# Tutorial: Local troubleshooting of a Cloud Run service

This tutorial shows how a service developer can troubleshoot a broken Cloud Run service using Stackdriver tools for discovery and a local development workflow for investigation.

This step-by-step "case study" companion to the troubleshooting guide (https://cloud.google.com/run/docs/troubleshooting) uses a sample project that results in runtime errors when deployed, which you troubleshoot to find and fix the problem.

You can use this tutorial with Cloud Run (fully managed) or Cloud Run for Anthos on Google Cloud. You cannot use this tutorial with Cloud Run for Anthos deployed on VMware due to Stackdriver support limitations (https://cloud.google.com/gke-on-prem/docs/concepts/logging-and-monitoring#logging_and_monitoring)
.

## Objectives

- Write, build, and deploy a service to Cloud Run
- Use Stackdriver Error Reporting and Stackdriver Logging to identify an error
- Retrieve the container image from Container Registry for a root cause analysis
- Fix the "production" service, then improve the service to mitigate future problems

## Costs

This tutorial uses billable components of Cloud Platform, including:

- Cloud Build (https://cloud.google.com/cloud-build/)
- Container Registry (https://cloud.google.com/container-registry/)
- Cloud Run or Cloud Run for Anthos on Google Cloud (https://cloud.google.com/run/)

- Stackdriver Logging (https://cloud.google.com/logging)
- Stackdriver Error Reporting (https://cloud.google.com/error-reporting)

Use the Pricing Calculator (https://cloud.google.com/products/calculator/) to generate a cost estimate based on your projected usage.

New Cloud Platform users might be eligible for a free trial (https://cloud.google.com/free/).

## Before you begin

1. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).

2. In the 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 THE PROJECT SELECTOR PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/PROJECTSELECT

3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (https://cloud.google.com/billing/docs/how-to/modify-project).

4. ENABLE THE CLOUD RUN API (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/APIS/LIBRARY/RUN.GOOG

5. Install and initialize (https://cloud.google.com/sdk/docs/) the Cloud SDK.

6. For Cloud Run for Anthos on Google Cloud install the gcloud kubectl component:

   ```
   gcloud components install kubectl
   ```

7. Update components:

   ```
   gcloud components update
   ```

8. If you are using Cloud Run for Anthos on Google Cloud, create a new cluster using the instructions in Setting up Cloud Run for Anthos on Google Cloud (https://cloud.google.com/run/docs/gke/setup).
9. If you are using Cloud Run for Anthos on Google Cloud, install curl (https://curl.haxx.se/dlwiz/?type=bin) to try out the service
10. Follow the instructions to install Docker locally (https://docs.docker.com/install/)

## Setting up gcloud defaults

To configure gcloud with defaults for your Cloud Run service:

1. Set your default project:

```
gcloud config set project PROJECT-ID
```

   Replace ***PROJECT-ID*** with the name of the project you created for this tutorial.

2. If you are using Cloud Run (fully managed), configure gcloud for your chosen region:

```
gcloud config set run/region REGION
```

   Replace ***REGION*** with the supported Cloud Run region (#follow-cloud-run) of your choice.

3. If you are using Cloud Run for Anthos on Google Cloud, configure gcloud for your cluster:

```
gcloud config set run/cluster CLUSTER-NAME
gcloud config set run/cluster_location REGION
```

   Replace

   - ***CLUSTER-NAME*** with the name you used for your cluster,
   - ***REGION*** with the supported cluster location of your choice.

## Assembling the code

Build a new Cloud Run greeter service step-by-step. As a reminder, this service creates a runtime error on purpose for the troubleshooting exercise.

1. Create a new project:

NODE.JS | PYTHON | GO | JAVA

Create a Node.js project by defining the service package, initial dependencies, and some common operations.

a. Create a new `hello-service` directory:

```
mkdir hello-service
cd hello-service
```

b. Create a new Node.js project by generating a `package.json` file:

```
npm init --yes
npm install --save express@4
```

c. Open the new `package.json` file in your editor and configure a `start` script to run `node index.js`. When you're done, the file will look like this:

```
{
  "name": "hello-service",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
      "start": "node index.js",
      "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
      "express": "^4.17.1"
  }
}
```

If you continue to evolve this service beyond the immediate tutorial, consider filling in the description, author, and evaluate the license. For more details, read the package.json documentation (https://docs.npmjs.com/creating-a-package-json-file).

