App Dev - Harnessing Cloud Trace and Cloud Monitoring: Node.js

Overview

In this lab, you leverage Cloud Trace and Cloud Monitoring to diagnose and fix a performance issue in the running application and use Cloud Monitoring to view performance metrics of the application.

Objectives

In this lab, you learn how to perform the following tasks:

- Enable, install, and configure Cloud Trace.
- View Trace information to diagnose a performance issue.
- Fix the performance issue and verify the performance improvement.
- Monitor Google Cloud Platform products using Cloud Monitoring.

Task 1. Preparing the case study application

In this section, you access Cloud Shell, clone the git repository containing the quiz application, configure environment variables. and run the application.

Clone source code in Cloud Shell

1. To clone the repository for the class, execute the following command:

```
git clone --depth=1
https://github.com/GoogleCloudPlatform/training-data-analyst
```

2. Create a soft link as a shortcut to the working directory:

```
ln -s
~/training-data-analyst/courses/developingapps/v1.3/nodejs/stackdriver-tra
ce-monitoring ~/stackdriver-trace-monitoring
```

Configure and run the case study application

1. Navigate to the directory that contains the sample files for this lab:

```
cd ~/stackdriver-trace-monitoring/start
```

2. To configure the Quiz application, execute the following command:

```
. prepare_environment.sh
```

If prompted, enter `Y` to all unauthenticated invocations.

Note: This script file:

- Creates an App Engine application.
- Exports environment variables `GCLOUD_PROJECT` and `GCLOUD_BUCKET.`
- Runs `npm install.`
- Creates entities in Cloud Datastore.
- Creates a Cloud Spanner instance, database, and tables.
- Creates two Google Cloud Pub/Sub topics.
- Creates two Cloud Functions.
- Prints out the Google Cloud Platform Project ID.

3. To complete the preparation of the **Cloud Shell** environment to run the application, execute the following commands:

```
gcloud iam service-accounts create quiz-account --display-name "Quiz Account"

gcloud iam service-accounts keys create key.json --iam-account=quiz-account@$DEVSHELL_PROJECT_ID.iam.gserviceaccount.com

export GOOGLE_APPLICATION_CREDENTIALS=key.json

gcloud projects get-iam-policy $DEVSHELL_PROJECT_ID --format json > iam.json

node setup/add_iam_policy_to_service_account.js

gcloud projects set-iam-policy $DEVSHELL_PROJECT_ID iam_modified.json
```

Note: These commands:

- Create a service account.
- Create and download the key file for the service account.
- Export an environment variable `GOOGLE_APPLICATION_CREDENTIALS` referencing the key file.
- Create the necessary IAM permissions to enable the service account to access all required APIs.

In production on Compute Engine or Container Engine, you would grant access to APIs either using a custom service account or via scopes. Typically, you would allow GCP to manage key rotation.

Task 2. Harnessing Cloud Trace

In this section, you enable the Trace API, write the code to create and start the Cloud Trace in the Quiz application, and view application timings across multiple products in the Cloud Console.

Enable the Cloud Trace API

1. In the **Cloud Console**, on the **Navigation menu**, click **APIs & services**.
2. Click **Library**.
3. For **Search for APIs & services**, type **Google Trace**.
4. Click **Cloud Trace API**.
5. If the API is not enabled, click **Enable**.

Write code to set up Cloud Trace

1. Return to **Cloud Shell**.
2. To install the Node.js agent for Cloud Trace, execute the following command:

```
npm install --save @google-cloud/trace-agent
```

3. In **Cloud Shell**, click **Open Editor** to open in a new window.

4. Navigate to **stackdriver-trace-monitoring/start**.
5. Open **...frontend/app.js**.

Note: Update code within the sections marked as follows:

```
// TODO
```

```
// END TODO
```

To maximize your learning, review the code, inline comments, and related API documentation.

6. At the top of the `app.js` file, load the `'@google-cloud/trace-agent'` module and then `start()` it:

frontend/app.js

```
// TODO: Load the trace-agent and start it

// Trace must be started before any other code in the

// application.

require('@google-cloud/trace-agent').start({

  projectId: config.get('GCLOUD_PROJECT')

});

// END TODO
```

7. Save the file.

Run the web application and view trace data

1. To start the application, execute the following command:

```
npm start
```

2. In **Cloud Shell**, click **Web preview** > **Preview on port 8080** to preview the quiz application.
3. Return to the **Cloud Platform Console**.
4. On the **Navigation menu**, click **Trace**.

Note: The Trace overview page opens. However, it may take a few minutes for the first request to populate. After this happens, trace data is visible with just a few seconds' latency.

5. While waiting for the first Trace to populate, return to the **Cloud Shell** window and look for error messages.

Note: Sometimes, due to a timing issue after enabling APIs, you might see errors about the Cloud Trace and/or Debug agent.

If you see errors, stop the Quiz application, restart after a minute, and then refresh the Quiz application home page.

6. Return to **Trace** in the **Cloud Platform Console**.
7. After your first Trace has populated, click **Trace List**.

Note: You should see the Latency, HTTP Method, and URL for the request.

