Cloud Run Using Pubsub Triggers

A few months back Google announced something really cool, a little known project called Cloud Run — and from the initial announcement, Cloud Run caused some serious noise. Cloud Run comes out at a very interesting time. Just as developers are coming **in mass** to Kubernetes, Google came out with a new tool that makes it so whether you know the difference between a *Deployment* and an *Ingress* you can still create amazing tools that run built on the power of Kubernetes and Google’s GKE.

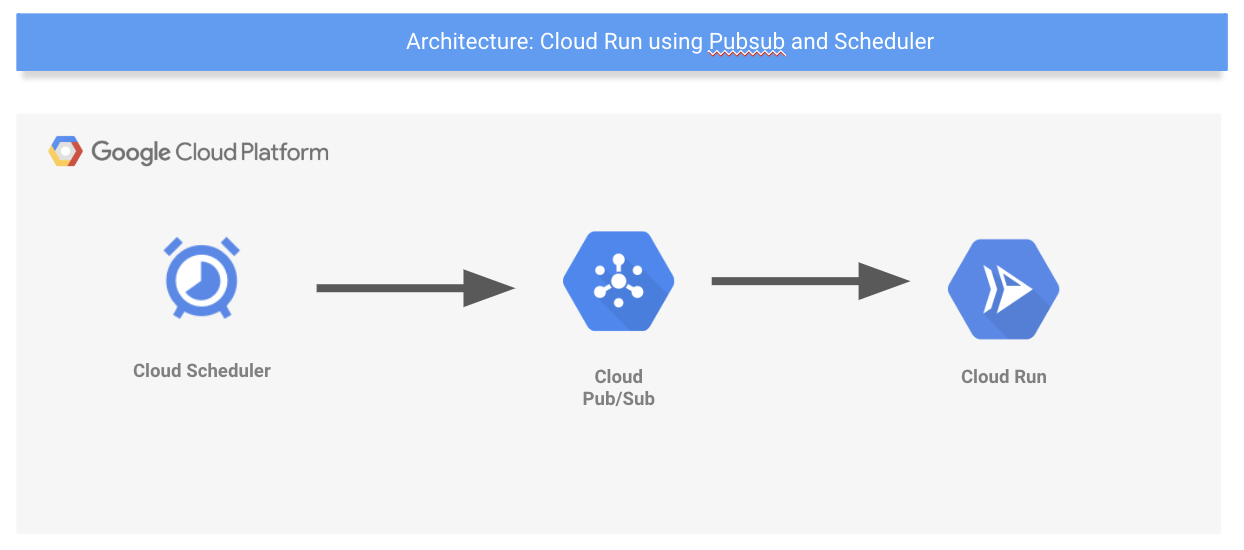
*If you want to go further down the rabbit hole you’ll learn that under the covers Google is using another new project that is gaining steam called*[*KNative*](https://knative.dev/)*. Knative uses the power of Kubernetes and — with some black magic — turns Kubernetes into a Serverless platform that allows you to ramp up your workload based on usage and ramp down your workload all the way to zero if there is no traffic.*

With Cloud Run the big benefit is with just your Dockerized application and a few clicks you can have your application running on Kubernetes in an environment you can trust to scale up and down as your usage scales.

Now, in this article, I am going to quickly show you that you can have everything running in no time. Without further ado, let’s get started.

Cloud Run Example Project Using Pubsub and Scheduler

As a quick little project, we are going to make a system that uses Pubsub to trigger your Cloud Run workload. I felt the easiest way to trigger these Pubsub messages was to use Cloud Scheduler to trigger Pubsub on an interval that we can set (every minute in this example). What will this look like? Like this.



Cloud Scheduler To Cloud Run via Pubsub

Our Cloud Run workload is written in NodeJS but if you look at the final code you’ll see that the language doesn’t really matter for this example because of how simple it is.

All of the code is viewable in [my Github Repo](https://github.com/jonbcampos/cloudrun-series/tree/master/getting-started).

