Streaming IoT Kafka to Google Cloud Pub/Sub

## Overview

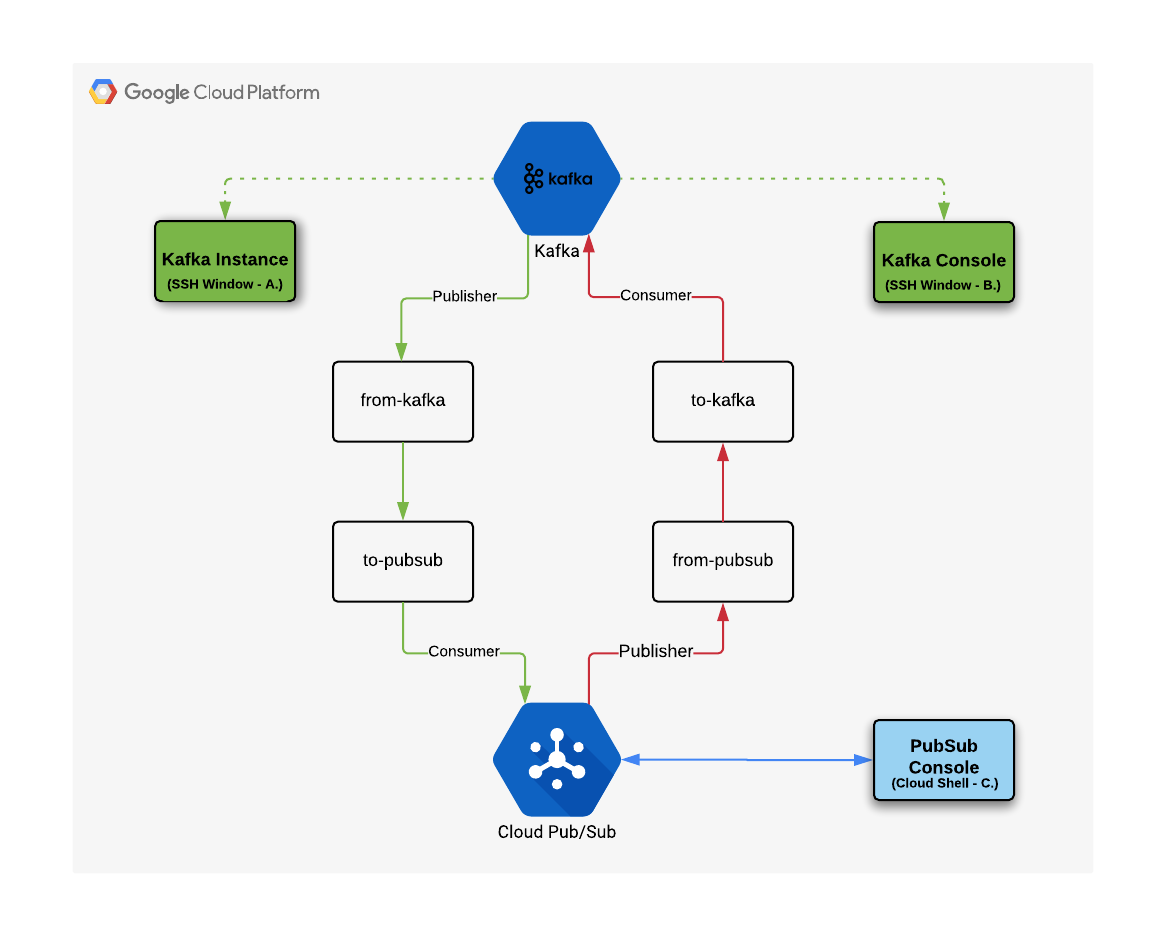
In this lab, you launch a Confluent Kafka instance and use it to communicate with Cloud Pub/Sub.

In this lab, you learn how to:

* Initiate Cloud Launcher to create an instance of Confluent Kafka
* Configure a Kafka connector to integrate with Pub/Sub
* Setup topics and subscriptions for message communication
* Perform basic testing of both Kafka and Cloud Pub/Sub services
* Connect IoT Core to Cloud Pub/Sub

For this lab you will be using multiple SSH sessions.You might want to think about how to organize them on your desktop. Leave these windows open during your lab.

|  |  |  |
| --- | --- | --- |
| Name | Identifier | Description |
| Kafka Instance | SSH Window A. | This window will be used to run the application in the background. |
| Kafka Console | SSH Window B.(Publisher/Consumer) | This window is used to access the Kafka command line. |
| Cloud PubSub Console | Cloud Shell C. | This shell is required to set up PubSub and view the queue. |



## Introduction

With the announcement of the Google Cloud Confluent managed Kafka offering, it has never been easier to use Google Cloud's great data tools with Kafka. You can use the Apache Beam Kafka.io connector to go straight into Dataflow, but this may not always be the right solution.