2. Create an HTTP service to handle incoming requests:

NODE.JS PYTHON GO JAVA

run/hello-broken/index.js (https://github.com/GoogleCloudPlatform/nodejs-docs-samples/blob/master/run/hello-broken/index.js)

/GOOGLECLOUDPLATFORM/NODEJS-DOCS-SAMPLES/BLOB/MASTER/RUN/HELLO-BROKEN/INDEX.JS)

FEEDBACK (#)

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  console.log('hello: received request.');

  const {NAME} = process.env;
  if (!NAME) {
    // Plain error logs do not appear in Stackdriver Error Reporting.
    console.error('Environment validation failed.');
    console.error(new Error('Missing required server parameter'));
    return res.status(500).send('Internal Server Error');
  }
  res.send(`Hello ${NAME}!`);
});
const port = process.env.PORT || 8080;
app.listen(port, () => {
  console.log(`hello: listening on port ${port}`);
});
```

3. Create a `Dockerfile` to define the container image used to deploy the service:

NODE.JS PYTHON GO JAVA

run/hello-broken/Dockerfile (https://github.com/GoogleCloudPlatform/nodejs-docs-samples/blob/master/run/hello-broken/Dockerfile)

)OGLECLOUDPLATFORM/NODEJS-DOCS-SAMPLES/BLOB/MASTER/RUN/HELLO-BROKEN/DOCKERFILE)

FEEDBACK (#)

```
# Use the official lightweight Node.js 10 image.
# https://hub.docker.com/_/node
FROM node:10-slim

# Create and change to the app directory.
WORKDIR /usr/src/app

# Copy application dependency manifests to the container image.
# A wildcard is used to ensure copying both package.json AND package-lock.jso
# Copying this first prevents re-running npm install on every code change.
COPY package*.json ./

# Install production dependencies.
# If you add a package-lock.json, speed your build by switching to 'npm ci'.
# RUN npm ci --only=production
RUN npm install --only=production

# Copy local code to the container image.
COPY . ./

# Run the web service on container startup.
CMD [ "npm", "start" ]
```

## Shipping the code

Shipping code consists of three steps: building a container image with Cloud Build, uploading the container image to Container Registry, and deploying the container image to Cloud Run.

To ship your code:

1. Build your container and publish on Container Registry:

   ```
   gcloud builds submit --tag gcr.io/PROJECT-ID/hello-service
   ```

   Where ***PROJECT-ID*** is your GCP project ID. You can check your current project ID with `gcloud config get-value project`.

   Upon success, you should see a SUCCESS message containing the ID, creation time, and image name. The image is stored in Container Registry and can be re-used if desired.

2. Run the following command to deploy your app:

```
gcloud run deploy hello-service --image gcr.io/PROJECT-ID/hello-service
```

Replace ***PROJECT-ID*** with your GCP project ID. `hello-service` is both the container image name and name of the Cloud Run service. Notice that the container image is deployed to the service and region (Cloud Run) or cluster (Cloud Run for Anthos on Google Cloud) that you configured previously under Setting up gcloud (#setting-up-gcloud)

If deploying to Cloud Run (fully managed), respond `y`, "Yes", to the "allow unauthenticated" prompt. See Managing Access (https://cloud.google.com/run/docs/securing/managing-access) for more details on IAM-based authentication.

Wait until the deployment is complete: this can take about half a minute. On success, the command line displays the service URL.

## Trying it out

Try out the service to confirm you have successfully deployed it. Requests should fail with a HTTP 500 or 503 error (members of the class 5xx Server errors (https://en.wikipedia.org/wiki/List_of_HTTP_status_codes#5xx_Server_errors)). The tutorial walks through troubleshooting this error response.

- For Cloud Run (fully managed), the service is auto-assigned a navigable URL.

  Navigate to this URL with your web browser:

  1. Open a web browser
  2. Find the service URL output by the earlier deploy command.

     If the deploy command did not provide a URL then something went wrong. Review the error message and act accordingly: if no actionable guidance is present, review the troubleshooting guide (https://cloud.google.com/run/docs/troubleshooting) and possibly retry the deployment command.
  3. Navigate to this URL by copying it into your browser's address bar and pressing **ENTER**.
  4. See the HTTP 500 or HTTP 503 error.

  If you receive a HTTP 403 error, you may have rejected `allow unauthenticated invocations` at the deployment prompt. Grant unauthenticated access to the service to fix

this:

