Setup and Requirements

1. Prepare GCP project
2. Enable Cloud Trace API

```
# gcloud services enable cloudtrace.googleapis.com
```

3. Enable Container Registry API

```
# gcloud services enable containerregistry.googleapis.com
```

4. Enable Cloud Compute Engine API and Container API (for creating and using GKE)

```
# gcloud services enable compute.googleapis.com container.googleapis.com
```

Get Source Code

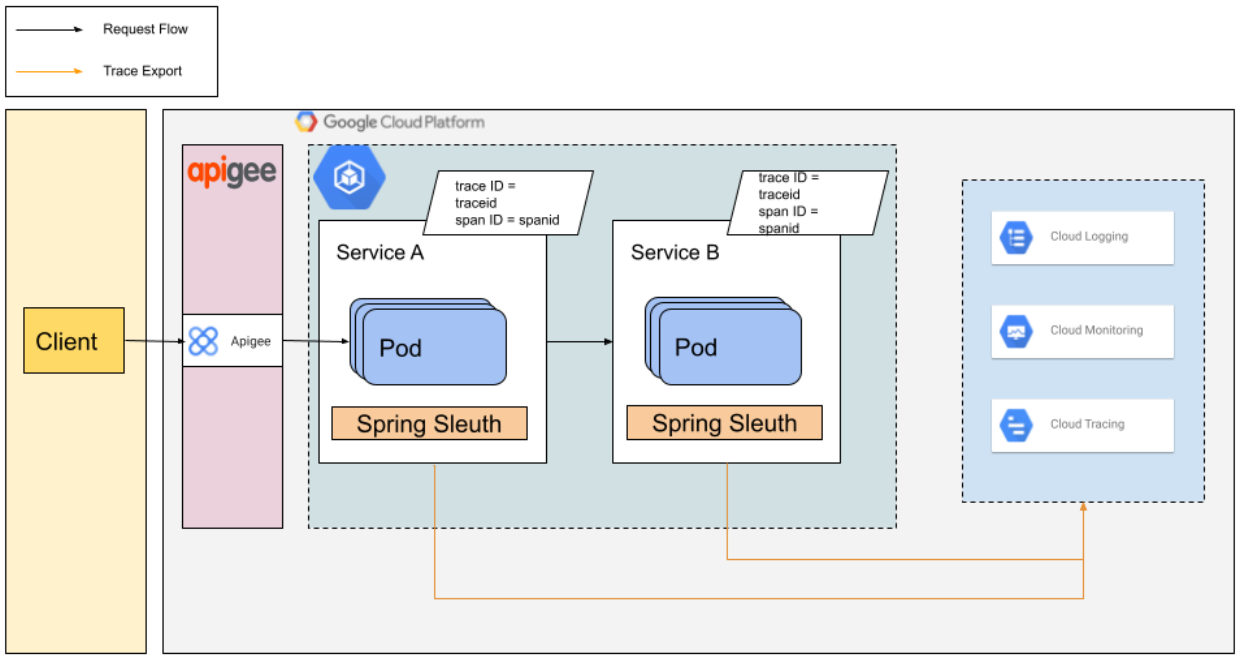
First Service:

```
/microservice-trace/spring/trace-service-a
  |--src
    |--test
    |--main
      |--java
        |--com
          |--example
            |--demo
              |--DemoApplication.java
              |--WorkController.java
      |--resources
        |--application.properties
  |--mvnw.cmd
  |--mvnw
  |--HELP.md
  |--pom.xml
```

Second Service:

```
/microservice-trace/spring/trace-service-b
  |--src
    |--test
    |--main
      |--java
        |--com
          |--example
            |--demo
              |--DemoApplication.java
              |--MeetingController.java
      |--resources
        |--application.properties
  |--mvnw.cmd
  |--mvnw
  |--HELP.md
  |--pom.xml
```

Request Flow:



