

# Using Flex Templatesbookmark

 [cloud.google.com/dataflow/docs/guides/templates/using-flex-templates](https://cloud.google.com/dataflow/docs/guides/templates/using-flex-templates)

This tutorial shows you how to create and run a Dataflow Flex Template job with a custom Docker image using `gcloud` command-line tool. This tutorial walks you through a streaming pipeline example that reads JSON-encoded messages from Pub/Sub, transforms message data with Beam SQL, and writes the results to a BigQuery table.

To know more about Flex Templates, see [Dataflow templates](#).

## Objectives

- Build a Docker container image.
- Create and run a Dataflow Flex Template.

## Costs

This tutorial uses billable components of Google Cloud, including:

- Dataflow
- Pub/Sub
- Cloud Storage
- Cloud Scheduler
- App Engine
- Container Registry
- Cloud Build
- BigQuery

Use the [Pricing Calculator](#) to generate a cost estimate based on your projected usage.

## Before you begin

1. If you're new to Google Cloud, [create an account](#) to evaluate how our products perform in real-world scenarios. New customers also get \$300 in free credits to run, test, and deploy workloads.
2. In the Google 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 project selector](#)

3. Make sure that billing is enabled for your Cloud project. [Learn how to confirm that billing is enabled for your project](#).

4. Enable the Dataflow, Compute Engine, Logging, Cloud Storage, Cloud Storage JSON, BigQuery, Pub/Sub, Resource Manager, App Engine, Cloud Scheduler, and Cloud Build APIs.

Enable the APIs

5. Create a service account:

1. In the Cloud Console, go to the **Create service account** page.

Go to Create service account

2. Select a project.
3. In the **Service account name** field, enter a name. The Cloud Console fills in the **Service account ID** field based on this name.

In the **Service account description** field, enter a description. For example, `Service account for quickstart`.

4. Click **Create**.
5. Click the **Select a role** field.

Under **Quick access**, click **Basic**, then click **Owner**.

**Note:** The **Role** field affects which resources your service account can access in your project. You can revoke these roles or grant additional roles later. In production environments, do not grant the Owner, Editor, or Viewer roles. For more information, see [Granting, changing, and revoking access to resources](#).

6. Click **Continue**.
7. Click **Done** to finish creating the service account.

Do not close your browser window. You will use it in the next step.

6. Create a service account key:

1. In the Cloud Console, click the email address for the service account that you created.
2. Click **Keys**.
3. Click **Add key**, then click **Create new key**.
4. Click **Create**. A JSON key file is downloaded to your computer.
5. Click **Close**.
7. Set the environment variable `GOOGLE_APPLICATION_CREDENTIALS` to the path of the JSON file that contains your service account key. This variable only applies to your current shell session, so if you open a new session, set the variable again.

**Example: Linux or macOS**

**Example: Windows**

When you finish this tutorial, you can avoid continued billing by deleting the resources you created. See [Cleaning up](#) for more detail.

## Creating the example source and sink

---

This section explain how to create the following:

- A streaming source of data using Pub/Sub
- A dataset to load the data into BigQuery

### Create a Cloud Storage bucket

---

Use the `gsutil mb` command:

```
export BUCKET="my-storage-bucket"
gsutil mb gs://$BUCKET
```

### Create a Pub/Sub topic and a subscription to that topic

---

Use the `gcloud` command-line tool:

```
export TOPIC="messages"
export SUBSCRIPTION="ratings"
gcloud pubsub topics create $TOPIC
gcloud pubsub subscriptions create --topic $TOPIC $SUBSCRIPTION
```

### Create a Cloud Scheduler job

---

In this step, we use the `gcloud` command-line tool to create and run a Cloud Scheduler job that publishes "positive ratings" and "negative ratings."

1. Create a Cloud Scheduler job for this Google Cloud project.

```
gcloud scheduler jobs create pubsub positive-ratings-publisher \
  --schedule="* * * * *" \
  --topic="$TOPIC" \
  --message-body='{"url": "https://beam.apache.org/", "review": "positive"}'
```

This creates and runs a publisher for "positive ratings" that publishes 1 message per minute.

