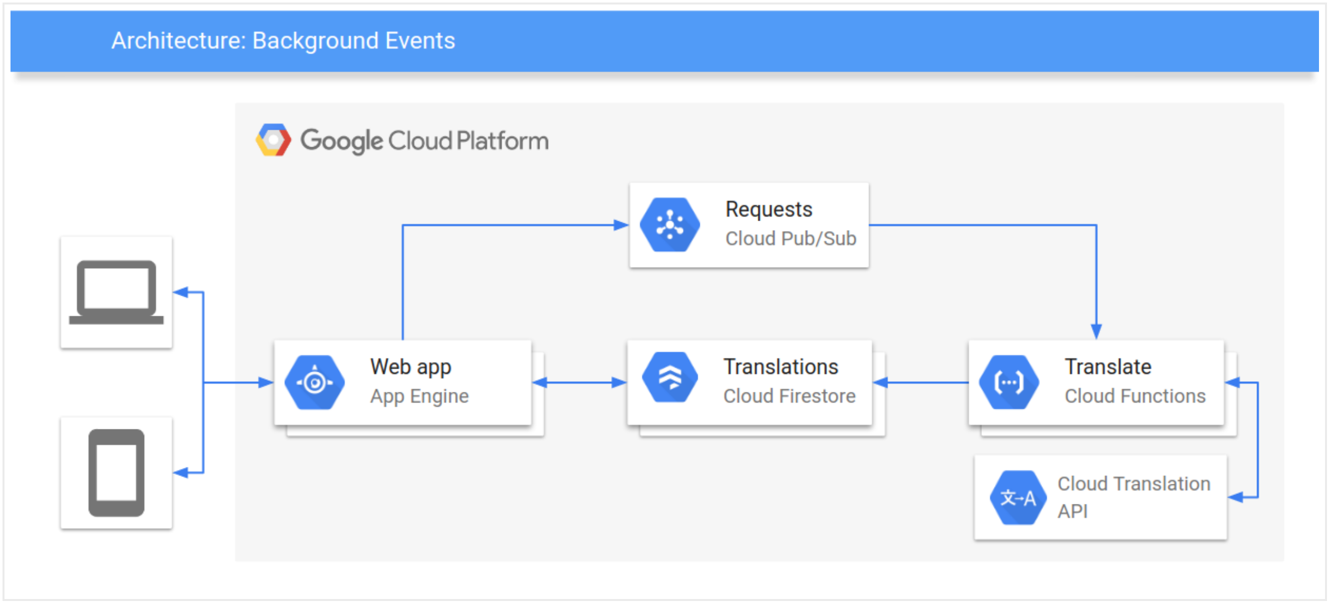
Node.js (https://cloud.google.com/nodejs/) Guides

Background processing with Node.js

Many apps need to do background processing outside of the context of a web request. This tutorial creates a web app that lets users input text to translate, and then displays a list of previous translations. The translation is done in a background process to avoid blocking the user's request.

The following diagram illustrates the translation request process.



Here is the sequence of events for how the tutorial app works:

1. Visit the web page to see a list of previous translations, stored in Firestore.
2. Request a translation of text by entering an HTML form.
3. The translation request is published to Pub/Sub.
4. A Cloud Function subscribed to that Pub/Sub topic is triggered.

5. The Cloud Function uses Cloud Translation to translate the text.
6. The Cloud Function stores the result in Firestore.

This tutorial is intended for anyone who is interested in learning about background processing with Google Cloud. No prior experience is required with Pub/Sub, Firestore, App Engine, or Cloud Functions. However, to understand all of the code, some experience with Node.js, JavaScript, and HTML is helpful.

Objectives

- Understand and deploy a Cloud Function.
- Understand and deploy an App Engine app.
- Try the app.

Costs

This tutorial uses the following billable components of Google Cloud:

- App Engine (https://cloud.google.com/appengine/pricing)
- Cloud Functions (https://cloud.google.com/functions/pricing)
- Firestore (https://cloud.google.com/firestore/pricing)
- Pub/Sub (https://cloud.google.com/pubsub/pricing)
- Cloud Translation (https://cloud.google.com/translate/pricing)

To generate a cost estimate based on your projected usage, use the pricing calculator (https://cloud.google.com/products/calculator). New Google Cloud users might be eligible for a free trial (https://cloud.google.com/free-trial).