```
gcloud run services add-iam-policy-binding hello-service \
  --member="allUsers" \
  --role="roles/run.invoker"
```

For more information, read Allowing public (unauthenticated) access (https://cloud.google.com/run/docs/authenticating/public).

- For Cloud Run for Anthos on Google Cloud, without a custom domain (https://cloud.google.com/run/docs/gke/custom-domains) you are not provided a navigable URL for your service.

  Instead, use the provided URL and the IP address of the service's ingress gateway to create a `curl` command that can make requests to your service:

  1. To get the external IP for the Istio ingress gateway:

     ```
     kubectl get svc ISTIO-GATEWAY -n NAMESPACE
     ```

     Replace ***ISTIO-GATEWAY*** and ***NAMESPACE*** as follows:

     | Cluster version | ISTIO-GATEWAY | NAMESPACE |
     |---|---|---|
     | **1.15.3-gke.19** and greater<br>**1.14.3-gke.12** and greater<br>**1.13.10-gke.8** and greater | **istio-ingress** | **gke-system** |
     | All other versions | **istio-ingressgateway** | **istio-system** |

     where the resulting output looks something like this:

     ```
     NAME            TYPE           CLUSTER-IP     EXTERNAL-IP  PORT(S)
     ISTIO-GATEWAY    LoadBalancer   XX.XX.XXX.XX   pending      80:32380/TCP,44
     ```

     The ***EXTERNAL-IP*** for the Load Balancer is the IP address you must use.

  2. Run a `curl` command using this `GATEWAY_IP` address in the URL.

     ```
     curl -G -H "Host: SERVICE-DOMAIN" https://EXTERNAL-IP/
     ```

Replace ***SERVICE-DOMAIN*** with the default assigned domain of your service. You can obtain this by taking the default URL and removing the protocol `http://`.

3. See the HTTP 500 or HTTP 503 error.

If your cluster is configured with a routable default domain (https://cloud.google.com/run/docs/gke/default-domain), skip the steps above and instead copy the URL into your web browser.

## Investigating the problem

Visualize that the HTTP 5xx error encountered above in Trying it out (#trying_it_out) was encountered as a production runtime error. This tutorial walks through a formal process for handling it. Although production error resolution processes vary widely, this tutorial presents a particular sequence of steps to show the application of useful tools and techniques.

To investigate this problem you will work through these phases:

- Collect more details on the reported error to support further investigation and set a mitigation strategy.
- Relieve user impact by deciding to push forward in a fix or rollback to a known-healthy version.
- Reproduce the error to confirm the correct details have been gathered and that the error is not a one-time glitch
- Perform a root cause analysis on the bug to find the code, configuration, or process which created this error

At the start of the investigation you have a URL, timestamp, and the message "Internal Server Error".

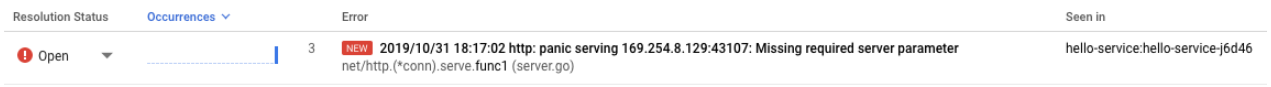
### Gathering further details

Gather more information about the problem to understand what happened and determine next steps.

Use available Stackdriver tools to collect more details:

1. Use the Stackdriver Error Reporting console, which provides a dashboard with details and recurrence tracking for errors with a recognized stack trace (https://en.wikipedia.org/wiki/Stack_trace).

   GO TO STACKDRIVER ERROR REPORTING CONSOLE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/ERR

   List of recorded errors. Errors are grouped by message across revisions, services, and platforms.

2. Click on the error to see the stack trace details, noting the function calls made just prior to the error.

   The "Stack trace sample" in the error details page shows a single instance of the error. You can review each individual instances.

3. Use Stackdriver Logging to review the sequence of operations leading to the problem, including error messages that are not included in the Stackdriver Error Reporting console because of a lack of a recognized error stack trace (https://cloud.google.com/error-reporting/docs/formatting-error-messages):

   GO TO STACKDRIVER LOGGING CONSOLE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/LOGGING)

   - If using Cloud Run (fully managed), select **Cloud Run Revision > hello-service** from the first drop-down box. This will filter the log entries to those generated by your service.
   - If using Cloud Run for Anthos on Google Cloud select **Kubernetes Container > hello-service** from the first drop-down box.

   Read more about viewing logs in Cloud Run (https://cloud.google.com/run/docs/logging#viewing_logs)

## Rollback to a healthy version

If this is an established service, known to work, there will be a previous revision of the service on Cloud Run. This tutorial uses a new service with no previous versions, so you cannot do a rollback.

However, if you have a service with previous versions you can roll back to, follow Viewing revision details (https://cloud.google.com/run/docs/managing/revisions#viewing_revision_details) to extract the container name and configuration details necessary to create a new working deployment of your service.

