# Global Yugabyte Database Deployment into Google Cloud (Part 1)

Single database deployment into 37 Google Cloud regions

Christoph Bussler

Published in Google Cloud - Community

10 min read · May 30

## Goal

This is a first step in my journey to a global Yugabyte database deployment as database layer for a global distributed application.

The goal is to have a single global database with nodes in all regions of a cloud, here Google Cloud (currently there are 37 regions).

The main reasons are

- **Data residency requirements:** data has to reside in certain continents, economic zones, regions or even countries.
- **Consistent data globally:** all constraints, like schemas, tables, enumerations and terminology, or multiplicities, must be the same globally.
- **Global consistent analytics:** analytics must be possible not only in each region individually, but based on all available data globally.

There are many more aspects and requirements, however, the above are the most important drivers.

### A first setup

To start, I set up the deployment for a Yugabyte database with its nodes running in all Google Cloud regions — one node in each region.

This blog discusses a few items and observations made on that journey when installing Yugabyte directly on compute instances (and not using Yugabyte's managed offering).

### Note

The initial deployment I am discussing in this blog is not a production deployment as it forces transactions to be coordinated on a global cross-region basis incurring significant latency. Furthermore, one node in each region means that the region is offline if the node in the region becomes nonoperational. This is meant to be a starting point for architecture and deployment refinement like for example geo-partitioning.

I want to point out that Yugabyte has alternative offerings, including a managed service — I am not focusing on those offerings for the time being.

## Deployment

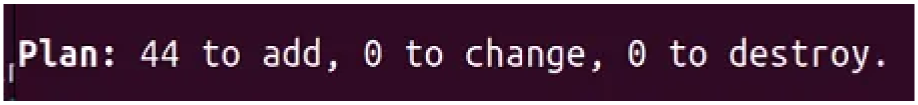
The deployment of the compute instances for Yugabyte is straight forward as outlined next.

### Terraform

I use Terraform for the setup of the virtual private cloud (VPC), firewall rules, project meta data and compute instances in Google Cloud. I created two Terraform scripts, one that contains all resources except those representing compute instances. A second script contains only the resources for the compute instances that host Yugabyte nodes. The second script is generated, as outlined next.

I follow a generative approach to Terraform to cover all regions by generating the compute instance Terraform resource specifications with a Python script based on a configurable list of regions and a resource template. The result is a Terraform file that only contains the compute instance resource specifications as created by the generator.

Applying the two Terraform scripts currently creates 44 resources (37 are compute instances) as indicated by its output:



```
Plan: 44 to add, 0 to change, 0 to destroy.
```

### Compute instances

Since this is initial phase to setup a global database, I select a small compute instance type: e2-medium. This is sufficient for development and sufficiently economical for my cloud bill as well.

The image on the compute instances is Ubuntu 22.04 suitable for a Yugabyte installation.

Google Cloud supports adding a compute instance startup script to a project's metadata and I use this approach to ensure that all compute instances have the same version of Yugabyte installed in the same location on the compute instance's file system following the exact same installation protocol.

### Naming convention for compute instances

The compute instances are named "yb-instance-" followed by the Google Cloud region. For example, "yb-instance-us-west2". Each compute instance runs one Yugabyte node.

This naming convention indicates by name the compute instance's zone and that it hosts a Yugabyte node.

### Steampipe

Steampipe is a great tool as it makes the resources in a cloud project accessible by SQL queries. The following shows a few queries against a full deployment.

```
> SELECT name FROM gcp.gcp_compute_instance;
+-------------------------------------+
| name                                |
+-------------------------------------+
| yb-instance-us-central1             |
| yb-instance-europe-central2         |
| yb-instance-europe-north1           |
| yb-instance-us-west4                |
| yb-instance-asia-northeast1         |
| yb-instance-southamerica-west1      |
| yb-instance-europe-west9            |
| yb-instance-asia-south1             |
| yb-instance-europe-west3            |
| yb-instance-southamerica-east1      |
| yb-instance-us-west1                |
| yb-instance-europe-west8            |
| yb-instance-europe-southwest1       |
| yb-instance-northamerica-northeast2 |
...
```

```
> SELECT count(*) FROM gcp.gcp_compute_instance;
+-------+
| count |
+-------+
| 37    |
+-------+
```

As you can see, Steampipe queries are a convenient way to query the resources in a Google Cloud project.

## Starting up

Once all Google Cloud resources are deployed starting up the Yugabyte nodes takes place.

### Yugabyte

Currently I use Yugabyte version v2.18.