Locally run the app

```
# cd trace-demo-service-A
# ./mvnw -DskipTests spring-boot:run
# curl localhost:8080
```

Using Cloud Trace With Spring Sleuth

Add the Spring Cloud GCP Trace dependency

1. pom.xml:

```
<project>
  ...
  <dependencies>
    ...
    <!-- Add Cloud Trace Starter -->
    <dependency>
      <groupId>com.google.cloud</groupId>
      <artifactId>spring-cloud-gcp-starter-trace</artifactId>
    </dependency>
  </dependencies>
  ...
</project>
```

2. increase sample rate to 100%

By default, Spring Cloud Sleuth does not sample every request. To make the testing a little easier, increase the sample rate to 100% in application.properties to ensure each test request is traced:

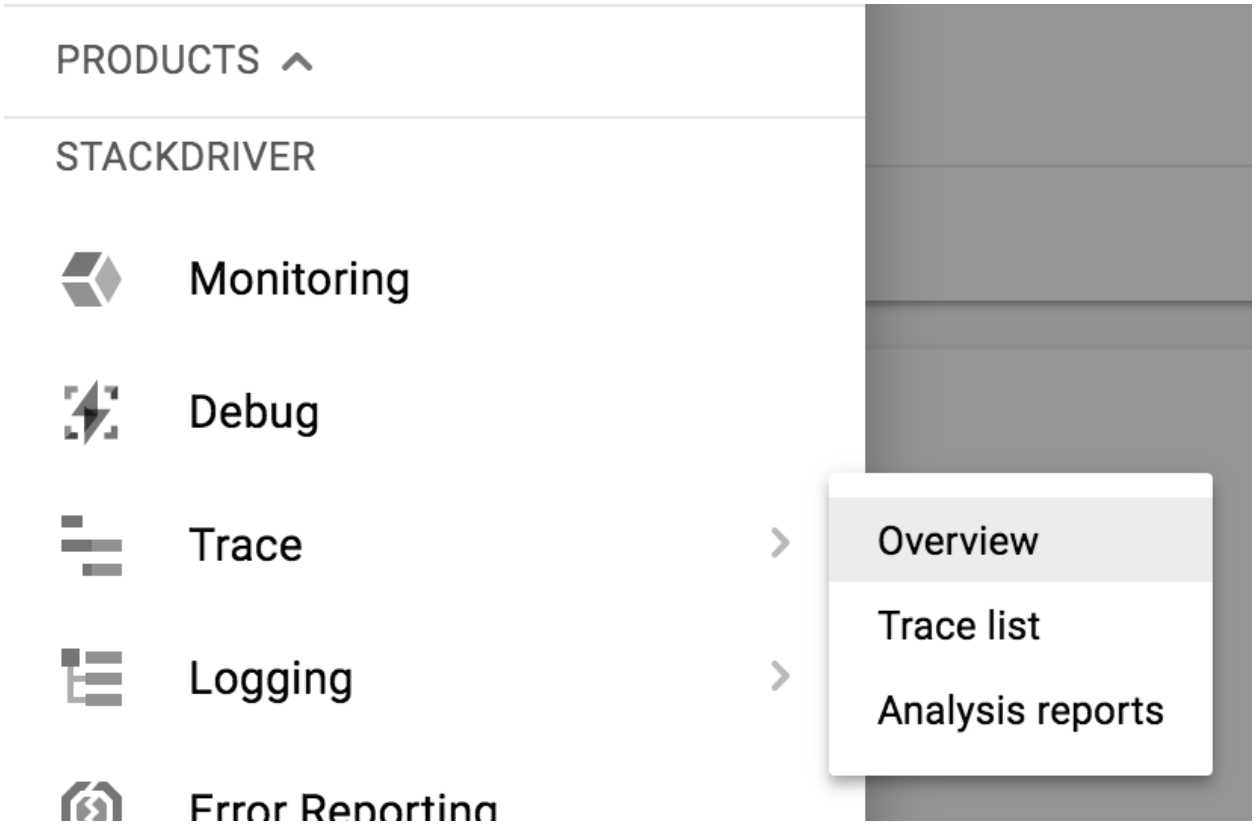
```
# echo "
spring.sleuth.sampler.probability=1.0
" > src/main/resources/application.properties
```

