DOWN PERIOD: 60 seconds

Create an HTTP load balancer

1. From the **Compute Engine** list in the Developers Console project, click **HTTP load balancing.**

2. Click **Create an HTTP load balancer**.

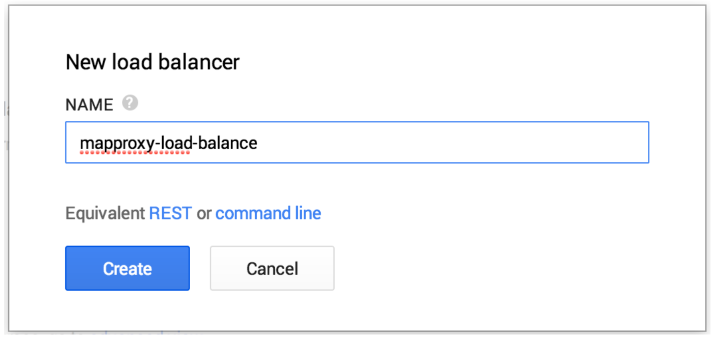


HTTP load balancers

This project has no HTTP load balancers

Create an HTTP load balancer, or learn about HTTP load balancing.

3. Give the load balancer a name. Then click **Create**.

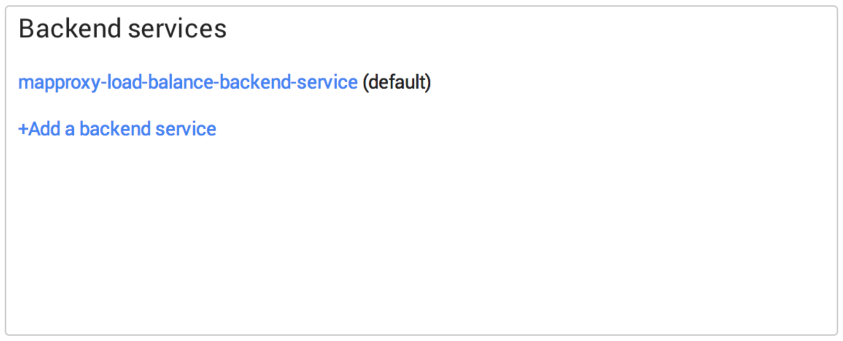


New load balancer

NAME: mapproxy-load-balance

Equivalent REST or command line

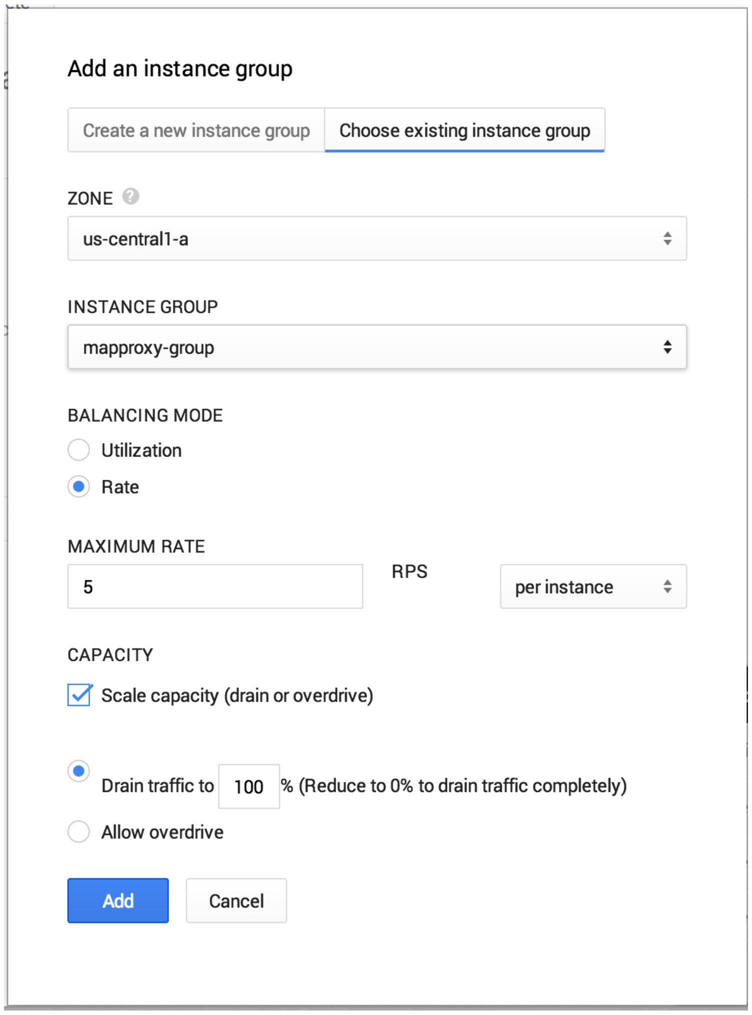
4. Once the load balancer is created, click the default backend service to edit it.