8. Return to the quiz application and click **Create Question**.
9. Fill in the form using the following values:

Form Field	Value
Author	`Your Name`
Quiz	`Google Cloud Platform`
Title	`Which Google Cloud product allows you to see request timing?`
Answer 1	`Debugger`
Answer 2	`Error Reporting`
Answer 3	`Logging`
Answer 4	**`Trace`** (select the Answer 4 radio button)

10. Click **Save**.
11. Return to the **Cloud Trace** window in the **Cloud Platform Console** and enable **Auto Reload**.

Note: Look at the Trace list.

Eventually, two new requests are displayed:

1. GET /questions/add
2. POST /questions/add
12. Click on the POST request against the **/questions/add** URL.

Note: You should see the total time to handle the request and the call made against the Cloud Datastore.

13. Return to the quiz application, and click **Take Test**.
14. Click **People**.
15. Complete the test, and submit feedback.
16. Return to the **Cloud Trace** window in the **Cloud Console**.

Note: Look at the Trace list again. Eventually a set of new requests is displayed.

Find the following requests and click on them:

1. POST /api/quizzes/people
2. POST /api/quizzes/feedback/people

These represent the calls made from the client-side application against the quiz application.

Once again, you should see the total time taken to complete the request.

Notice that the calls are sequential. This is because the Cloud Pub/Sub request needs the correct answers from Question entities in Cloud Datastore to send answer data to Cloud Pub/Sub for Storage in Cloud Spanner.

17. In the quiz application, click **Quite Interesting Quiz** in the toolbar, and then click **Leaderboard**.
18. Return to the **Cloud Trace** window in the **Cloud Console** and review the Trace data for the new request.

Note: You should see that this request includes calls to Cloud Spanner.

Task 3. Diagnosing a performance problem with Cloud Trace

In this section, you continue to explore the quiz application with Cloud Trace and identify and resolve a performance issue.

Identify a performance problem with Cloud Trace

1. Return to the quiz application and click **Take Test**.
2. Click **Places**.
3. Complete the test, and submit feedback.
4. Return to the **Cloud Trace** window in the **Cloud Console**.

Note: Look at the Trace list again. Eventually a set of new requests will be displayed.

Find the following requests and click on them:

1. POST /api/quizzes/places
2. POST /api/quizzes/feedback/places

These represent the calls made from the client-side application against the quiz application.

Once again, you should see the total time taken to complete the request.

Notice that the calls are sequential. This is because the Cloud Pub/Sub request needs data from Cloud Datastore to send the answer data to Cloud Pub/Sub.

Note: However, you should also see that each call to Cloud Pub/Sub is also sequential; this is not OK!

5. Return to the quiz application and click **GCP**.
6. Complete the test, and submit feedback.
7. Return to the **Cloud Trace** window in the **Cloud Console**.

Note: Look at the Trace list once again. Eventually a set of new requests will be displayed.

Find the following requests and click on them:

1. POST /api/quizzes/gcp
2. POST /api/quizzes/feedback/gcp

These represent the calls made from the client-side application against the quiz application.

Once again, you should see the total time taken to complete the request.

Notice that the calls are sequential. This is because the Cloud Pub/Sub request needed data from Cloud Datastore to send the answer data to Cloud Pub/Sub.

Note: However, you should also see that once again, each call to Cloud Pub/Sub is also sequential.

With four questions in the quiz, it's taking four times as long to send the Cloud Pub/Sub messages!

This is definitely not OK!

Modify application code to resolve the performance problem

1. Return to the **Cloud Shell** and launch the **Code editor** if it is not opened already.
2. Navigate to **stackdriver-trace-monitoring/start/**.
3. Open `...frontend/api/index.js`.
4. Find the statement that publishes answer messages in sequence, and modify it to perform the operations in parallel.

Note: Fortunately, this is a very easy change to make! The code change is shown below.

api/index.js before the change:

```
// TODO: Sends the answers to Pub/Sub in parallel
```

```
      // Sends the answers to Pub/Sub in sequence

      // Change sequence to parallel in the next statement

      sequence(answersWithCorrect.map(answer => () =>
publisher.publishAnswer(answer))).then(() => {

        // Waits until all the Pub/Sub messages have been acknowledged
before returning to the client

        const score = answersWithCorrect.filter(a => a.answer ==
a.correct).length;

        res.status(200).json({ correct: score, total: questions.length });

      });

      // END TODO
```

Copied!

content_copy

api/index.js after the change:

```
      // TODO: Sends the answers to Pub/Sub in parallel

       // Changed to parallel

       parallel(answersWithCorrect.map(answer => () =>
publisher.publishAnswer(answer))).then(() => {
```

```
        // Waits until all the Pub/Sub messages have been acknowledged
before returning to the client

        const score = answersWithCorrect.filter(a => a.answer ==
a.correct).length;

        res.status(200).json({ correct: score, total: questions.length });

      });

       // END TODO
```

Confirm problem resolution with Cloud Trace

1. Return to the **Cloud Shell**, stop the application by pressing **Ctrl+C**, and then start it again.
2. Return to the Quiz application, and take the **People**, **Places**, and **GCP** tests again.
3. Return to the **Cloud Trace** window in the **Cloud Console**.