Setting Up A Basic Node Project

The NodeJS project is fairly simple. A tiny Express application and one POST endpoint for our PubSub subscription to push data into.

|  |
| --- |
| // imports |
|  | const express = require("express"); |
|  | const bodyParser = require("body-parser"); |
|  | const app = express(); |
|  | // setup |
|  | app.use(bodyParser.json()); |
|  | // POST endpoint |
|  | app.post("/", (req, res) => { |
|  | // parse body |
|  | if (!req.body) { |
|  | const msg = "no Pub/Sub message received"; |
|  | console.error(`error: ${msg}`); |
|  | res.status(400).send(`Bad Request: ${msg}`); |
|  | return |
|  | } |
|  | if (!req.body.message) { |
|  | const msg = "invalid Pub/Sub message format"; |
|  | console.error(`error: ${msg}`); |
|  | res.status(400).send(`Bad Request: ${msg}`); |
|  | return |
|  | } |
|  | // parse message |
|  | const pubSubMessage = req.body.message; |
|  | const originatorMessage = Buffer.from(pubSubMessage.data, "base64").toString().trim(); |
|  | const now = new Date(); |
|  | // log output |
|  | console.log(`${originatorMessage} says the time is ${now.toString()}`); |
|  | res.status(204).send() |
|  | }); |
|  | // start server |
|  | const PORT = process.env.PORT || 8080; |
|  | app.listen(PORT, () => console.log(`listening on port ${PORT}`)); |

Index.js File Consuming Pubsub Message

You can look deeper into [each line of code in the getting started project in Github](https://github.com/jonbcampos/cloudrun-series/tree/master/getting-started).

Build And Deploy The Project

To build the project into a deployable Dockerized application we just need to run one quick command to [build and store the application within GCP](https://github.com/jonbcampos/cloudrun-series/blob/master/getting-started/scripts/build.sh).

gcloud builds submit --tag gcr.io/${GCLOUD\_PROJECT}/${IMAGE\_NAME} .

Dockerize Application

This is it! This is the moment. The all-important moment where we [publish our application to Cloud Run](https://github.com/jonbcampos/cloudrun-series/blob/master/getting-started/scripts/deploy.sh). Are you ready for this?

*gcloud beta run deploy ${PROJECT\_NAME} \*

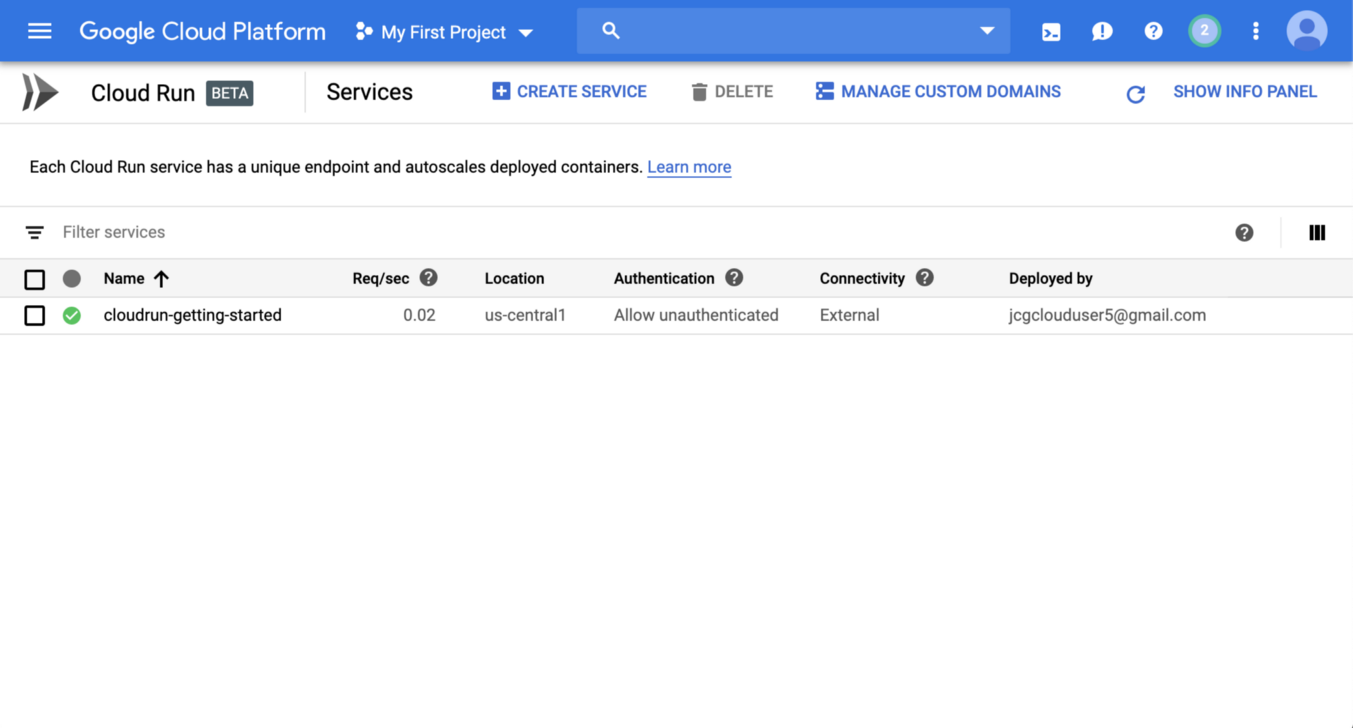
*--image gcr.io/${GCLOUD\_PROJECT}/${IMAGE\_NAME} \*