## Reproducing the error

Using the details (#gather-details) you obtained previously, confirm the problem consistently occurs under test conditions.

Send the same HTTP request by trying it out (#trying-it-out) again, and see if the same error and details are reported. It may take some time for error details to show up.

Because the sample service in this tutorial is read-only and doesn't trigger any complicating side effects, reproducing errors in production is safe. However, for many real services, this won't be the case: you may need to reproduce errors in a test environment or limit this step to local investigation.

Reproducing the error establishes the context for further work. For example, if developers cannot reproduce the error further investigation may require additional instrumentation of the service.

## Performing a root cause analysis

Root cause analysis is an important step in effective troubleshooting (https://landing.google.com/sre/sre-book/chapters/effective-troubleshooting/) to ensure you fix the problem instead of a symptom.

Previously in this tutorial, you reproduced the problem on Cloud Run which confirms the problem is active when the service is hosted on Cloud Run. Now reproduce the problem locally to determine if the problem is isolated to the code or if it only emerges in production hosting.

1. If you have not used Docker CLI locally with Container Registry, authenticate it with gcloud:

   ```
   gcloud auth configure-docker
   ```

   For alternative approaches see Container Registry authentication methods (https://cloud.google.com/container-registry/docs/advanced-authentication).

2. If the most recently used container image name is not available, the service description has the information of the most recently deployed container image:

   ```
   gcloud run services describe hello-service
   ```

   Find the container image name inside the `spec` object. A more targeted command can directly retrieve it:

   ```
   gcloud run services describe hello-service \
      --format="value(spec.template.spec.containers.image)"
   ```

   This command reveals a container image name such as `gcr.io/PROJECT-ID/hello-service`.

3. Pull the container image from the Container Registry to your environment, this step might take several minutes as it downloads the container image:

   ```
   docker pull gcr.io/PROJECT-ID/hello-service
   ```

   Later updates to the container image that reuse this name can be retrieved with the same command. If you skip this step, the `docker run` command below pulls a container image if one is not present on the local machine.

4. Run locally to confirm the problem is not unique to Cloud Run:

   ```
   PORT=8080 && docker run --rm -e PORT=$PORT -p 9000:$PORT \
      gcr.io/PROJECT-ID/hello-service
   ```

   Breaking down the elements of the command above,

   - The `PORT` environment variable is used by the service to determine the port to listen on inside the container.

- The `run` command starts the container, defaulting to the entrypoint command defined in the Dockerfile or a parent container image.
- The `--rm` flag deletes the container instance on exit.
- The `-e` flag assigns a value to an environment variable. `-e PORT=$PORT` is propagating the `PORT` variable from the local system into the container with the same variable name.
- The `-p` flag publishes the container as a service available on localhost at port 9000. Requests to localhost:9000 will be routed to the container on port 8080. This means output from the service about the port number in use will not match how the service is accessed.
- The final argument `gcr.io/PROJECT-ID/hello-service` is a container image `tag`, a human-readable label for a container image's sha256 hash identifier. If not available locally, docker attempts to retrieve the image from a remote registry.

In your browser, open http://localhost:9000 (http://localhost:9000). Check the terminal output for error messages that match those on Stackdriver.

If the problem is not reproducible locally, it may be unique to the Cloud Run environment. Review the Cloud Run troubleshooting guide (https://cloud.google.com/run/docs/troubleshooting) for specific areas to investigate.

In this case the error is reproduced locally.

Now that the error is doubly-confirmed as persistent and caused by the service code instead of the hosting platform, it's time to investigate the code more closely.

For purposes of this tutorial it is safe to assume the code inside the container and the code in the local system is identical.

Revisit the error report's stack trace and cross-reference with the code to find the specific lines at fault.

NODE.JS | PYTHON | GO | JAVA

Find the source of the error message in the file `index.js` around the line number called out in the stack trace shown in the logs:

run/hello-broken/index.js (https://github.com/GoogleCloudPlatform/nodejs-docs-samples/blob/master/run/hello-broken/index.js)

/GOOGLECLOUDPLATFORM/NODEJS-DOCS-SAMPLES/BLOB/MASTER/RUN/HELLO-BROKEN/INDEX.JS)

FEEDBACK (#)