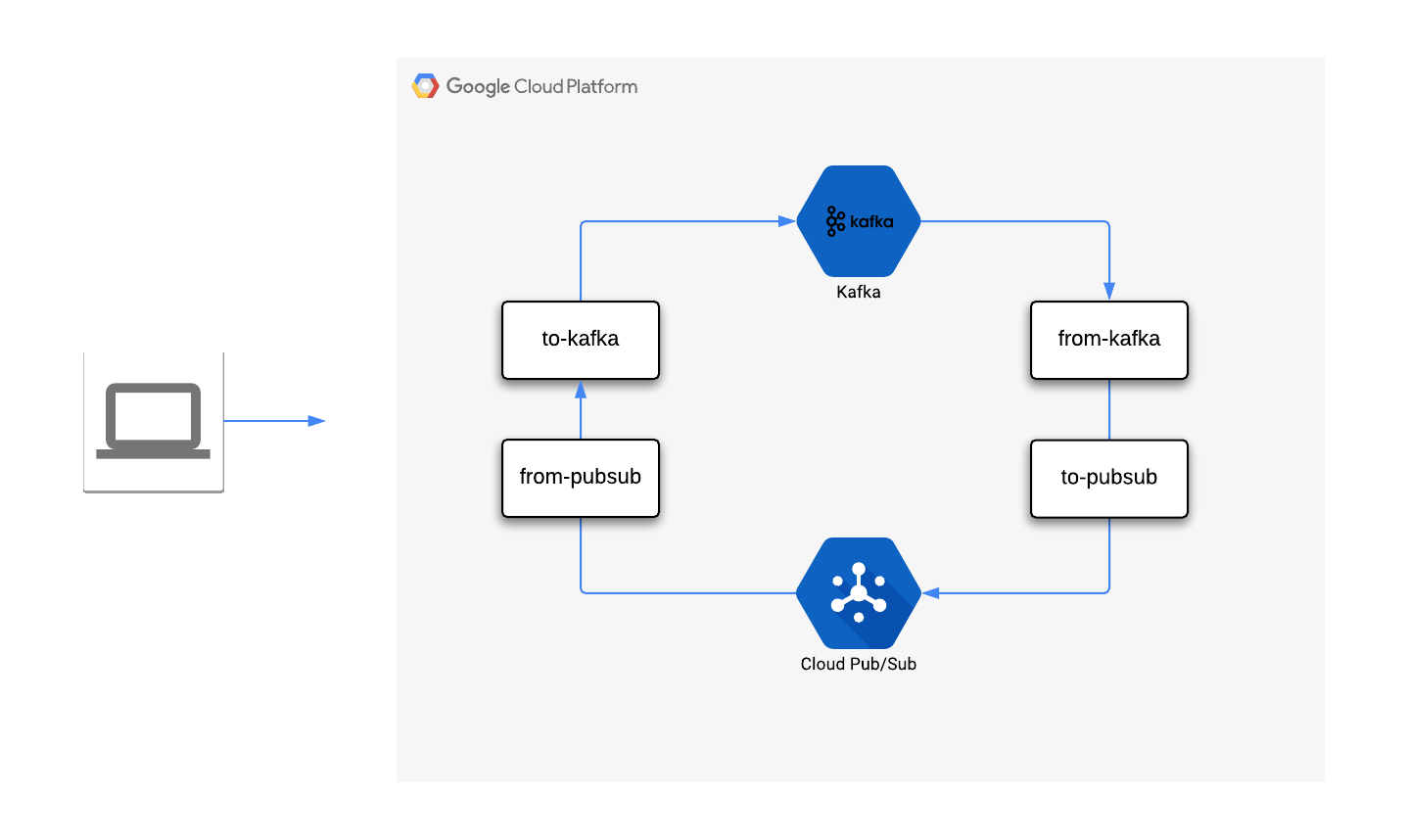
Whether Kafka is provisioned in the Cloud or on premise, you might want to push to a subset of Pub/Sub topics. Why? For the flexibility of having Pub/Sub as your GCP event notifier. Then you could not only choreograph Dataflow jobs, but also use topics to trigger Cloud Functions.

