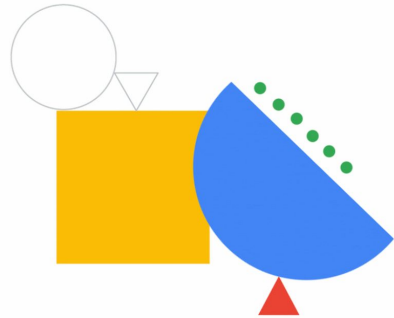


Integrating with Cloud Databases



In this module, we discuss how to you can integrate your functions with cloud databases.

You learn how to securely connect your functions to read and write data from and to Firestore and Memorystore databases in Google Cloud.

Agenda

01	Connecting Cloud Functions to Memorystore
02	Using environment variables
03	Connecting Cloud Functions to Firestore
04	Using secrets with Cloud Functions
05	Lab: Integrating Cloud Functions with Firestore
06	Quiz



In this module, you learn about connecting Cloud Functions to:

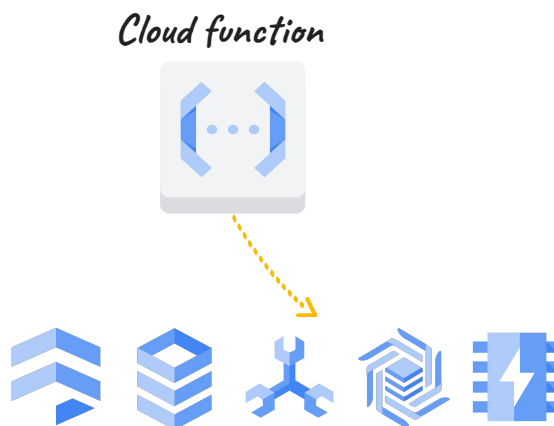
- Memorystore
- Firestore

We also discuss how Cloud Functions can use environment variables, and access secrets stored in Secret Manager.

You'll complete a lab on integrating Cloud Functions with Firestore, and also do a short quiz on the topics in this module.

Integrate Cloud Functions with Cloud databases

- Firestore
- Cloud SQL
- Cloud Spanner
- Cloud Bigtable
- Memorystore (in-memory cache service)



Google Cloud

You can integrate Cloud Functions with these cloud databases:

- Firestore
- Cloud SQL
- Cloud Spanner
- Cloud Bigtable

and with Memorystore, Google Cloud's in-memory cache service. In this module, we discuss how you can connect your Cloud Functions to Memorystore and Firestore.

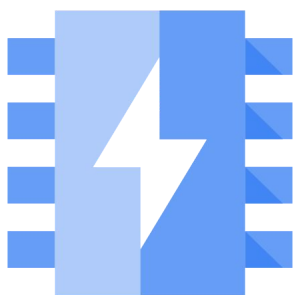
For information about connecting to other Cloud databases, refer to the documentation:

[Connect Cloud Functions to Cloud SQL](#)

[Connect Cloud Functions to Cloud Bigtable](#)

[Connect Cloud Functions to Cloud Spanner](#)

Memorystore



- ✓ Fully managed in-memory cache solution
- ✓ Supports [Redis](#) and [Memcached](#)
- ✓ Automates provisioning, replication, failover, and patching
- ✓ Integrated with IAM and Cloud Monitoring

Google Cloud

Memorystore is a Google Cloud service that provides a highly available, scalable, and secure in-memory cache solution for Redis and Memcached.

It's a fully managed service that automates provisioning, replication, failover, and patching.

It's also integrated with IAM for secure access, and with Cloud Monitoring for service monitoring and alerting.

For a complete list of features, see the [Memorystore documentation](#).

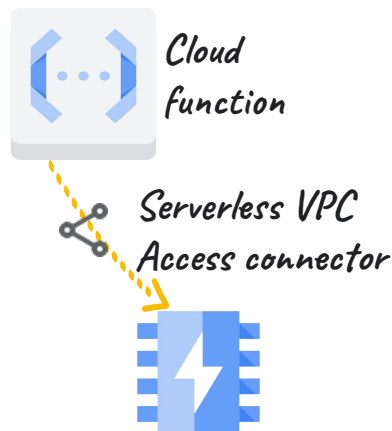
Redis is an open source in-memory data structure store used as a database, cache, message broker, and streaming engine.

Memcached is an open source distributed memory object caching system.

Connect Cloud Functions to Memorystore (Redis)

To access Memorystore from Cloud Functions:

- Determine your Redis instance's authorized VPC network.
- Create a Serverless VPC Access connector.
- Attach the connector to the Redis instance's authorized VPC network.

