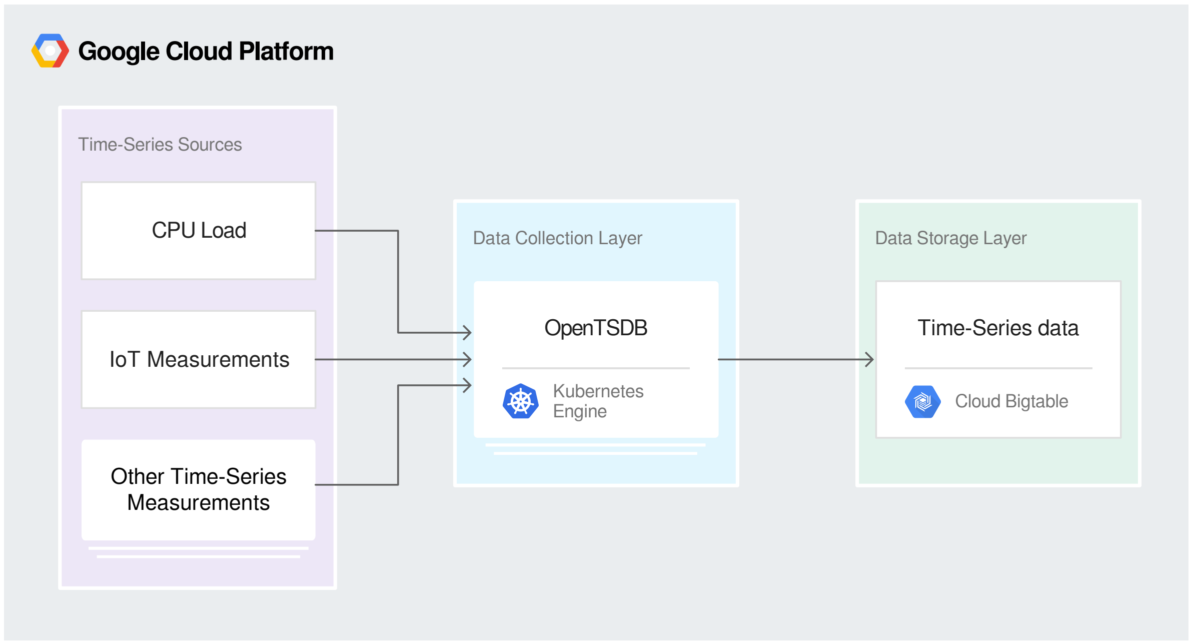
Solutions (https://cloud.google.com/solutions/) Solutions

Using OpenTSDB to Monitor Time-Series Data

This tutorial describes how to collect, record, and monitor time-series data (https://wikipedia.org/wiki/Time_series) on Google Cloud using OpenTSDB (http://opentsdb.net/) running on Google Kubernetes Engine (GKE) (https://cloud.google.com/kubernetes-engine/) and Bigtable (https://cloud.google.com/bigtable/).

Time-series data is a highly valuable asset that you can use for several apps, including trending, monitoring, and machine learning. You can generate time-series data from server infrastructure, app code, and other sources. OpenTSDB can collect and retain large amounts of time-series data with a high degree of granularity.

This tutorial details how to create a scalable data collection layer using GKE and work with the collected data using Bigtable. The following diagram illustrates the high-level architecture of the solution:



Objectives

- Create a new Bigtable instance.
- Create a new GKE cluster.
- Deploy OpenTSDB to your GKE cluster.
- Send time-series metrics to OpenTSDB.
- Visualize metrics using OpenTSDB and Grafana (https://grafana.com/).

Costs

This tutorial uses billable components of Google Cloud, including:

- Compute Engine
- GKE
- Bigtable
- Cloud Storage

Use the Pricing Calculator (https://cloud.google.com/products/calculator/) to generate a cost estimate based on your projected usage.

New Google Cloud users might be eligible for a free trial (https://cloud.google.com/free-trial).

Before you begin

1. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).