### Operation scripts

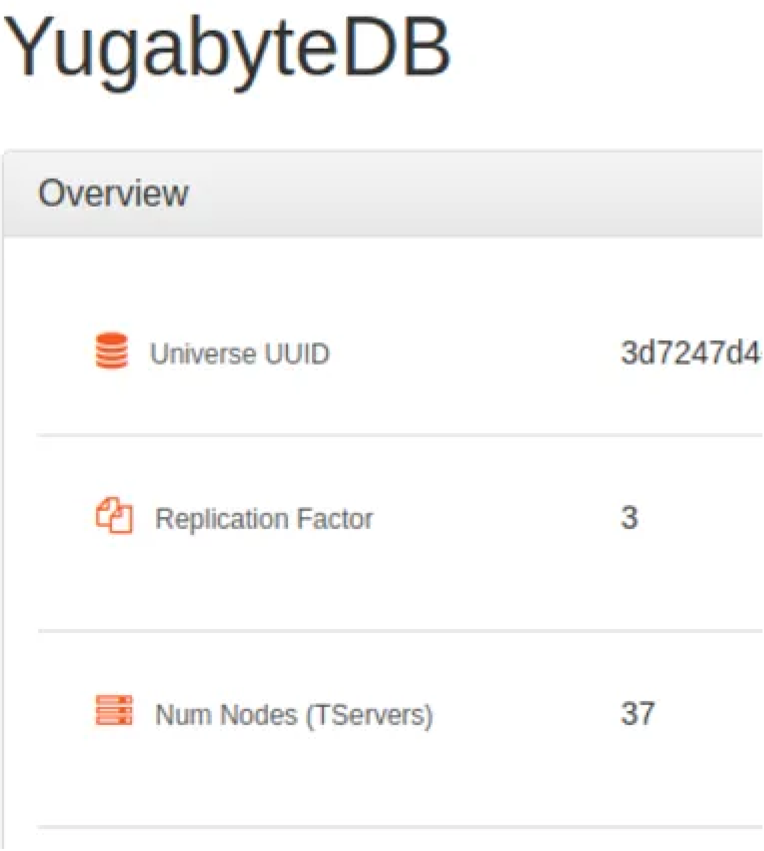
A startup script starts one Yugabyte master and one Yugabyte tserver on one compute instance in every regions.

As indicated earlier, a compute instance startup script installs the Yugabyte software when the compute instance is created. Starting a master and tserver is done by a separate operations bash script for now.

This can be improved by instead placing a Yugabyte configuration file on each compute instance so that starting up Yugabyte takes place automatically when the compute instance is started. I have not yet setup this approach right now but plan to. Once done I’ll report this approach in a future blog.

### Yugabyte web interface

Once Yugabyte masters and tservers are running, the Yugabyte administration web interface is available (for master by default on port 7000, and for tserver by default on port 9000). Here a screen shot of the master administration web interface as example:



YugabyteDB

Overview

Universe UUID 3d7247d4

Replication Factor 3

Num Nodes (TServers) 37

## Use case

As use case for this and future work I chose vehicle management. Vehicles are managed in all regions, and as physical objects they are local by default in the general case.

### Vehicle management

Yugabyte, based on PostgreSQL, is a multi-modal database. I start building the use case over time based on an initial table that has some relational columns, and a document-oriented column based on JSONB.

First, I create a database, a role and a schema. Afterwards I add a table and a few entries representing vehicles. For executing these operations I connect to the database in a region and execute the commands as shown next.

### Database, role and schema

The following shows a complete history of database queries and their results. The commands are executed in `yb-instance-us-west1`, as the prompt indicates.

1. Logging in to Yugabyte's query shell as `yugabyte`:

```
me@yb-instance-us-west1:/yb/yugabyte-2.18.0.0$ ./bin/ysqlsh -h 10.138.0.2 -p
5433 -U yugabyte
ysqlsh (11.2-YB-2.18.0.0-b0)
Type "help" for help.
```

2. Creating database and role:

```
yugabyte=# DROP
yugabyte-# DATABASE IF EXISTS global_management;
NOTICE: database "global_management" does not exist, skipping
DROP DATABASE
yugabyte=#
yugabyte=# CREATE
yugabyte-# DATABASE global_management;
CREATE DATABASE
```

```
yugabyte=# DROP USER IF EXISTS globmandev;
NOTICE: role "globmandev" does not exist, skipping
DROP ROLE
yugabyte=#
yugabyte=# CREATE USER globmandev WITH ENCRYPTED PASSWORD 'globmandev';
CREATE ROLE
yugabyte=# GRANT ALL PRIVILEGES ON DATABASE global_management TO globmandev;
GRANT
```

3. Logging in with newly created role:

```
yugabyte=# exit
me@yb-instance-us-west1:/yb/yugabyte-2.18.0.0$ ./bin/ysqlsh -h 10.138.0.2 -p
```

```
5433 -U globmandev -d global_management
ysqlsh (11.2-YB-2.18.0.0-b0)
Type "help" for help.
```

4. Create schema and table:

```
global_management=> DROP SCHEMA IF EXISTS vehicle_management;
NOTICE: schema "vehicle_management" does not exist, skipping
DROP SCHEMA
global_management=>
global_management=> CREATE SCHEMA vehicle_management;
CREATE SCHEMA
```

```
global_management=> DROP TABLE IF EXISTS
vehicle_management.vehicle_management;
NOTICE: table "vehicle_management" does not exist, skipping
DROP TABLE
global_management=>
global_management=> CREATE TABLE vehicle_management.vehicle_management
global_management-> (
global_management(> uuid UUID PRIMARY KEY,
global_management(> vin VARCHAR UNIQUE,
global_management(> properties JSONB
global_management(> );
CREATE TABLE
```

