Getting Started with API Gateway: Challenge Lab

Task 1. Create a Cloud Run function

Note: Cloud Run functions (2nd gen) depend on the Cloud Run Admin APIs. The Cloud Run Admin APIs have been enabled for you at the start of this lab. It may however take a few minutes for all of the enabled services to propagate. If you experience an issue when deploying your Cloud Run function, wait a few minutes then try again.

Create a new Cloud Run function (`2nd gen`) called `gcfunction` in the `us-central1` region using `Node.js 22` and allowing unauthenticated invocations. For now, simply have the function return "Hello World!" when invoked.

```
export REGION=


gcloud auth list


gcloud services enable apigateway.googleapis.com



mkdir lab

cd lab



cat > index.js <<EOF

/**

 * Responds to any HTTP request.
```

```
 *
 * @param {!express:Request} req HTTP request context.
 * @param {!express:Response} res HTTP response context.
 */
exports.helloWorld = (req, res) => {
  let message = req.query.message || req.body.message || 'Hello World!';
  res.status(200).send(message);
};

EOF




cat > package.json <<EOF
{
  "name": "sample-http",
  "version": "0.0.1"
}

EOF

sleep 40

export PROJECT_NUMBER=$(gcloud projects describe $DEVSHELL_PROJECT_ID --
format="json(projectNumber)" --quiet | jq -r '.projectNumber')

# Set the service account email
SERVICE_ACCOUNT="service-$PROJECT_NUMBER@gcf-admin-robot.iam.gserviceaccount.com"

# Get the current IAM policy
IAM_POLICY=$(gcloud projects get-iam-policy $DEVSHELL_PROJECT_ID --format=json)

# Check if the binding exists
```

```
if [[ "$IAM_POLICY" == *"$SERVICE_ACCOUNT"* && "$IAM_POLICY" ==
*"roles/artifactregistry.reader"* ]]; then

  echo "IAM binding exists for service account: $SERVICE_ACCOUNT with role
roles/artifactregistry.reader"

else

  echo "IAM binding does not exist for service account: $SERVICE_ACCOUNT with role
roles/artifactregistry.reader"


  # Create the IAM binding

  gcloud projects add-iam-policy-binding $DEVSHELL_PROJECT_ID \

    --member=serviceAccount:$SERVICE_ACCOUNT \

    --role=roles/artifactregistry.reader


  echo "IAM binding created for service account: $SERVICE_ACCOUNT with role
roles/artifactregistry.reader"

fi




gcloud functions deploy GCFunction \

  --runtime=nodejs20 \

  --trigger-http \

  --gen2 \

  --allow-unauthenticated \

  --entry-point=helloWorld \

  --region=$REGION \

  --max-instances 5 \

  --source=./
```

Task 2. Create an API Gateway

Once the Cloud Run function is deployed, configure an API Gateway to proxy requests to the backend.

Create a file named `openapispec.yaml` (using the code below), which references the Cloud Run function deployed in Task 1.

Use `openapispec.yaml` when deploying the API Gateway with the following properties:

Name	Value
Display Name	gcfunction API (wherever requested)
API ID	gcfunction-api
Select a service account	Compute Engine default service account
Location	`us-central1`
Config Name	gcfunction-api

```
swagger: '2.0'
info:
  title: gcfunction API
  description: Sample API on API Gateway with a Google Cloud Run
functions backend
  version: 1.0.0
schemes:
- https
produces:
- application/json
x-google-backend:
  address: https://gcfunction-383419317379.us-central1.run.app
paths:
  /gcfunction:
    get:
      summary: gcfunction
      operationId: gcfunction
```

```
      responses:
       '200':
          description: A successful response
          schema:
            type: string
```

