

**GCP – HOL -Session 10**

# **Slack HOL session - Slash Commands**

This HOL session demonstrates using Cloud Functions to implement a Slack Slash Command that searches the Google Knowledge Graph API.

## Objectives

* Create a Slash Command in Slack.
* Write and deploy an HTTP Cloud Function.
* Search the Google Knowledge Graph API using the Slash Command.

## Costs

This HOL session uses billable components of Cloud Platform, including:

* Google Cloud Functions

## Before you begin

1. Select or create a GCP project.

**Note**: If you don't plan to keep the resources you create in this procedure, create a new project instead of selecting an existing project. After you finish following these steps, you can delete the project, removing all resources associated with the project.

1. Make sure that billing is enabled for your Google Cloud Platform project.
2. Enable the Cloud Functions and Google Knowledge Graph Search APIs.
3. Update gcloud components:

gcloud components update

1. Prepare your development environment.

## Visualizing the flow of data

The flow of data in the Slack Slash Command HOL session application involves several steps:

1. The user executes the /kg <search\_query> Slash Command in a Slack channel.
2. Slack sends the command payload to the Cloud Function's trigger endpoint.
3. The Cloud Function sends a request with the user's search query to the Knowledge Graph API.
4. The Knowledge Graph API responds with any matching results.
5. The Cloud Function formats the response into a Slack message.
6. The Cloud Function sends the message back to Slack.
7. The user sees the formatted response in the Slack channel.

It may help to visualize the steps:

**Preparing the function**

1. Clone the sample app repository to your local machine:

git clone https://github.com/GoogleCloudPlatform/nodejs-docs-samples.git

Alternatively, you can [download the sample](https://github.com/GoogleCloudPlatform/nodejs-docs-samples/archive/master.zip)as a zip file and extract it.

1. Change to the directory that contains the Cloud Functions sample code:

cd nodejs-docs-samples/functions/slack/

1. Configure the app:

Using the config.default.json file as a template, create a config.json file in the app directory with the following contents:

{

"SLACK\_TOKEN": "**YOUR\_SLACK\_TOKEN**",

"KG\_API\_KEY": "**YOUR\_KG\_API\_KEY**",

}

* + Replace **YOUR\_SLACK\_TOKEN** with the verification token provided by Slack in the **Basic information** page of your app configuration.
  + Replace **YOUR\_KG\_API\_KEY** with the Knowledge Graph API Key you just created.

## Deploying the function

To deploy the function that is executed when you (or Slack) make an HTTP POST request to the function's endpoint, run the following command in the directory that contains the Cloud Functions sample code:

gcloud functions deploy kgSearch --runtime nodejs8 --trigger-http

## Configuring the application

After the function is deployed, you need to create a Slack Slash Command that sends the query to your Cloud Function every time the command is triggered:

1. Create a [Slack App](https://api.slack.com/apps) to host your Slack Slash Command. Associate it with a Slack team where you have permissions to install integrations.
2. Go to **Slash commands** and click the **Create new command** button.
3. Enter /kg as the name of the command.
4. Enter the URL for the command:

https://***YOUR\_REGION***-***YOUR\_PROJECT\_ID***.cloudfunctions.net/kgSearch

where **YOUR\_REGION** is the region where your Cloud Function is deployed and **YOUR\_PROJECT\_ID** is your Cloud project ID.

Both values are visible in your terminal when your function finishes deploying.

1. Click **Save**.
2. Go to **Basic Information**.
3. Click **Install your app to your workspace** and follow the instructions on screen to enable the application for your workspace.

Your Slack Slash Command should come online shortly.

## Understanding the code

### **Importing dependencies**

The application must import several dependencies in order to communicate with Google Cloud Platform services:

[functions/slack/index.js](https://github.com/GoogleCloudPlatform/nodejs-docs-samples/blob/master/functions/slack/index.js)

const config = require('./config.json');  
const {google} = require('googleapis');  
  
// Get a reference to the Knowledge Graph Search component  
const kgsearch = google.kgsearch('v1');

### **Receiving the webhook**

The following function is executed when you (or Slack) make an HTTP POST request to the function's endpoint:

[functions/slack/index.js](https://github.com/GoogleCloudPlatform/nodejs-docs-samples/blob/master/functions/slack/index.js)

/\*\*  
 \* Receive a Slash Command request from Slack.  
 \*  
 \* Trigger this function by making a POST request with a payload to:  
 \* https://[YOUR\_REGION].[YOUR\_PROJECT\_ID].cloudfunctions.net/kgsearch  
 \*  
 \* @example  
 \* curl -X POST "https://us-central1.your-project-id.cloudfunctions.net/kgSearch" --data '{"token":"[YOUR\_SLACK\_TOKEN]","text":"giraffe"}'  
 \*  
 \* @param {object} req Cloud Function request object.  
 \* @param {object} req.body The request payload.  
 \* @param {string} req.body.token Slack's verification token.  
 \* @param {string} req.body.text The user's search query.  
 \* @param {object} res Cloud Function response object.  
 \*/  
exports.kgSearch = (req, res) => {  
  return Promise.resolve()  
    .then(() => {  
      if (req.method !== 'POST') {  
        const error = new Error('Only POST requests are accepted');  
        error.code = 405;  
        throw error;  
      }  
  
