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:
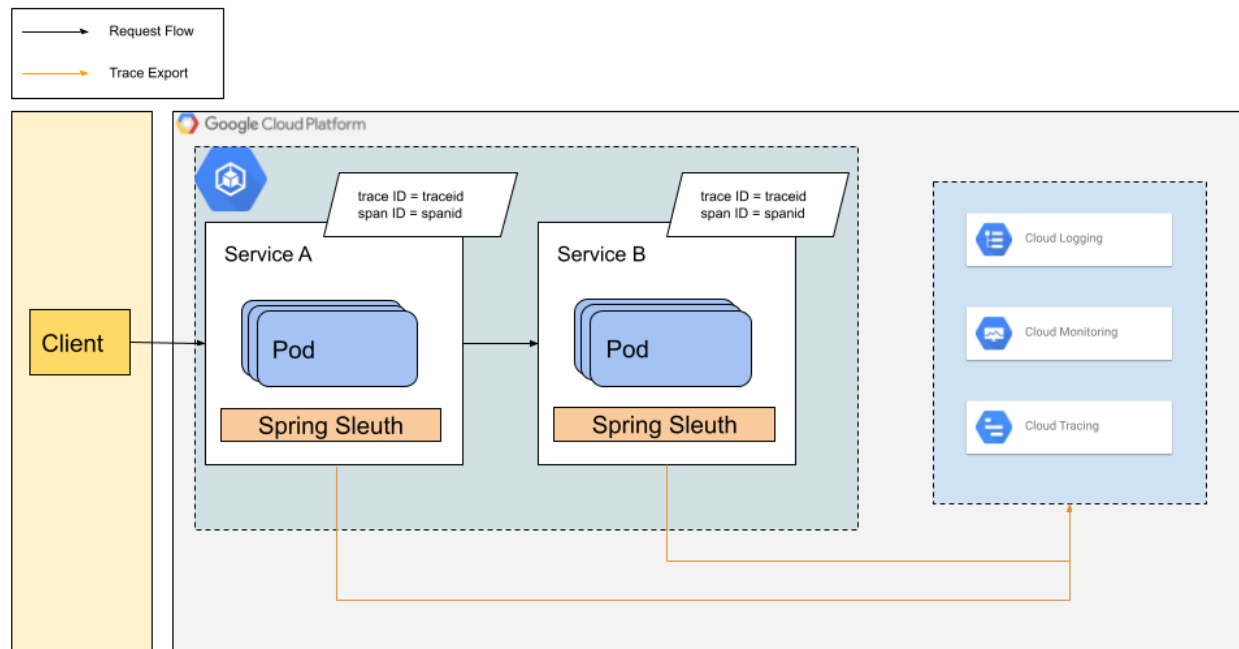
Locally run the app

```
# 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:

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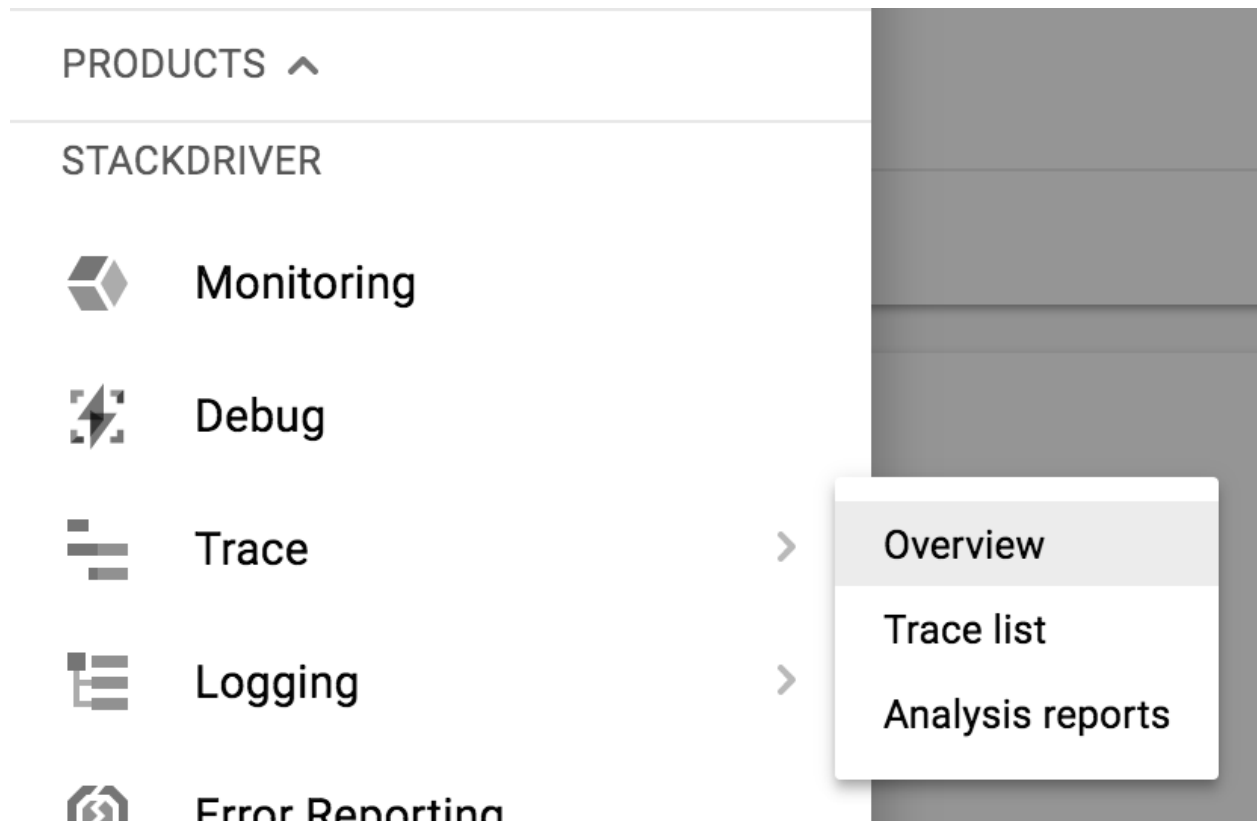
```
# echo "
spring.sleuth.sampler.probability=1.0
" > src/main/resources/application.properties
```

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
```