When you finish this tutorial, you can avoid continued billing by deleting the resources you created. For more information, see Cleaning up (#clean-up).

Before you begin

1. Sign in (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).

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/PROJECTSELECT

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).

4. Enable the Firestore, Cloud Functions, Pub/Sub, and Cloud Translation APIs.

   ENABLE THE APIS (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/FLOWS/ENABLEAPI?APIID=FIRESTOR

5. In the Google Cloud Console, open the app in Cloud Shell (https://cloud.google.com/shell).

   GO TO CLOUD SHELL (HTTPS://CLOUD.GOOGLE.COM/CONSOLE/CLOUDSHELL/OPEN?GIT_BRANCH

   Cloud Shell provides command-line access to your cloud resources directly from the browser. Open Cloud Shell in your browser and click **Proceed** to download the sample code and change into the app directory.

6. In Cloud Shell, configure the `gcloud` tool to use your Google Cloud project:

```
# Configure gcloud for your project
gcloud config set project YOUR_PROJECT_ID
```

Understanding the Cloud Function

- The function starts by importing several dependencies like Firestore and Translation. The global Firestore and Translation clients are initialized so they can be reused between function invocations. That way, you don't have to initialize new clients for every function invocation, which would slow down execution.

background/function/translate.js (https://github.com/GoogleCloudPlatform/nodejs-getting-started/blob/master/background/function/translate.js)

.OUDPLATFORM/NODEJS-GETTING-STARTED/BLOB/MASTER/BACKGROUND/FUNCTION/TRANSLATE.JS)

```
const {Firestore} = require('@google-cloud/firestore');
const {Translate} = require('@google-cloud/translate');

const firestore = new Firestore();
const translate = new Translate();
```

- The Translation API translates the string to the language you selected.

background/function/translate.js (https://github.com/GoogleCloudPlatform/nodejs-getting-started/blob/master/background/function/translate.js)

.OUDPLATFORM/NODEJS-GETTING-STARTED/BLOB/MASTER/BACKGROUND/FUNCTION/TRANSLATE.JS)

```
const [
  translated,
  {
    data: {translations},
  },
] = await translate.translate(original, language);
const originalLanguage = translations[0].detectedSourceLanguage;
console.log(
  `Translated ${original} in ${originalLanguage} to ${translated} in ${language
);
```

- The Cloud Function parses the Pub/Sub message to get the text to translate and the desired target language.

  Then, the app requests a translation and stores the result in Firestore.

background/function/translate.js (https://github.com/GoogleCloudPlatform/nodejs-getting-started/blob/master/background/function/translate.js)

.OUDPLATFORM/NODEJS-GETTING-STARTED/BLOB/MASTER/BACKGROUND/FUNCTION/TRANSLATE.JS)

```
// translate translates the given message and stores the result in Firestore.
// Triggered by Pub/Sub message.
exports.translate = async pubSubEvent => {
  const {language, original} = JSON.parse(
    Buffer.from(pubSubEvent.data, 'base64').toString()
  );

  const [
    translated,
    {
      data: {translations},
    },
  ] = await translate.translate(original, language);
  const originalLanguage = translations[0].detectedSourceLanguage;
  console.log(
    `Translated ${original} in ${originalLanguage} to ${translated} in ${langua
  );

  // Store translation in firestore.
  await firestore
    .collection('translations')
    .doc()
    .set({
      language,
      original,
      translated,
      originalLanguage,
    });
};
```

Deploying the Cloud Function

- In the same directory as the `translate.js` file, deploy the Cloud Function with a Pub/Sub trigger:

```
gcloud functions deploy translate --runtime nodejs10 --trigger-topic translate
```

Understanding the app

There are two main components for the web app:

- A Node.js HTTP server to handle web requests. The server has the following two endpoints:

  - `/`: Lists all of the existing translations and shows a form users can submit to request new translations.
  - `/request-translation`: Form submissions are sent to this endpoint, which publishes the request to Pub/Sub to be translated asynchronously.
- An HTML template that is filled in with the existing translations by the Node.js server.