### Data

1. Insert data

```
global_management=> DELETE
global_management-> FROM vehicle_management.vehicle_management
global_management-> WHERE TRUE;
DELETE 0
global_management=>
global_management=> INSERT INTO vehicle_management.vehicle_management
global_management-> VALUES ('090b3de9-d506-4bc7-9b10-39d2f87d6a4d',
global_management(> 'ZHWGC5AU4BLA10126',
global_management(> '{
```

```
global_management'> "empty": true
global_management'> }');
INSERT 0 1
global_management=> INSERT INTO vehicle_management.vehicle_management
global_management-> VALUES ('39eddf36-0bc6-4f07-8396-6bb1dfcfbafa',
global_management(> 'ZFF74UFL0E0196736',
global_management(> '{
global_management'> "empty": true
global_management'> }');
INSERT 0 1
```

2. Query the inserted data:

```
global_management=> SELECT * FROM vehicle_management.vehicle_management;
                 uuid                 |        vin        |   properties
--------------------------------------+-------------------+-----------------
 090b3de9-d506-4bc7-9b10-39d2f87d6a4d | ZHWGC5AU4BLA10126 | {"empty": true}
 39eddf36-0bc6-4f07-8396-6bb1dfcfbafa | ZFF74UFL0E0196736 | {"empty": true}
(2 rows)
global_management=>
```

### Global fun

Let’s have some global fun. By that I mean I start over executing the database commands from above on an initial Yugabyte installation, however, this time from different regions to demonstrate that Yugabyte is synchronizing globally.

Yugabyte provides a helpful database function that helps me illustrating this demonstration: the ability to query the specific region the connection takes place. When executed it returns the specific region:

```
SELECT yb_server_region();
```

Moving on to our global fun, the following shows the order of database commands as well as the region they were executed in over time (going west):

```
us-west1 - create database
--------------------------

me@yb-instance-us-west1:/yb/yugabyte-2.18.0.0$ ./bin/ysqlsh -h 10.138.0.2 -p
5433 -U yugabyte
ysqlsh (11.2-YB-2.18.0.0-b0)
Type "help" for help.

yugabyte=# SELECT yb_server_region();
 yb_server_region
------------------
 us-west1
(1 row)

yugabyte=# DROP
yugabyte-#     DATABASE IF EXISTS global_management;
NOTICE:  database "global_management" does not exist, skipping
DROP DATABASE
yugabyte=#
yugabyte=# CREATE
yugabyte-#     DATABASE global_management;
CREATE DATABASE
yugabyte=#
```

```
southamerica-east1 - create role
--------------------------------

me@yb-instance-southamerica-east1:/yb/yugabyte-2.18.0.0$ ./bin/ysqlsh -h
10.158.0.2 -p 5433 -U yugabyteysqlsh (11.2-YB-2.18.0.0-b0)
Type "help" for help.

yugabyte=# SELECT yb_server_region();
  yb_server_region
--------------------
 southamerica-east1
(1 row)

yugabyte=# DROP USER IF EXISTS globmandev;
NOTICE:  role "globmandev" does not exist, skipping
DROP ROLE
yugabyte=#
yugabyte=# CREATE USER globmandev WITH ENCRYPTED PASSWORD 'globmandev';
CREATE ROLE
yugabyte=# GRANT ALL PRIVILEGES ON DATABASE global_management TO globmandev;
```

```
GRANT
yugabyte=#
```

```
europe-north1 - create schema
-----------------------------

me@yb-instance-europe-north1:/yb/yugabyte-2.18.0.0$ ./bin/ysqlsh -h
10.166.0.2 -p 5433 -U globmandev -d global_management
ysqlsh (11.2-YB-2.18.0.0-b0)
Type "help" for help.

global_management=> SELECT yb_server_region();
 yb_server_region
------------------
 europe-north1
(1 row)

global_management=> DROP SCHEMA IF EXISTS vehicle_management;
NOTICE:  schema "vehicle_management" does not exist, skipping
DROP SCHEMA
global_management=>
global_management=> CREATE SCHEMA vehicle_management;
CREATE SCHEMA
global_management=
```

```
me-central1 - create table
--------------------------

me@yb-instance-me-central1:/yb/yugabyte-2.18.0.0$ ./bin/ysqlsh -h 10.212.0.2
-p 5433 -U globmandev -d global_management
ysqlsh (11.2-YB-2.18.0.0-b0)
Type "help" for help.

global_management=> SELECT yb_server_region();
 yb_server_region
------------------
 me-central1
(1 row)

global_management=> DROP TABLE IF EXISTS
vehicle_management.vehicle_management;
NOTICE:  table "vehicle_management" does not exist, skipping
DROP TABLE
global_management=>
```

```
global_management=> CREATE TABLE vehicle_management.vehicle_management
global_management-> (
global_management(>     uuid       UUID PRIMARY KEY,
global_management(>     vin        VARCHAR UNIQUE,
global_management(>     properties JSONB
global_management(> );
CREATE TABLE
global_management=>
```