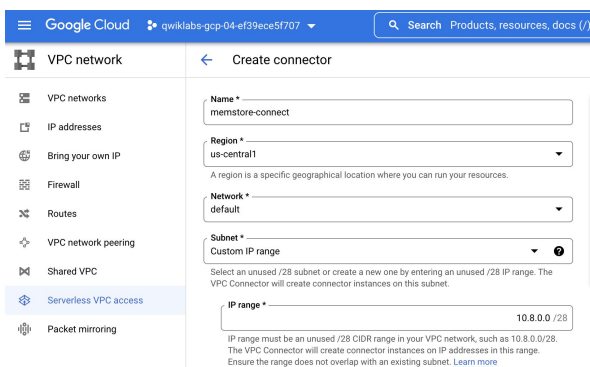


You can connect to a Redis instance from Cloud Functions by using Serverless VPC Access.

Once you've determined your Redis instance's authorized VPC network, you create a Serverless VPC Access connector in the same region as your function.

Then, attach the connector to the Redis instance's authorized VPC network.

Creating the VPC connector



The screenshot shows the Google Cloud console interface for creating a VPC connector. The left sidebar lists various VPC network services, with 'Serverless VPC access' highlighted. The main panel is titled 'Create connector' and contains the following fields:

- Name ***: A text input field containing 'memstore-connect'.
- Region ***: A dropdown menu set to 'us-central1'.
- Network ***: A dropdown menu set to 'default'.
- Subnet ***: A dropdown menu set to 'Custom IP range'.
- IP range ***: A text input field containing '10.8.0.0/28'.

Below the IP range field, there is a note: 'IP range must be an unused /28 CIDR range in your VPC network, such as 10.8.0.0/28. The VPC Connector will create connector instances on IP addresses in this range. Ensure the range does not overlap with an existing subnet. [Learn more](#)'.

gcloud CLI

```
gcloud compute networks vpc-access
connectors create CONNECTOR_NAME \
--network VPC_NETWORK \
--region REGION \
--range IP_RANGE
```

```
gcloud compute networks vpc-access
connectors describe CONNECTOR_NAME \
--region REGION
```

Specify the network, region, and IP address range when the VPC connector is created.

Ensure that the connector is in a Ready state before using it.

Connect Cloud Functions to Memorystore (Redis)

Deploy the function:

- Specify the path and name of the connector.
- Set environment variables for the function code to connect to the Redis host and port.

gcloud CLI

```
gcloud functions deploy visit_count \
--runtime python37 \
--trigger-http \
--region [REGION] \
--vpc-connector
projects/[PROJECT_ID]/locations/[REGI
ON]/connectors/[CONNECTOR_NAME] \
--set-env-vars
REDISHOST=[REDIS_IP],REDISPORT=[REDIS_
PORT]
```

Deploy the function specifying the path and name of the connector, and environment variables for the Redis host IP address and port.

Your function code can then use these environment variables to instantiate a client that connects to the Redis service.

Invoke the function

To invoke the function:

- Send an HTTP GET request to the function url.

gcloud CLI

```
curl -H "Authorization: bearer $(gcloud  
auth print-identity-token)" \  
  
https://[REGION]-[PROJECT_ID].cloudfunc  
tions.net/visit_count
```

Function source code on GitHub.

To invoke the function, send an HTTP GET request to the functions URL endpoint.

The [source code](#) for this function can be found on GitHub.

Environment variables

gcloud CLI

```
gcloud functions deploy FUNCTION_NAME  
--set-env-vars F00=bar,BAZ=boo  
FLAGS...
```

```
# from a file  
gcloud functions deploy FUNCTION_NAME  
--env-vars-file env.yaml FLAGS...
```

```
# contents of file env.yaml
```

```
F00: bar  
BAZ: boo
```

Cloud Functions environment variables are:

- Key-value pairs.
- Set up during function deployment.
- Accessed by function code at runtime.
- Stored in the Cloud Functions backend.
- Bound or scoped to a single function.
- Added, updated, or removed using the Google Cloud console or the gcloud CLI.

You set environment variables for Cloud Functions at deployment time. These variables are accessible by your function code at runtime, or as configuration information for the buildpack system.

They are stored in the Cloud Functions backend, are bound to a single function, and exist within the same function lifecycle.

You can provide the environment variable key-value pairs when you deploy your function with the gcloud CLI or in the Google Cloud console.

You can also store them in a YAML file in source control and provide the name of the file during function deployment.

For more information on using environment variables with Cloud Functions, see the [documentation](#).