So how do you exchange messages between Kafka and Pub/Sub? This is where the Pub/Sub Kafka Connector comes in handy. In this lab you will learn the basic steps to start working with a Kafka connector on Google Cloud.

**Tip:**In this lab Cloud Launcher is used to create a single instance of Kafka. This Kafka instance connects to Cloud Pub/Sub and exchanges event messages between the two services.

In the real world, Kafka would likely be run in a cluster, but for this lab you will develop a simpler (and cheaper) usage pattern.

The following diagram illustrates the high-level architecture for this lab.



## Setting up the environment

### **Launch a Kafka instance via Marketplace**

1. In the GCP Console, in the left-hand **Navigation menu**, click **Marketplace**.
2. Locate the Apache Kafka deployment by searching for **Kafka**.
3. Click on the Google deployment of Kafka and read about the service provided by the software.
4. Click on the blue **Launch On Compute Engine** button.
5. You can accept the default setting, or make changes based on your requirements. For example, selecting a Zone that is closer in locality. Check off the Terms of Service box, then click **Deploy**.

Once the deployment completes, a new VM instance named kafka-1-vm is available in the zone you selected.

Next you need to alter the Cloud API scope on the VM created to enable the cloud Pub/Sub API.

1. Click on the instance name on the right side of the screen, then **Stop** the Kafka virtual machine instance. Refresh your screen and wait until you see the instance has stopped before continuing. It will take 1-2 minutes. Refresh your screen to make sure you're looking at the most recent information. You will see a new icon next to the name indicating it is stopped.
2. Click **Edit** then scroll down and select **Allow full access to all Cloud APIs** in the Access scopes section, then click **Save**.
3. Restart the Kafka virtual machine instance by clicking **Start**. There will be a confirmation message when it has restarted. Refresh your screen to make sure you're lookin at the most recent information.

## Configure the Kafka VM instance

1. In the Console, go to **Navigation Menu** > **Compute Engine** and open a SSH cloud shell to the Kafka VM. (This is SSH Window A.)
2. In the SSH window, set an environment variable to the project identifier.

export PROJECT\_ID=[PROJECT\_ID]

1. Copy the generated jar file from the storage bucket to the Kafka VM.

gsutil cp gs://cloud-training/gsp285/binary/cps-kafka-connector.jar .

1. Create the destination sub-directory for the Kafka connector:

sudo mkdir -p /opt/kafka/connectors

1. Move the downloaded jar file to the directory created for the Kafka application:

sudo mv ./cps-kafka-connector.jar /opt/kafka/connectors/

1. Update the java connector file permissions to be executable:

sudo chmod +x /opt/kafka/connectors/cps-kafka-connector.jar

1. Change the current directory to **/opt/kafka/config**:

cd /opt/kafka/config

1. Using an editor create **cps-sink-connector.properties**:

sudo nano cps-sink-connector.properties

1. Add the following content, replacing PROJECT\_ID with your GCP Project ID:

name=CPSSinkConnector

connector.class=com.google.pubsub.kafka.sink.CloudPubSubSinkConnector

tasks.max=10

topics=to-pubsub

cps.topic=from-kafka

cps.project=PROJECT\_ID

1. Using an editor create another file named **cps-source-connector.properties**:

sudo nano cps-source-connector.properties

1. Add the following content, replacing PROJECT\_ID with your GCP Project ID:

name=CPSSourceConnector

connector.class=com.google.pubsub.kafka.source.CloudPubSubSourceConnector

tasks.max=10

kafka.topic=from-pubsub

cps.subscription=to-kafka-sub

cps.project=PROJECT\_ID

The Kafka application is now configured to use the connector. Leave this SSH connection to the Kafka VM instance open, so we can finish the configuration and run the application later.

## Pub/Sub Topic and Subscription setup

1. In the Cloud Shell window, set an environment variable to the project identifier.

export PROJECT\_ID=[PROJECT\_ID]

1. Configure the Pub/Sub topics to communicate with Kafka:

gcloud pubsub topics create to-kafka from-kafka

1. Create a subscription for the to-kafka topic:

gcloud pubsub subscriptions create to-kafka-sub --topic=to-kafka --topic-project=$PROJECT\_ID

Pub/Sub is now configured with two topics. A subscription has also been created on the "to-kafka" topic using the PROJECT\_ID variable.

