Testing the Container Image Locally

During development, you can run and test your container image locally, prior to deploying. You can use Cloud Build or Docker installed locally (https://docs.docker.com/install/) to run and test locally, including running locally with access to Google Cloud services.

Running locally using Docker

To test your container image locally using Docker:

1. Use the Docker command:

```
PORT=8080 && docker run -p 9090:${PORT} -e PORT=${PORT} gcr.io/[PROJECT_ID]/[I
```

   Replace `[PROJECT-ID]` with your Google Cloud project ID and replace `[IMAGE]` with the name of your image.

   The `PORT` environment variable specifies the port your application will use to listen for HTTP or HTTPS requests. This is a requirement from the Container Runtime Contract (https://cloud.google.com/run/docs/reference/container-contract). In this example, we use port 8080.

2. Open http://localhost:9090 (http://localhost:9090) in your browser.

Refer to the Docker documentation (https://docs.docker.com/engine/reference/run/) to learn more about the Docker commands.

If you are new to working with containers, you may want to review the Docker Getting Started (https://docs.docker.com/get-started/) guide.

Running locally using Docker with access to Google Cloud services

If you are using Google Cloud client libraries to integrate your application with GCP services, and have not yet secured those services to control external access, you can set up your local

container to authenticate with Google Cloud services using Application Default Credentials (https://cloud.google.com/docs/authentication/production#providing_credentials_to_your_application).

1. Refer to Getting Started with Authentication (https://cloud.google.com/docs/authentication/getting-started) for instructions on generating, retrieving, and configuring your Service Account credentials.
2. Use the following Docker run flags to inject the credentials and configuration from your local system into the local container:
    a. Use the `--volume` (`-v`) flag to inject the credential file into the container (assumes you have already set your `GOOGLE_APPLICATION_CREDENTIALS` environment variable on your machine): `-v $GOOGLE_APPLICATION_CREDENTIALS:/tmp/keys/[FILE_NAME].json:ro`
    b. Use the `--environment` (`-e`) flag to set the `GOOGLE_APPLICATION_CREDENTIALS` variable inside the container: `-e GOOGLE_APPLICATION_CREDENTIALS=/tmp/keys/[FILE_NAME].json`

The following is a sample fully configured Docker run command:

```
PORT=8080 && docker run \
  -p 9090:${PORT} \
  -e PORT=${PORT} \
  -e K_SERVICE=dev \
  -e K_CONFIGURATION=dev \
  -e K_REVISION=dev-00001 \
  -e GOOGLE_APPLICATION_CREDENTIALS=/tmp/keys/[FILE_NAME].json \
  -v $GOOGLE_APPLICATION_CREDENTIALS:/tmp/keys/[FILE_NAME].json:ro \
  gcr.io/[PROJECT_ID]/[IMAGE]
```

Note that the path `/tmp/keys/[FILE_NAME].json` shown in the example above is a reasonable location to place your credentials inside the container.

However, other directory locations will also work. The crucial requirement is that the `GOOGLE_APPLICATION_CREDENTIALS` environment variable must match the bind mount location inside the container.

Note also, that with some Google Cloud services, you may want to use an alternate configuration to isolate local troubleshooting from production performance and data.

What's next

To learn how to deploy your built containers, follow Deploying Services (https://cloud.google.com/run/docs/deploying).

Last updated December 12, 2019.