2. In the Cloud Console, on the project selector page, select or create a Cloud project.

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

   GO TO THE PROJECT SELECTOR PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/PROJECTSELECT

3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (https://cloud.google.com/billing/docs/how-to/modify-project).

4. Enable the Bigtable, Bigtable Admin, Compute Engine, and Google Kubernetes Engine APIs.

   ENABLE THE APIS (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/FLOWS/ENABLEAPI?APIID=BIGTABLE,

Make note of the **Project ID** (https://console.cloud.google.com/home/dashboard) for use in a later step.

Preparing your environment

You will use Cloud Shell (https://cloud.google.com/shell/docs/overview) to enter commands in this tutorial. Cloud Shell gives you access to the command line in Cloud Console, and includes Cloud SDK and other tools you need for Google Cloud development. Cloud Shell appears as a window at the bottom of Cloud Console. It can take several minutes to initialize, but the window appears immediately.

1. Activate Cloud Shell.

   ACTIVATE CLOUD SHELL (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/?CLOUDSHELL=TRUE)

   Note: As an alternative to Cloud Shell, which comes with the Cloud SDK pre-installed, you can choose to install Cloud SDK on your local workstation (https://cloud.google.com/sdk/) and continue working from there. If you use your local environment, you will need to alter a few of the instructions in this tutorial.

2. Set the default Compute Engine zone to the zone where you are going to create your Bigtable cluster, for example `us-central1-f`.

```
gcloud config set compute/zone us-central1-f
```

3. Clone the git repository containing the sample code.

```
git clone https://github.com/GoogleCloudPlatform/opentsdb-bigtable.git
```

4. Navigate to the sample code directory.

```
cd opentsdb-bigtable
```

Creating a Bigtable instance

This tutorial uses Bigtable to store the time-series data that you collect. You must create a Bigtable instance to do that work.

Bigtable is a key/wide-column store that works especially well for time-series data, as is explained in Bigtable Schema Design for Time Series Data (https://cloud.google.com/bigtable/docs/schema-design-time-series). Bigtable supports the HBase API, which makes it easy for you to use software designed to work with Apache HBase (https://hbase.apache.org/), such as OpenTSDB. You can learn about the HBase schema used by OpenTSDB in the OpenTSDB documentation (http://opentsdb.net/docs/build/html/user_guide/backends/hbase.html).

A key component of OpenTSDB is the AsyncHBase (https://github.com/OpenTSDB/asynchbase) client, which enables it to bulk-write to HBase in a fully asynchronous, non-blocking, thread-safe manner. When you use OpenTSDB with Bigtable, AsyncHBase is implemented as the AsyncBigtable (https://github.com/OpenTSDB/asyncbigtable) client.

The ability to easily scale to meet your needs is a key feature of Bigtable. This tutorial uses a single-node development cluster, because it is sufficient for the task and is economical. Start your projects in a development cluster and move to a larger production cluster when you are ready to work with production data. The Bigtable documentation includes detailed discussion about performance and scaling (https://cloud.google.com/bigtable/docs/performance) to help you pick a cluster size for your own work.

Follow these steps to create your Bigtable instance:

1. Go to the **Create Instance** page in the Cloud Console.

   GO TO THE CREATE INSTANCE PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/BIGTABLE/CREATE-

2. Enter a name for your instance in the **Instance name** box. You can use `OpenTSDB instance` or another name of your choosing. The page automatically sets **Instance ID** and **Cluster ID** after you enter your instance name.
3. Set **Instance type** to **Development**.
4. In **Zone**, select **us-central1-f** or the zone from which you are going to run OpenTSDB.
5. Click **Create** to create the instance.

Make note of the values of **Instance ID** and **Zone**. You will use them in a later step.

Creating a GKE cluster

GKE provides a managed Kubernetes (https://kubernetes.io/) environment. After you create a GKE cluster, you can deploy Kubernetes pods (https://kubernetes.io/docs/concepts/workloads/pods/pod/) to it. This tutorial uses GKE and Kubernetes pods to run OpenTSDB.