*--region ${INSTANCE\_REGION} \*

*--platform managed*

Deploy To Cloud Run

That’s it! That is how hard it is. Okay, it is crazy easy and now completely scalable. Seriously, this is basically production-ready.



Cloud Run Project Running

Everything from now on is downhill to support our application.

Add Pubsub

Our application is running but nothing is really happening at this point because we don’t have Pubsub wired up. We will [now add Pubsub into our GCP Project](https://github.com/jonbcampos/cloudrun-series/blob/master/getting-started/scripts/add-pubsub.sh).

echo "enable pubsub"

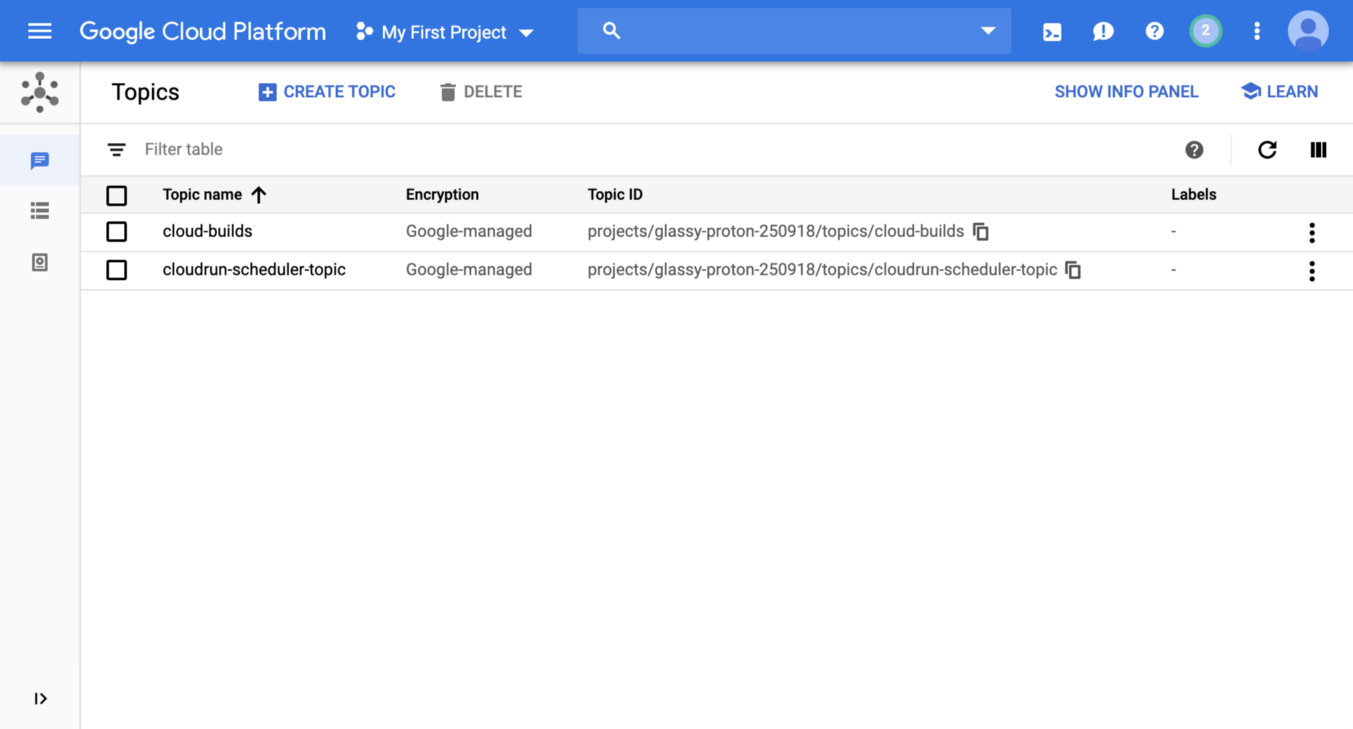
gcloud services enable pubsub.googleapis.com

