DATA ANALYTICS

Turn any Dataflow pipeline into a reusable template

Sameer Farooqui
Cloud Data Engineer

Prathap Parvathareddy
Cloud Data Engineer

October 1, 2020

As data analysis grows within an organization, business teams need the ability to run batch and streaming jobs and leverage the code written by engineers. But re-running existing code often requires setting up a development environment and making slight code changes, which is challenging for people without a programming background.

With this challenge in mind, we recently introduced Dataflow Flex Templates, which make it even easier to turn any Dataflow pipeline into a reusable template that anyone can run.

Existing classic templates let developers share batch and streaming Dataflow pipelines via templates so everyone can run a pipeline without a development environment or writing code. However, classic templates were rigid for a couple of reasons:

First, since the Dataflow pipeline execution graph is permanently fixed when the developer converts the pipeline into a shareable template, classic templates could then

only be run to accomplish the exact task the developer originally had in mind. For example, choosing a source to read from, such as Cloud Storage or BigQuery, had to be determined at the template creation stage and could not be dynamic based on a user's choice during template execution. So developers sometimes had to create several templates with minor variations (such as whether the source was Cloud Storage or BigQuery).

Second, the developer had to select the pipeline source and sink from a limited list of options because of classic templates' dependency on the ValueProvider interface. Implementing ValueProvider allows a developer to defer the reading of a variable to whenever the template is actually run. For example, a developer may know that the pipeline will read from Pub/Sub but wants to defer the name of the subscription for the user to pick at runtime. In practice, this means that developers of external storage and messaging connectors needed to implement Apache Beam's ValueProvider interface to be used with Dataflow's classic templates.

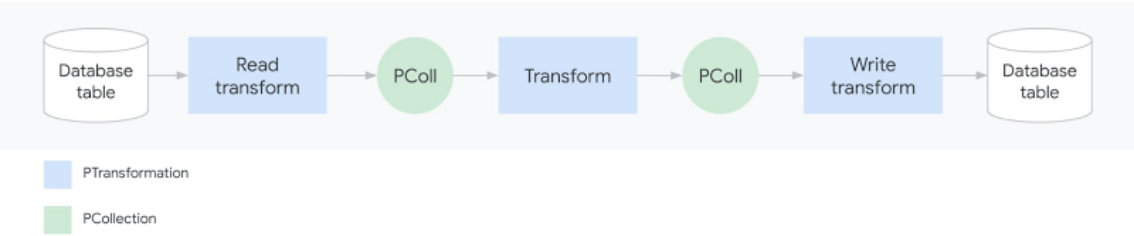
The new architecture of Flex Templates effectively removes both limitations, so we recommend using Flex Templates moving forward.

Flex Templates bring more flexibility over classic templates by allowing minor variations of Dataflow jobs to be launched from a single template and allowing the use of any source or sink I/O. Since the execution graph is now built dynamically when the template is executed (instead of during the template creation process), minor variations can be made to accomplish different tasks with the same underlying template, such as changing the source or sink file formats. Flex Templates also remove the ValueProvider dependency, so any input and output source can be used.

Next, we'll offer a developer's guide to why and how to create custom Flex Templates.

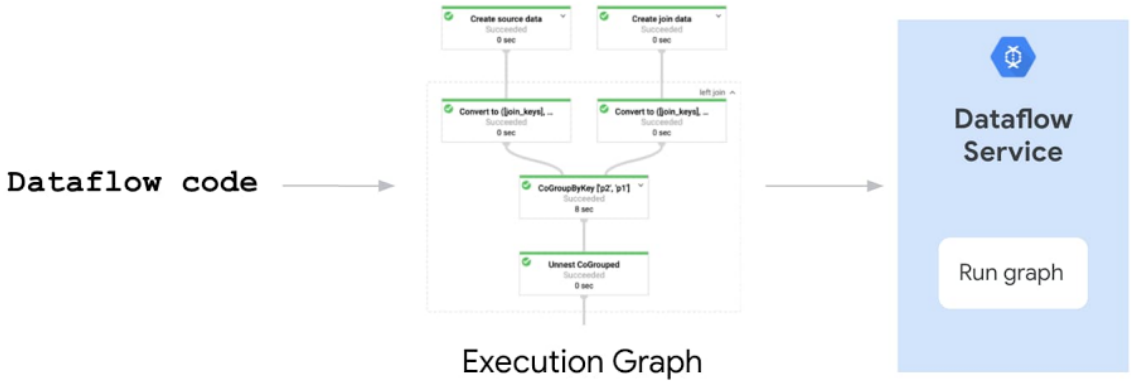
Why sharing Dataflow pipelines has been challenging