2. Start the Cloud Scheduler job.

```
gcloud scheduler jobs run positive-ratings-publisher
```

3. Create and run another similar publisher for "negative ratings" that publishes 1 message every 2 minutes.

```
gcloud scheduler jobs create pubsub negative-ratings-publisher \
  --schedule="*/2 * * * *" \
  --topic="$TOPIC" \
  --message-body='{"url": "https://beam.apache.org/", "review": "negative"}'
gcloud scheduler jobs run negative-ratings-publisher
```

### Create a BigQuery dataset

---

Use the `bq mk` command:

```
export PROJECT="$(gcloud config get-value project)"
export DATASET="beam_samples"
export TABLE="streaming_beam_sql" bq mk --dataset "$PROJECT:$DATASET"
```

## Downloading the code sample

---

1. Download the code sample.

### JavaPython

Clone the [python-docs-samples repository](#) and navigate to the code sample for this tutorial.

```
git clone https://github.com/GoogleCloudPlatform/python-docs-samples.git
cd python-docs-samples/dataflow/flex-templates/streaming_beam
```

2. Export the `TEMPLATE_IMAGE` for this tutorial.

```
export TEMPLATE_IMAGE="gcr.io/$PROJECT/samples/dataflow/streaming-beam-
sql:latest"
```

## Setting up your development environment

---

### JavaPython

Use the [Apache Beam SDK](#) for Python with `pip` and Python version 2.7, 3.5, 3.6 or 3.7. Check that you have a working Python and `pip` installation by running:

```
python --version
python -m pip --version
```

If you do not have Python, find the installation steps for your operating system on the [Installing Python](#) page.

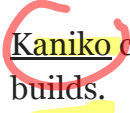
## Python only: Creating and building a container image

---

This section contains steps for Python users. If you are using Java, skip the following steps.

1. (Optional) Enable Kaniko cache use by default.

```
gcloud config set builds/use_kaniko True
```

 **Kaniko** caches container build artifacts, so using this option speeds up subsequent builds.

2. (Optional) Create the Dockerfile. You can customize the Dockerfile from this tutorial. The starter file looks like the following:

### Python

```
FROM gcr.io/dataflow-templates-base/python3-template-launcher-  
base  
  
ARG WORKDIR=/dataflow/template  
RUN mkdir -p ${WORKDIR}  
WORKDIR ${WORKDIR}  
# Due to a change in the Apache Beam base image in version 2.24, you must  
to install  
# libffi-dev manually as a dependency. For more information:  
# https://github.com/GoogleCloudPlatform/python-docs-samples/issues/4891  
RUN apt-get update && apt-get install -y libffi-dev && rm -rf  
/var/lib/apt/lists/* COPY requirements.txt .  
COPY streaming_beam.py . ENV  
FLEX_TEMPLATE_PYTHON_REQUIREMENTS_FILE="${WORKDIR}/requirements.txt"  
ENV FLEX_TEMPLATE_PYTHON_PY_FILE="${WORKDIR}/streaming_beam.py" RUN pip  
install -U -r ./requirements.txt
```

This Dockerfile contains the `FROM`, `ENV`, and `COPY` commands, which you can read about in the [Dockerfile reference](#).

### **Note:**

Do not explicitly install any job dependencies/packages in the above Dockerfile. Container deployed using the above Dockerfile is only for Dataflow Job initiation purposes. Job packages/dependencies that are to be installed must be referenced in the 'requirements' file.

Images starting with `gcr.io/PROJECT/` are saved into your project's Container Registry, where the image is accessible to other Google Cloud products.

3. Build the Docker image using a Dockerfile with Cloud Build.

**Note:** If you created your own Dockerfile from the previous step, you must update `TEMPLATE_IMAGE`.

```
gcloud builds submit --tag $TEMPLATE_IMAGE .
```

## Creating a Flex Template

To run a template, you need to create a template spec file in a Cloud Storage containing all of the necessary information to run the job, such as the SDK information and metadata.

The `metadata.json` file in this example contains additional information for the template such as the `name`, `description`, and input `parameters` fields.