This configuration allows messages to be consumed by Pub/Sub. Go look at Pub/Sub in Console

1. Now create a subscription for traffic published from Kafka:

gcloud pubsub subscriptions create from-kafka --topic=from-kafka

## Start the Kafka VM application instance

Now you will set up the Kafka VM Topic.

1. Return to the Kafka VM instance (SSH Window A.) and submit the following command:

cd /opt/kafka

1. Create a topic that will exchange information to Pub/Sub:

kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic to-pubsub

1. Create a topic that will receive messages from Pub/Sub:

kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic from-pubsub

1. You need to make a couple of edits to the existing file located in **/opt/kafka/config/connect-standalone.properties** by running:

sudo nano /opt/kafka/config/connect-standalone.properties

1. First, uncomment the plugin path that is located at the bottom of the file and updated it to the following:

plugin.path=/opt/kafka/connectors

1. Second, toggle the key and value converter for schema enforcement to the setting false; close the editor when you're done:

key.converter.schemas.enable=false

value.converter.schemas.enable=false

1. Move back to the user home directory:

cd ~

1. Use an editor to create a new file named **run-connector.sh** and add these contents to it:

#!/bin/bash

/opt/kafka/bin/connect-standalone.sh /opt/kafka/config/connect-standalone.properties \

/opt/kafka/config/cps-sink-connector.properties \

/opt/kafka/config/cps-source-connector.properties

1. Update the file permissions to allow it to be executed from the command line:

sudo chmod +x ./run-connector.sh

1. Start the connect service:

./run-connector.sh

The Kafka service should now be running on the VM. **Leave this session open** so that any errors can be seen.

## Data exchange between Kafka and Pub/Sub

Test Kafka to Pub/Sub (producer/consumer) communication by opening a new SSH window where the Kafka commands will be run.

1. Open a new SSH connection to the **Kafka VM**, this is SSH Window B. Enter the following command to initiate a Kafka console:

kafka-console-producer.sh --broker-list localhost:9092 --topic to-pubsub

1. From the Kafka console enter the following data elements, then press **Ctrl+c** to terminate the command entry:

{"message":"Hello"}

{"message":"Big Data"}

1. Return to **Cloud Shell** to see the information submitted earlier:

gcloud pubsub subscriptions pull from-kafka --auto-ack --limit=10

**Note:** You may need to run this command a couple of times to see results.

Cloud Shell Example Output (do not copy)

┌────────────────────┬─────────────────┬────────────┐

│ DATA │ MESSAGE\_ID │ ATTRIBUTES │

├────────────────────┼─────────────────┼────────────┤

│ {message=Hello} │ 169722717841938 │ │

│ {message=Big Data} │ 169724507782910 │ │

└────────────────────┴─────────────────┴────────────┘

Kafka to Pub/Sub messaging is configured and working as expected.

### **Test Pub/Sub to Kafka**

1. In SSH Window B. enter the following command:

kafka-console-consumer.sh --bootstrap-server=localhost:9092 --value-deserializer=org.apache.kafka.common.serialization.StringDeserializer --topic from-pubsub

1. In the active Cloud Shell window, publish a topic to be consumed by Kafka

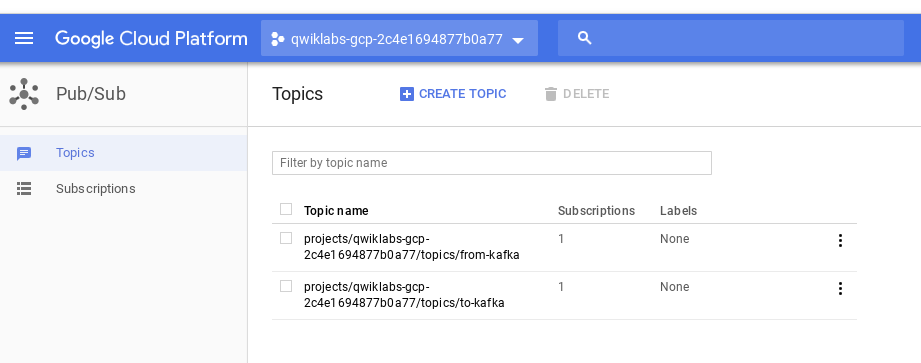
gcloud pubsub topics publish to-kafka --attribute=data=Hello