Test and Verify Trace

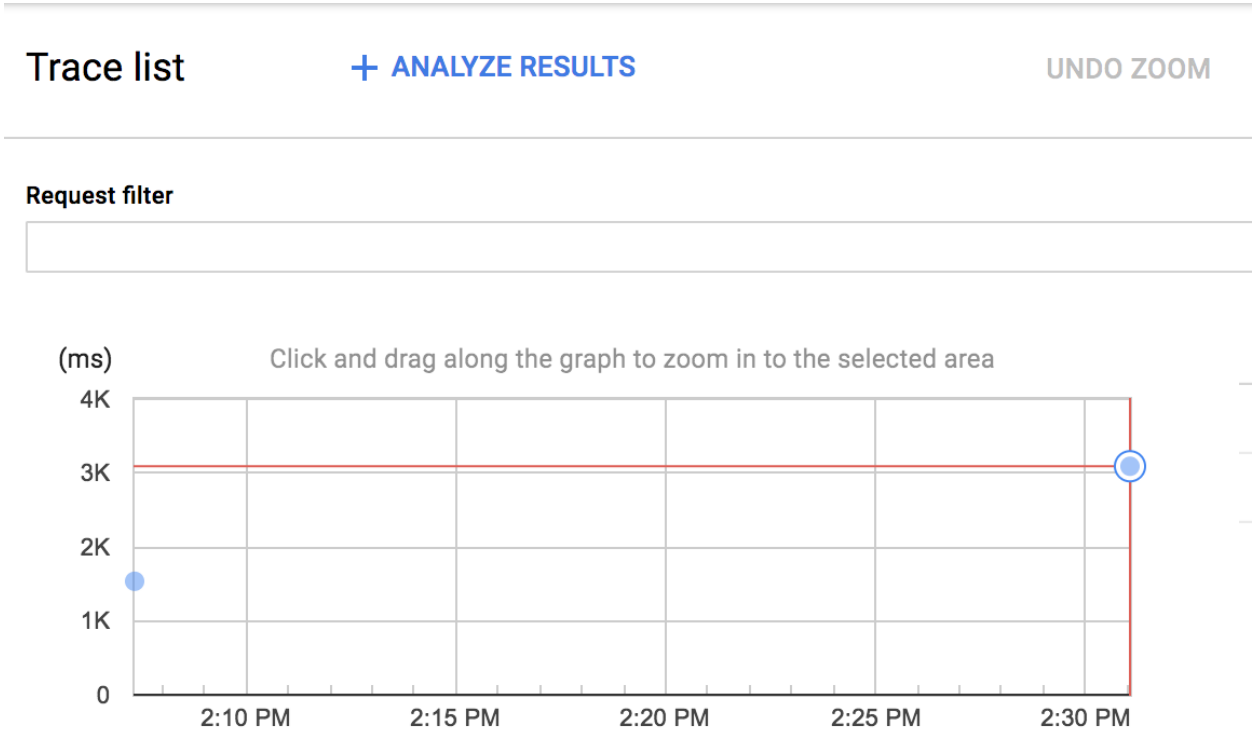
Test again, the trace data should be sent to Cloud Trace:

```
# export GOOGLE_CLOUD_PROJECT=`gcloud config list --format 'value(core.project)'`
# ./mvnw -DskipTests spring-boot:run
# curl localhost:8080
```