echo "create topic"

gcloud pubsub topics create ${TOPIC\_NAME}

Enable Pubsub

This script enables Pubsub and creates a Pubsub Topic.



Created Pubsub Message

The stage is set. We just need to set up the Cloud Scheduler dispatch the Pubsub Topic.

Add The Scheduler

[Cloud Scheduler](https://cloud.google.com/scheduler/) is a really slick tool that allows you to set up a Cron schedule that will fire off either an HTTP message or Pubsub message on a regular interval. For our project, we are having Cloud Scheduler send out a Pubsub message every minute to simulate an application dispatching Pubsub messages. Again, [one line of bash script later and we are good to go](https://github.com/jonbcampos/cloudrun-series/blob/master/getting-started/scripts/add-scheduler.sh).

gcloud scheduler jobs create pubsub ${SCHEDULER\_NAME} \

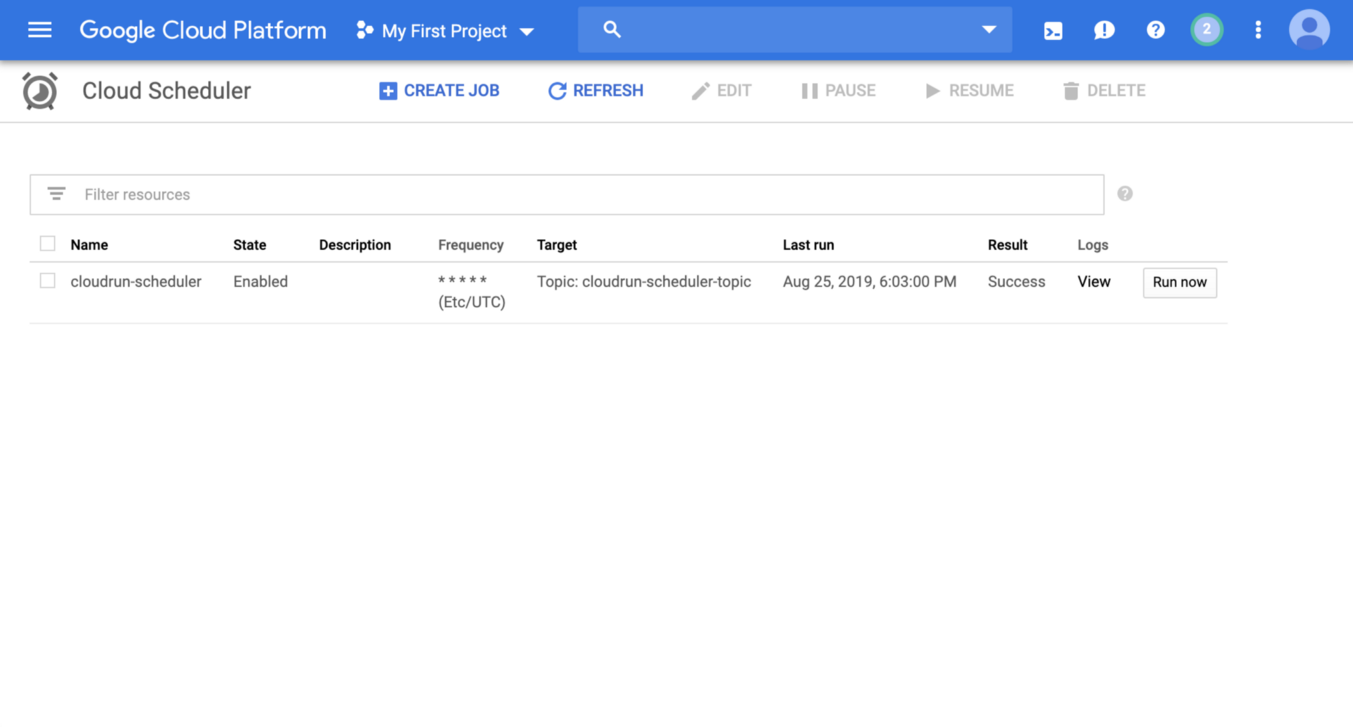