Retrieving environment variables in your function

Python

```
import os

def process(request):
    foo_var = os.environ.get('F00',
    'Specified environment variable is not
    set.')

    return 'F00: {}'.format(foo_var)
```

To access runtime environment variables in your function:

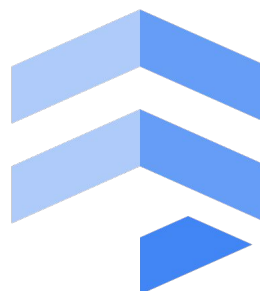
- Use the `os` module in Python.
- Use the `process.env` property in Node.js.

To access runtime environment variables in your function written in Python, use the `os` module.

To access environment variables in other language runtimes, view the [samples](#) in the documentation.

Firestore

- ✓ Fully managed serverless NoSQL document database
- ✓ Store documents containing key-value pairs
- ✓ Provides high availability, scalability, and multi-region replication



Google Cloud

Firestore is a Google Cloud service that provides a fully managed serverless NoSQL document database.

It provides high availability, scalability with no maintenance windows or downtime required, and multi-region replication.

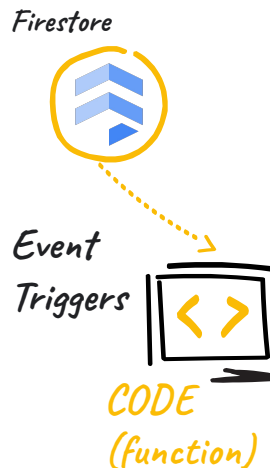
You store a set of key-value pairs known as a document in Firestore and all documents are stored in collections.

For a complete list of features, see the [Firestore documentation](#).

Extend Firestore with Cloud Functions

Use Cloud Functions to extend Firestore's capabilities:

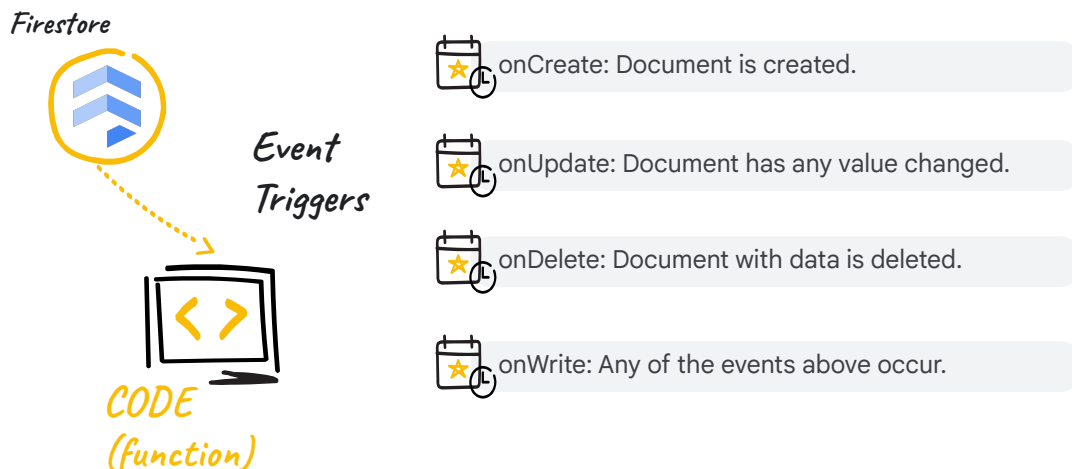
- ✓ Handle events that are triggered by changes in the Firestore database.
- ✓ Implement server-side function code for your app without running your own servers.



You can extend Firestore's capabilities with Cloud Functions by handling events that are triggered by changes in your Firestore database.

By implementing function code to handle these events, you can easily add server-side functionality to your application without running your own servers.

Firestore function triggers



You can implement your function code to handle Firestore events that occur when a document is created, updated, or deleted, or when any of these events occur.

These events are exposed by the Cloud Functions for Firebase SDK that you use in your function source code.

Firestore triggers for Cloud Functions are available only for [Firestore in Native mode](#). It is not available for Firestore in Datastore mode.

Writing functions for Firestore events

- To trigger a function, specify:

- a document path
- an event type

- A document path can reference:

- A specific document
- A wildcard pattern

Node.js

*Event
type*

*Document
path*

```
// Listen for changes
in document `john` in
the `users` collection.
```

```
exports.changeUser =
functions.firestore
```

```
.document('users/john')
.onWrite((change,
context) => {
  // ... Your code here
});
```

```
// Listen for changes in
all documents in the
`users` collection.
```

```
exports.createUser =
functions.firestore
```

```
.document('users/{userId}')
.onCreate((change, context)
=> {
  // ... Your code here
});
```

Functions that are triggered on Firestore events must specify a document path and an event type.