Note: It will take several minutes (~10 minutes) for the Create Gateway operation to complete. To check the status of the creation and deployment process, you can click the Notifications icon (bell icon) in the top main navigation bar to display a status notification. Please ensure that the icon status for **Creating gateway "gcfunction API"** has a green check next to it before proceeding.

```
cat > openapispec.yaml <<EOF
swagger: '2.0'
info:
 title: GCFunction API
 description: Sample API on API Gateway with a Google Cloud Functions backend
 version: 1.0.0
schemes:
 - https
produces:
 - application/json
paths:
 /GCFunction:
   get:
    summary: gcfunction
    operationId: gcfunction
    x-google-backend:
     address: https://$REGION-$DEVSHELL_PROJECT_ID.cloudfunctions.net/GCFunction
    responses:
     '200':
       description: A successful response
       schema:
        type: string

EOF
```

```
PROJECT_NUMBER=$(gcloud projects describe $DEVSHELL_PROJECT_ID --
format="value(projectNumber)")



export API_ID="gcfunction-api-$(cat /dev/urandom | tr -dc 'a-z' | fold -w ${1:-8} | head -n 1)"



gcloud api-gateway apis create $API_ID --project=$DEVSHELL_PROJECT_ID


gcloud api-gateway api-configs create gcfunction-api \

  --api=$API_ID --openapi-spec=openapispec.yaml \

  --project=$DEVSHELL_PROJECT_ID --backend-auth-service-account=$PROJECT_NUMBER-
compute@developer.gserviceaccount.com



  gcloud api-gateway gateways create gcfunction-api \

  --api=$API_ID --api-config=gcfunction-api \

  --location=$REGION --project=$DEVSHELL_PROJECT_ID




# Set the service account email

SERVICE_ACCOUNT="service-$PROJECT_NUMBER@gcf-admin-robot.iam.gserviceaccount.com"


# Get the current IAM policy

IAM_POLICY=$(gcloud projects get-iam-policy $DEVSHELL_PROJECT_ID --format=json)


# Check if the binding exists

if [[ "$IAM_POLICY" == *"$SERVICE_ACCOUNT"* && "$IAM_POLICY" ==
*"roles/artifactregistry.reader"* ]]; then

  echo "IAM binding exists for service account: $SERVICE_ACCOUNT with role
roles/artifactregistry.reader"
```

```
else
  echo "IAM binding does not exist for service account: $SERVICE_ACCOUNT with role roles/artifactregistry.reader"

  # Create the IAM binding
  gcloud projects add-iam-policy-binding $DEVSHELL_PROJECT_ID \
    --member=serviceAccount:$SERVICE_ACCOUNT \
    --role=roles/artifactregistry.reader

  echo "IAM binding created for service account: $SERVICE_ACCOUNT with role roles/artifactregistry.reader"
fi



gcloud functions deploy GCFunction \
  --runtime=nodejs20 \
  --trigger-http \
  --gen2 \
  --allow-unauthenticated \
  --entry-point=helloWorld \
  --region=$REGION \
  --max-instances 5 \
  --source=./
```

Task 3. Create a Pub/Sub Topic and Publish Messages via API Backend

The development team would like the API backend to publish messages to a new Pub/Sub topic named `demo-topic`.

Create a new Pub/Sub topic (`demo-topic`) and push messages to it in the Cloud Run function deployed earlier. Be sure to keep the option to create a default subscription enabled when creating the topic.

Use the snippet below to update the `package.json` file and `index.js` code in the Cloud Run function deployed in Task 1.

package.json

```
{
  "dependencies": {
    "@google-cloud/functions-framework": "^3.0.0",
    "@google-cloud/pubsub": "^3.4.1"
  }
}
```

Copied!

content_copy

index.js

```
/**
 * Responds to any HTTP request.
 *
 * @param {!express:Request} req HTTP request context.
 * @param {!express:Response} res HTTP response context.
 */
const {PubSub} = require('@google-cloud/pubsub');
const pubsub = new PubSub();
const topic = pubsub.topic('demo-topic');
const functions = require('@google-cloud/functions-framework');

exports.helloHttp = functions.http('helloHttp', (req, res) => {

  // Send a message to the topic
  topic.publishMessage({data: Buffer.from('Hello from Cloud Run
functions!')});
  res.status(200).send("Message sent to Topic demo-topic!");
});
```

Copied!

content_copy

Redeploy the Cloud Run function once the `index.js` and `package.json` files have been updated.

Next, invoke the Cloud Run function via API Gateway. If done correctly, a message will be published to the topic `demo-topic` you've created in this task.

Note: It will take several minutes (~5 minutes) for the messages published to appear in the Messages section of the subscription after invoking the API Gateway endpoint.

```
gcloud pubsub topics create demo-topic


cat > index.js <<EOF
/**
 * Responds to any HTTP request.
 *
 * @param {!express:Request} req HTTP request context.
 * @param {!express:Response} res HTTP response context.
 */
const {PubSub} = require('@google-cloud/pubsub');
const pubsub = new PubSub();
const topic = pubsub.topic('demo-topic');
exports.helloWorld = (req, res) => {

  // Send a message to the topic
  topic.publishMessage({data: Buffer.from('Hello from Cloud Functions!')});
  res.status(200).send("Message sent to Topic demo-topic!");
};
EOF




cat > package.json <<EOF
{
  "name": "sample-http",
  "version": "0.0.1",
  "dependencies": {
    "@google-cloud/pubsub": "^3.4.1"
```

```
  }
}
EOF



gcloud functions deploy GCFunction \
  --runtime=nodejs20 \
  --trigger-http \
  --gen2 \
  --allow-unauthenticated \
  --entry-point=helloWorld \
  --region=$REGION \
  --max-instances 5 \
  --source=./
```