--schedule="\* \* \* \* \*" \

--topic=${TOPIC\_NAME} \

--message-body="scheduler system"

Create a Scheduler Job

Cloud Scheduler is now dispatching the topic every minute.



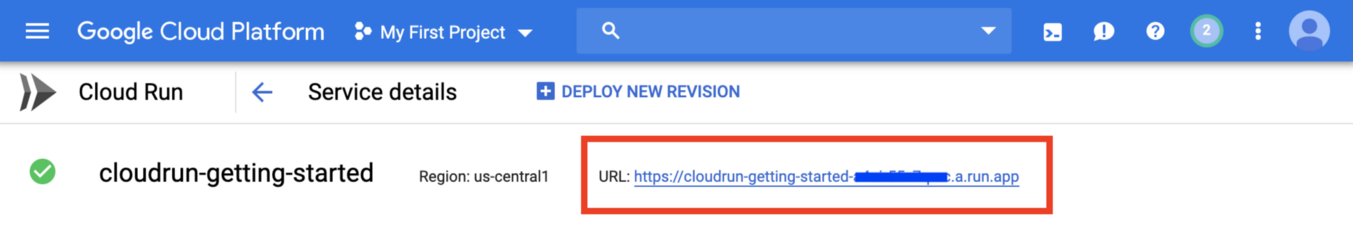
Created Cloud Scheduler

The last thing we need to do is set up a subscription to push the topic into our Cloud Run workload.

Add A Subscription

This is the last/most complicated part of the process. We need to create a service account, get the URL for our Cloud Run endpoint and then create the subscription to route the Pubsub messages to Cloud Run.

First, the URL. This can be found in the Cloud Run dashboard very easily.



Cloud Run URL

Next, we need to create the subscription. [This is done with a few lines of a bash script.](https://github.com/jonbcampos/cloudrun-series/blob/master/getting-started/scripts/add-subscription.sh)