The HTTP server

- In the `server` directory, `app.js` starts by setting up the app and registering HTTP handlers:

background/server/app.js (https://github.com/GoogleCloudPlatform/nodejs-getting-started/blob/master/background/server/app.js)

GOOGLECLOUDPLATFORM/NODEJS-GETTING-STARTED/BLOB/MASTER/BACKGROUND/SERVER/APP.JS)

```
// This app is an HTTP app that displays all previous translations
// (stored in Firestore) and has a form to request new translations. On form
// submission, the request is sent to Pub/Sub to be processed in the background

// TOPIC_NAME is the Pub/Sub topic to publish requests to. The Cloud Function t
// process translation requests should be subscribed to this topic.
const TOPIC_NAME = 'translate';

const express = require('express');
const bodyParser = require('body-parser');
const {PubSub} = require('@google-cloud/pubsub');
const {Firestore} = require('@google-cloud/firestore');

const app = express();
const port = process.env.PORT || 8080;

const firestore = new Firestore();

const pubsub = new PubSub();
const topic = pubsub.topic(TOPIC_NAME);

// Use handlebars.js for templating.
app.set('views', __dirname);
app.set('view engine', 'html');
app.engine('html', require('hbs').__express);
app.use(bodyParser.urlencoded({extended: true}));

app.get('/', index);
app.post('/request-translation', requestTranslation);
app.listen(port, () => console.log(`Listening on port ${port}!`));
```

- The index handler (`/`) gets all existing translations from Firestore and fills an HTML template with the list:

background/server/app.js (https://github.com/GoogleCloudPlatform/nodejs-getting-started/blob/master/background/server/app.js)

GOOGLECLOUDPLATFORM/NODEJS-GETTING-STARTED/BLOB/MASTER/BACKGROUND/SERVER/APP.JS)

```
// index lists the current translations.
async function index(req, res) {
  const translations = [];
  const querySnapshot = await firestore.collection('translations').get();
  querySnapshot.forEach(doc => {
    console.log(doc.id, ' => ', doc.data());
    translations.push(doc.data());
  });

  res.render('index', {translations});
}
```

- New translations are requested by submitting an HTML form. The request translation handler, registered at `/request-translation`, parses the form submission, validates the request, and publishes a message to Pub/Sub:

background/server/app.js (https://github.com/GoogleCloudPlatform/nodejs-getting-started/blob/master/background/server/app.js)

GOOGLECLOUDPLATFORM/NODEJS-GETTING-STARTED/BLOB/MASTER/BACKGROUND/SERVER/APP.JS)

```
// requestTranslation parses the request, validates it, and sends it to Pub/Sub
function requestTranslation(req, res) {
  const language = req.body.lang;
  const original = req.body.v;

  const acceptableLanguages = ['de', 'en', 'es', 'fr', 'ja', 'sw'];
  if (!acceptableLanguages.includes(language)) {
    throw new Error(`Invalid language ${language}`);
  }

  console.log(`Translation requested: ${original} -> ${language}`);

  const buffer = Buffer.from(JSON.stringify({language, original}));
  topic.publish(buffer);
  res.sendStatus(200);
}
```

The HTML template

The HTML template is the basis for the HTML page shown to the user so they can see previous translations and request new ones. The template is filled in by the HTTP server with the list of existing translations.

- The `<head>` element of the HTML template includes metadata, style sheets, and JavaScript for the page:

[background/server/index.html](https://github.com/GoogleCloudPlatform/nodejs-getting-started/blob/master/background/server/index.html) (https://github.com/GoogleCloudPlatform/nodejs-getting-started/blob/master/background/server/index.html)