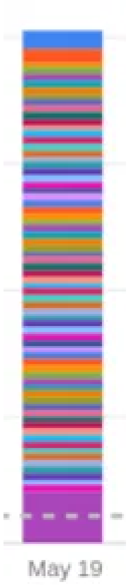
```
asia-northeast1 - insert data
-----------------------------

me@yb-instance-asia-northeast1:/yb/yugabyte-2.18.0.0$ ./bin/ysqlsh -h
10.146.0.2 -p 5433 -U globmandev -d global_management
ysqlsh (11.2-YB-2.18.0.0-b0)
Type "help" for help.

global_management=> SELECT yb_server_region();
 yb_server_region
------------------
 asia-northeast1
(1 row)

global_management=> DELETE
global_management-> FROM vehicle_management.vehicle_management
global_management-> WHERE TRUE;
DELETE 0
global_management=>
global_management=> INSERT INTO vehicle_management.vehicle_management
global_management-> VALUES ('090b3de9-d506-4bc7-9b10-39d2f87d6a4d',
global_management(>         'ZHWGC5AU4BLA10126',
global_management(>         '{
global_management'>           "empty": true
global_management'>         }');
INSERT 0 1
global_management=> INSERT INTO vehicle_management.vehicle_management
global_management-> VALUES ('39eddf36-0bc6-4f07-8396-6bb1dfcfbafa',
global_management(>         'ZFF74UFL0E0196736',
global_management(>         '{
global_management'>           "empty": true
global_management'>         }');
INSERT 0 1
global_management=>
```

```
australia-southeast2 - run query
--------------------------------

me@yb-instance-australia-southeast2:/yb/yugabyte-2.18.0.0$ ./bin/ysqlsh -h
10.192.0.2 -p 5433 -U globmandev -d global_management
ysqlsh (11.2-YB-2.18.0.0-b0)
Type "help" for help.

global_management=> SELECT yb_server_region();
   yb_server_region
----------------------
 australia-southeast2
(1 row)

global_management=> SELECT * FROM vehicle_management.vehicle_management;
                 uuid                 |        vin        |   properties
--------------------------------------+-------------------+-----------------
 090b3de9-d506-4bc7-9b10-39d2f87d6a4d | ZHWGC5AU4BLA10126 | {"empty": true}
 39eddf36-0bc6-4f07-8396-6bb1dfcfbafa | ZFF74UFL0E0196736 | {"empty": true}
(2 rows)

global_management=>
```

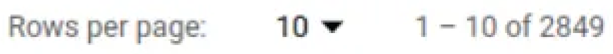
## Interesting tidbits

### Billing

I never thought billing can be interesting :-), however, when I reviewed the charges I thought the following two items are noteworthy. First, the color coding of a bar in the billing diagram indicates the variety of cloud resources:



What was even more unexpected to me was the huge number of SKUs that are involved when operating on a global scope:


Rows per page: 10 ▾ 1 – 10 of 2849

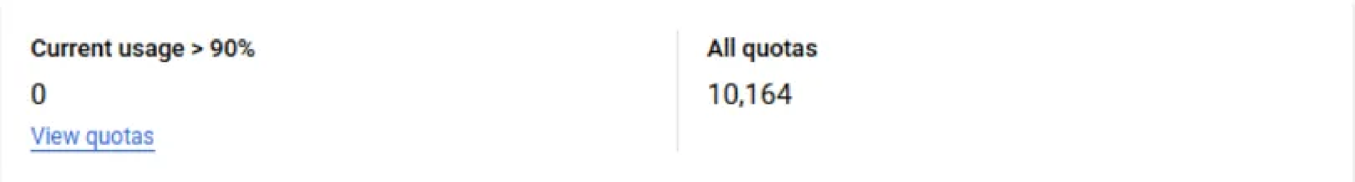
**Quota**

While ramping up the number of regions over time to get familiar with the Google Cloud behavior in combination with Yugabyte, I exceeded the default global compute CPU quota of currently 32 CPUs as I needed a few more to cover all regions:

```
│ Error: Error waiting for instance to create: Quota 'CPUS_ALL_REGIONS' exceeded.
│
│
│   with google_compute_instance.yb-instance-northamerica-northeast2,
│   on compute_instances.tf line 451, in resource "google_compute_instance" "yb-i
│  451: resource "google_compute_instance" "yb-instance-northamerica-northeast2"
```

I applied for quota extension with Google Cloud and after its approval I was able to finally create compute instances in all regions.

So you might wonder, why is this noteworthy? Well, in the quota extension application process I got aware of how many different types of quotas exist, and I did not expect this high number (10+k):



Current usage > 90%
0
View quotas

All quotas
10,164

### Stability

Both, Google Cloud and Yugabyte are extremely stable in my experience, no issue showed up so far. Google Cloud diligently creates and tears down compute instance as Terraform instructs it to do, and Yugabyte synchronizes globally — I did not have to change any of the default Yugabyte configuration settings.

Of course, there is no load on the system, and no dataset size to speak of right now. But having a stable basis is a good point to continue from.

## It's a canvas for future work

While the current status of my work is certainly not a deployment fit for production, this first step provides me with a canvas to work on further details, aspects, functionality and explore what it means to setup a global database in all regions of a cloud.

Some of the possible improvements are

- Store configuration files for starting up Yugabyte's master and tservers on compute instances so that the parameter settings are available on startup and restart
- Deploy database resources with Terraform using a Terraform provider like cyrilgdn
- Setup Yugabyte geo-partitioning for the use case to start managing data locally

These are some of the possibilities for improvements, more are on the list and even more emerge over time.