An Apache Beam pipeline commonly reads input data (from the source), transforms it (using transforms like ParDo) and writes the output (to the sink):

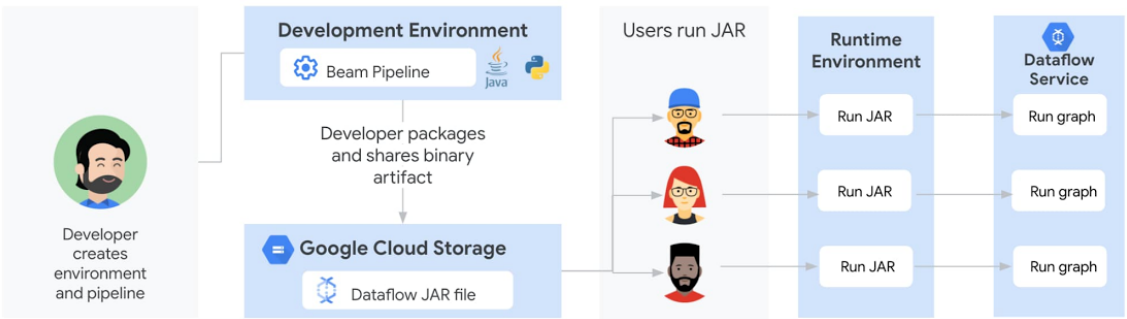


A simple Dataflow pipeline

Pipelines can be significantly more sophisticated, with multiple input sources, series of chained transforms (DAG of steps), and multiple output sinks. Once an Apache Beam pipeline is constructed, it can be deployed and executed in various runners such as Dataflow, Spark, Flink or Direct Runner (for local runs). Templates are a Dataflow-specific feature that makes it easier to re-run pipelines on the Dataflow runner.



But what exactly does "running a pipeline" mean? When you run a Dataflow pipeline, the Apache Beam SDK executes the code locally and builds an execution graph converting the sources, sinks and transforms into nodes. The execution graph object is then translated (serialized) into JSON format and submitted to the Dataflow service. Finally, the Dataflow service performs several validations (API, quota and IAM checks), optimizes the graph, and creates a Dataflow job to execute the pipeline.

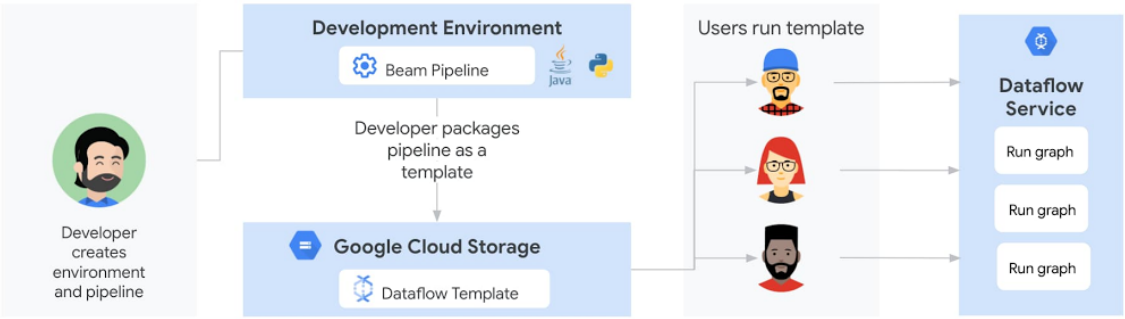


Sharing a Java-based Dataflow pipeline before Templates

Prior to Dataflow templates, it was challenging for developers to share pipelines with coworkers. In the past, a developer would start by creating a development environment (with the JDK, Apache Beam SDK, and Maven or Gradle typically installed for Java or Python, and pip typically installed for Python), write the code, build a binary artifact (fat JAR with dependencies or Python equivalent) and share the artifact either through an artifactory or Cloud Storage. Users would then set up local runtime environments and fetch the binary into their individual environments for execution. If the pipeline was written in Java, the runtime environment would need Java JRE installed; if the pipeline was Python-based, all Python packages the developer used would need to be installed. Finally, when the user ran the pipeline, an execution graph would get generated and sent to the Dataflow service to run the pipeline on cloud.

There were several points where something could break in these steps, and creating a runtime environment was a non-trivial task for users without a technical background. Even scheduling pipelines on a VM (using cron) or third-party schedulers needed a similar runtime environment to exist, complicating the automation process.