LECLOUDPLATFORM/NODEJS-GETTING-STARTED/BLOB/MASTER/BACKGROUND/SERVER/INDEX.HTML)

```
<html>

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Translations</title>

    <link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Mater
    <link rel="stylesheet" href="https://code.getmdl.io/1.3.0/material.indigo-p
    <script defer src="https://code.getmdl.io/1.3.0/material.min.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.
    <script>
        $(document).ready(function() {
            $("#translate-form").submit(function(e) {
                e.preventDefault();
                // Get value, make sure it's not empty.
                if ($("#v").val() == "") {
                    return;
                }
                $.ajax({
                    type: "POST",
                    url: "/request-translation",
                    data: $(this).serialize(),
                    success: function(data) {
                        // Show snackbar.
                        console.log(data);
                        var notification = document.querySelector('.mdl-js-snac
                        $("#snackbar").removeClass("mdl-color--red-100");
                        $("#snackbar").addClass("mdl-color--green-100");
                        notification.MaterialSnackbar.showSnackbar({
                            message: 'Translation requested'
                        });
                    },
                    error: function(data) {
                        // Show snackbar.
                        console.log("Error requesting translation");
                        var notification = document.querySelector('.mdl-js-snac
                        $("#snackbar").removeClass("mdl-color--green-100");
                        $("#snackbar").addClass("mdl-color--red-100");
                        notification.MaterialSnackbar.showSnackbar({
                            message: 'Translation request failed'
                        });
                    }
                });
            });
        });
    </script>
    <style>
        .lang {
            width: 50px;
        }
        .translate-form {
            display: inline;
        }
    </style>
</head>
```

The page pulls in Material Design Lite (MDL) (https://getmdl.io/) CSS and JavaScript assets. MDL lets you add a Material Design (https://material.io/design/) look and feel to your websites.

The page uses JQuery (https://jquery.com/) to wait for the document to finish loading and set a form submission handler. Whenever the request translation form is submitted, the page does minimal form validation to check that the value isn't empty, and then sends an asynchronous request to the `/request-translation` endpoint.

Finally, an MDL snackbar (https://getmdl.io/components/#snackbar-section) appears to indicate if the request was successful or if it encountered an error.

- The HTML body of the page uses an MDL layout (https://getmdl.io/components/index.html#layout-section) and several MDL components (https://getmdl.io/components/index.html) to display a list of translations and a form to request additional translations:

background/server/index.html (https://github.com/GoogleCloudPlatform/nodejs-getting-started/blob/master/background/server/index.html)

LECLOUDPLATFORM/NODEJS-GETTING-STARTED/BLOB/MASTER/BACKGROUND/SERVER/INDEX.HTML)

```
<body>
    <div class="mdl-layout mdl-js-layout mdl-layout--fixed-header">
        <header class="mdl-layout__header">
            <div class="mdl-layout__header-row">
                <!-- Title -->
                <span class="mdl-layout-title">Translate with Background Proces
            </div>
        </header>
        <main class="mdl-layout__content">
            <div class="page-content">
                <div class="mdl-grid">
                <div class="mdl-cell mdl-cell--1-col"></div>
                    <div class="mdl-cell mdl-cell--3-col">
                        <form id="translate-form" class="translate-form">
                            <div class="mdl-textfield mdl-js-textfield mdl-text
                                <input class="mdl-textfield__input" type="text"
                                <label class="mdl-textfield__label" for="v">Tex
                            </div>
                            <select class="mdl-textfield__input lang" name="lan
                                <option value="de">de</option>
                                <option value="en">en</option>
                                <option value="es">es</option>
                                <option value="fr">fr</option>
                                <option value="ja">ja</option>
                                <option value="sw">sw</option>
                            </select>
                            <button class="mdl-button mdl-js-button mdl-button-
                                name="submit">Submit</button>
                        </form>
                    </div>
                    <div class="mdl-cell mdl-cell--8-col">
                        <table class="mdl-data-table mdl-js-data-table mdl-shad
                            <thead>
                                <tr>
                                    <th class="mdl-data-table__cell--non-numeri
                                    <th class="mdl-data-table__cell--non-numeri
                                </tr>
                            </thead>
                            <tbody>
                                {{#each translations}}
                                <tr>
                                    <td class="mdl-data-table__cell--non-numeri
                                        <span class="mdl-chip mdl-color--primar
                                            <span class="mdl-chip__text mdl-col
                                        </span>
                                    {{ original }}
                                    </td>
                                    <td class="mdl-data-table__cell--non-numeri
                                        <span class="mdl-chip mdl-color--accent
                                            <span class="mdl-chip__text mdl-col
                                        </span>
                                        {{ translated }}
                                    </td>
                                </tr>
                                {{/each}}
                            </tbody>
                        </table>
                        <br/>
                        <button class="mdl-button mdl-js-button mdl-button--rai
                            Refresh
                        </button>
                    </div>
                </div>
            </div>
            <div aria-live="assertive" aria-atomic="true" aria-relevant="text"
                <div class="mdl-snackbar__text mdl-color-text--black"></div>
                <button type="button" class="mdl-snackbar__action"></button>
            </div>
        </main>
    </div>
</body>

</html>
```

Deploying the web app

You can use the App Engine standard environment (https://cloud.google.com/appengine/docs/standard/nodejs) to build and deploy an app that runs reliably under heavy load and with large amounts of data.