```
const {NAME} = process.env;
if (!NAME) {
  // Plain error logs do not appear in Stackdriver Error Reporting.
  console.error('Environment validation failed.');
  console.error(new Error('Missing required server parameter'));
  return res.status(500).send('Internal Server Error');
}
```

Examining this code, the following actions are taken when the `NAME` environment variable is not set:

- An error is logged to Stackdriver
- An HTTP error response is sent

The problem is caused by a missing variable, but the root cause is more specific: the code change adding the hard dependency on an environment variable did not include related changes to deployment scripts and runtime requirements documentation.

# Fixing the root cause

Now that we have collected the code and identified the potential root cause, we can take steps to fix it.

- Check whether the service works locally with the `NAME` environment available in place:
    1. Run the container locally with the environment variable added:

```
PORT=8080 && docker run --rm -e PORT=$PORT -p 9000:$PORT \
 -e NAME="Local World!" \
 gcr.io/PROJECT-ID/hello-service
```

    2. Navigate your browser to http://localhost:9000 (http://localhost:9000)
    3. See "Hello Local World!" appear on the page
- Modify the running Cloud Run service environment to include this variable:
    1. Run the services update command to add an environment variable:

```
gcloud run services update hello-service \
  --set-env-vars NAME=Override
```

2. Wait a few seconds while Cloud Run creates a new revision based on the previous revision with the new environment variable added.

- Confirm the service is now fixed:
  1. Navigate your browser to the Cloud Run service URL.
  2. See "Hello Override!" appear on the page.
  3. Verify that no unexpected messages or errors appear in Stackdriver Logging or Stackdriver Error Reporting.

# Improving future troubleshooting speed

In this sample production problem, the error was related to operational configuration. There are code changes that will minimize the impact of this problem in the future.

- Improve the error log to include more specific details.
- Instead of returning an error, have the service fall back to a safe default. If using a default represents a change to normal functionality, use a warning message for monitoring purposes.

Let's step through removing the `NAME` environment variable as a hard dependency.

1. Remove the existing `NAME`-handling code:

NODE.JS PYTHON GO JAVA

run/hello-broken/index.js (https://github.com/GoogleCloudPlatform/nodejs-docs-samples/blob/master/run/hello-broken/index.js)

/GOOGLECLOUDPLATFORM/NODEJS-DOCS-SAMPLES/BLOB/MASTER/RUN/HELLO-BROKEN/INDEX.JS)

FEEDBACK (#)

```
const {NAME} = process.env;
if (!NAME) {
  // Plain error logs do not appear in Stackdriver Error Reporting.
  console.error('Environment validation failed.');
```

```
  console.error(new Error('Missing required server parameter'));
  return res.status(500).send('Internal Server Error');
}
```

2. Add new code that sets a fallback value:

NODE.JS PYTHON GO JAVA

run/hello-broken/index.js (https://github.com/GoogleCloudPlatform/nodejs-docs-samples/blob/master/run/hello-broken/index.js)

/GOOGLECLOUDPLATFORM/NODEJS-DOCS-SAMPLES/BLOB/MASTER/RUN/HELLO-BROKEN/INDEX.JS)

FEEDBACK (#)