Google Cloud Yugabyte Global Database Data Google Cloud Platform

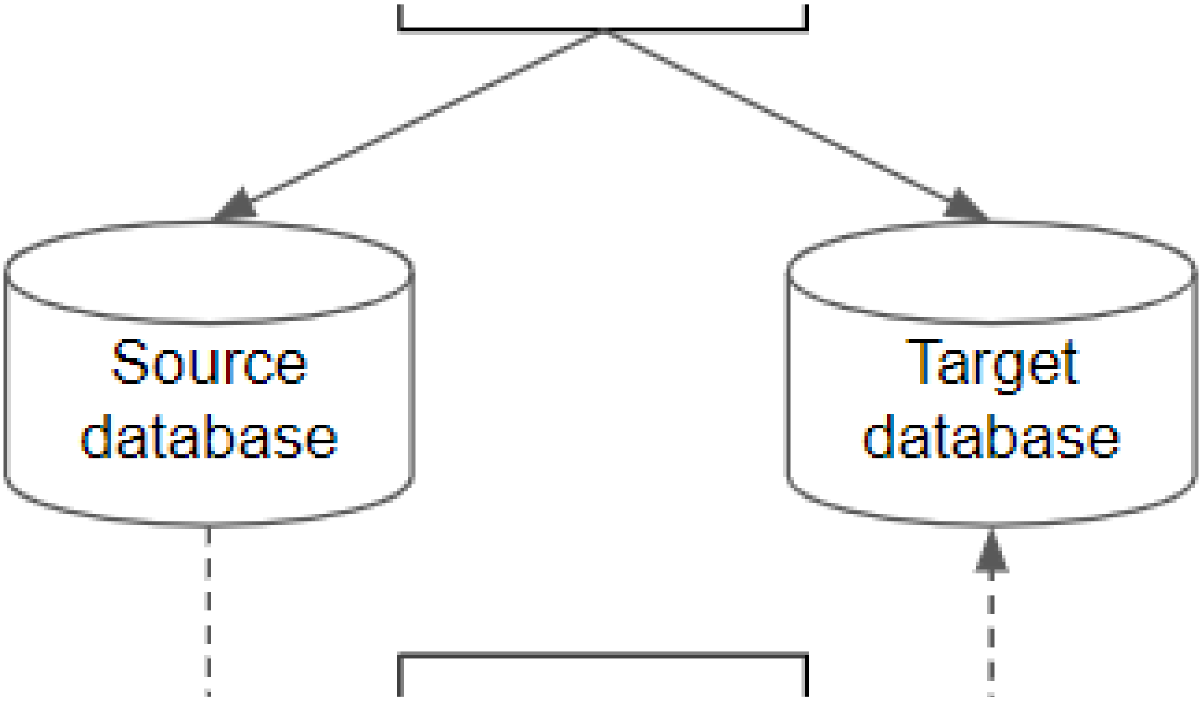
Edit profile

## Written by Christoph Bussler

111 Followers · Writer for Google Cloud - Community

www.real-programmer.com

### More from Christoph Bussler and Google Cloud - Community



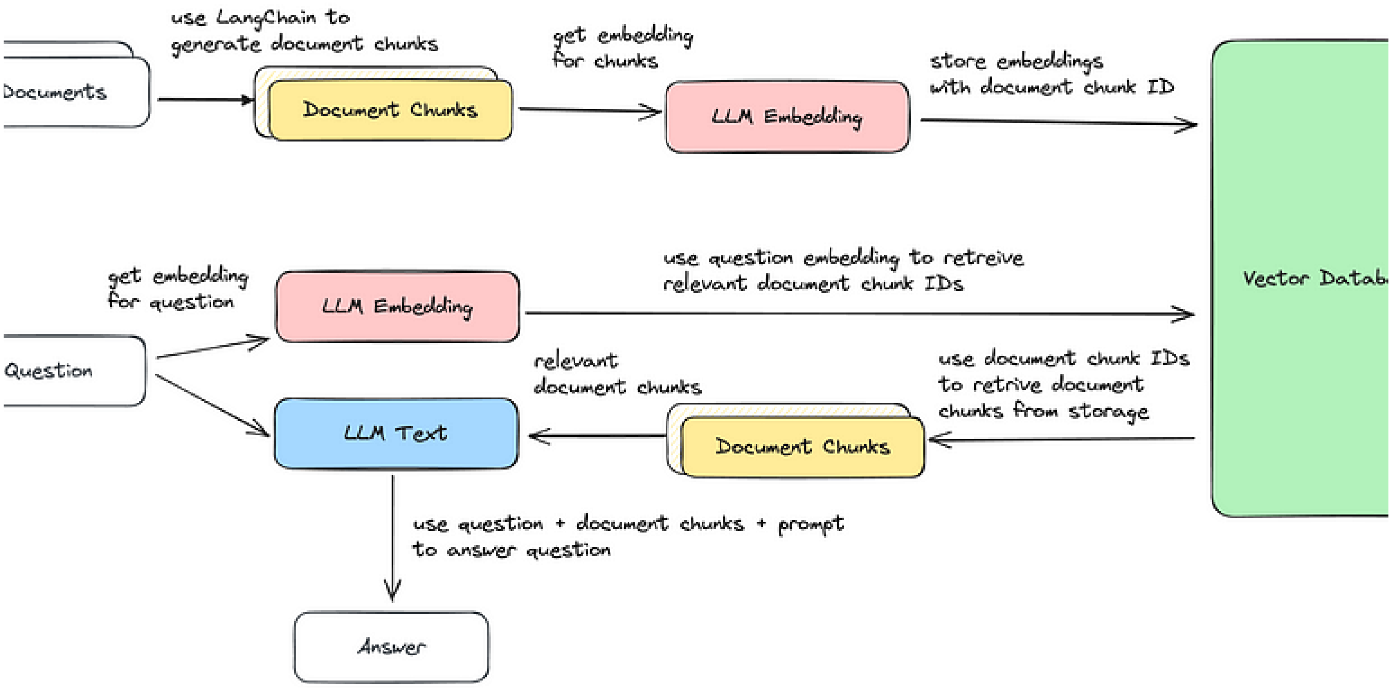
Christoph Bussler in Google Cloud - Community