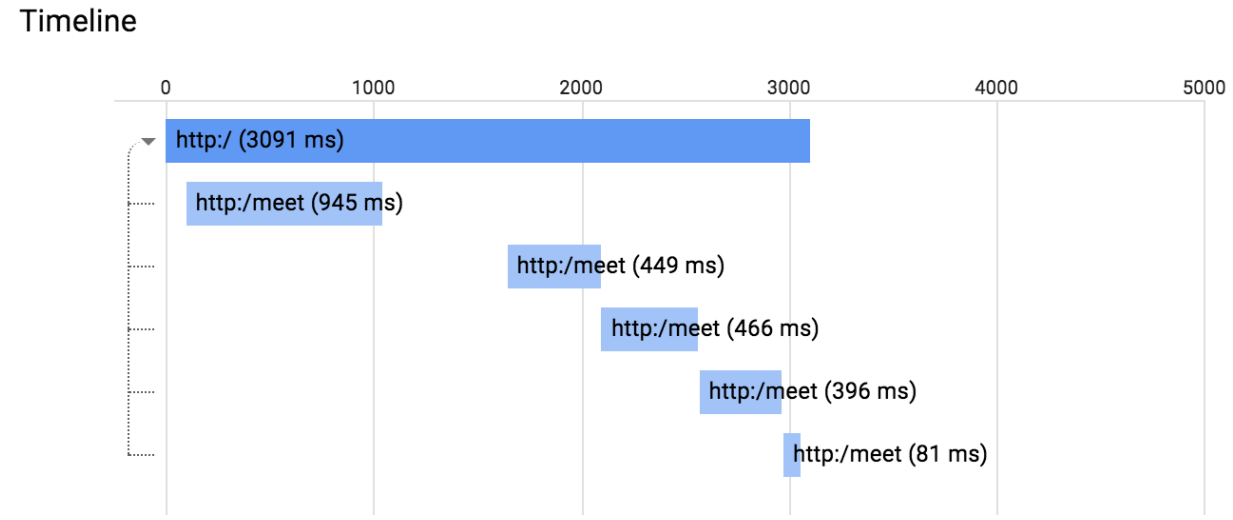
From Cloud Trace Console, navigate to **Operations** → **Trace** → **Trace list**:



We expect to see the trace data similar as below:



Click the blue dot to see trace details:



Package the Java app as a Docker container

1. Create the JAR deployable for the app.

```
# cd ~/demo/trace-demo-service-a
# ./mvnw -DskipTests package
```

2. Use Jib to create the container image and push it to the Container Registry.

```
# export GOOGLE_CLOUD_PROJECT=`gcloud config list --format="value(core.project)"`

# ./mvnw -DskipTests com.google.cloud.tools:jib-maven-plugin:build \
  -Dimage=gcr.io/$GOOGLE_CLOUD_PROJECT/trace-demo-service-a-java:v1
```

3. Repeat step 1 and 2 for **trace-demo-service-b**

```
# cd ~/demo/trace-demo-service-b
# ./mvnw -DskipTests package

# export GOOGLE_CLOUD_PROJECT=`gcloud config list --format="value(core.project)"`

# ./mvnw -DskipTests com.google.cloud.tools:jib-maven-plugin:build \
  -Dimage=gcr.io/$GOOGLE_CLOUD_PROJECT/trace-demo-service-b-java:v1
```

4. We should be able to see the container image listed in the console by navigating to the Container Registry Images page in the Cloud Console. They are project-wide Docker images, which Kubernetes can access and orchestrate.

Create GKE cluster

Run the command to create a GKE cluster (in us-central1-c zone):

```
# gcloud container clusters create trace-demo-cluster \
  --num-nodes 2 \
  --machine-type n1-standard-1 \
  --zone us-central1-c

Creating cluster trace-demo-cluster...done.
Created [https://container.googleapis.com/v1/projects/...].
kubeconfig entry generated for hello-dotnet-cluster.
NAME                 ZONE           MASTER_VERSION
trace-demo-cluster  us-central1-c  ...
```

```
```

About Cluster Size:

In this demo, the cluster with minimum size (2 GCE nodes) was created. To create the cluster with more nodes, change the parameter `--num-nodes`.