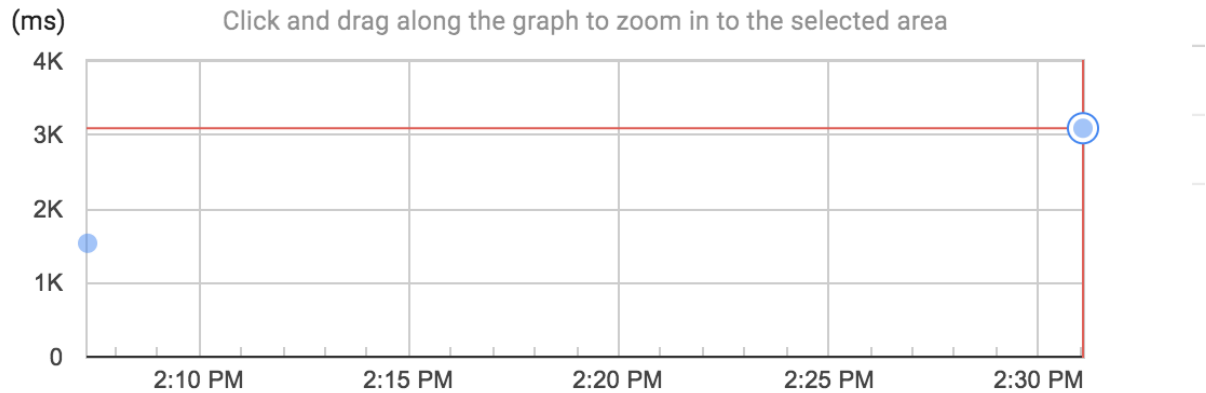From Cloud Trace Console, navigate to **Operations** → **Trace** → **Trace list**:

We expect to see the trace data similar as below:
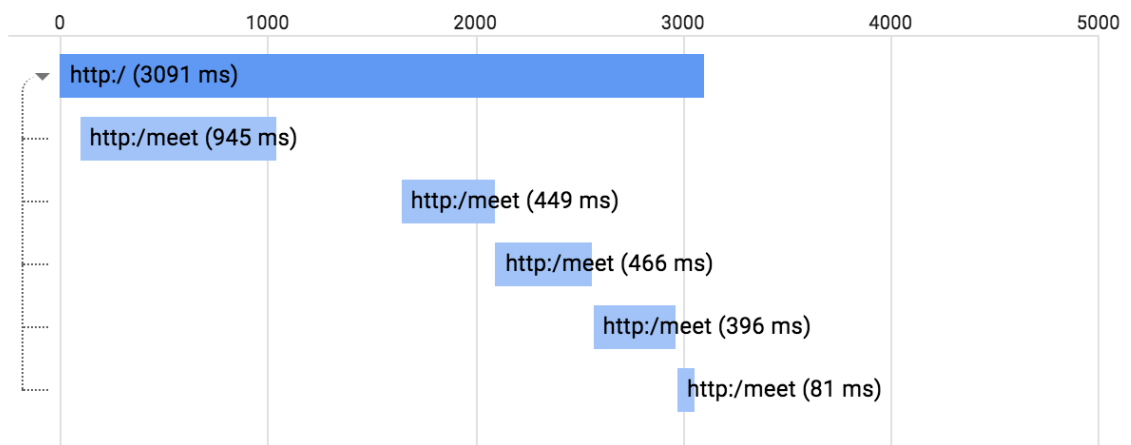
Trace list   + ANALYZE RESULTS   UNDO ZOOM