### Online Database Migration by Dual-Write: This is not for Everyone

(to be more precise: for almost no-one)

47 min read · Jun 23, 2020

50 2



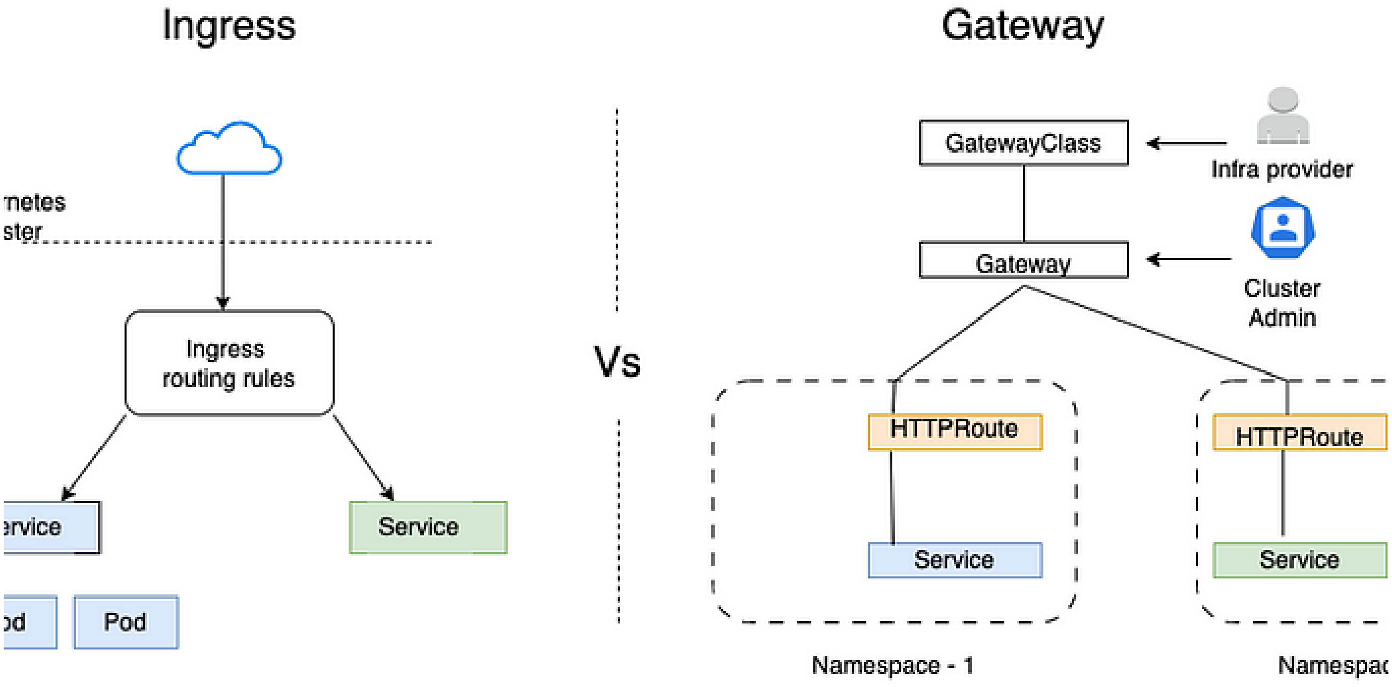
Sascha Heyer in Google Cloud - Community