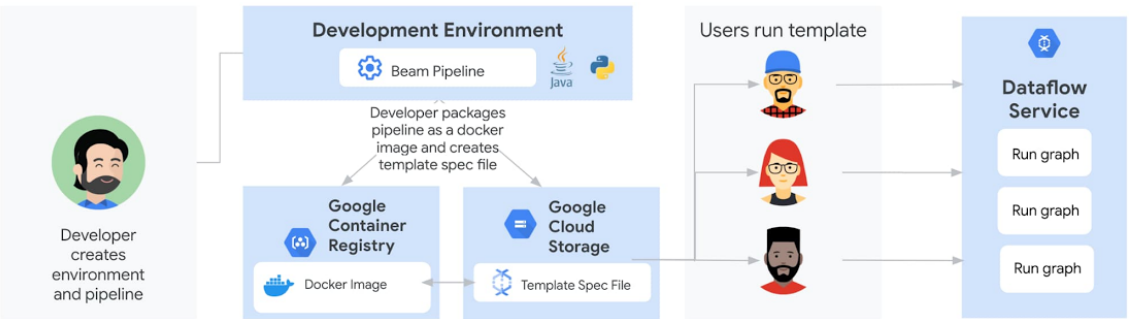
Sharing Dataflow pipelines with classic templates



Sharing a Dataflow pipeline with classic templates

Classic templates significantly improve the user experience for rerunning Dataflow pipelines. When the developer runs the pipeline code in the development environment, the pipeline now gets converted into a Dataflow template stored on Cloud Storage. The staged template consists of the translated JSON execution graph along with dependencies (pipeline artifacts and third-party libraries). The execution graph is permanently fixed and the user cannot change the shape of the DAG. Once the Cloud Storage bucket permissions have been adjusted to share with users, they can invoke the pipeline while passing in any required parameters directly via a gcloud command, a REST API, or the Dataflow UI in Google Cloud Console. Users no longer need to build and configure a runtime environment. Cloud Scheduler can also be used to easily trigger the pipeline to be run on a regular schedule without the need of a runtime environment.

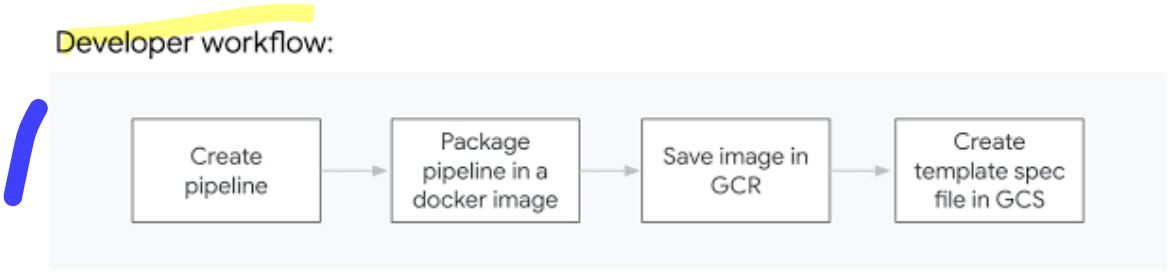
Sharing Dataflow pipelines with Flex Templates