About Cluster Network:

In this demo, the cluster is configured to be running on the **default** VPC network in the specified region/zone. The default network is a predefined VPC network when the GCP project is created. It has subnetworks in all the regions and it configured the basic firewall rules/routes to communicate with other VPCs and the internet.

About Cluster node machine type:

In this demo, the cluster uses the n1-standard-1 GCE node machine type[1]. It has 1 vCPU and 3.75 GB memory. To use other machine types, use the parameter --machine-type.

[1] https://cloud.google.com/compute/docs/general-purpose-machines#n1_machines

About Cluster Service Account:

In this demo, the cluster uses the default Compute Engine service account. To use other user defined service account, use the parameter --service-account

Deploy Java app to GKE Cluster

Helm Structure

1. trace-demo-service-a

```
trace-demo-service-a
  |--charts
  |--templates
  |--Chart.yam
  |--values.yaml
```

2. trace-demo-service-b

```
trace-demo-service-b
  |--charts
  |--templates
  |--Chart.yam
  |--values.yaml
```

Create Deployment and Services

1. Deploy one instance of the demo service A to Kubernetes using the **kubectl run** command.

```
# kubectl create deployment trace-demo-service-a \
  --image=gcr.io/$GOOGLE_CLOUD_PROJECT/trace-demo-service-a-java:v1
```

2. View the deployment and pod that we created

```
# kubectl get deployments
NAME         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
trace-demo-service-a   1         1         1            1           37s

# kubectl get pods
NAME                          READY     STATUS    RESTARTS   AGE
trace-demo-service-a-714049816-ztzrb   1/1       Running   0          57s
```

3. Expose the deployment by creating a Kubernetes LoadBalancer service.

```
# kubectl create service loadbalancer trace-demo-service-a --tcp=8080:8080
```

4. Find the publicly accessible IP address of the service

```
# kubectl get services
NAME         CLUSTER-IP     EXTERNAL-IP      PORT(S)    AGE
trace-demo-service-a   10.3.253.62    aaa.bbb.ccc.ddd  8080/TCP    1m
kubernetes   10.3.240.1     <none>           443/TCP    5m
```

5. Repeat step 1~4 for **trace-demo-service-b and expose the service to a different port 8081**

```
# kubectl create deployment trace-demo-service-b \
  --image=gcr.io/$GOOGLE_CLOUD_PROJECT/trace-demo-service-b-java:v1

# kubectl get deployments
NAME         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
trace-demo-service-b   1         1         1            1           37s

# kubectl get pods
NAME                          READY     STATUS    RESTARTS   AGE
trace-demo-service-b-714059816-cbzrb   1/1       Running   0          57s

# kubectl create service loadbalancer trace-demo-service-b --tcp=8080:8080

# kubectl get services
NAME         CLUSTER-IP     EXTERNAL-IP      PORT(S)    AGE
trace-demo-service-b   10.3.253.62    aaa.bbb.ccc.ddd  8081/TCP    1m
kubernetes   10.3.240.1     <none>           443/TCP    5m
```

Test from public IP

1. Find out the public IP of demo-trace-service-a:

```
# kubectl get services
NAME         CLUSTER-IP     EXTERNAL-IP      PORT(S)    AGE
trace-demo-service-a   10.3.253.62    aa1.bb1.cc1.dd1  8080/TCP    1m
kubernetes   10.3.240.1     <none>           443/TCP    5m

trace-demo-service-b   10.3.253.62    aa2.bb2.cc2.dd2  8081/TCP    1m
kubernetes   10.3.240.1     <none>           443/TCP    5m
```

2. Make a request to **aa1.bb1.cc1.dd1:8080**

```
# curl aa1.bb1.cc1.dd1:8080
```

```
# export GOOGLE_CLOUD_PROJECT=`gcloud config list --format 'value(core.project)'`
# ./mvnw -DskipTests spring-boot:run
```

```
# curl localhost:8080
```