The document path can reference a specific document or a wildcard pattern, and must not contain a trailing slash.

Reading and writing Firestore data

Node.js

```
exports.updateUser =  
functions.firestore  
.document('users/{userId}').onUpdate((change, context)  
=> {  
  const newValue = change.after.data();  
  const previousValue = change.before.data();  
  return change.after.ref.set(...);  
});
```

Document data after update

Document data before update

Document reference

Google Cloud

When a function is triggered, a snapshot of the data related to the event is available to the function. You can use this snapshot to read from or write to the document that triggered the event. The document snapshot data before and after the update is available to the function.

Each function invocation is associated with a specific document in your Firestore database. You can access that document as a `DocumentReference` in the `ref` property of the snapshot returned to your function.

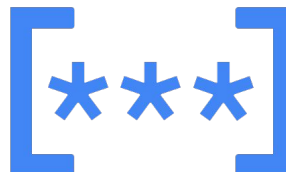
This `DocumentReference` comes from the [Firestore Node.js SDK](#) and includes methods that enable you to modify the document that triggered the function.

You can also read and write data of documents other than the one that triggered the function using the [Firebase Admin SDK](#).

Secrets

A secret is an object that contains:

- A collection of metadata that includes:
 - Replication locations
 - Labels
 - Permissions
- Secret versions that store the secret data.



Secret Manager:

- A service that enables you to store, manage, and access secrets.

When your function code needs to access certain databases or APIs, it will need to provide credentials like a database username/password or API key.

This kind of sensitive information should be securely stored and accessed by the function when it executes.

You can use secrets and Google Cloud's [Secret Manager](#) to store this information.

A secret is an object that contains a collection of metadata like replication locations, labels, permissions, and other information; and the secret versions.

A secret version stores the actual secret data such as an API key or password as a text string or binary blob.

To create and manage secrets, you must enable the Secret Manager API.

Accessing secrets from Cloud Functions

To access a secret from your function:

- Grant the function's runtime service account `roles/secretmanager.secretAccessor` role on the secret.
- Make the secret available to the function as:
 - A file by mounting the secret as a volume.
 - An environment variable.

gcloud CLI

```
# secret mounted as a volume

gcloud functions deploy FUNCTION_NAME
--runtime RUNTIME --set-secrets \
'SECRET_FILE_PATH=SECRET:VERSION'

# secret passed as an environment
variable

gcloud functions deploy FUNCTION_NAME
--runtime RUNTIME --set-secrets \
'ENV_VAR_NAME=SECRET:VERSION'
```

Google Cloud

To access a secret, your function must first be granted access to the secret. This is achieved by granting the appropriate role (`roles/secretmanager.secretAccessor`) to the function's runtime service account on the secret.

To make the secret available to your function:

- Mount the secret as a volume so that the function can access it from a file, or
- Pass the secret to the function as an environment variable.

Mounting the secret as a volume allows your function to reference the latest version of the secret each time the file is read.

Because environment variables are resolved at function instance startup time, with this method, your function would have access to a specific version of the secret.

Accessing secrets from a different project

- Grant the function's runtime service account `roles/secretmanager.secretAccessor` role on the secret.
- Make the secret available to the function by specifying the secret's resource path with the format:
`projects/PROJECT_ID/secrets/SECRET_NAME`

gcloud CLI

```
# secret in a different project
mounted as a volume

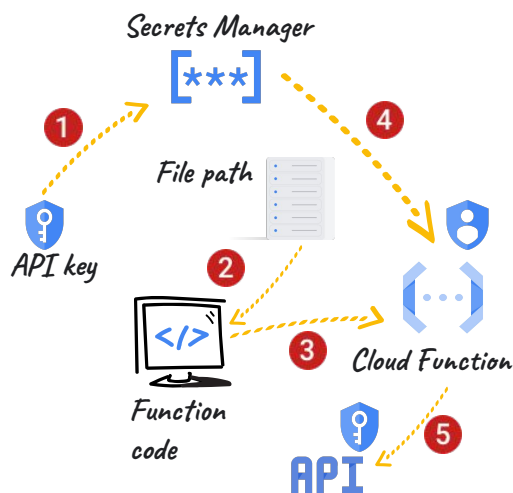
gcloud functions deploy FUNCTION_NAME
--runtime RUNTIME --set-secrets
'SECRET_FILE_PATH=SECRET_RESOURCE_PATH
:VERSION'
```

To provide your function access to a secret that is in a different project than your function:

- Grant your function's runtime service account access to the secret as previously described.
- Make the secret available to the function by specifying the secret's resource path that includes the project ID and secret name.