This tutorial uses the App Engine standard environment to deploy the HTTP frontend.

The `app.yaml` configures the App Engine app:

background/server/app.yaml (https://github.com/GoogleCloudPlatform/nodejs-getting-started/blob/master/background/server/app.yaml)

OGLECLOUDPLATFORM/NODEJS-GETTING-STARTED/BLOB/MASTER/BACKGROUND/SERVER/APP.YAML)

```
runtime: nodejs10
```

- From the same directory as the `app.yaml` file, deploy your app to the App Engine standard environment:

  ```
  gcloud app deploy
  ```

Testing the app

After you've deployed the Cloud Function and App Engine app, try requesting a translation.

1. To view the app in your browser, go to `https://`*`YOUR_GOOGLE_CLOUD_PROJECT`*`.appspot.com`.

   There is a page with an empty list of translations and a form to request new translations.
2. In the **Text to translate** field, enter some text to translate, for example, `Hello, World`.
3. Select a language from the drop-down list that you want to translate the text to.
4. Click **Submit**.
5. To refresh the page, click **Refresh** ↻. There is a new row in the translation list. If you don't see a translation, wait a few more seconds and try again. If you still don't see a translation, see the next section about debugging the app.

Debugging the app

If you cannot connect to your App Engine app or don't see new translations, check the following:

1. Check that the `gcloud` deploy commands successfully completed and didn't output any errors. If there were errors, fix them, and try deploying the Cloud Function (#deploying-the-cloud-function) and the App Engine app (#deploying-the-web-app) again.
2. In the Google Cloud Console, go to the Logs Viewer page.

   GO TO LOGS VIEWER PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/LOGS/VIEWER)

   a. In the **Recently selected resources** drop-down list, click **GAE Application**, and then click **All module_id**. You see a list of requests from when you visited your app. If you don't see a list of requests, confirm you selected **All module_id** from the drop-down list. If you see error messages printed to the Cloud Console, check that your app's code matches the code in the section about understanding the app (#understanding-the-app).

   b. In the **Recently selected resources** drop-down list, click **Cloud Function**, and then click **All function name**. You see a function listed for each requested translation. If not, check that the Cloud Function and App Engine app are using the same Pub/Sub topic:

      - In the `background/server/app.js` file, check that the `TOPIC_NAME` constant is `"translate"`.
      - When you deploy the Cloud Function (#deploying_the_cloud_function), be sure to include the `--trigger-topic=translate` flag.

Cleaning up

To avoid incurring charges to your Google Cloud Platform account for the resources used in this tutorial:

Delete the Google Cloud 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/PRO

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 the App Engine instance

1. In the Cloud Console, go to the **Versions** page for App Engine.

   GO TO THE VERSIONS PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/APPENGINE/VERSIONS)

2. Select the checkbox for the non-default app version you want to delete.

   Note: The only way you can delete the default version of your App Engine app is by deleting your project. However, you can stop the default version in the Cloud Console (https://console.cloud.google.com/appengine/versions). This action shuts down all instances associated with the version. You can restart these instances later if needed.

   In the App Engine standard environment, you can stop the default version only if your app has manual or basic scaling.

3. Click **Delete** 🗑 to delete the app version.

Delete the Cloud Function

- Delete the Cloud Function you created in this tutorial:

```
gcloud functions delete Translate
```

What's next

- Try additional Cloud Functions tutorials (https://cloud.google.com/functions/docs/tutorials/).
- Learn more about App Engine (https://cloud.google.com/appengine/docs/).
- Try Cloud Run (https://cloud.google.com/run/docs/quickstarts/prebuilt-deploy), which lets you run stateless containers on a fully managed environment or in your own Google Kubernetes Engine cluster.