OpenTSDB separates its storage from its application layer, which enables it to be deployed across multiple instances simultaneously. By running in parallel, it can handle a large amount of time-series data. Packaging OpenTSDB into a Docker container enables easy deployment at scale using GKE.

Create a Kubernetes cluster by running the following command. This operation can take a few minutes to complete:

```
gcloud container clusters create opentsdb-cluster --scopes \
"https://www.googleapis.com/auth/bigtable.admin",\
"https://www.googleapis.com/auth/bigtable.data"
```

Adding the two extra scopes to your GKE cluster allows your OpenTSDB container to interact with Bigtable. You can pull images (https://cloud.google.com/container-registry/docs/access-control) from Google Container Registry (https://cloud.google.com/container-registry/) without adding a scope for Cloud Storage, because the cluster can read from Cloud Storage by default. You might need additional scopes (https://developers.google.com/identity/protocols/googlescopes) in other deployments.

The rest of this tutorial uses a prebuilt container, `gcr.io/cloud-solutions-images/opentsdb-bigtable:v1` located in Container Registry (https://cloud.google.com/container-registry/). The Dockerfile (https://github.com/GoogleCloudPlatform/opentsdb-bigtable/blob/master/build/Dockerfile) and ENTRYPOINT (https://github.com/GoogleCloudPlatform/opentsdb-bigtable/blob/master/build/docker-entrypoint.sh) script used to build the container are located in the `build` folder of the tutorial repository.

Creating a ConfigMap with configuration details

Kubernetes provides a mechanism called the ConfigMap (https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/) to separate configuration details from the container image to make apps more portable. The configuration for OpenTSDB is specified in `opentsdb.conf`. A ConfigMap containing `opentsdb.conf` is included with the sample code. You must edit it to reflect your instance details.

Create the ConfigMap

Edit the OpenTSDB configuration to use the project name, instance identifier, and zone that you used when creating your instance.

1. To open the code editor built into Cloud Shell, click the pencil icon in the toolbar at the top of the Cloud Shell window.
2. Select `opentsdb-config.yaml` under `opentsdb/configmaps` to open it in the editor.
3. Replace the placeholder text with the project name, instance identifier, and zone you set earlier in the tutorial.

4. From the Cloud Shell prompt, create a ConfigMap from the updated `opentsdb-config.yaml`:

   ```
   kubectl create -f configmaps/opentsdb-config.yaml
   ```

Note: OpenTSDB offers you a variety of configuration options (http://opentsdb.net/docs/build/html/user_guide/configuration.html). To change your configuration, modify the `opentsdb.conf` ConfigMap and apply (https://kubernetes.io/docs/concepts/cluster-administration/manage-deployment/#kubectl-apply) it to push the changes to the cluster. Some changes require you to restart processes.

Creating OpenTSDB tables in Bigtable

Before you can read or write data using OpenTSDB, you need to create the necessary tables (http://opentsdb.net/docs/build/html/installation.html#create-tables) in Bigtable to store that data. Follow these steps to create a Kubernetes job (https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/) that creates the tables.

1. Launch the job:

   ```
   kubectl create -f jobs/opentsdb-init.yaml
   ```

2. The job can take up to a minute or more to complete. Verify the job has completed successfully by periodically running this command:

   ```
   kubectl describe jobs
   ```

   The output should indicate that one job has succeeded under the heading `Pods Statuses`.

3. Get the table creation job logs by running the following commands:

   ```
   pods=$(kubectl get pods  --show-all --selector=job-name=opentsdb-init \
   --output=jsonpath={.items..metadata.name})

   kubectl logs $pods
   ```

When you get the logs, examine the bottom of the output, which should indicate each table that was created. This job runs several table creation commands, each taking the form of `create 'TABLE_NAME'`. Look for a line of the form `0 row(s) in 0.0000 seconds`, where the actual command duration is listed instead of `0.0000`.