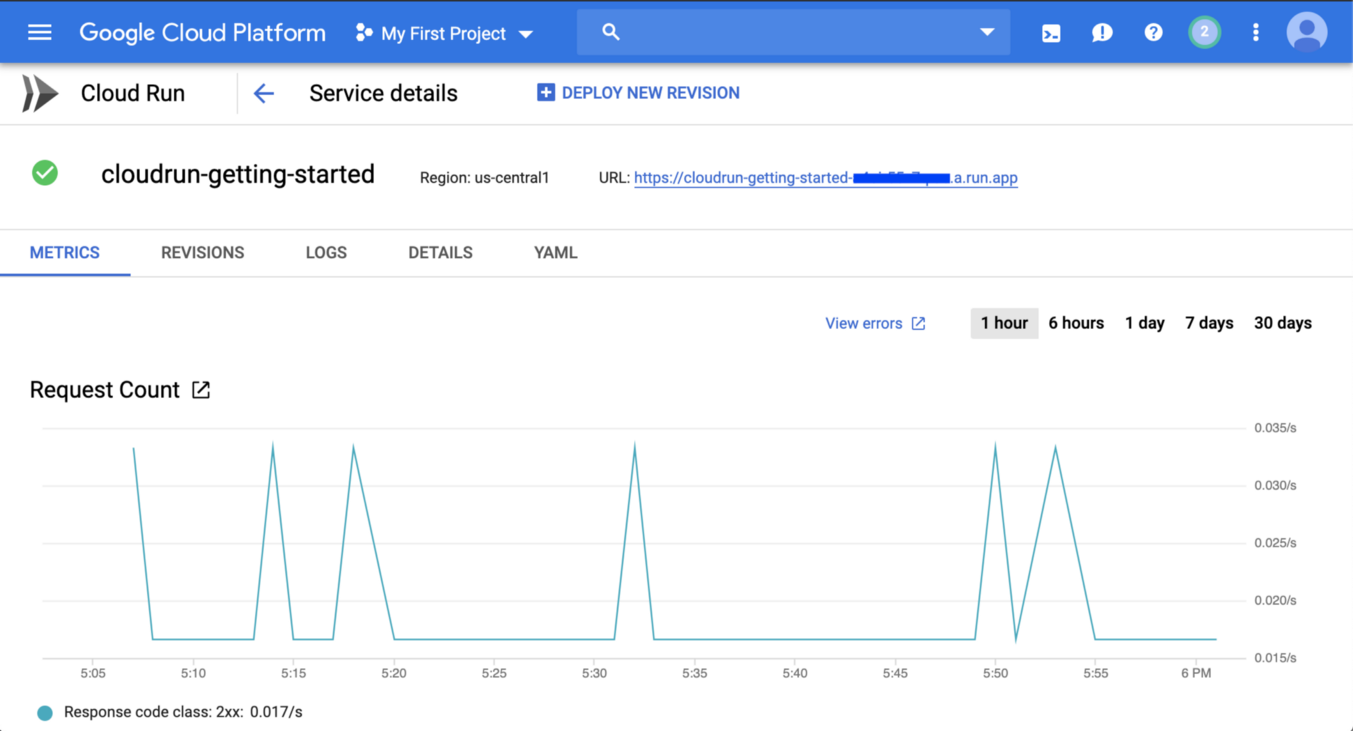
|  |
| --- |
|  |
|  | echo "create iam binding"  gcloud projects add-iam-policy-binding ${GCLOUD\_PROJECT} \ |
|  | --member=serviceAccount:service-${PROJECT\_NUMBER}@gcp-sa-pubsub.iam.gserviceaccount.com \ |
|  | --role=roles/iam.serviceAccountTokenCreator |
|  |  |
|  | echo "create service account" |
|  | gcloud iam service-accounts create ${INVOKER\_NAME} \ |
|  | --display-name "Cloud Run Pub/Sub Invoker" |
|  |  |
|  | echo "iam roles" |
|  | gcloud beta run services add-iam-policy-binding ${PROJECT\_NAME} \ |
|  | --member=serviceAccount:${INVOKER\_NAME}@${GCLOUD\_PROJECT}.iam.gserviceaccount.com \ |
|  | --role=roles/run.invoker |
|  |  |
|  | echo "create subscription" |
|  | gcloud beta pubsub subscriptions create ${SUBSCRIPTION\_NAME} \ |
|  | --topic ${TOPIC\_NAME} \ |
|  | --push-endpoint=${SERVICE\_URL} \ |
|  | --push-auth-service-account=${INVOKER\_NAME}@${GCLOUD\_PROJECT}.iam.gserviceaccount.com |

Create Subscription

As soon as this is completed the Pubsub messages will be routed to Cloud Run. Let’s go see our handy work.