Check SSH Window B. for the Kafka VM Example Output:

{"message":"","data":"hello"}

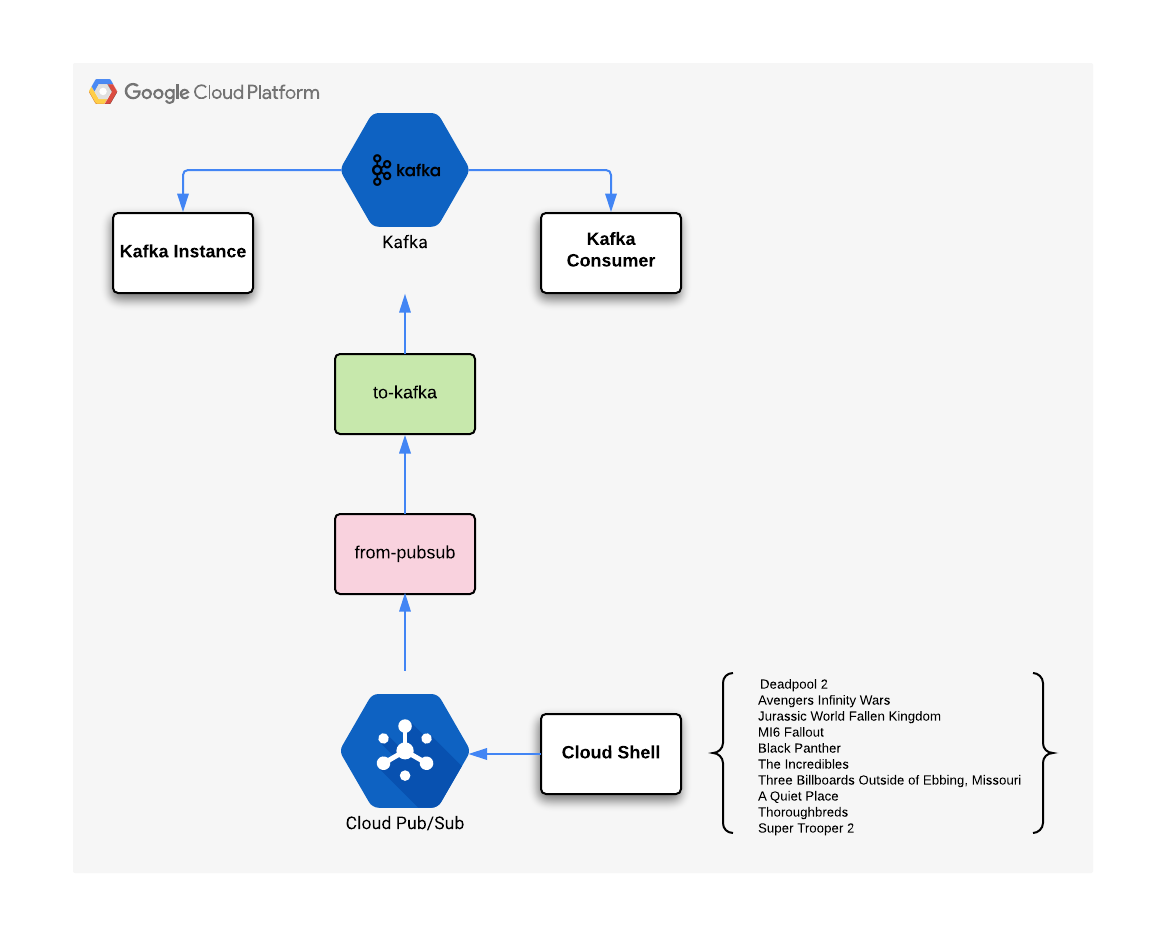
Pub/Sub to Kafka message is configured and working as expected.

**Ctrl+c** to stop this process.

If you go to **Navigation menu** > **Pub/Sub**, you will see the two topics: 

## Pub/Sub to Kafka Testing

Your architecture for testing Pub/Sub to Kafka is as illustrated below:



**Note:**Ensure that a Kafka instance is actually running in the background - there should still be an open window showing the output from the instance.

If the application instance is not currently running, open a new SSH connection to Kafka and run the command ./run-connector.sh at the command line.