Backend services

mapproxy-load-balance-backend-service (default)

5. For the backend service, chose the instance group you previously created. For the balancing mode, chose **Rate**, and for MapProxy, 5 RPS is appropriate for each of our n1-standard-2 virtual machines. Because you previously selected 80% as our HTTP Load Balancing Utilization threshold, once your first instance starts receiving a sustained 4 RPS, a new instance will be created. Once the two instances reach 8 RPS, a third instance will be created, and so on, until the upper limit of the number of instances you've configured.


Add an instance group

Create a new instance group | Choose existing instance group

ZONE: us-central1-a

INSTANCE GROUP: mapproxy-group

BALANCING MODE: Utilization [✓ Rate]

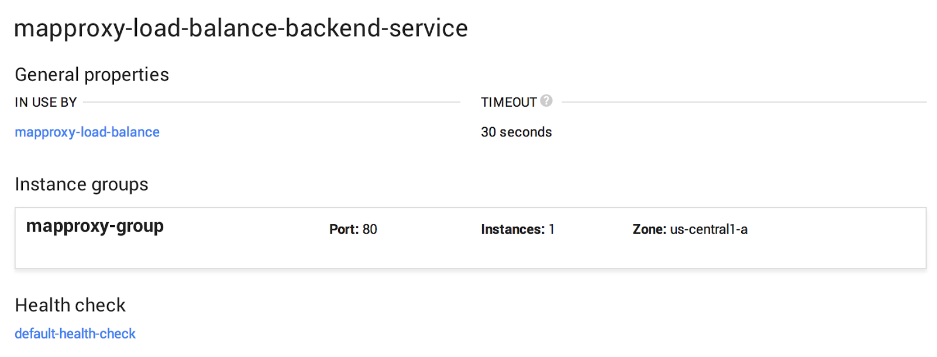
MAXIMUM RATE: 5 RPS per instance

CAPACITY: [✓] Scale capacity (drain or overdrive)

[✓] Drain traffic to 100 % (Reduce to 0% to drain traffic completely)

Allow overdrive

6. Once the backend service is configured, click the **default-health-check** link.



mapproxy-load-balance-backend-service

General properties

IN USE BY	TIMEOUT
mapproxy-load-balance	30 seconds

Instance groups

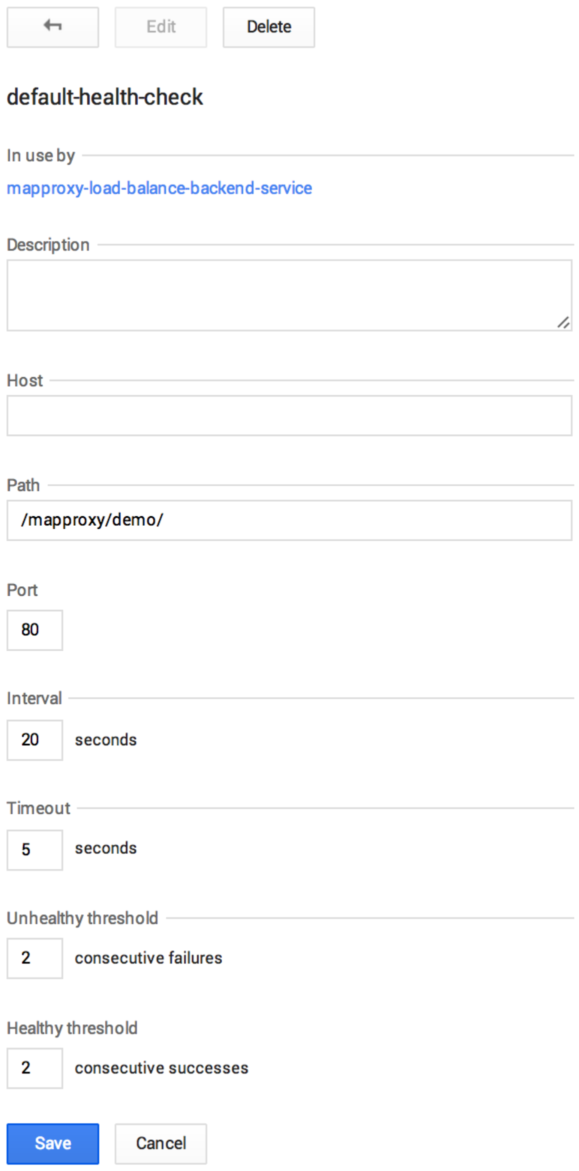
mapproxy-group	Port: 80	Instances: 1	Zone: us-central1-a

Health check

default-health-check

The HTTP Load Balancer will run a health check to make sure that all your instance groups are healthy and responding to requests. By default, it will just load the root of your web servers.