1. Create a template spec file containing all of the information necessary to run the job, such as the SDK information and metadata.

```
export TEMPLATE_PATH="gs://$BUCKET/samples/dataflow/templates/streaming-beam-  
sql.json"
```

## 2. Build the Flex Template.

### JavaPython

```
gcloud dataflow flex-template build $TEMPLATE_PATH \
  --image "$TEMPLATE_IMAGE" \
  --sdk-language "PYTHON" \
  --metadata-file "metadata.json"
```

The template is now available through the template file in the Cloud Storage location that you specified.

## Running a Flex Template pipeline

You can now run the Apache Beam pipeline in Dataflow by referring to the template file and passing the template parameters required by the pipeline.

### 1. Run the template.

#### JavaPython

```
export REGION="us-central1" gcloud dataflow flex-template run "streaming-
beam-`date +%Y%m%d-%H%M%S`" \
  --template-file-gcs-location "$TEMPLATE_PATH" \
  --parameters input_subscription="$SUBSCRIPTION" \
  --parameters output_table="$PROJECT:$DATASET.$TABLE" \
  --region "$REGION"
```

Alternatively, run the template with a REST API request.

```
curl -X POST \
  "https://dataflow.googleapis.com/v1b3/projects/$PROJECT/locations/us-
central1/flexTemplates:launch" \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $(gcloud auth print-access-token)" \
  -d '{
    "launch_parameter": {
      "jobName": "streaming-beam-sql-`(date +%Y%m%d-%H%M%S)`",
      "parameters": {
        "inputSubscription": "'$SUBSCRIPTION'",
        "outputTable": "'$PROJECT:$DATASET.$TABLE'"
      },
      "containerSpecGcsPath": "'$TEMPLATE_PATH"
```

After you execute the command to run the Flex Template, the Dataflow returns a Job ID with the job status **Queued**. It might take several minutes before the job status reaches **Running** and you can access the job graph.

### 2. Check the results in BigQuery by running the following query:

```
bq query --use_legacy_sql=false 'SELECT * FROM
`'$PROJECT.$DATASET.$TABLE`''`'
```

While this pipeline is running, you can see new rows appended into the BigQuery table every minute.

## Cleaning up

---

After you've finished this tutorial, you can clean up the resources you created on Google Cloud so you won't be billed for them in the future. The following sections describe how to delete or turn off these resources.

### Clean up the Flex Template resources

---

1. Stop the Dataflow pipeline.

```
gcloud dataflow jobs list \
  --filter 'NAME:streaming-beam-sql AND STATE=Running' \
  --format 'value(JOB_ID)' \
  --region "$REGION" \
  | xargs gcloud dataflow jobs cancel --region "$REGION"
```

2. Delete the template spec file from Cloud Storage.

```
gsutil rm $TEMPLATE_PATH
```

3. Delete the Flex Template container image from Container Registry.

```
gcloud container images delete $TEMPLATE_IMAGE --force-delete-tags
```

### Clean up Google Cloud project resources

---

1. Delete the Cloud Scheduler jobs.

```
gcloud scheduler jobs delete negative-ratings-publisher
gcloud scheduler jobs delete positive-ratings-publisher
```

2. Delete the Pub/Sub subscription and topic.

```
gcloud pubsub subscriptions delete $SUBSCRIPTION
gcloud pubsub topics delete $TOPIC
```

3. Delete the BigQuery table.

```
bq rm -f -t $PROJECT:$DATASET.$TABLE
```

4. Delete the BigQuery dataset, this alone does not incur any charges.

The following command also deletes all tables in the dataset. The tables and data cannot be recovered.

```
bq rm -r -f -d $PROJECT:$DATASET
```

5. Delete the Cloud Storage bucket, this alone does not incur any charges.

The following command also deletes all objects in the bucket. These objects cannot be recovered.

```
gsutil rm -r gs://$BUCKET
```

## Limitations

---

The following limitations apply to Flex Templates jobs:

- You must use a Google-provided base image to package your containers using Docker.
- `waitUntilFinish` (Java) and `wait_until_finish` (Python) are not supported.

## What's next

---

- For more information, read [Configuring Flex Templates](#).
- Explore reference architectures, diagrams, tutorials, and best practices about Google Cloud. Take a look at our [Cloud Architecture Center](#).

Rate and review