1. In the consumer/producer SSH session (SSH Window B.) enter the following command:

kafka-console-consumer.sh --bootstrap-server=localhost:9092 --value-deserializer=org.apache.kafka.common.serialization.StringDeserializer --topic from-pubsub

1. From Cloud Shell you'll create some example content. Use an editor and make a text file named movies.txt, and add the following contents to it:

Deadpool 2

Avengers Infinity Wars

Jurassic World Fallen Kingdom

MI6 Fallout

Black Panther

The Incredibles

Three Billboards Outside of Ebbing Missouri

A Quiet Place

Thoroughbreds

Super Trooper 2

1. Enter the following script to publish your movie messages to the Kafka consumer:

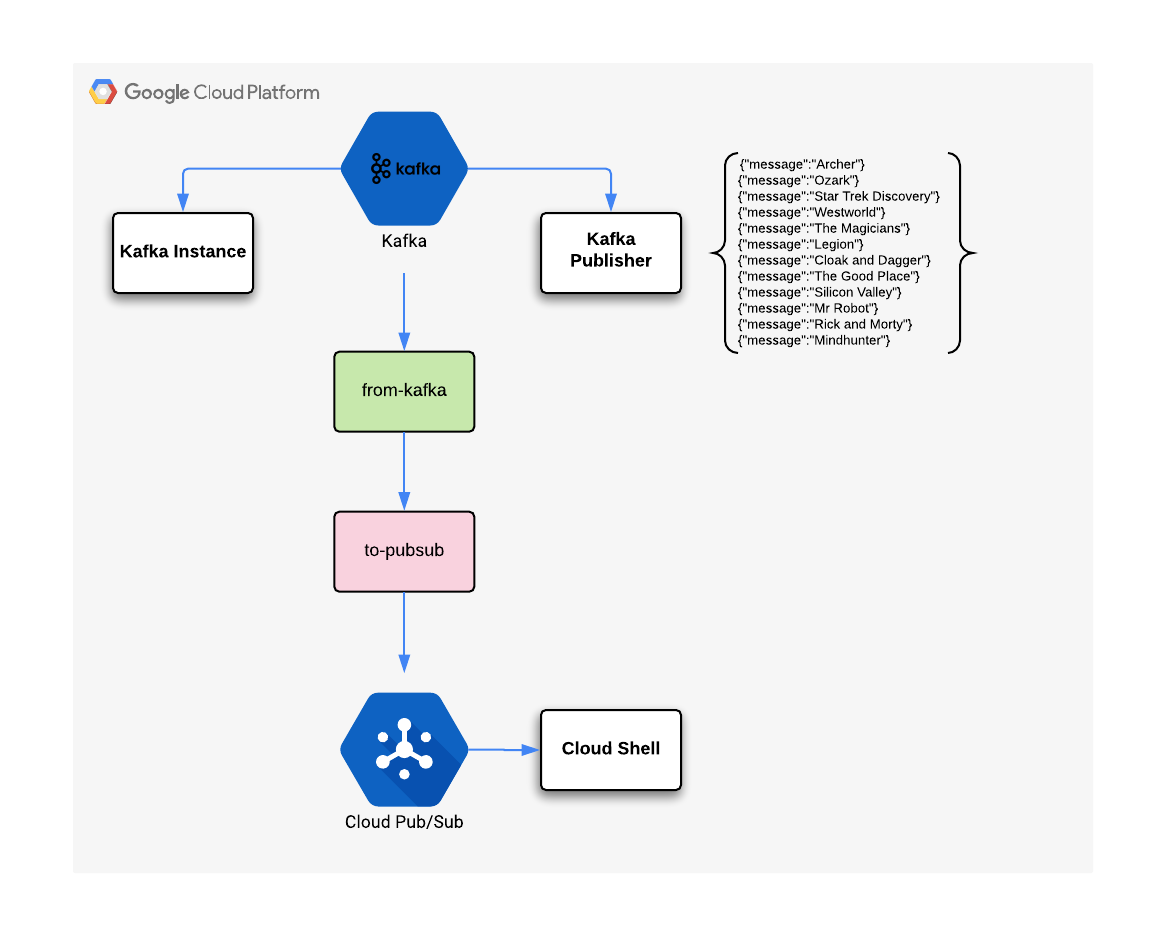
while read i; do gcloud pubsub topics publish to-kafka --attribute=data="$i"; done < movies.txt

From the above command a stream of messages should be observable on the Kafka consumer window.

In this example we have sent a steam of information between two services. As the example demonstrates, exchanging information once configured is fairly straightforward.

### **Kafka to Pub/Sub testing**

Our architecture for testing Kafka to Pub/Sub is illustrated below:



1. **Ctrl+c** in SSH Window B, then Use an editor to create a text file named tv.json and add the following contents:

{"message":"Archer"}

{"message":"Ozark"}

{"message":"Star Trek Discovery"}

{"message":"Westworld"}

{"message":"The Magicians"}

{"message":"Legion"}

{"message":"Cloak and Dagger"}

{"message":"The Good Place"}

{"message":"Silicon Valley"}

{"message":"Mr Robot"}

{"message":"Rick and Morty"}

{"message":"Mindhunter"}

1. Now use the following script to publish your TV messages to the Pub/Sub consumer:

kafka-console-producer.sh --broker-list localhost:9092 --topic to-pubsub < tv.json

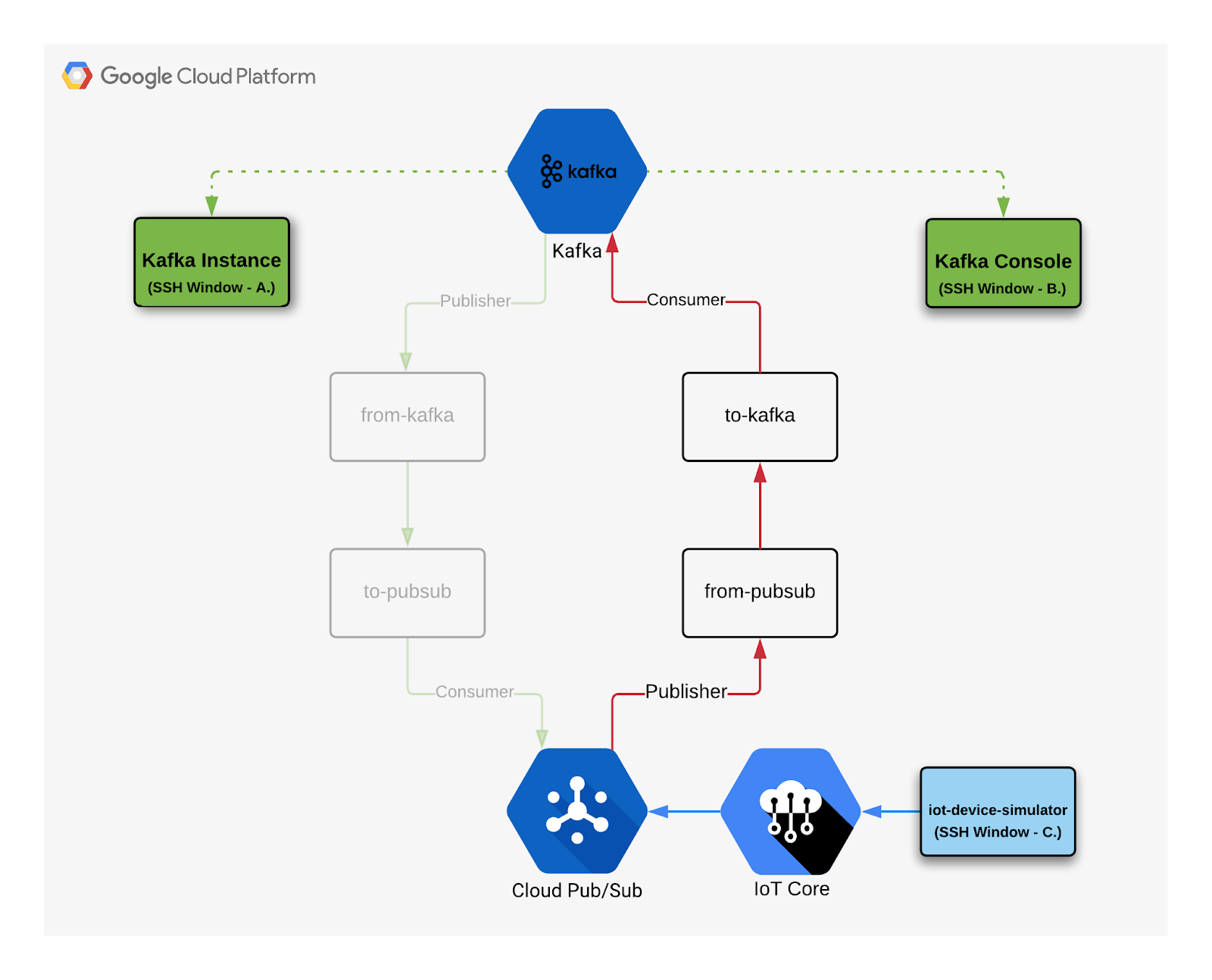
1. From the Cloud Shell window, run the following command to view the messages that have been published from Kafka:

gcloud pubsub subscriptions pull from-kafka --auto-ack --limit=10

In this example you have sent a stream of information between two services. When passing information via Kafka, the message content is formatted as JSON.