7. To make sure that the health check is actually invoking the WSGI application of MapProxy, change the **Path** to `/mapproxy/demo/` Also, change the **Interval** to 20 seconds.



default-health-check

In use by
mapproxy-load-balance-backend-service

Description

Host

Path
/mapproxy/demo/

Port
80

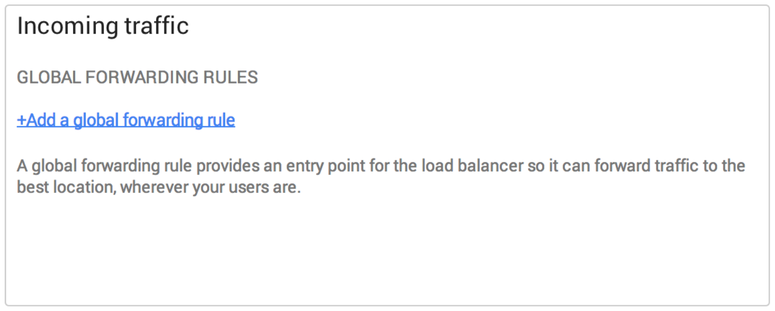
Interval
20 seconds

Timeout
5 seconds

Unhealthy threshold
2 consecutive failures

Healthy threshold
2 consecutive successes

8. Click **Save** and return to the HTTP Load Balancer UI. Then click **Add a global forwarding rule**.



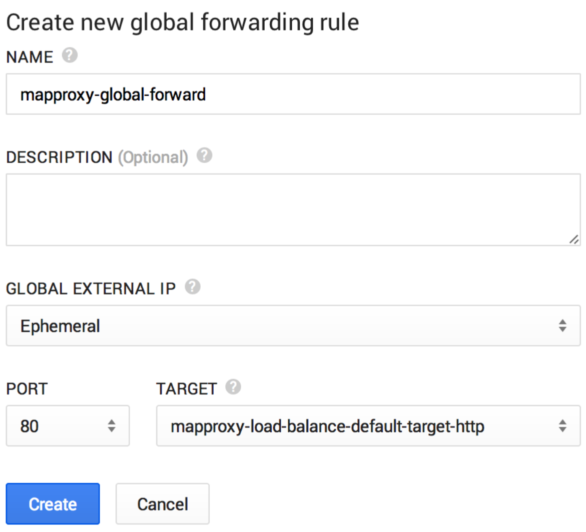
Incoming traffic

GLOBAL FORWARDING RULES

+Add a global forwarding rule

A global forwarding rule provides an entry point for the load balancer so it can forward traffic to the best location, wherever your users are.

9. Give the rule a name, and chose either an **Ephemeral** or **Static IP**. We recommend choosing **Static IP**, because you'll share this IP address with users or map it to a custom URL.



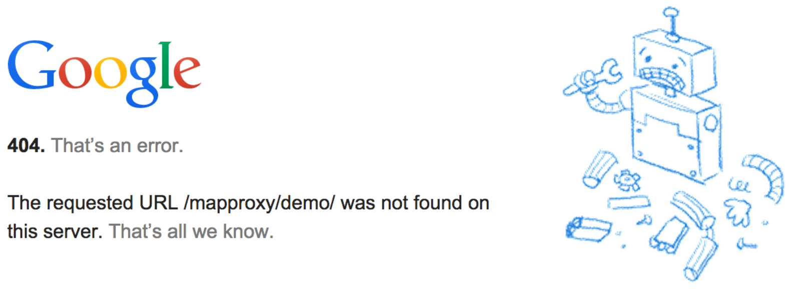
Create new global forwarding rule

NAME	mapproxy-global-forward
DESCRIPTION (Optional)	
GLOBAL EXTERNAL IP	Ephemeral
PORT	80
TARGET	mapproxy-load-balance-default-target-http

10. Click **Create**.

    Your HTTP load balancer and instance groups will be pushed to production.

    As the service is being turned on, you'll see a 404 error at your ephemeral or static IP address `/mapproxy/demo/`.



Google

404. That's an error.

The requested URL /mapproxy/demo/ was not found on this server. That's all we know.

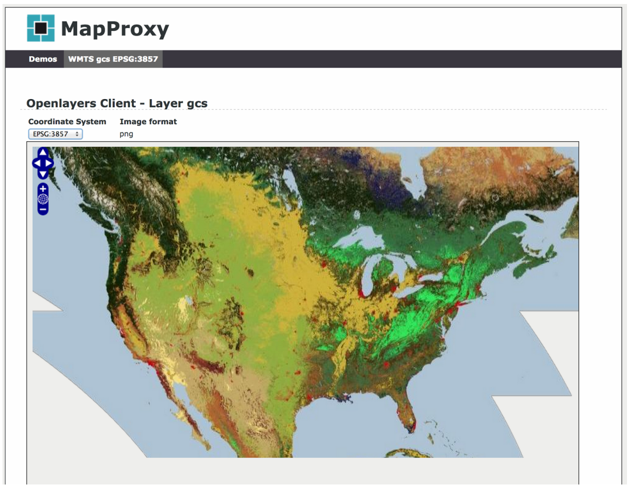
Shortly after, you'll start to see 500 errors during the DNS propagation phase.

Error: Server Error

The server encountered a temporary error and could not complete your request.

Please try again in 30 seconds.

After about 5 minutes, the service and DNS propagation should be complete, and you'll see your live Google Compute Engine MapProxy Service. Your service ready to serve at Google scale, with HTTP load balancing, autoscaling, and the ability to create OGC endpoints around tiles served from Google Cloud Storage.



Openlayers Client - Layer gcs

Coordinate System: EPSG:3857
Image format: png

[See the demo]