Request filter

(ms)   Click and drag along the graph to zoom in to the selected area

4K

3K

2K

1K

0
        2:10 PM   2:15 PM   2:20 PM   2:25 PM   2:30 PM

Click the blue dot to see trace details:

Timeline

        0   1000   2000   3000   4000   5000

http:/ (3091 ms)

  http:/meet (945 ms)

                http:/meet (449 ms)

                  http:/meet (466 ms)

                      http:/meet (396 ms)

                        http:/meet (81 ms)

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  ...
```

Deploy Java app to GKE Cluster

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
```
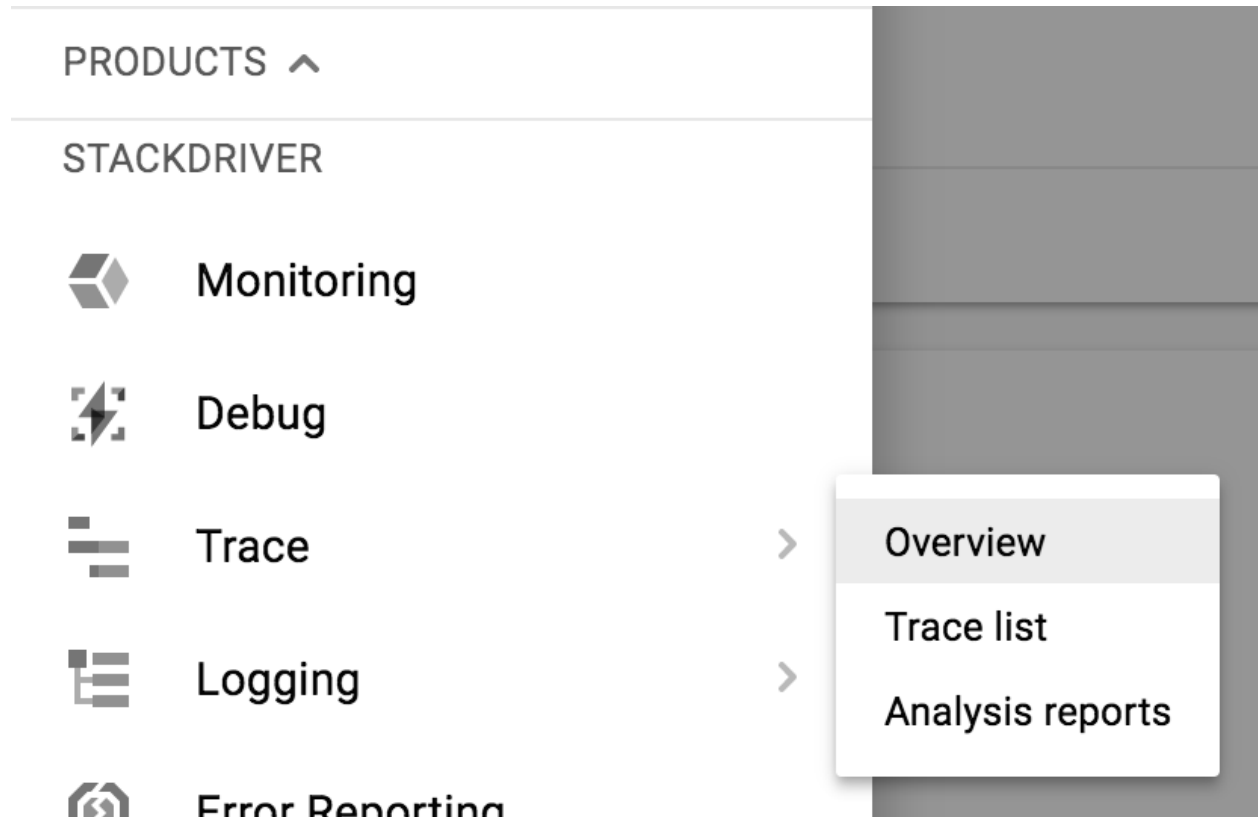
2. Login into the pod:

```
# 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**:

PRODUCTS ⌃

STACKDRIVER

Monitoring

Debug

Trace   >   Overview

                                  Trace list

Logging   >   Analysis reports

Error Reporting

We expect to see the trace data similar as below:

Trace list

Request filter

(ms)

Click and drag along the graph to zoom in to the selected area

4K

3K

2K

1K

0

2:10 PM   2:15 PM   2:20 PM   2:25 PM   2:30 PM