## IOT Simulator - IoT core

Extending your architecture allows the opportunity to explore further integration. In this section the IoT core service will be used to demonstrate connectivity of IoT devices, as illustrated below:



1. Create a new VM named iot-device-simulator and select the **Allow full access to all Cloud APIs** radio button.`
2. SSH into the instance and update it by running the following:

sudo apt-get update && sudo apt-get --only-upgrade install kubectl google-cloud-sdk google-cloud-sdk-app-engine-grpc google-cloud-sdk-pubsub-emulator google-cloud-sdk-app-engine-go google-cloud-sdk-cloud-build-local google-cloud-sdk-datastore-emulator google-cloud-sdk-app-engine-python google-cloud-sdk-cbt google-cloud-sdk-bigtable-emulator google-cloud-sdk-app-engine-python-extras google-cloud-sdk-datalab google-cloud-sdk-app-engine-java

1. Install python openssl & git packages:

sudo apt-get install python-pip openssl git -y

1. Install python components for messaging:

sudo pip install pyjwt paho-mqtt cryptography

1. Clone a git repository to gain access to the lab specific tools:

git clone http://github.com/GoogleCloudPlatform/training-data-analyst

1. Add an environment variable for the current project id - replace [PROJECT\_ID] with the GCP project id:

export PROJECT\_ID=[PROJECT\_ID]

1. Add an environment variable for the region - replace [MY\_REGION] with the iot-device-simulator VM GCP region:

export MY\_REGION=[MY\_REGION]

1. Create a device registry named iotlab-registry:

gcloud beta iot registries create iotlab-registry \

--project=$PROJECT\_ID \

--region=$MY\_REGION \

--event-notification-config=topic=projects/$PROJECT\_ID/topics/to-kafka

1. Change the working directory to the iotlab directory:

cd $HOME/training-data-analyst/quests/iotlab/

1. Create a cryptographic key pair that will allow IoT devices to connect to Cloud Pub/Sub:

openssl req -x509 -newkey rsa:2048 -keyout rsa\_private.pem \

-nodes -out rsa\_cert.pem -subj "/CN=unused"

The simulated devices to be created provide temperature readings from around the world. In the example we will setup an IoT device for Buenos Aires and read values from it into Pub/Sub.

1. Create a simulated device for Buenos Aires based on the current project settings:

gcloud beta iot devices create temp-sensor-buenos-aires \

--project=$PROJECT\_ID \

--region=$MY\_REGION \

--registry=iotlab-registry \

--public-key path=rsa\_cert.pem,type=rs256

1. Download the CA root certificates from pki.google.com:

wget https://pki.google.com/roots.pem

**Note:**Before the IoT device simulator is started, make sure that a:

* Background Kafka instance is running
* Kafka consumer instance is ready to accept messages

1. In the SSH session for iot-device-simulator, run the following code to begin generating temperature readings to be consumed by Cloud Pub/Sub.

python cloudiot\_mqtt\_example\_json.py \

--project\_id=$PROJECT\_ID \

--cloud\_region=$MY\_REGION \

--registry\_id=iotlab-registry \

--device\_id=temp-sensor-buenos-aires \

--private\_key\_file=rsa\_private.pem \

--message\_type=event \

--algorithm=RS256

1. In the consumer/producer SSH session (SSH Window B.) enter the following command:

kafka-console-consumer.sh --bootstrap-server=localhost:9092 --value-deserializer=org.apache.kafka.common.serialization.StringDeserializer --topic from-pubsub

Once the python command is running, it will send a stream of messages via PubSub to the Kafka instances:

* iot-device-simulator is displaying the list of temperatures.
* SSH Window B. (Kafka consumer) is receiving the temperatures.

At this point, the architecture has been extended to include IoT Core. The example provides a simulated approach that can be further extended to include real devices.