Your output should include a section that looks something like the following:

```
create 'tsdb-uid',
  {NAME => 'id', COMPRESSION => 'NONE', BLOOMFILTER => 'ROW'},
  {NAME => 'name', COMPRESSION => 'NONE', BLOOMFILTER => 'ROW'}
0 row(s) in 1.3680 seconds

Hbase::Table - tsdb-uid

create 'tsdb',
  {NAME => 't', VERSIONS => 1, COMPRESSION => 'NONE', BLOOMFILTER => 'ROW'}
0 row(s) in 0.6570 seconds

Hbase::Table - tsdb

create 'tsdb-tree',
  {NAME => 't', VERSIONS => 1, COMPRESSION => 'NONE', BLOOMFILTER => 'ROW'}
0 row(s) in 0.2670 seconds

Hbase::Table - tsdb-tree

create 'tsdb-meta',
  {NAME => 'name', COMPRESSION => 'NONE', BLOOMFILTER => 'ROW'}
0 row(s) in 0.5850 seconds

Hbase::Table - tsdb-meta
```

You only need to run this job once. It returns an error message if the tables already exist. You can continue the tutorial using existing tables, if present.

Note: Bigtable automatically performs data compression (https://cloud.google.com/bigtable/docs/overview#data_compression), so it disables user-configurable compression at an HBase level.

Data Model

The tables you just created will store data points from OpenTSDB. In a later step, you will write time-series data into these tables. Time-series data points are organized and stored as follows:

Field	Required	Description	Example
`metric`	Required	Item that is being measured - the default key	`sys.cpu.user`
`timestamp`	Required	Epoch time of the measurement	1497561091
`value`	Required	Measurement value	89.3

tags	At least one tag is required	Qualifies the measurement for querying purposes	`hostname=www`
			`cpu=0`
			`env=prod`

Deploying OpenTSDB

The rest of this tutorial provides instructions for making the sample scenario work. The following diagram shows the architecture you will use:

Diagram of the architecture used in this tutorial to write, read, and visualize time-series data.

This tutorial uses two Kubernetes deployments (https://kubernetes.io/docs/concepts/workloads/controllers/deployment/). One deployment sends metrics into OpenTSDB and the other reads from it. Using two deployments prevents long-running reads and writes from blocking each other. The pods in each deployment use the same container. OpenTSDB provides a daemon called `tsd` (http://opentsdb.net/docs/build/html/user_guide/cli/tsd.html) that runs in each container.

A single `tsd` process can handle a high throughput of events every second. To distribute load, each deployment in this tutorial creates 3 replicas of the read and write pods.

Create a deployment for writing metrics

The configuration information for the writer deployment is in `opentsdb-write.yaml` in the `deployments` folder of the tutorial repository. Use the following command to create it:

```
kubectl create -f deployments/opentsdb-write.yaml
```

Create a deployment for reading metrics

The configuration information for the reader deployment is in **`opentsdb-read.yaml`** in the **`deployments`** folder of the tutorial repository. Use the following command to create it:

```
kubectl create -f deployments/opentsdb-read.yaml
```

In a production deployment, you can increase the number of **`tsd`** pods running manually or by using autoscaling (https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/) in Kubernetes. Similarly, you can increase the number of instances in your GKE cluster manually or by using Cluster Autoscaler (https://cloud.google.com/kubernetes-engine/docs/cluster-autoscaler).

Creating OpenTSDB services

In order to provide consistent network connectivity to the deployments, you will create two Kubernetes services (https://kubernetes.io/docs/concepts/services-networking/service/). One service writes metrics into OpenTSDB and the other reads.

Create the service for writing metrics

The configuration information for the metrics writing service is contained in **`opentsdb-write.yaml`** in the **`services`** folder of the tutorial repository. Use the following command to create the service:

```
kubectl create -f services/opentsdb-write.yaml
```

This service is created inside your GKE cluster and is accessible to other services running in your cluster. In the next section of this tutorial, you write metrics to this service.

Note: You can expose the service to the rest of your network by using an internal load balancer (https://cloud.google.com/solutions/prep-kubernetes-engine-for-prod#connecting_to_a_cluster_from_inside_gcp)
or you can expose it to the internet

(https://cloud.google.com/solutions/prep-kubernetes-engine-for-prod#connecting_from_inside_a_cluster_to_external_services) by adding a LoadBalancer (https://kubernetes.io/docs/tasks/access-application-cluster/create-external-load-balancer/) in the service definition.