Sharing a Dataflow pipeline with Flex Templates

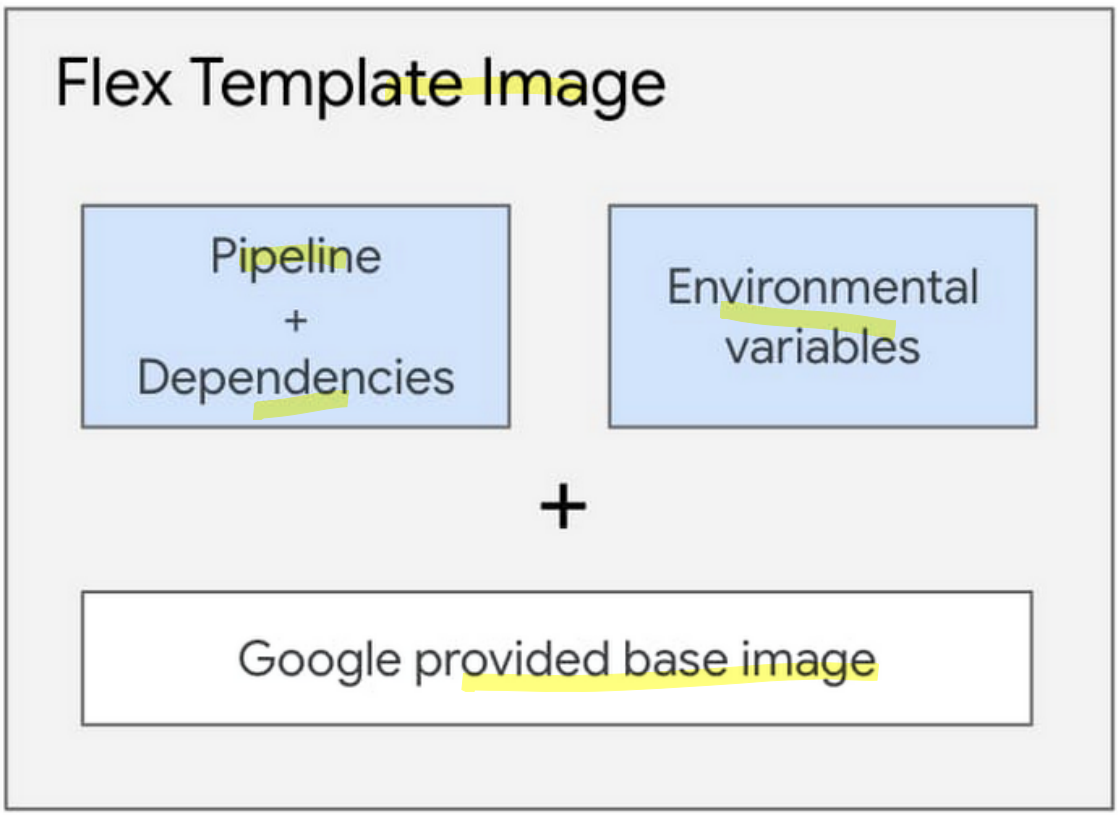
Similar to classic templates, with Flex Templates staging and execution are still separate steps. However, the runnable pipeline artifact that gets staged is different; instead of staging a template file in Cloud Storage, developers now stage a Docker image in Google Container Registry.

Additionally, a developer does not need to run the pipeline to create a Flex Template. Instead, the developer packages the pipeline code/binaries, including dependencies, into a Docker image and stores it in Container Registry, then creates a template spec file stored in Cloud Storage.



Four steps in the developer workflow

The staged image is built using a Google-provided base image and contains the pipeline artifacts with dependencies and environmental variables:



Components inside the Flex Template Docker image

The Docker image does not contain the JSON serialized execution graph. For Java-based pipelines, the image contains the JAR file; for Python pipelines, the image contains the Python code itself. Only when a user actually runs the Flex Template does the graph construction phase start within a new container and the execution graph is constructed based on the parameters the user provides at runtime. This allows for execution graphs to be dynamically constructed based on final input parameters from the user.

The file in Cloud Storage is not the Flex Template, but rather the template spec file. This spec file contains all of the necessary information to run the job, such as the Container Registry image location, SDK language, metadata such as the name and description of the template and any required or optional parameters the template needs. Similar to classic templates, regex can be used to validate the input parameters provided by the user.

Users can execute the Flex Template using a gcloud command, calling the REST API, or using the Dataflow UI in Google Cloud Console referring to a template spec file stored in Cloud Storage and providing required parameters. Automating and scheduling a recurring job can also be done via Cloud Scheduler or Terraform (support for Airflow is under development).

Comparing classic vs. Flex Templates

The following table summarizes the similarities and differences between classic and Flex templates:

Feature	Classic	Flex
Any authorized user can invoke template via Google Cloud Console, gcloud cmd line tool, or REST API	✓	✓
Running pipeline does not require recompiling code	✓	✓
Run pipeline without development environment and associated dependencies	✓	✓
Runtime parameters to customize execution of the pipeline	✓	✓
Separation of staging and execution steps	✓	✓
Job execution graph can be changed after the template is created		✓
Support IOs beyond ValueProvider		✓
Support using SQL as a parameter		✓
Reduce runtime errors by running validations upon job graph construction		✓