```
const NAME = process.env.NAME || 'World';
if (!process.env.NAME) {
  console.log(
    JSON.stringify({
      severity: 'WARNING',
      message: `NAME not set, default to '${NAME}'`,
    })
  );
}
```

3. Test locally by re-building and running the container through the affected configuration cases:

```
docker build --tag gcr.io/PROJECT-ID/hello-service .
```

Confirm the `NAME` environment variable still works:

```
PORT=8080 && docker run --rm -e $PORT -p 9000:$PORT \
 -e NAME="Robust World" \
 gcr.io/PROJECT-ID/hello-service
```

Confirm the service works without the `NAME` variable:

```
PORT=8080 && docker run --rm -e $PORT -p 9000:$PORT \
 gcr.io/PROJECT-ID/hello-service
```

If the service does not return a result, confirm the removal of code in the first step did not remove extra lines, such as those used to write the response.

4. Deploy this by revisiting the Deploy your code (#shipping_the_code) section.

   Each deployment to a service creates a new revision and automatically starts serving traffic when ready.

   To clear the environment variables set earlier:

   ```
   gcloud run services update hello-service --clear-env-vars
   ```

Add the new functionality for the default value to automated test coverage for the service.

# Finding other issues in the logs

You may see other issues in the Log Viewer for this service. For example, an unsupported system call will appear in the logs as a "Container Sandbox Limitation".

For example, the Node.js services sometimes result in this log message:

```
Container Sandbox Limitation: Unsupported syscall statx(0xffffff9c,0x3e1ba8e86d88,0x
```

In this case, the lack of support does not impact the hello-service sample service.

## Cleaning up

If you created a new project for this tutorial, delete the project (#delete-project). If you used an existing project and wish to keep it without the changes added in this tutorial, delete resources created for the tutorial (#delete-resources).

### Deleting the project

The easiest way to eliminate billing is to delete the project that you created for the tutorial.

To delete the project:

**Caution**: Deleting a project has the following effects:

- **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.
- **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an `appspot.com` URL, delete selected resources inside the project instead of deleting the whole project.

If you plan to explore multiple tutorials and quickstarts, reusing projects can help you avoid exceeding project quota limits.

1. In the Cloud Console, go to the **Manage resources** page.

   GO TO THE MANAGE RESOURCES PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/IAM-ADMIN/PRC

2. In the project list, select the project you want to delete and click **Delete** .
3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

## Deleting tutorial resources

1. Delete the Cloud Run service you deployed in this tutorial:

   ```
   gcloud run services delete SERVICE-NAME
   ```

   Where ***SERVICE-NAME*** is your chosen service name.

   You can also delete Cloud Run services from the Google Cloud Console (https://console.cloud.google.com/run).

2. Remove the gcloud default configurations you added during tutorial setup.

   If you use Cloud Run (fully managed), remove the region setting:

   ```
   gcloud config unset run/region
   ```

   If you use Cloud Run for Anthos on Google Cloud, remove the cluster configuration:

   ```
   gcloud config unset run/cluster run/cluster
   gcloud config unset run/cluster run/cluster_location
   ```

3. Remove the project configuration:

```
gcloud config unset project
```

4. Delete other Google Cloud resources created in this tutorial:
    - Delete the container image (https://cloud.google.com/container-registry/docs/managing#deleting_images) named `gcr.io/<var>PROJECT-ID</var>/hello-service` from Container Registry.

## What's next

- Learn more about how to use Stackdriver Logging (https://cloud.google.com/logging/docs) and Stackdriver Error Reporting (https://cloud.google.com/error-reporting/docs) to gain insight into production behavior.
- For more information about Cloud Run troubleshooting, see the troubleshooting guide (https://cloud.google.com/run/docs/troubleshooting#sandbox).
- Try out other Google Cloud features for yourself. Have a look at our tutorials (https://cloud.google.com/docs/tutorials).

---



*Last updated January 7, 2020.*