## Generative AI - Document Retrieval and Question Answering with LLMs

Apply LLMs to your domain-specific data

· 7 min read · Jun 4

97 4



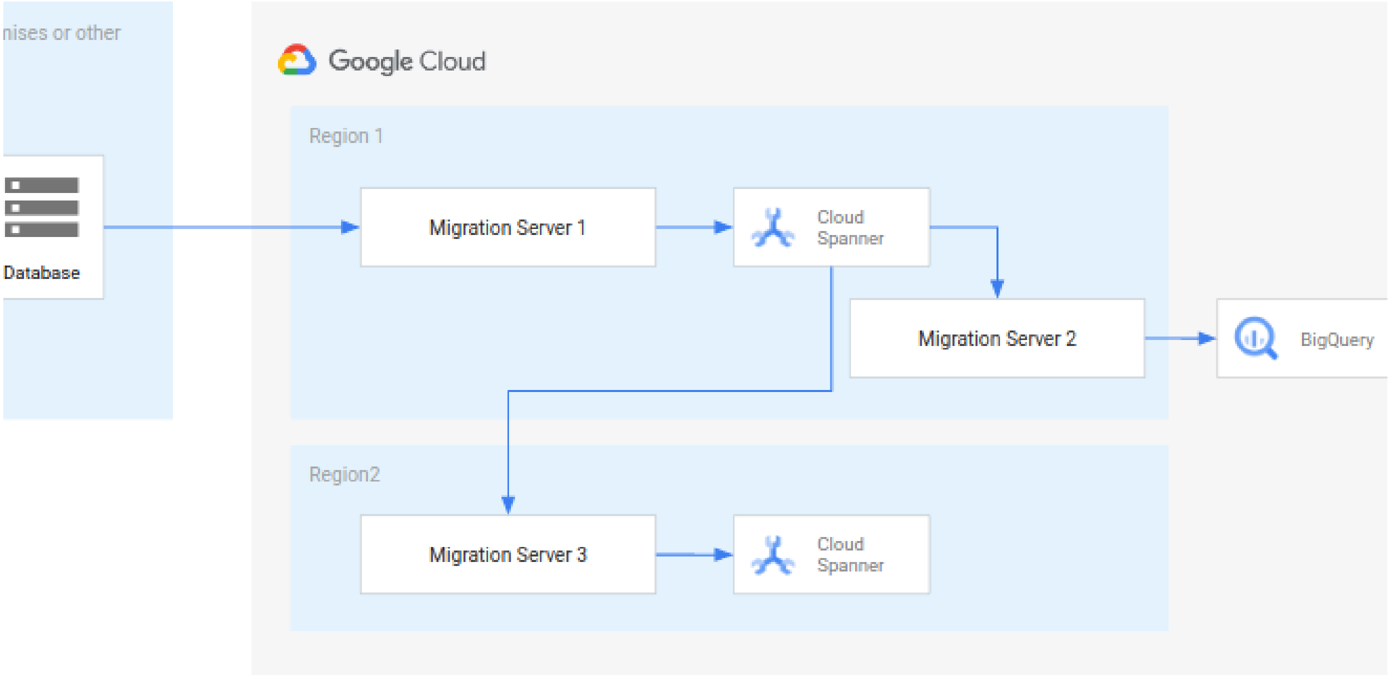
Harsh Manvar in Google Cloud - Community

## Kubernetes Ingress Vs Gateway API

Understanding the Differences between Kubernetes Ingress and Gateway API for Effective Traffic Management

5 min read · Feb 22

356 1



Christoph Bussler in Google Cloud - Community

## Zero downtime database migration and replication to and from Cloud Spanner

by Christoph Bussler, Szabolcs Rozsnyai

12 min read · Jul 13, 2020

64

See all from Christoph Bussler

See all from Google Cloud - Community

## Recommended from Medium

Nico Hein in Apache Airflow

## Running a Multi-Tenant Airflow Cluster

Apache Airflow is a versatile and well maintained open source tool for orchestrating big data and analytics workloads. In this blog post…

7 min read · Jun 21

55

Trinquier Vannick in Google Cloud - Community

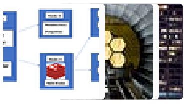
## Evaluating your existing GCP resources