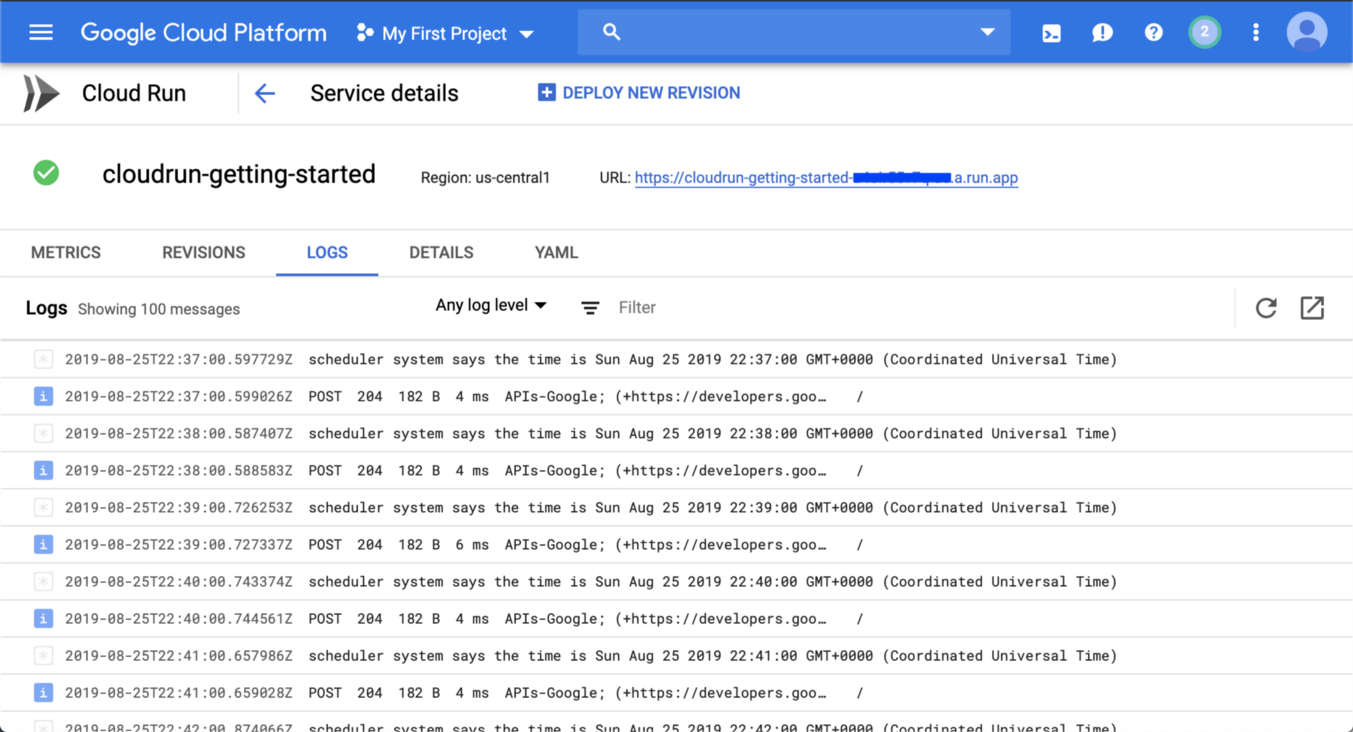
Watch The Logs Roll In

When we return to Cloud Run we can quickly see that something is already running.



Results coming through Cloud Run Metrics

And if we go to the logs for this Service we can see the data as it comes through.



Logs Showing Results

And we are officially done. Everything is running through into our service using Pubsub via Cloud Scheduler. It’s pretty slick to see if all running.

Conclusion

Cloud Run is an amazing service. If you have a Google Kubernetes Engine (GKE) instance running you can include Cloud Run as an Add-On or you can use the managed Cloud Run instance as we’ve used here. Either way, you will have an amazing platform to build the biggest applications.