Test from internal pod

1. Find out the pod name of demo-trace-service-a

```
# kubectl get pods
NAME                              READY     STATUS    RESTARTS   AGE
trace-demo-service-a-714049816-ztzrb   1/1       Running   0          57s
```

2. Login into the pod:

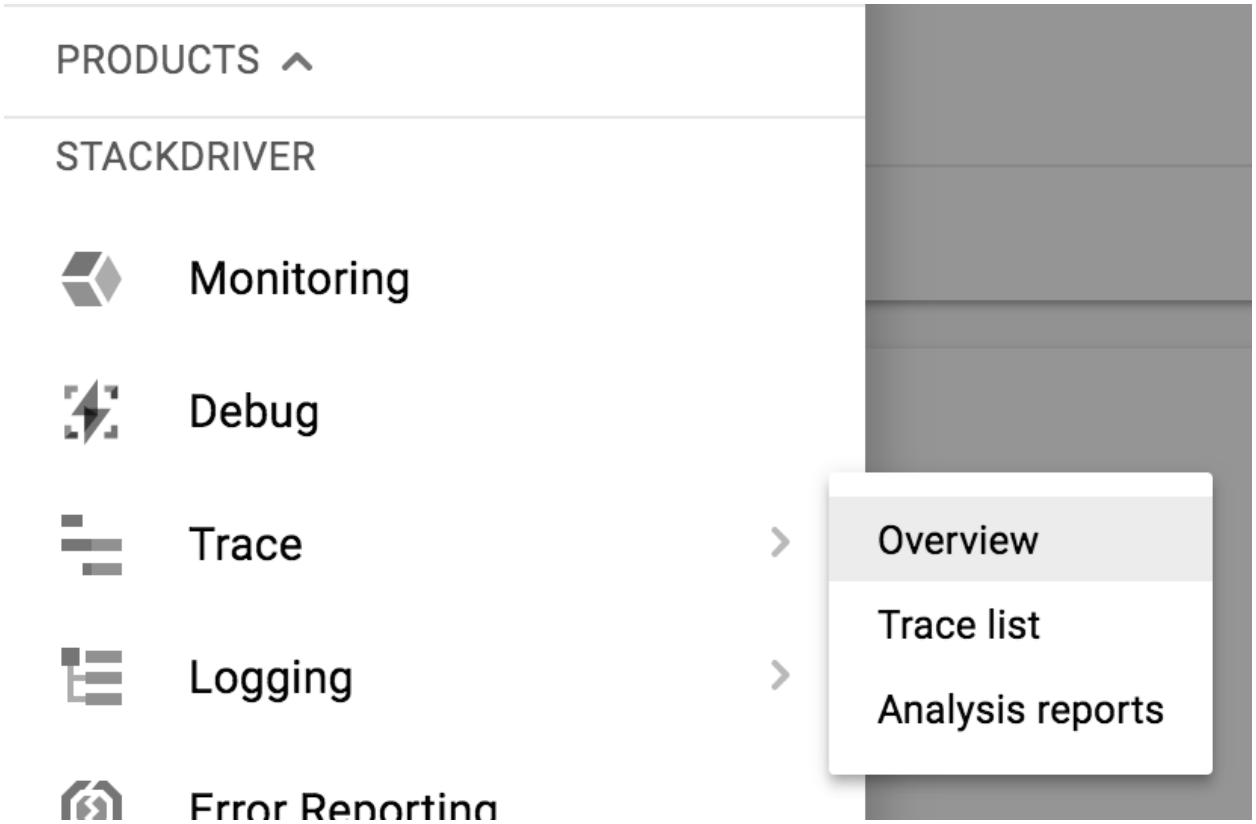
```
# kubectl exec -it trace-demo-service-a-714049816-ztzrb -- /bin/bash
```