How to build a service to validate existing GCP resources from a CAI export

9 min read · Jun 20

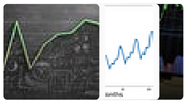
9

## Lists



### New_Reading_List

173 stories · 6 saves

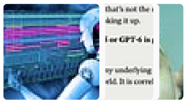


### Predictive Modeling w/ Python

18 stories · 39 saves

### General Coding Knowledge

20 stories · 29 saves



### Now in AI: Handpicked by Better Programming

248 stories · 14 saves

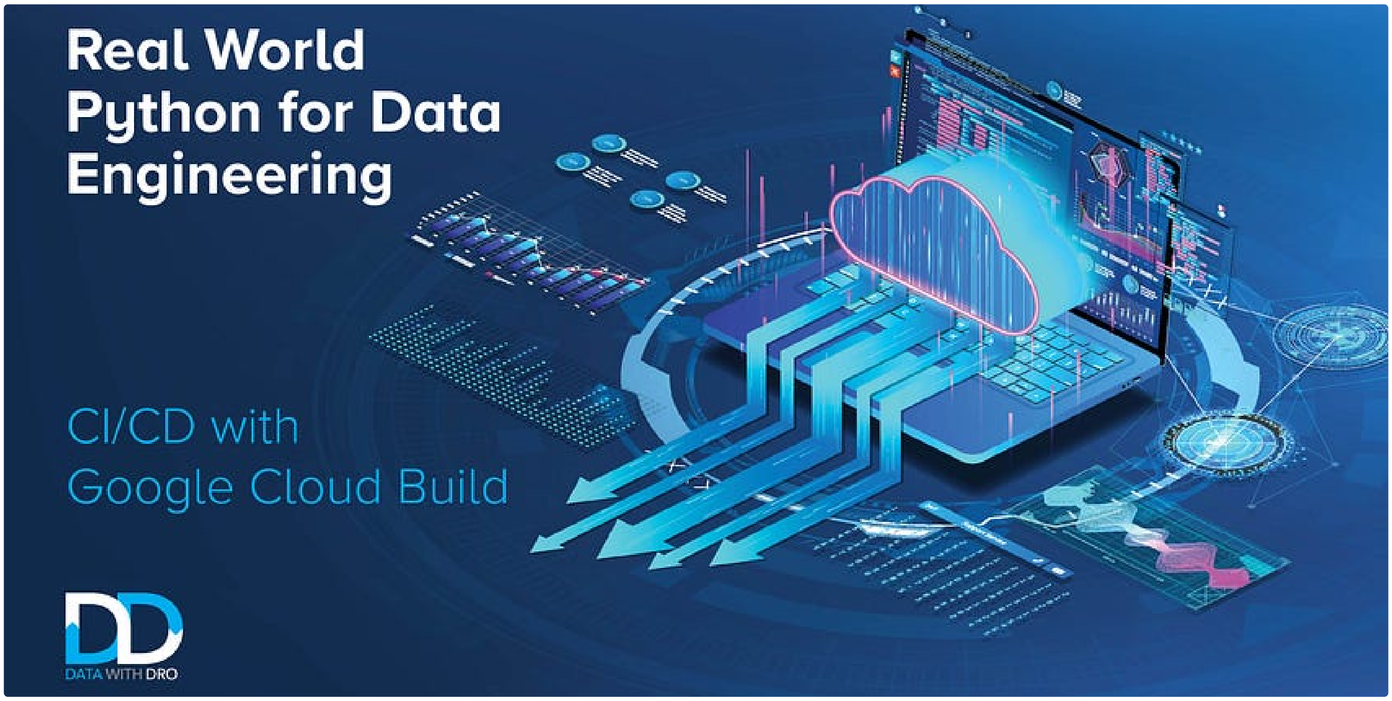
François Blanchard

## How to Correctly Use BigQuery LAST_VALUE

I recently had to use the useful navigation BigQuery function LAST_VALUE, but the result was not what I expected.

3 min read · 5 days ago

6 1



Danilo Drobac

### 4) CI/CD with Google Cloud Build

Implement DevOps best practices with Continuous Integration and Continuous Deployment using Google Cloud Build

11 min read · Dec 28, 2022

3

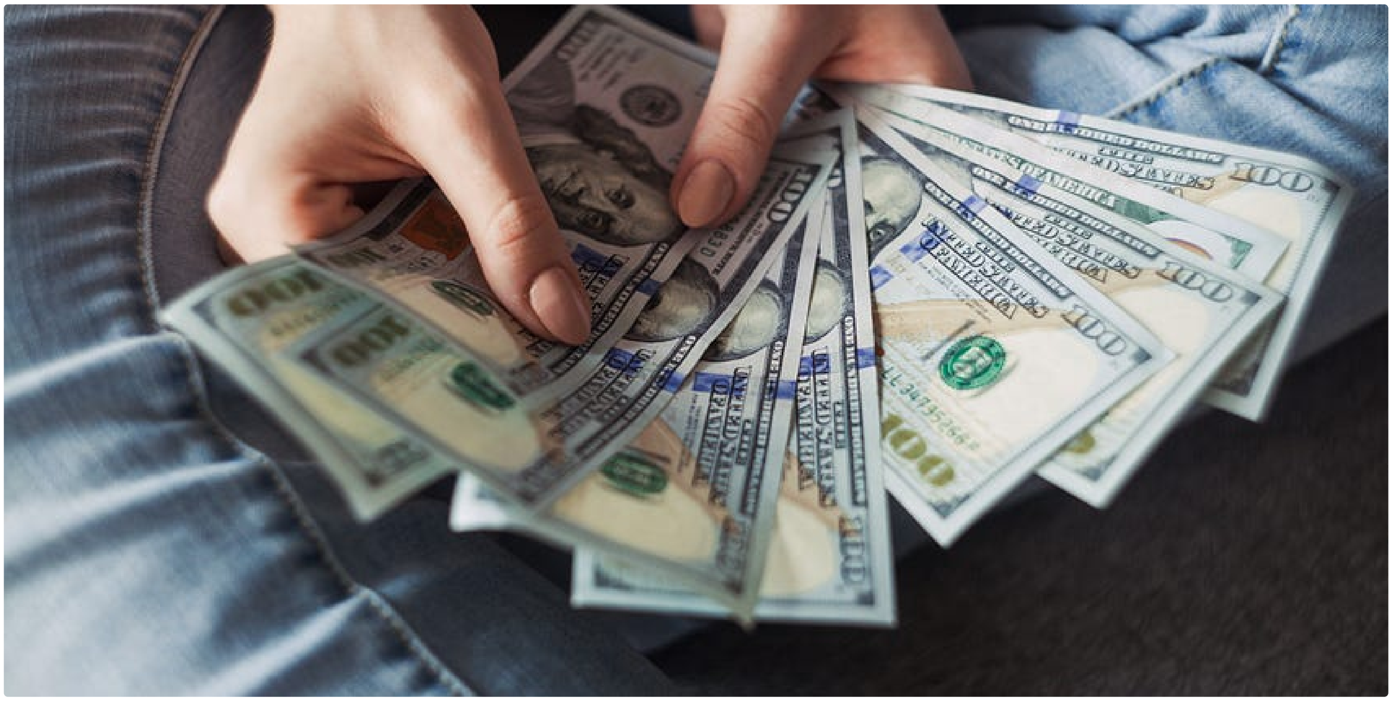
Arslan Mirza in Level Up Coding

## How to Master Serverless Computing: Best Practices for Google Cloud Functions

Maximizing Efficiency, Security, and Scalability for Your Serverless Applications

✦ · 10 min read · Mar 12

105 1



Vincent Junitio Ungu in Blibli.com Tech Blog

### Estimate Your BigQuery Storage Cost

Have you been using BigQuery as a data warehouse to retrieve data using Structured Query Language (SQL)?

6 min read · Jun 12

4

See more recommendations