Note: Look at the Trace list once again. Eventually a set of new requests will be displayed.

Find the following requests and click on them:

1. POST /api/quizzes/people|places|gcp
2. POST /api/quizzes/feedback/people|places|gcp

These represent the calls made from the client-side application against the quiz application.

Once again, you should see the total time taken to complete the request.

Note: This time, you should see that all of the Pub/Sub messages have been dispatched in parallel. The time taken to complete processing of the requests is significantly lower.

Task 4. Visualizing application metrics with Cloud Monitoring

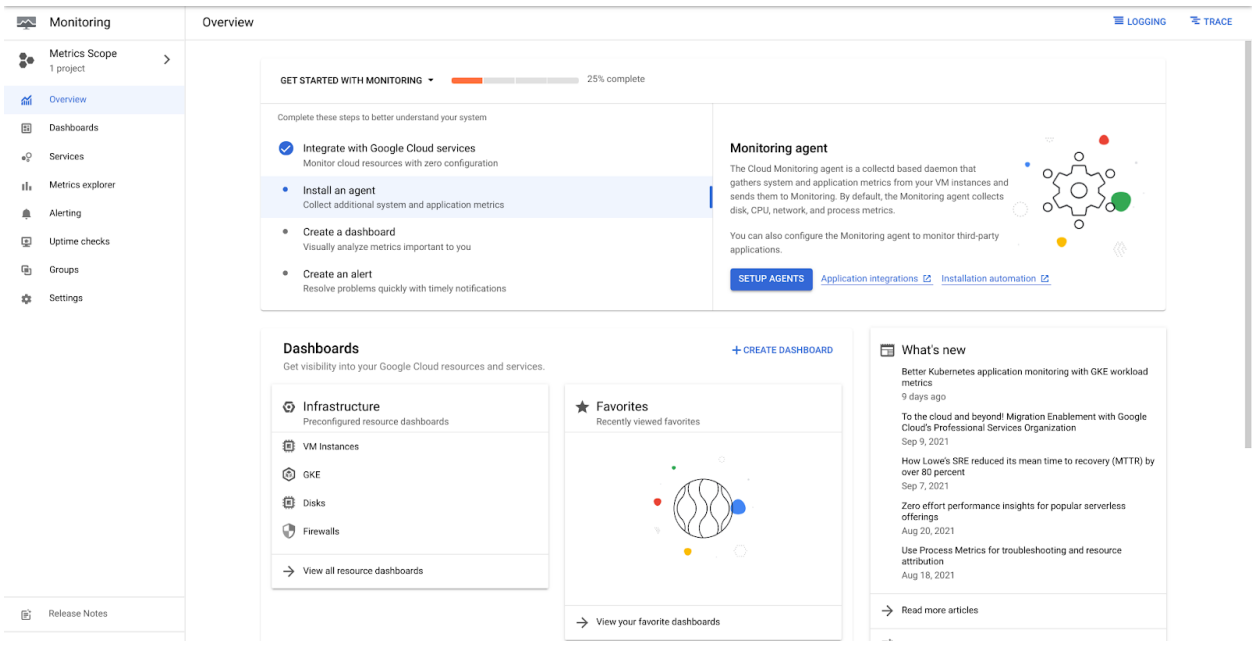
In this section, you continue to explore the quiz application with Cloud Monitoring by exploring metrics and create dashboards.

Create a Monitoring workspace

You will now setup a Monitoring workspace that's tied to your Google Cloud Project. The following steps create a new account that has a free trial of Monitoring.

1. In the Cloud Console, click on **Navigation menu** > **Monitoring**.
2. Wait for your workspace to be provisioned.

When the Monitoring dashboard opens, your workspace is ready.



Create a dashboard and explore metrics

1. In the left-hand pane, click **Dashboards**.
2. Click **+Create Dashboard**.
3. For **New Dashboard Name**, type **Quiz Application Metrics**.
4. Click **Line Chart**.
5. Under **Resource & Metric** field, click on **VM Instance - CPU Utilization** dropdown.
6. Enable **Show only active resources & metrics**.

Note: Examples of resources that are active:

- cloud_function
- datastore_request
- gae_app (this is because of Cloud Datastore)

- gcs_bucket
- global
- pubsub_subscription
- pubsub_topic
- spanner_instance

7. Click on each resource in turn, and select a few metrics that are of interest to you.

Note: Here are some example metrics to explore:

- cloudfunctions/function/execution_count
- datastore/api/request_count
- storage/api/request_count
- logging/log_entry_count
- pubsub/subscription/num_outstanding_messages
- pubsub/topic/message_sizes
- spanner/api/request_count

8. Create several charts, including your own selection of resource and metrics

Task 5. Bonus: Deploy the Quiz Application into Container or App Engine

Use the notes from the appropriate labs to do this. After the application is deployed, you'll be able to monitor both the frontend and the Google Cloud resources.