Create the service for reading metrics

The configuration information for the metrics reading service is contained in `opentsdb-read.yaml` in the `services` folder of the tutorial repository. Use the following command to create the service:

```
kubectl create -f services/opentsdb-read.yaml
```

Writing time-series data to OpenTSDB

There are several mechanisms to write data (http://opentsdb.net/docs/build/html/user_guide/writing/index.html) into OpenTSDB. After you define service endpoints, you can direct processes to begin writing data to them. This tutorial uses Heapster (https://github.com/kubernetes/heapster) to demonstrate writing data. Your Heapster deployment collects data about GKE and publishes metrics from the GKE cluster on which you are running OpenTSDB.

Use the following command to deploy Heapster to your cluster:

```
kubectl create -f deployments/heapster.yaml
```

Examining time-series data with OpenTSDB

You can query time-series metrics by using the `opentsdb-read` service endpoint that you deployed earlier in the tutorial. You can use the data in a variety of ways. One common option is to visualize it. OpenTSDB includes a basic interface to visualize metrics that it collects. This tutorial uses Grafana (https://grafana.com/), a popular alternative for visualizing metrics that provides additional functionality.

Set up Grafana

Running Grafana in your cluster requires a similar process to that you used to set up OpenTSDB. In addition to creating a ConfigMap and a deployment, you need to configure port forwarding (https://kubernetes.io/docs/tasks/access-application-cluster/port-forward-access-application-cluster/) so that you can access Grafana while it is running in your GKE cluster.

Use the following steps to set up Grafana

1. Create the Grafana `ConfigMap` using the configuration information in `grafana.yaml` in the `configmaps` folder of the tutorial repository.

   ```
   kubectl create -f configmaps/grafana.yaml
   ```

2. Create the Grafana deployment using the configuration information in `grafana.yaml` in the `deployments` folder of the tutorial repository.

   ```
   kubectl create -f deployments/grafana.yaml
   ```

3. Get the name of the Grafana pod in the cluster and use it to set up port forwarding.

   ```
   grafana=$(kubectl get pods --show-all --selector=app=grafana \
     --output=jsonpath={.items..metadata.name})

   kubectl port-forward $grafana 8080:3000
   ```

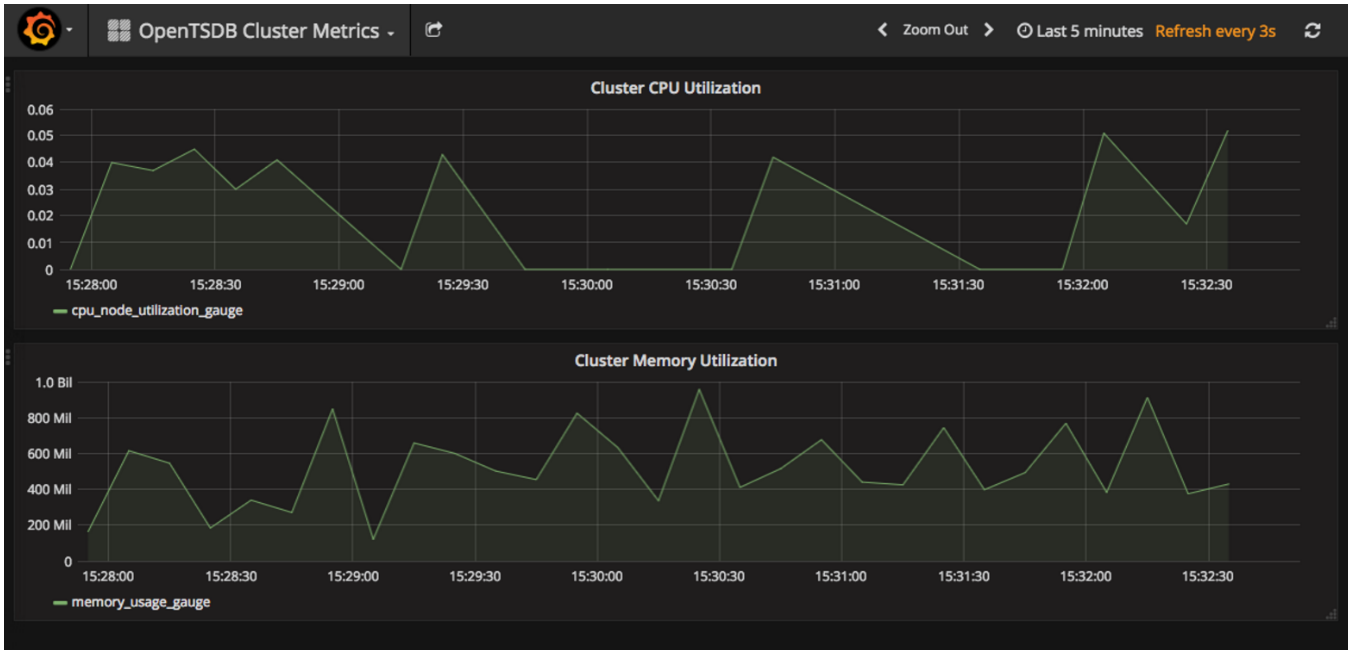
4. Verify that forwarding was successful. The output should match the following:

   ```
   Forwarding from 127.0.0.1:8080 -> 3000
   Forwarding from [::1]:8080 -> 3000
   ```

Connect to the Grafana web interface

In Cloud Shell, click **Web Preview** (https://cloud.google.com/shell/docs/using-web-preview) and then select **Preview on port 8080**.

A new browser tab opens and connects to the Grafana web interface. After a few moments, the browser displays graphs like the following:



This deployment of Grafana has been customized for this tutorial. The files `configmaps/grafana.yaml` and `deployments/grafana.yaml` configure Grafana to connect to the `opentsdb-read` service, allow anonymous authentication, and display some basic cluster metrics. A deployment of Grafana in production would implement the proper authentication mechanisms and use richer time-series graphs.

Cleaning up

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

1. Delete the GKE cluster to terminate all the artifacts previously created with the `kubectl create` command:

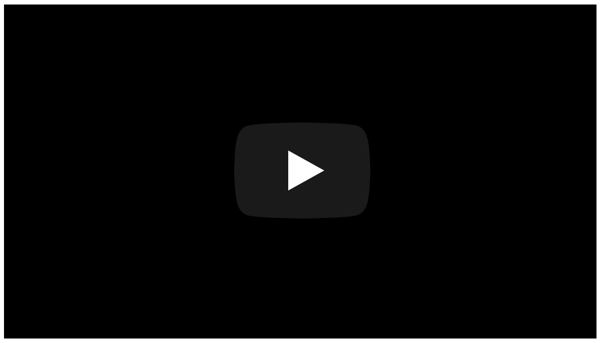
   ```
   gcloud container clusters delete opentsdb-cluster
   ```

   To delete the GKE cluster, confirm by typing `Y` or pressing **Enter**.

2. To delete the Bigtable cluster, click **Products & services** in Google Cloud Console. Click **Bigtable**, select the cluster that you created earlier, and click **Delete**.

What's next

- To learn how to improve the performance of your uses of OpenTSDB, consult Bigtable Schema Design for Time Series Data (https://cloud.google.com/bigtable/docs/schema-design-time-series).
- The video Bigtable in Action (https://www.youtube.com/watch?v=KaRbKdMInuc#t=37m15s) from Google Cloud Next 17 describes field promotion and other performance considerations.



- The documentation on cluster scopes (https://cloud.google.com/sdk/gcloud/reference/container/clusters/create#--scopes) for GKE clusters describes default scopes, such as Cloud Storage, and scopes you can add for other Google services.
- Try out other Google Cloud Platform features for yourself. Have a look at our tutorials (https://cloud.google.com/docs/tutorials).

Last updated November 19, 2019.