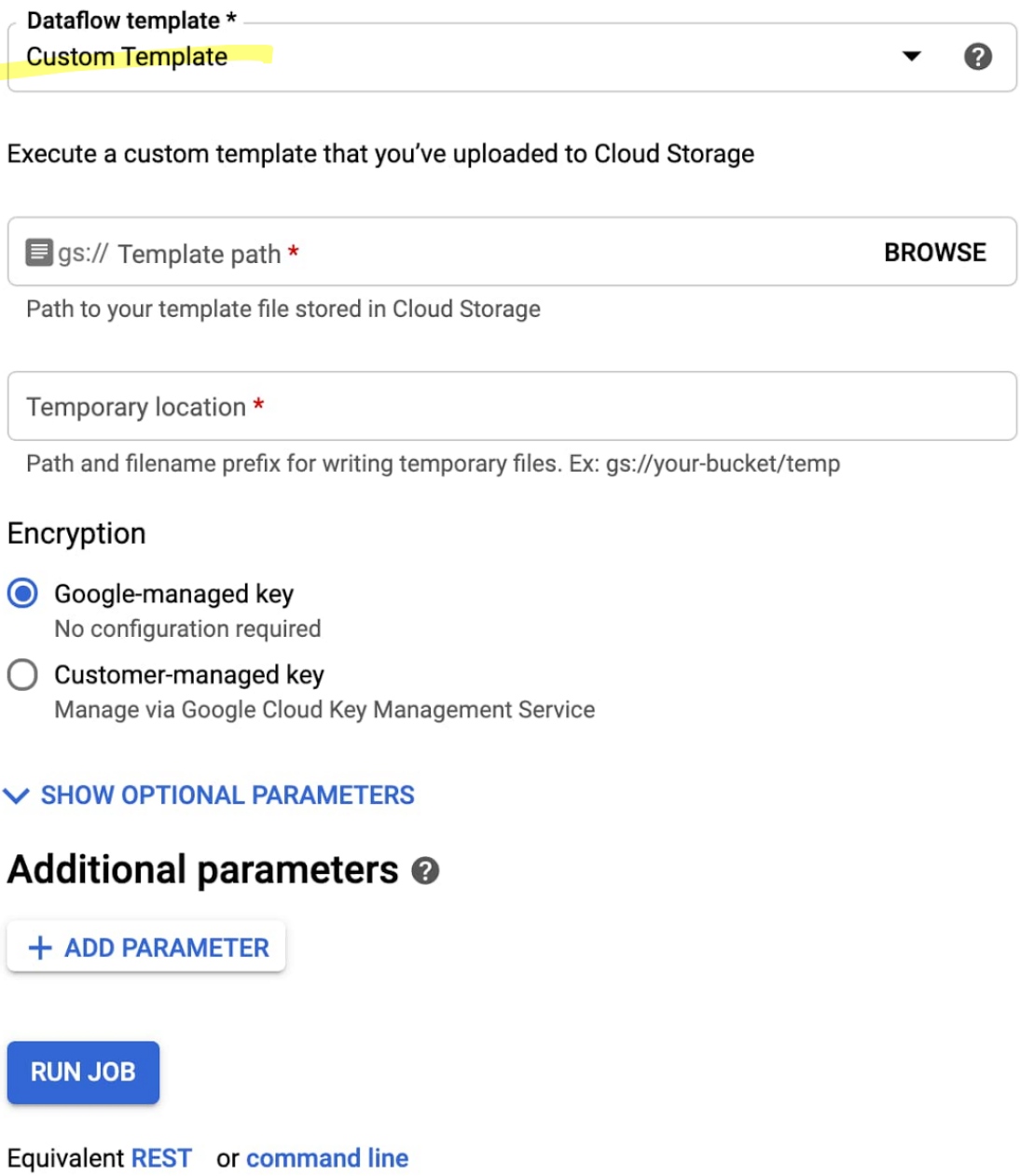
Create your first Dataflow Flex Template

If you are new to Dataflow templates, we recommend starting with our Google Cloud-provided templates for moving data between systems with minimal processing. These are production-quality templates that can be easily run from the Dataflow UI in Google Cloud Console.

If you want to automate a task that is not in the provided templates, follow our Using Flex Templates guide. The 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.

You can also review the source code for the Google-provided templates and review our examples for generating random data, decompressing data in Cloud Storage, analyzing tweets, or doing data enrichment tasks like obfuscating data before writing it to BigQuery.

Finally, when you're ready to share the Flex Template with users, the Google Cloud Console UI provides an option to select a Custom Template and then asks for a Cloud Storage path of its location:



Creating a custom template from Google Cloud Console

Learn more about Dataflow on our site, and check out our presentation on Flex Templates at Beam Summit.

Thanks to contributors to the design and development of the new release of Dataflow Flex Templates, in no particular order: Mehran Nazir, Sameer Abhyankar, Yunqing Zhou, Arvind Ram Anantharam, Runpeng Chen, and the rest of the Dataflow team.

POSTED IN: DATA ANALYTICS—GOOGLE CLOUD PLATFORM