3. Make a request to localhost:8080

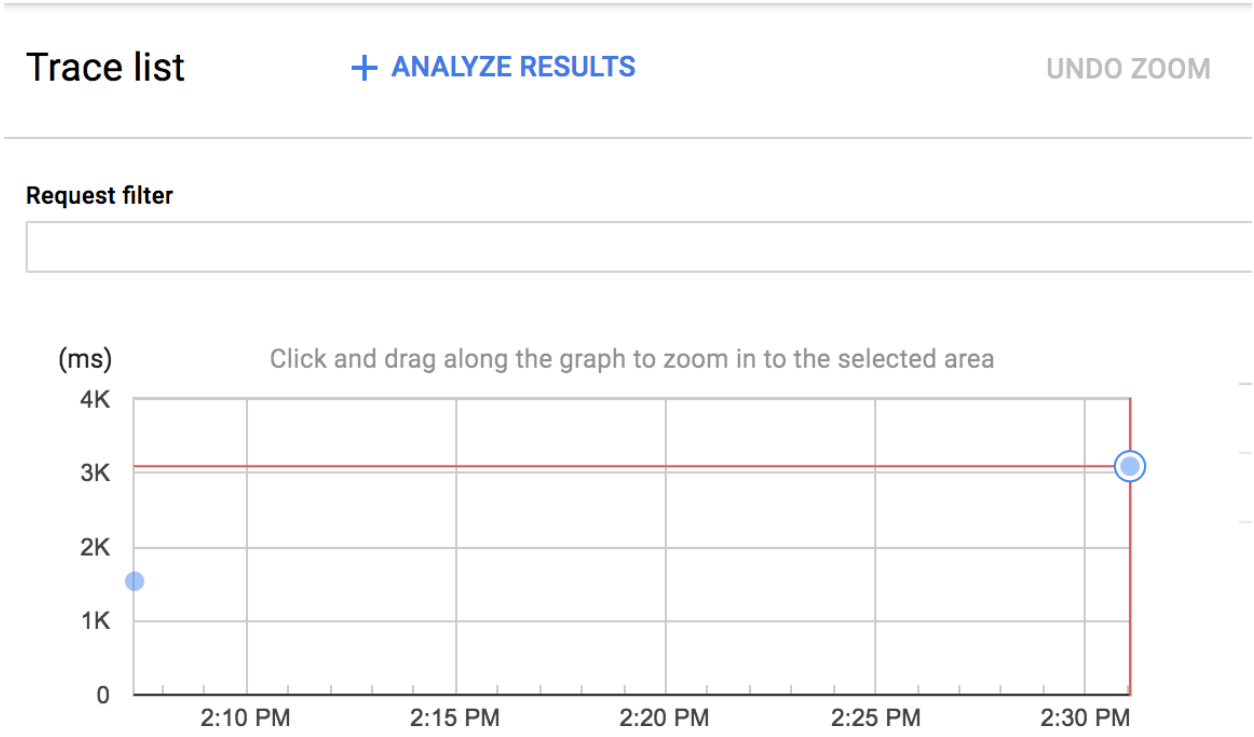
```
$ curl localhost:8080
```

Verify Trace from Cloud Trace Console

From Cloud Trace Console, navigate to **Operations** → **Trace** → **Trace list**:



We expect to see the trace data similar as below:



Upgrade the Service

1. package and push the new image to Container registry

```
./mvnw -DskipTests package \
  com.google.cloud.tools:jib-maven-plugin:build \
  -Dimage=gcr.io/$GOOGLE_CLOUD_PROJECT/trace-demo-service-a-java:v2
```

2. Edit the deployment and change the image

```
# kubectl set image deployment/trace-demo-service-a \

trace-demo-service-a-java=gcr.io/$GOOGLE_CLOUD_PROJECT/trace-demo-service-a-java:v2

deployment "trace-demo-service-a" image updated
```