Serverless Computing (https://cloud.google.com/products/serverless/)
Cloud Run: Serverless Computing (https://cloud.google.com/run/)
Documentation (https://cloud.google.com/run/docs/) Guides

Hosting webhooks targets

This guide shows how to host a webhook target in a Cloud Run service.

Cloud Functions vs Cloud Run

Cloud Functions (https://cloud.google.com/functions/docs/) and Cloud Run both provide good solutions for hosting your webhook targets. Generally, Cloud Functions is quick to set up, good for prototyping, and ideal for lower volume workflows. Cloud Run provides more flexibility and is able to handle larger volumes with concurrency.

Use Cloud Run if:

- You're using languages or runtimes not supported in Cloud Functions
- You want longer request timeouts (up to 15 minutes)
- You're expecting large volume and need concurrency (80 concurrent requests per container instance)

Creating a webhook target in Cloud Run

Using Cloud Run, you can define a webhook target in any language you choose. You only need to create an HTTP endpoint that can accept the data. Typically this is done with a `POST`, for example:

```
@app.route('/', methods=['POST'])
def index():
    data = request.get_json()
```

In the above example, the index page of the URL is configured to accept only `POST` requests and expects data to be delivered via a JSON payload.

Integrating with the webhook provider

Most services that provide HTTP callbacks require you to verify URL ownership. This is usually done by sending some kind of token, message, or secret and expecting a valid response. You'll need to obtain these requirements from the service provider. Using the same example above, this could look like:

```
def index():
    data = request.get_json()
    return data['challenge']
```

After the provider verifies your ownership, you'll need to add authorization on your end as well.

Authorizing requests

A webhook target is an open and public URL. Most services provide a token or a secret to ensure that the incoming requests are from authorized services. Because the URL is public, you cannot prevent malicious attempts to send data to the webhook target. However, using tokens or secrets ensures you only process data from authorized sources.

In order to verify the request, you need to store your copy of the secret either as an environment variable or using some kind of key management system. Each request should have a secret or token in the request headers or the JSON payload, and you must check it to ensure the source is valid.

```
def index():
    request_secret = request.headers['Secret']
    if request_secret != os.environ['SECRET']:
        return ('Unauthorized', 401)
```

If the webhook provider does not support a secret or other authentication mechanism, anyone with the URL of your webhook target will be able to send messages. In this case, your webhook implementation should be safe to expose to the public internet.

Note: If you're creating webhooks to send data between multiple Cloud Run instances or Cloud Functions, use the built in authentication (https://cloud.google.com/run/docs/triggering/authenticating/service-to-service) to protect your HTTP

Target. When prompted to allow unauthenticated requests (https://cloud.google.com/run/docs/triggering/authenticating/public), respond no.

Responding to requests

Most services require you to respond to a request within a set amount of time, as specified by the service. Some webhooks have built-in retry methods if there is an error response, such as an HTTP status code of 4xx or 5xx, so you'll need to return a successful status code (2xx) to let the service know the event was processed properly.

```
def index():
    data = request.get_json()
    return ('', 200)
```

Timeouts

Both Cloud Run and the webhooks provider have timeouts. The shorter of the two will apply to your application. If your data processing exceeds the time allotted by either Cloud Run or the webhooks provider, you'll need to use a product that allows you to complete your processing asynchronously, such as Pub/Sub (https://cloud.google.com/pubsub/docs/) or Cloud Tasks (https://cloud.google.com/tasks/). These products allow you to quickly hand off the data, immediately return a success response to the webhooks provider, and continue the processing without the timeout concern. These are also good options for handling failures and retries.

Common webhooks patterns

Type	Examples
Relaying Data	Sending a notification via Firebase Cloud Messaging whenever the webhook is called.
Storing Data	Storing the data in BigQuery for later analysis.
Triggering Actions	Fulfilling actions on Dialogflow, posting replies on Twitter, or pushing to your staging environment whenever new code is committed in GitHub.

What's next

- Learn more about webhooks (HTTP Triggers) on Cloud Functions (https://cloud.google.com/functions/docs/calling/)
- Set up webhooks notifications on Stackdriver (https://cloud.google.com/monitoring/support/notification-options#webhooks)
- Send Pub/Sub messages to a webhook using push subscriptions (https://cloud.google.com/pubsub/docs/push)
- Fulfill actions on Dialogflow (https://cloud.google.com/dialogflow/docs/fulfillment-overview) with webhooks

---

Last updated December 13, 2019.