Using a secret

- 1 Use Secrets Manager to create and store a secret whose value is an API key.
- 2 Write your function code to access the secret.
- 3 Deploy the function setting the secret and the access method.
- 4 To access the secret, grant access to the function's runtime service account.
- 5 Call the API with the API key.



Google Cloud

A sample use case of using a secret is when your function must access an external service or API, that requires an API key to identify the calling application.

The key is considered a sensitive piece of information so it is stored as a secret using Secrets Manager.

Write your function code to access the secret by reading the contents of a file or environment variable. The file path is provided when the function is deployed.

When deploying the function, provide the secret name and the method to access the secret. The method can be a mounted file path or environment variable.

To access the secret, the function's runtime service account must be granted the *Secret Manager Secret Accessor* role on the secret.

Call the API from the function with the secret value which is the API key.

Lab



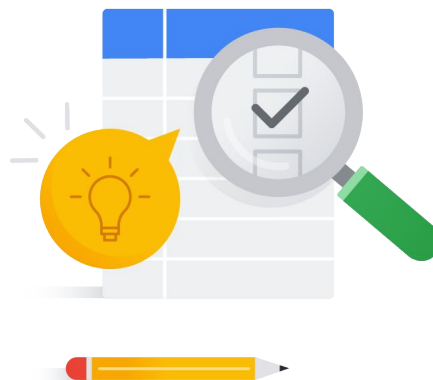
45 min



Individual

Integrating Cloud Functions with Firestore

Develop a function to read, write, and update data in a Firestore database.



In this lab, you develop a Cloud function to read and write a document to a Firestore database, and update that document by responding to create events in Firestore.

Lab Instructions



45 min



Individual



Tasks

- Set up Firestore by creating a Firebase project and database.
- Write an HTTP function that creates a document entity in the Firestore database.
- Write an event-driven function that is triggered when a document is created in the Firestore database.
- Access a secret from a cloud function.

1

Set up Firestore.

2

Develop an HTTP function for Firestore.

3

Develop an event-driven function for Firestore.

4

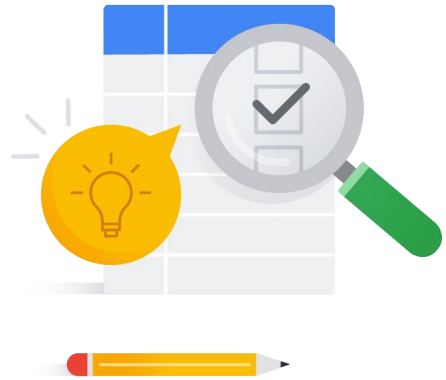
Use Secrets with Cloud Functions.

Quiz

5 min

Group

Integrating with Cloud Databases



Let's do a short quiz on this module.

Quiz | Question 1

Question

Which three statements about using environment variables with Cloud Functions are correct?

Environment variables:

- A. Are provided during function deployment.
- B. Can be shared and used by multiple functions.
- C. Can be stored in a YAML file, whose filename can be provided during function deployment.
- D. Are key-value pairs that are accessed by the function code at runtime.
- E. Cannot be added, updated, or removed.

Quiz | Question 2

Question

Which statements about triggering Cloud Functions from Firestore database events are correct? Select two.

- A. A function cannot be triggered when a Firestore document is deleted.
- B. A function can be triggered when a Firestore document is created, updated, or deleted.
- C. A function can be triggered when a specific field in a Firestore document is changed.
- D. To trigger a function, you must specify an event type and the document path.
- E. A function can only be triggered on changes to an individual Firestore document.

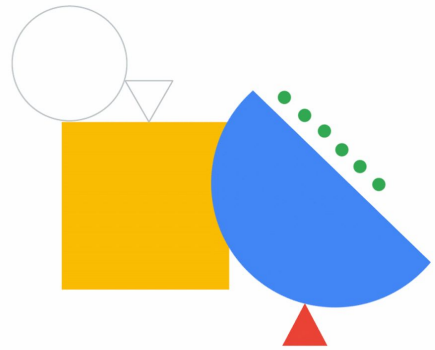
Quiz | Question 3

Question

What are two methods of making a secret available to a cloud function?

- A. Provide the secret name and value as query parameters to the function.
- B. Provide the secret as an environment variable when deploying the function.
- C. Mount the secret as a volume so that the function can access the secret from a file.
- D. Secrets cannot be accessed from Cloud Functions.

Review: Integrating with Cloud Databases



In this module, you learned how to connect Cloud Functions to Memorystore, and Firestore.

We discussed how Cloud Functions can use environment variables, and access secrets stored in Secret Manager.

You also completed a lab on integrating Cloud Functions with Firestore.