      // Verify that this request came from Slack  
      verifyWebhook(req.body);  
  
      // Make the request to the Knowledge Graph Search API  
      return makeSearchRequest(req.body.text);  
    })  
    .then(response => {  
      // Send the formatted message back to Slack  
      res.json(response);  
    })  
    .catch(err => {  
      console.error(err);  
      res.status(err.code || 500).send(err);  
      return Promise.reject(err);  
    });  
};

The following function authenticates the incoming request by checking for the verification token generated by Slack:

[functions/slack/index.js](https://github.com/GoogleCloudPlatform/nodejs-docs-samples/blob/master/functions/slack/index.js)

/\*\*  
 \* Verify that the webhook request came from Slack.  
 \*  
 \* @param {object} body The body of the request.  
 \* @param {string} body.token The Slack token to be verified.  
 \*/  
function verifyWebhook(body) {  
  if (!body || body.token !== config.SLACK\_TOKEN) {  
    const error = new Error('Invalid credentials');  
    error.code = 401;  
    throw error;  
  }  
}

### **Querying the Knowledge Graph API**

The following function sends a request with the user's search query to the Knowledge Graph API:

[functions/slack/index.js](https://github.com/GoogleCloudPlatform/nodejs-docs-samples/blob/master/functions/slack/index.js)

/\*\*  
 \* Send the user's search query to the Knowledge Graph API.  
 \*  
 \* @param {string} query The user's search query.  
 \*/  
function makeSearchRequest(query) {  
  return new Promise((resolve, reject) => {  
    kgsearch.entities.search(  
      {  
        auth: config.KG\_API\_KEY,  
        query: query,  
        limit: 1,  
      },  
      (err, response) => {  
        console.log(err);  
        if (err) {  
          reject(err);  
          return;  
        }  
  
        // Return a formatted message  
        resolve(formatSlackMessage(query, response));  
      }  
    );  
  });  
}

### **Formatting the Slack message**

Finally, the following function formats the Knowledge Graph result into a richly formatted Slack message that will be displayed to the user:

[functions/slack/index.js](https://github.com/GoogleCloudPlatform/nodejs-docs-samples/blob/master/functions/slack/index.js)

/\*\*  
 \* Format the Knowledge Graph API response into a richly formatted Slack message.  
 \*  
 \* @param {string} query The user's search query.  
 \* @param {object} response The response from the Knowledge Graph API.  
 \* @returns {object} The formatted message.  
 \*/  
function formatSlackMessage(query, response) {  
  let entity;  
  
  // Extract the first entity from the result list, if any  
  if (  
    response &&  
    response.data &&  
    response.data.itemListElement &&  
    response.data.itemListElement.length > 0  
  ) {  
    entity = response.data.itemListElement[0].result;  
  }  
  
  // Prepare a rich Slack message  
  // See https://api.slack.com/docs/message-formatting  
  const slackMessage = {  
    response\_type: 'in\_channel',  
    text: `Query: ${query}`,  
    attachments: [],  
  };  
  
  if (entity) {  
    const attachment = {  
      color: '#3367d6',  
    };  
    if (entity.name) {  
      attachment.title = entity.name;  
      if (entity.description) {  
        attachment.title = `${attachment.title}: ${entity.description}`;  
      }  
    }  
    if (entity.detailedDescription) {  
      if (entity.detailedDescription.url) {  
        attachment.title\_link = entity.detailedDescription.url;  
      }  
      if (entity.detailedDescription.articleBody) {  
        attachment.text = entity.detailedDescription.articleBody;  
      }  
    }  
    if (entity.image && entity.image.contentUrl) {  
      attachment.image\_url = entity.image.contentUrl;  
    }  
    slackMessage.attachments.push(attachment);  
  } else {  
    slackMessage.attachments.push({  
      text: 'No results match your query...',  
    });  
  }  
  
  return slackMessage;  
}

### **Slack API timeouts**

The Slack API expects your function to respond within 3 seconds of receiving a webhook request.

The commands in this HOL session typically take less than 3 seconds to respond. For longer-running commands, we recommend configuring a function to push requests (including their [response\_url](https://api.slack.com/slash-commands" \l "responding_response_url)) to a Pub/Sub topic that acts as a task queue.

Then, you can create a second function triggered by Pub/Sub that processes those tasks and sends results back to Slack's response\_url.

## Using the Slash command

1. Test the command manually:

curl -X POST "https://***YOUR\_REGION***-***YOUR\_PROJECT\_ID***.cloudfunctions.net/kgSearch" -H "Content-Type: application/json" --data '{"token":"***YOUR\_SLACK\_TOKEN***","text":"giraffe"}'

where

* + **YOUR\_REGION** is the region where your function is deployed. This is visible in your terminal when your function finishes deploying.
  + **YOUR\_PROJECT\_ID** is your Cloud project ID. This is visible in your terminal when your function finishes deploying.
  + **YOUR\_SLACK\_TOKEN** is the token provided by Slack in your Slash Command configuration.

1. Watch the logs to be sure the executions have completed:

gcloud functions logs read --limit 100

1. Type the command into your Slack channel:

/kg giraffe

## Cleaning up

To avoid incurring charges to your Google Cloud Platform account for the resources used in this HOL session: