

Google Cloud

Partner Certification Academy



# Professional Cloud Developer

pls-academy-pcd-student-slides-2-2309

The information in this presentation is classified:

## **Google confidential & proprietary**

⚠ This presentation is shared with you under NDA.

- Do **not** record or take screenshots of this presentation.
- Do **not** share or otherwise distribute the information in this presentation with anyone **inside** or **outside** of your organization.

Thank you!



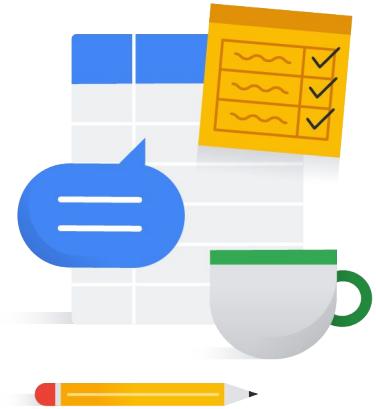
Google Cloud

## Session logistics

- When you have a question, please:
  - Click the Raise hand button in Google Meet.
  - Or add your question to the Q&A section of Google Meet.
  - Please note that answers may be deferred until the end of the session.
- These slides are available in the Student Lecture section of your Qwiklabs classroom.
- The session is **not recorded**.
- Google Meet does not have persistent chat.
  - If you get disconnected, you will lose the chat history.
  - Please copy any important URLs to a local text file as they appear in the chat.

## Program issues or concerns?

- For questions regarding Cloud Skills Boost access, Qwiklabs issues, voucher queries, etc.
  - [cloud-partner-training@google.com](mailto:cloud-partner-training@google.com)
- For questions regarding Partner Advantage access
  - <https://support.google.com/googlecloud/topic/9198654>



Google Cloud

# Partner Certification Academy

A differentiated learning experience for the busy professional



Our goal is to help you prepare for Google Cloud certification exams

These programs may include:

- On-demand learning
- Self-paced labs
- Mentor-led workshops
- A voucher for the exam

The workshop sessions:

- **Are NOT training sessions - that's the purpose of the on-demand content.**
- Help you review key concepts on the exam guide.
- Will NOT discuss actual exam questions.

# Professional Cloud Developer (PCD)

A Professional Cloud Developer builds scalable and highly available applications using Google-recommended tools and best practices. This individual has experience with cloud-native applications, developer tools, managed services, and next-generation databases. A Professional Cloud Developer also has proficiency with at least one general-purpose programming language and instruments their code to produce metrics, logs, and traces.

The Professional Cloud Developer exam assesses your ability to:

- ✓ Design highly scalable, available, reliable cloud-native applications
- ✓ Deploy applications
- ✓ Manage deployed applications
- ✓ Build and test applications
- ✓ Integrate Google Cloud services

<https://cloud.google.com/learn/certification/cloud-developer>



Google Cloud

## Experience level assumptions

- Per the [certification site](#): “*3+ years of industry experience including 1+ years designing and managing solutions using Google Cloud*”
- Workshops do not cover “programming 101” topics
  - Discussion centers on topics specific to Google Cloud services, e.g.,
    - Authentication and authorization
    - Best practices, etc.
- If you lack experience with Google Cloud
  - Additional content is provided as part of this program which provides a basic overview on Google Cloud compute and storage options

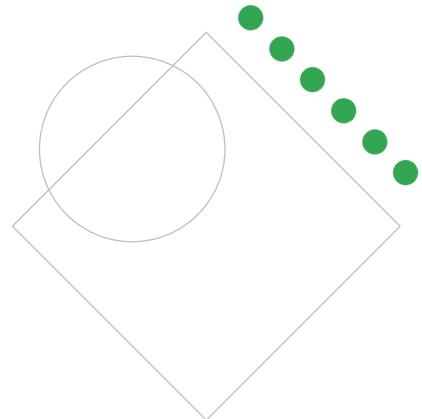


Google Cloud

# Module Agenda

- 01** Google Cloud Client Libraries and Google API Libraries
- 02** Google Cloud APIs
- 03** Service Accounts Review
- 04** Application Authentication

# Google Cloud Client Libraries and Google API Libraries



Google Cloud

# Guide: Client Libraries

This guide provides an overview of:

- Google Cloud Client Libraries
- Google API Client Libraries
- How to use them with Cloud Code

## Guide: Client Libraries

### Library options

- [Cloud Client Libraries](#) are the latest and **recommended** client libraries for calling Google Cloud APIs
  - [Cloud Client Libraries by language](#) contains
    - Links to installation instructions for various programming languages
    - Reference documentation for each language
- [Google API Client Libraries](#) are the original set of libraries.
  - Use these when you encounter the (rare) occurrence of a Google Cloud API that isn't supported by the Cloud Client Libraries.
  - The also support non-Google Cloud services such as
    - [Google Drive API](#): Allows you to interact with user files on Google Drive.
    - [Google Sheets API](#): Enables reading from and writing to Google Sheets.
    - [Google Calendar API](#): Lets you integrate with Google Calendar to create, access, or modify calendar events.
  - Check out the [API Client Explorer](#) for more examples

### Using Cloud Client Libraries with Cloud Code

- Cloud Code provides tools in [VS Code](#), [IntelliJ](#), and [Cloud Shell](#) for developing cloud applications, allowing you to easily use Cloud Client Libraries without leaving your IDE.

## Client Libraries

Google Cloud

## Two choices of client libraries

<a href="#">Google Cloud Client Libraries</a>	<a href="#">Google API Client Libraries</a>
<ul style="list-style-type: none"><li>• <b>Recommended option</b> for accessing Cloud APIs programmatically, where available</li><li>• Provides idiomatic code in each language to make Cloud APIs simple and intuitive to use</li><li>• Provides a consistent style across client libraries to simplify working with multiple Cloud services</li><li>• Some support gRPC which provides performance benefits</li></ul>	<ul style="list-style-type: none"><li>• <b>Use when there is no Cloud Client Library for your preferred language or a specific service is not supported</b></li><li>• Has auto-generated interface code that might not be as idiomatic as the Cloud Client Libraries.</li><li>• Provides access to the API's REST interface only; gRPC is not supported.</li></ul>

They both

- Handle all the low-level details of communication with the server, including authentication
- Can be installed using familiar package management tools such as npm and pip

# Google Cloud Libraries are available in multiple programming languages

- Install the entire Google Cloud Library
  - Or install libraries for individual Google Cloud services, e.g. Cloud Storage
- Each GitHub Library repository has installation instructions for getting started in the chosen language

Google Cloud Client Library	Installation & Reference
Go	<ul style="list-style-type: none"> <li>• <a href="#">GitHub Repo</a></li> <li>• <a href="#">Library Reference</a></li> <li>• <a href="#">Supported Go Versions</a></li> </ul>
Java	<ul style="list-style-type: none"> <li>• <a href="#">GitHub Repo</a></li> <li>• <a href="#">Library Reference</a></li> <li>• <a href="#">Supported Java Versions</a></li> </ul>
Node.js	<ul style="list-style-type: none"> <li>• <a href="#">GitHub Repo</a></li> <li>• <a href="#">Library Reference</a></li> </ul>
Python	<ul style="list-style-type: none"> <li>• <a href="#">GitHub Repo</a></li> <li>• <a href="#">Library Reference</a></li> </ul>
Ruby	<ul style="list-style-type: none"> <li>• <a href="#">GitHub Repo</a></li> <li>• <a href="#">Library Reference</a></li> </ul>
PHP	<ul style="list-style-type: none"> <li>• <a href="#">GitHub Repo</a></li> <li>• <a href="#">Library Reference</a></li> </ul>
C#	<ul style="list-style-type: none"> <li>• <a href="#">GitHub Repo</a></li> <li>• <a href="#">Library Reference</a></li> </ul>
C++	<ul style="list-style-type: none"> <li>• <a href="#">GitHub Repo</a></li> <li>• <a href="#">Library Reference</a></li> </ul>

[Cloud Client Libraries](#)

Google Cloud

<https://cloud.google.com/apis/docs/cloud-client-libraries>

## Example -Using Python to create a Cloud Storage bucket

```
from google.cloud import storage
def create_bucket(name, storage_class):
    storage_client = storage.Client()
    bucket = storage_client.bucket(name)
    bucket.storage_class =
        storage_client.bucket(storage_class)
    new_bucket = storage_client.create_bucket(
        bucket, location="us")
    return new_bucket
```

- Import the Cloud Client Library for Cloud Storage
- Instantiate a client
- Specify the bucket name
- Specify the storage class
- Create a multi-regional bucket

Google Cloud

Here's an example of using the Python Cloud Client Library to create a Cloud Storage bucket.

Every package provides a client that interacts with an API. Your application runs with a particular identity, which is typically a service account. This example imports the Cloud Storage client library, instantiates the client by using the default credentials provided by the service account, and creates a cloud bucket.

The Cloud Client Libraries let you easily manage your Google Cloud resources by using the natural style of the language you have chosen.

# Google API Client Library

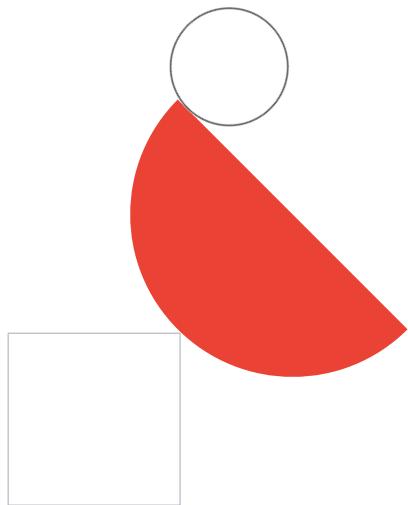
- Provides programmatic access to Google Maps, Google Drive, YouTube, and many other Google products
- Older library as compared to Google Cloud Client Libraries
  - Use Google Cloud Client Libraries if possible

The screenshot shows the official Google API Client Libraries page. At the top, there's a search bar and a link to 'Google API Client Libraries'. Below the header, a green banner states: 'The Google API Client Libraries provide simple, flexible, powerful access to many Google APIs.' A section titled 'Access Google APIs more easily' explains that Google APIs give programmatic access to various Google products like Maps, Drive, and YouTube, and that client libraries make coding easier and code more robust. It also mentions the Identity Platform products. Below this, a section titled 'Libraries for Google APIs' displays icons for Java, Python, PHP, .NET, JavaScript, Objective-C, Dart, Ruby, Node.js, and Go.

[Google API Client Libraries](#)

Google Cloud

# Google Cloud APIs



Google Cloud

# Guide: Google Cloud APIs

Topics covered include

- How to get started
- API documentation
- API Explorer
- Working with quotas

## Guide: Google Cloud APIs

### Getting started with Cloud APIs

- This page explains how developers can get started using Google Cloud APIs.  
Included topics are:
  - Creating a free account
  - Creating a project
  - Discovering and enabling APIs
  - Enabling billing
  - Building applications with the Google Cloud Client Libraries

### Youtube video: Getting started with Google Cloud

- In this video, you learn how to navigate the Cloud Console and locate components needed to build cloud applications.



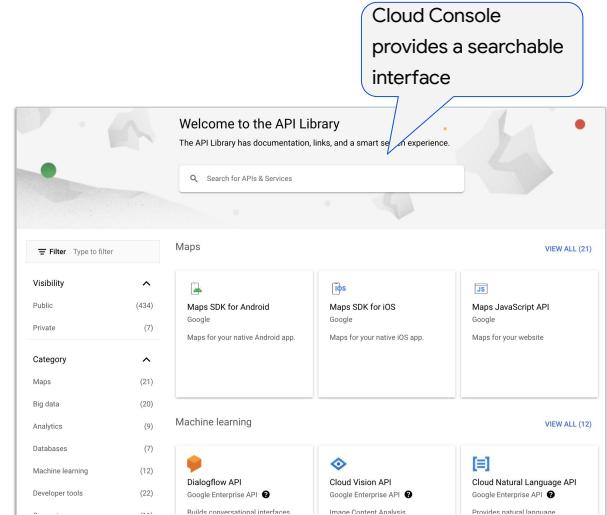
<https://www.youtube.com/watch?v=Iw9q0xQAU2Y>

Google Cloud APIs

Google Cloud

# Google Cloud APIs

- Programmatic interfaces to Google services
- Typically runs on one or more subdomains of `googleapis.com`
  - E.g. `compute.googleapis.com`, `pubsub.googleapis.com`
- APIs are enabled and used within a project
  - Some Cloud APIs are enabled by default
  - Others must be enabled within a project\*
    - E.g., Must enable the Compute Engine API prior to creating a virtual machine



\*Developer must have **Service Controller** and **Service Consumer** role to enable APIs and view quotas

Google Cloud

# API Explorer

- Provides documentation for each API call
  - All APIs support REST
    - Some support [gRPC](#)
      - Look at the specific API documentation for details
- Access the APIs from
  - [Client libraries](#) in many popular programming languages
  - Mobile/web apps via the [Firebase SDKs](#)
  - [Google Cloud command line \(CLI\) tools](#)
  - [Google Cloud console](#)

The screenshot shows the Google APIs Explorer interface. At the top, there's a search bar and language selection (English). Below that is a table titled "Google APIs Explorer Directory" with columns for "Title" and "Description". The table lists several APIs:

Title	Description
<a href="#">Abusive Experience Report API</a>	Views Abusive Experience Report data, and gets a list of sites that have a significant number of abusive experiences.
<a href="#">Accelerated Mobile Pages (AMP) URL API</a>	Retrieves the list of AMP URLs (and equivalent AMP Cache URLs) for a given list of public URLs(s).
<a href="#">Access Context Manager API</a>	An API for setting attribute based access control to requests to Google Cloud services.
<a href="#">ACME DNS API</a>	Google Domains ACME DNS API that allows users to complete ACME DNS-01 challenges for a domain.
<a href="#">Ad Exchange Buyer API II</a>	Accesses the latest features for managing Authorized Buyers accounts, Real-Time Bidding configurations and auction metrics, and Marketplace programmatic deals.
<a href="#">Ad Experience Report API</a>	Views Ad Experience Report data, and gets a list of sites that have a significant number of annoying ads.
<a href="#">Admin SDK API 1</a>	Admin SDK lets administrators of enterprise domains to view and manage resources like user, groups etc. It also provides audit and usage reports of domain.
<a href="#">Admin SDK API 1</a>	Admin SDK lets administrators of enterprise domains to view and manage resources like user, groups etc. It also provides audit and usage reports of domain.
<a href="#">Admin SDK API 1</a>	Admin SDK lets administrators of enterprise domains to view and manage resources like user, groups etc. It also provides audit and usage reports of domain.
<a href="#">AdMob API</a>	The AdMob API allows publishers to programmatically get information about their AdMob account.
<a href="#">AdSense Host API</a>	The AdSense Host API gives AdSense Hosts access to report generation, ad code generation, and publisher management capabilities.

[Google APIs Explorer Directory](#)

Google Cloud

## Example: Detect labels in an image by using the Cloud Vision API

- Quickstart: [Detect labels in an image by using the Cloud Vision API](#)
  - Create a storage bucket
  - Upload this image and make it public
    - <https://cloud.google.com/static/vision/docs/images/demo-image.jpg>
  - Enable the Cloud Vision API
  - Make a request to the Cloud Vision API by going to this link at the bottom of the quickstart
    - Edit the request body to match your Cloud Storage bucket name - example is shown on the next slide
    - Click **EXECUTE**
      - User account must have access **Storage Object Viewer** role on the bucket



Image stored in bucket

Google Cloud

Steps can be found in <https://cloud.google.com/vision/docs/detect-labels-image-api>

## Request body

```
{
  "requests": [
    {
      "features": [
        {
          "type": "LABEL_DETECTION"
        }
      ]
    },
    "image": {
      "source": {
        "imageUri": "gs://api-explorer-demo/demo-img.jpg"
      }
    }
  ]
}
```

Full path to file in the storage bucket

Partial results returned from the API call

200

```
{
  "responses": [
    {
      "labelAnnotations": [
        {
          "mid": "/m/083wq",
          "description": "Wheel",
          "score": 0.9770954,
          "topicality": 0.9770954
        },
        {
          "mid": "/m/0h9mv",
          "description": "Tire",
          "score": 0.9736712,
          "topicality": 0.9736712
        },
        {
          "mid": "/m/07mhn",
          "description": "Trousers",
          "score": 0.9619434,
          "topicality": 0.9619434
        }
      ]
    }
  ]
}
```

**Score:** % Confidence

**Topicality:** Importance of the entity to the overall theme.

Google Cloud

## Same example using the Google Cloud Python library

```
from google.cloud import vision_v1 as vision

def analyze_image(bucket_name, file_name):
    # Create a client for the Vision API
    client = vision.ImageAnnotatorClient()

    # Construct the image object with the storage URI
    image = vision.Image()
    image.source.image_uri = f'gs://{bucket_name}/{file_name}'

    # Make the request to the Vision API
    response = client.label_detection(image=image)

    # Print out the labels detected in the image
    for label in response.label_annotations:
        print(label.description, label.score)

if __name__ == "__main__":
    bucket_name = "api-explorer-demo"
    file_name = "demo-img.jpg"
    analyze_image(bucket_name, file_name)
```

Following best practices for authentication (Application Default Credentials)

Output

```
Bicycle 0.9850001931190491
Wheel 0.9770954251289368
Tire 0.9736711978912354
Trouser 0.9619433879852295
Bicycle wheel 0.9272286295890808
Sky 0.923011302947998
Bicycles--Equipment and supplies
0.92181795835495
Human 0.8934807777404785
Bicycle handlebar 0.8904159069061279
Bicycle tire 0.8738805055618286
```

Google Cloud

Detect labels in an image by using client libraries

<https://cloud.google.com/vision/docs/detect-labels-image-client-libraries>

# Calling REST APIs directly vs using the Google Cloud client libraries

- Use the REST APIs directly if
  - You're using a programming language not available with the Google Cloud client libraries
  - Need to use a feature that isn't exposed through the client libraries or want to handle the low-level details in a specific way
  - Need to use a newer version of an API that isn't yet supported by the client library, or need to use a specific older version
- In general, Google Cloud client libraries are a better option
  - Provides higher-level abstractions and hide much of the lower-level HTTP/REST detail
  - Retry logic and error handling is built in, making code cleaner and easier to maintain

## API quota and error handling

# All APIs have quotas

Proprietary + Confidential

- Quota is set at the project level
  - Prevents customers from consuming so many resources that they become unavailable to other customers
  - Helps you manage resources to avoid unexpected costs
- Type of quotas vary for service but typically fall into the following categories
  - Rate quotas - per minute or per day
    - Reset after a predefined time interval that is specific to each service
      - For example, Cloud SQL (MySQL) quota for queries per minute per user is set to 180
  - Allocation quotas - for resources that don't have a rate limit
    - For example, number of VMs used per project
      - When maximum is reached must delete a resource before creating another
  - Concurrent quotas
    - Restricts the total number of concurrent operations in flight at any given time
      - For example, Cloud SQL (MySQL) has a limit of 100,000 concurrent connections
- Look at specific API documentation for details, e.g., [Cloud SQL \(MySQL\)](#)

Google Cloud

# Handling quota related errors

- Rate quotas (maximum of N requests per X minutes)
  - [HTTP 429 rate limit exceeded](#)
  - To resolve these issues, retry the operation after a few seconds using [truncated exponential backoff](#)
- Allocation quotas
  - [HTTP 403 quota exceeded or resource exhausted](#)
  - Wait for at least 10 minutes before retrying
    - Consider optimizing your workload or requesting a [quota increase](#)
- Concurrent quotas
  - [HTTP 429 too many requests or too many operations](#)
  - Use exponential backoff when retrying operations that have hit concurrent limits

```
{  
  "code": 429,  
  "message": "Rate Limit Exceeded",  
  "errors": [  
    {  
      "message": "Rate Limit Exceeded",  
      "domain": "usageLimits",  
      "reason": "rateLimitExceeded"  
    }  
  ]  
}
```

[Complete list](#) of error codes and a short description of their cause

Note: The actual HTTP code/message returned may vary depending on the API. See the specific API documentation for details on error handling. [Cloud Storage example](#)

Google Cloud

## Exponential Backoff vs Truncated Exponential Backoff

- Exponential Backoff
  - Increasing the wait time between retries exponentially
    - After the first failed attempt, wait 't' seconds.
      - If it fails again, you wait '2t' seconds, then '4t' seconds, and so on
      - The delay between retries doubles with each subsequent retry
      - Leads to very long wait times after several failed attempts
    - Effective for situations where the cause of the error is likely to be temporary
  - Truncated Exponential Backoff
    - Variation of exponential backoff where the wait time is capped at a maximum value
      - Once it reaches the maximum cap, subsequent retries will use that maximum value
      - The cap ensures that the system doesn't wait excessively long periods between retries

Wikipedia: [Exponential backoff](#)

Google Cloud

## Python example of truncated exponential backoff

```
import time
import random
from googleapiclient.errors import HttpError

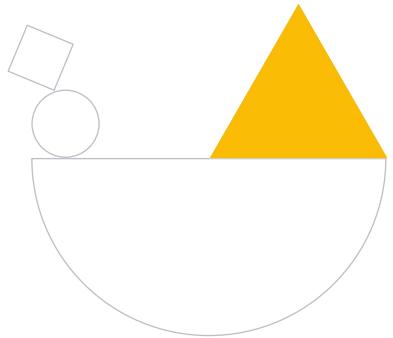
def make_request_with_exponential_backoff(api_call):
    max_retries = 5
    base_delay = 1 # start with a 1-second delay
    max_delay = 32 # max delay is 32 seconds

    retries = 0
    while retries <= max_retries:
        try:
            # Make the API call
            response = api_call()
            return response
        except HttpError as e:
            if e.resp.status in [500, 502, 503, 504]: # Retry on these HTTP errors
                delay = base_delay * (2 ** retries) # exponential delay
                delay = min(delay, max_delay) # don't exceed max delay
                time.sleep(delay + random.uniform(0, 0.1 * (2 ** retries)))
                retries += 1
            else:
                raise # re-raise the exception if it's not a retryable error
        raise Exception("Max retries exceeded")

# Example usage:
# response = make_request_with_exponential_backoff(lambda: some_google_api_call())
```

Google Cloud

# Service Accounts review



Google Cloud

# Guide: Service Accounts

Topics covered include

- Service account overview
- Service account security
- Best practices

## Guide: Service Accounts

### Service Accounts and Security

- Watch the 5 video series covering service accounts including topics such as:
  - What they are
  - How to secure them
  - Service Account Impersonation
  -



### Service Accounts and Security

- Service Accounts
  - Service Accounts overview
    - <https://cloud.google.com/iam/docs/service-account-overview>

## Service Accounts

Google Cloud

## Service Accounts

- A special type of principal in Google Cloud that provides IAM roles to an application and limits what the application can do
  - Applies to applications running within Google Cloud compute services
    - Compute Engine
    - Kubernetes Engine
    - Cloud Run
    - Cloud Functions
    - App Engine
  - May apply to applications running external to Google Cloud
    - Application assumes the identity of the service account for IAM purposes
- **Best practice:** Create custom service accounts for each use case and follow principle of least privilege when assigning IAM roles

## Example Service Account use cases

- Server-to-Server Interactions where user intervention is not required
  - For example, Pub/Sub triggering a Cloud Run service
    - Pub/Sub service account needs **Cloud Run Invoker** role
    - Cloud Run service account needs **Pub/Sub Subscriber** role
- Automated Tasks not initiated by a user
  - For example, batch jobs, data processing tasks, or automated backups
  - Service accounts needs the appropriate IAM roles required by the tasks
- Applications running on any Google Cloud compute resources
  - For example, a Python app deployed to Compute Engine which needs to query a BigQuery dataset
    - The service account needs **BigQuery Data Viewer** and **BigQuery User** roles

# Viewing Service Accounts in the Console

- Google Cloud creates default service accounts for the compute services when their APIs are enabled
  - Enables people new Google Cloud to create compute resources without have to learn IAM first
  - These accounts have the **Editor** role and don't follow the principle of least privilege
- Best practice:** Create custom service accounts with appropriate IAM roles given the use case

The screenshot shows the Google Cloud IAM & Admin interface. On the left, there's a sidebar with options like IAM, Identity & Organization, Policy Troubleshooter, Policy Analyzer, Organization Policies, Service Accounts (which is selected), Workload Identity Federat..., and Labels. The main area has two tabs at the top: 'VIEW BY PRINCIPALS' (selected) and 'VIEW BY ROLES'. Below that is a 'GRANT ACCESS' and 'REMOVE ACCESS' button, followed by a 'Filter' input field. A table lists three service accounts:

Type	Principal	Name	Role
447159861369-compute@developer.gserviceaccount.com	compute@developer.gserviceaccount.com	Compute Engine default service account	Editor
backend@bt-iam.iam.gserviceaccount.com	backend		Storage Object Viewer
bigrquery-qwiklab@bt-iam.iam.gserviceaccount.com	bigrquery-qwiklab		BigQuery Data Viewer BigQuery User

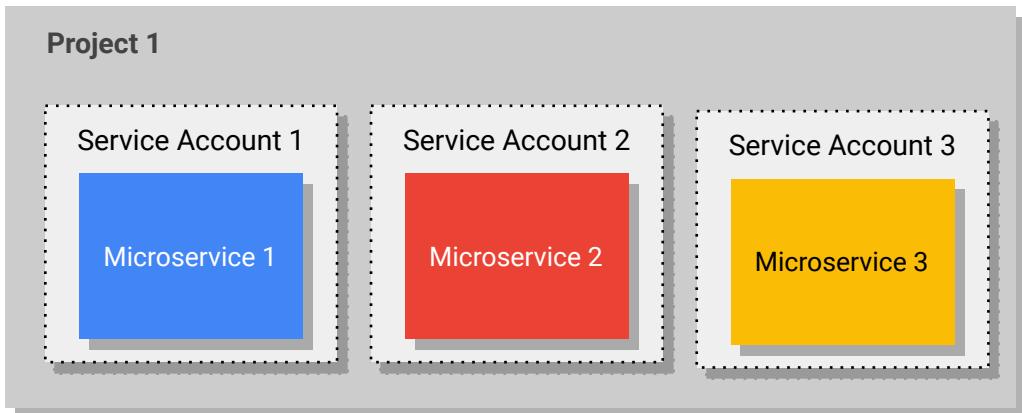
A blue callout bubble points to the first row with the text "Compute Engine default service account."

Custom service account.

Google Cloud

## Multiple service accounts can exist within a project

- Different service accounts for different use cases



# Creating Service Accounts in the Console

IAM & Admin > Service Accounts>  
Create Service account

Developers must have **Owner** or  
**Service Account Admin** role in  
order to create service accounts and  
assign roles to them

Email	Status	Name
bt-managed-instance-grp@appspot.gserviceaccount.com	✓	App Engine default service account
479845979764-compute@developer.gserviceaccount.com	✓	Compute Engine default service account
test-555@bt-managed-instance-grp.iam.gserviceaccount.com	✓	test

Google Cloud

## Creating service accounts

<https://cloud.google.com/iam/docs/creating-managing-service-accounts>

## Creating Service Accounts (continued)

Service account name →

**1 Service account details**

Service account name  
inventory-webserver

Display name for this service account

Service account ID \*  
inventory-webserver X C

Email address: inventory-webserver@bt-spaceinvaders-ke.iam.gserviceaccount.com

Service account description  
for VMs running the inventory system

Describe what this service account will do

---

**2 Grant this service account access to project (optional)**

Grant this service account access to bt-spaceinvaders-ke so that it has permission to complete specific actions on the resources in your project. [Learn more ↗](#)

Role	Storage Object Creator	IAM condition (optional) <span>?</span> <span>X</span>
Access to create objects in GCS.		<a href="#">+ ADD IAM CONDITION</a>
Role	Cloud Run Invoker	IAM condition (optional) <span>?</span> <span>X</span>
Can invoke a Cloud Run service.		<a href="#">+ ADD IAM CONDITION</a>
<a href="#">+ ADD ANOTHER ROLE</a>		

Add one or more roles →

Google Cloud

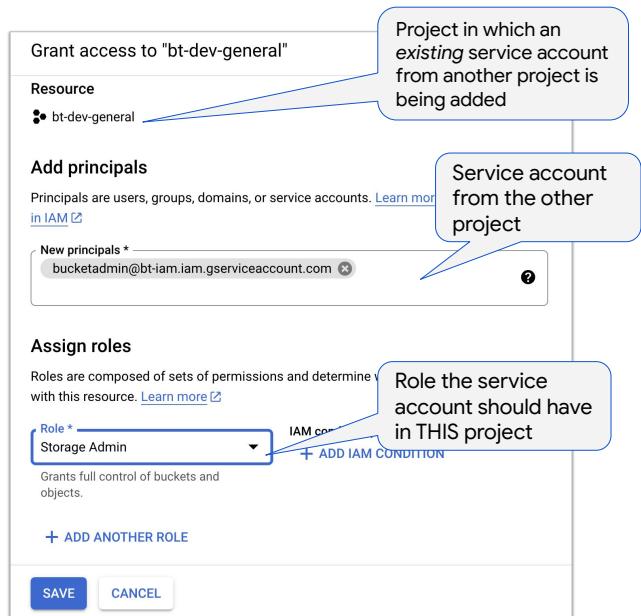
Here is an example of creating a service account.

First, give your service account a name. A best practice is to give the account a name that can be used to easily determine the purpose of the service account.

Next, you add one or more roles to associate with the service account. The roles will determine the permissions associated with the account.

## Cross-project usage

- Service accounts live in a specific project
  - Can be granted permissions to access resources in *other* projects
    - Used when resources in one project need to be accessed by apps/services running in a different project
      - E.g., Cloud Run service needs to access a Cloud Storage bucket located in another project
  - Allows for centralized management of service accounts while ensuring that resources in other projects can be accessed securely



[Where to create service accounts](#)

Google Cloud

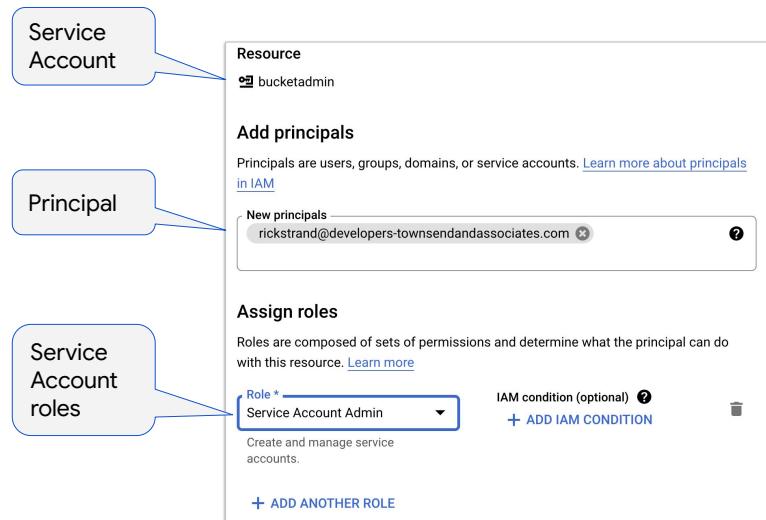
## Service Account have their own predefined roles

- Service Account Admin
  - Create and manage service accounts
- Service Account User
  - Can attach service account to resources (e.g., Compute Engine)
  - Can “impersonate” the service account and perform the tasks allowed by IAM given to the service account
- Service Account Key Admin
  - Create and manage (and rotate) service account keys
  - Keys are used by applications external to Google Cloud (**not best practice**)
- Service Account Token Creator
  - Short lived credentials represented as OAuth 2.0 access tokens, OpenID Connect ID tokens, self-signed JSON Web Tokens (JWTs), and self-signed binary objects (blobs)
  - Different use cases - applications external to Google Cloud, authentication of one cloud service to another, etc. (**best practice**)

All of these  
are discussed  
later in this  
module

Reasons  
discussed later in  
the module

# Assigning a service account role to a principal



Google Cloud

## Example of attaching Service Accounts to services

- Developer must have **Service Account User** role in order to attach service accounts to services

The screenshot displays two Google Cloud interface sections side-by-side, both titled "Identity and API access".

**Compute Engine:** This section is for creating a VM instance. It shows a "Service accounts" dropdown menu with "web-server" selected. A red box highlights this dropdown. Below it, a note states: "Requires the Service Account User role (roles/iam.serviceAccountUser) to be set for users who want to access VMs with this service account. [Learn more](#)".

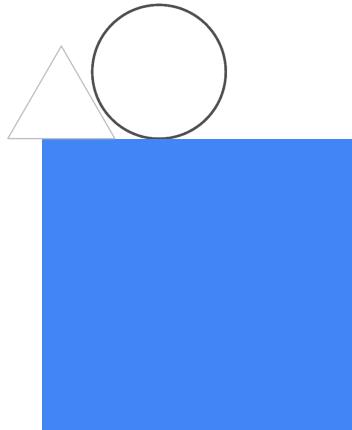
**Cloud Run:** This section is for creating a service. It also has a "Service account" dropdown menu with "storage-viewer" selected. A red box highlights this dropdown. Below it, a note states: "Identity to be used by the created revision".

Google Cloud

# Application Authentication

- **Overview**

- API keys
- Application Default Credentials
- Workload Identity
- Workload Identity Federation



Google Cloud

# Google API Authentication vs Authorization

- This module covers **application** authentication and authorization

Authentication



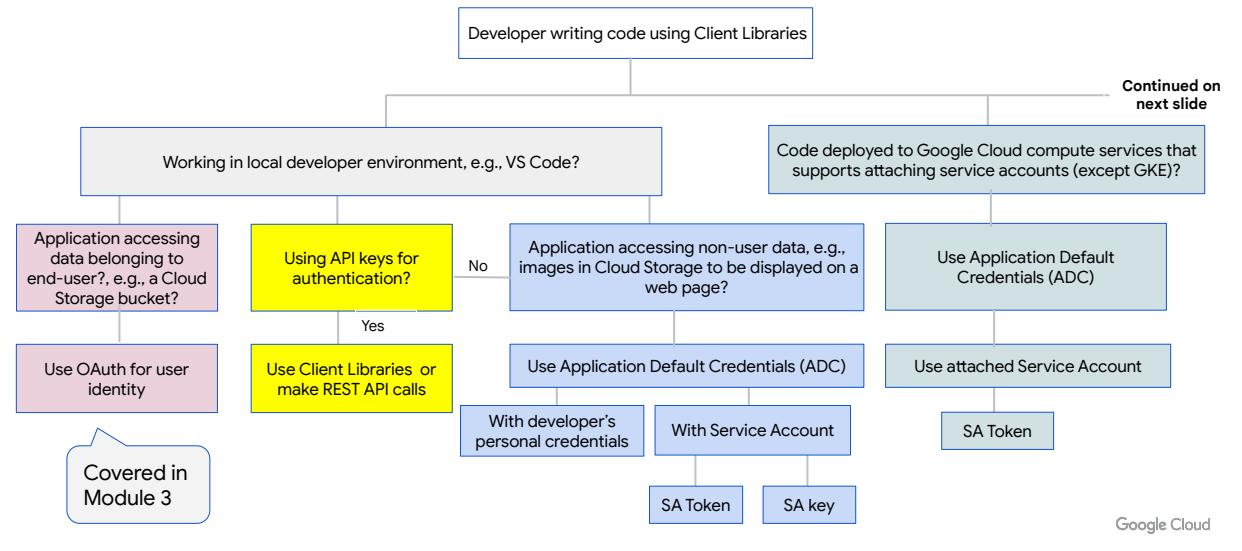
What is the application?

Authorization

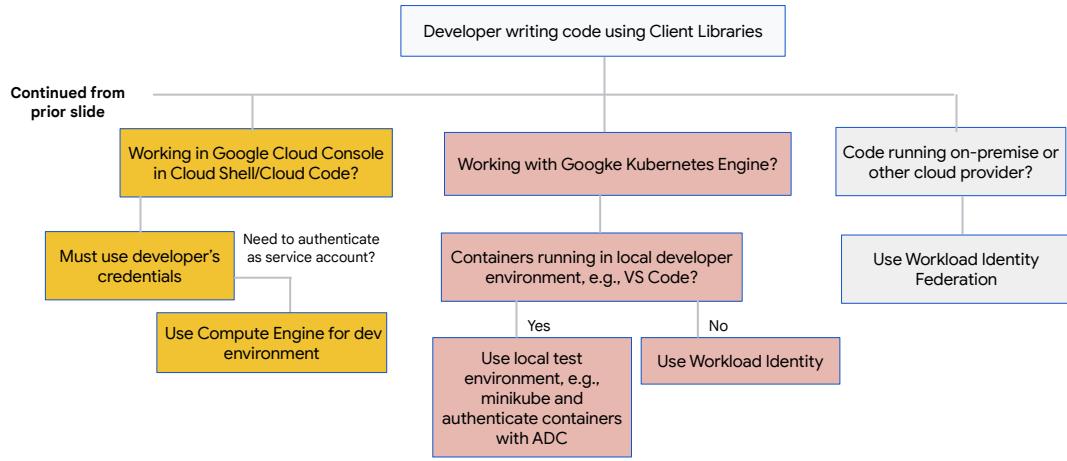


What is it allowed to do?

# Application authentication depends on where it is running



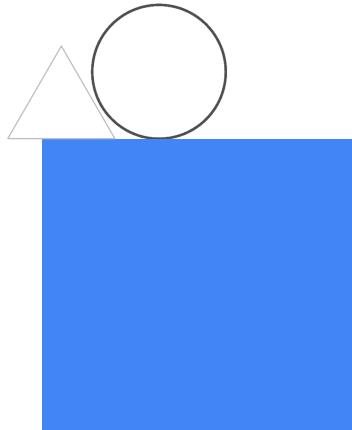
# Application authentication depends on where it is running (continued)



Google Cloud

# Application Authentication

- Overview
- **API keys**
- Application Default Credentials
- Workload Identity
- Workload Identity Federation



# Guide: API Keys

Topics covered include

- How to generate API Keys
- Security best practices
- Walk-through using the Cloud Natural Language API with an API Key

## Guide: API Keys

### API key authentication

- Most Google Cloud APIs don't support API keys. Check that the API that you want to use supports API keys before using this authentication method.
- When you use an API key to authenticate to an API, the API key does not identify a principal, nor does it provide any authorization information. Therefore, the request does not use Identity and Access Management (IAM) to check whether the caller has permission to perform the requested operation.
- The API key associates the request with a Google Cloud project for billing and quota purposes. Because API keys do not identify the caller, they are often used for accessing public data or resources.
  - Reference: [Authenticate by using API keys](#)

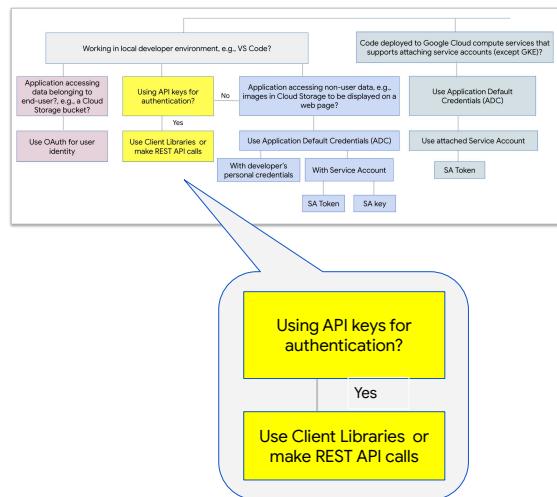
### API key security best practices

- [API security best practices](#) describes how to restrict access to API Keys for maximum security
- Youtube video:
  - [How to generate and restrict API Keys for Google Maps Platform](#)
    - This is specific to the Maps API, but illustrates how to generate and restrict API keys.

## API Keys

Google Cloud

# Authentication with API keys



Google Cloud

## What is a API key?

- Secret tokens usually in the form of an encrypted string
  - Used by many API providers as an easy way for developers to access their APIs
  - Typical use of an API key is to pass the key into a REST API call as a query parameter with the following format  
`http://library.googleapis.com/v1/publishers/mypublisher/books?key=API_KEY`
- API keys provide
  - Verification that the calling application has been granted access to the API and the API has been enabled at the Google Cloud project level
  - Usage information for billing and quota purposes
  - Identification of application or project that's making the call
    - ***Does not provide the identity of the caller***
    - **Most Google Cloud APIs do not accept API keys for this reason**
- API keys are often used for accessing public data or resources, where authentication/authorization is not required

Example Google Cloud APIs that accept keys:  
Cloud Translation API  
Cloud Natural Language API

Google Cloud

API keys provide project authorization

[https://cloud.google.com/endpoints/docs/openapi/when-why-api-key#api\\_keys\\_provide\\_project\\_authorization](https://cloud.google.com/endpoints/docs/openapi/when-why-api-key#api_keys_provide_project_authorization)

# Enabling APIs in Google Cloud

- APIs are enabled on a project by project basis
  - Billing for API usage is at the project level

The screenshot shows two panels of the Google Cloud API Library. The left panel is titled 'APIs & Services' and has a button labeled '+ ENABLE APIs AND SERVICES' circled in red. The right panel is titled 'API Library' and shows search results for 'maps'. A speech bubble points to the search bar with the text 'Search for the API'. Another speech bubble points to the 'Maps SDK for iOS' result with the text 'Select the one of interest'.

**Google Cloud API Library - Search Results for 'maps'**

Search results for 'maps':

- Maps SDK for iOS** (Google)
 

Add maps based on Google Maps data to your iOS application with the Maps SDK for iOS. The SDK automatically handles map servers, map display and response to user gestures such as clicks and drags.
- Maps SDK for Android** (Google)
 

Add maps based on Google Maps data to your Android application with the Maps SDK for Android. The SDK automatically handles map servers, map display and response to user gestures such as clicks and drags.
- Maps JavaScript API** (Google)
 

Add a map to your website, providing imagery and local data from the same source as Google Maps. Style the map with your own data on the map, bring the world to life with Street View, and use services like geocoding and directions.

Google Cloud

# Creating API keys

Pop-up appears when CREATE CREDENTIALS is clicked

**API key**  
Identifies your project using a simple API key to check quota and access

**OAuth client ID**  
Requests user consent so your app can access the user's data

**Service account**  
Enables server-to-server, app-level authentication using robot accounts

**Help me choose**  
Asks a few questions to help you decide which type of credential to use

- Authentication options
  - API keys
    - Not supported by most Google Cloud APIs
      - Keys do not identify a principal, nor provide any authorization information
  - OAuth when accessing data belonging to an end-user, e.g., Firestore documents
  - Service account when access data belonging to the application, e.g., images in a Cloud Storage bucket for a website

Google Cloud

Authenticate by using API keys

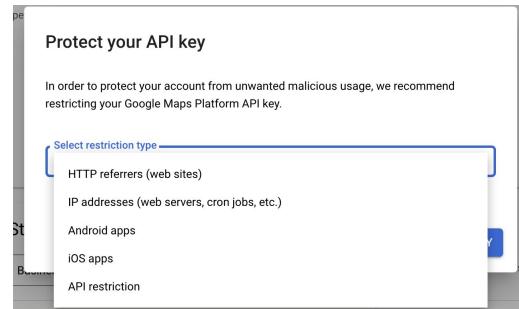
<https://cloud.google.com/docs/authentication/api-keys#using-with-rest>

Adding restrictions to API keys

<https://cloud.google.com/api-keys/docs/add-restrictions-api-keys>

# Restrict API key to specific resources

- Selecting one of the following:
  - HTTP referrers (websites) that are allowed to use the key
    - E.g., [www.example.com](http://www.example.com), [www.example2.com](http://www.example2.com)
  - IP addresses of callers that are allowed to use the key
    - "198.51.100.0/24", "198.51.100.1"
  - Android apps that are allowed to use the key
  - iOS apps that are allowed to use the key
- Only one restriction per key is allowed
  - Create another key if needed, e.g., one for Android and one for iOS apps



[Adding restrictions to API keys](#)

Google Cloud

Adding restrictions to API keys

<https://cloud.google.com/api-keys/docs/add-restrictions-api-keys>

# Securing API Keys

- API keys must be stored securely
  - Anyone who has access to the key can make API calls
- Never store keys in plain text files or in source control systems
- Use environment variables
  - Helps prevent the API key from being hard-coded into the application code or configuration files
- Restrict usage to specific web sites or IP addresses
- Use a third-party API key management service to store and manage the API keys
  - [Google Cloud Secret Manager](#) stores API keys, passwords, certificates, and other sensitive data

## Two ways to use an API Key

- Via [REST](#) API calls
- Using the [client libraries](#)
- Examples of both are on the following slides

# Method 1 - making a REST call

Making a POST  
request to the Cloud  
Translation API

- Pass the API key into a REST API call as a query parameter

```
curl -X POST -H "Content-Type: application/json" \
-d '{
  "q": "Welcome everyone. I hope you are enjoying the workshop.",
  "source": "en",
  "target": "es",
  "format": "text"
}' \
"https://translation.googleapis.com/language/translate/v2?key=\[my\_key\_here\]"
```

API key

[Using an API key with REST](#)

## Method 2 - Use a client library to make a REST API call

```
import requests
import json

# Define the endpoint URL and your API key
endpoint_url = "https://translation.googleapis.com/language/translate/v2"
api_key = "your-api-key-here" API key

# Define the payload for the POST request
data = {
    "q": "Welcome everyone. I hope you are enjoying the workshop.",
    "source": "en",
    "target": "es",
    "format": "text"
}

# Make the POST request
headers = {"Content-Type": "application/json"}
response = requests.post(f"{endpoint_url}?key={api_key}", data=json.dumps(data), headers=headers)

# Parse and print the response
response_data = response.json()
if response.status_code == 200:
    translated_text = response_data["data"]["translations"][0]["translatedText"]
    print(translated_text)
else:
    print(f"Request failed with status code {response.status_code}")
```

Note: Not best practice to store the API key in code.

Use [Secrets Manager](#) to store sensitive data.

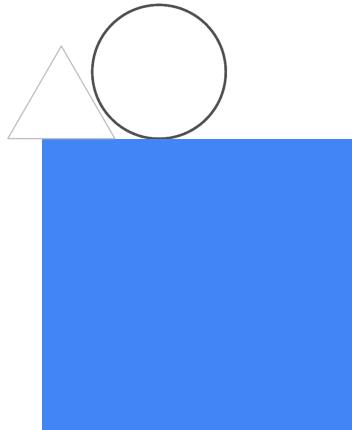
See [Using Secret Manager with Python](#) codelab for example usage

[Using an API key with client libraries](#)

Google Cloud

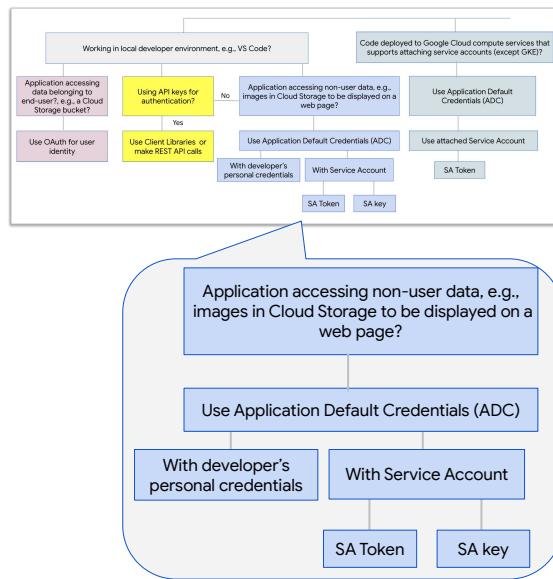
# Application Authentication

- Overview
- API keys
- **Application Default Credentials**
- Workload Identity
- Workload Identity Federation



Google Cloud

# Authentication with Application Default Credentials (ADC)



Google Cloud

# Guide: Application Default Credentials (ADC)

Topics covered include

- How ADC works
- How to set ADC up depending on where your code is running
  - Local environment
  - Google Cloud Compute
  - etc.

## Guide: Application Default Credentials (ADC)

### Application Default Credentials (ADC)

- Application Default Credentials (ADC) is a strategy used by the Google authentication libraries to automatically find credentials based on the application environment
  - [How to provide credentials to ADC](#)
    - Explains how to provide credentials based on where your code is running, e.g., local development environment, Google Cloud, etc.
  - [How Application Default Credentials works](#)
    - Explains the locations ADC searches to find credentials
  - [Troubleshoot your ADC setup](#)
    - Describes common problems encountered when using Application Default Credentials (ADC)

### Authenticate applications to Google Cloud

- Youtube video:
  - Describes the correct way to authenticate applications

Introductory

**Authenticate applications to Google Cloud**

Google Cloud

Next



## Application Default Credentials

Google Cloud

# Application Default Credentials (ADC)

Python example

- Strategy used by the [Google authentication library](#) to automatically find credentials based on the application environment
  - Uses [Application Default Credentials \(ADC\)](#) to find credentials you provide
    - Your code does not need to explicitly authenticate or manage tokens
      - Automatically managed by the authentication library
    - Code can run in either a development or production environment without changing how it authenticates to Google Cloud services and APIs
- **ADC is already integrated into the Cloud Client Libraries and Google API Client Libraries**
  - No need to add the Google Authentication library in your application and call it directly
- Benefits:
  - Consistent authentication across different environments
  - Reduces manual configuration and potential errors
  - Enhances security by promoting best practices.

# How to provide credentials to ADC

Method used depends on the environment where your code is running:

[Local development environment](#)

Next topic

[Compute Engine or other Google Cloud services that support attaching a service account](#)

[Cloud Shell or other Google Cloud cloud-based development environments](#)

[Google Kubernetes Engine](#)

[On-premises or another cloud provider](#)

Google Cloud

Crea

# How do Application Default Credentials (ADC) work in a local development environment?

- Three options
  - Each are explained on the following pages

1

## Using developer's credentials (easy to get started this way)

Developer needs IAM roles that reflect the permissions needed by the code

```
gcloud auth application-default login
```

2

## Using service account impersonation

Developer needs **Service Account Token Creator** role to impersonate the service account

```
gcloud auth application-default login --impersonate-service-account [SERVICE_ACCT_EMAIL_HERE]
```

Creates an access token which has a limited lifetime

### Best practice

3

## Using service account key

Developer needs **Service Account User** role plus **Service Account Key Admin** if need to create/download keys

Set the **GOOGLE\_APPLICATION\_CREDENTIALS** environment value to the path of the service account's key key

**AVOID THIS OPTION.** Keys don't expire by default

Google Cloud

## Review: Console view of IAM assignments

- Image shows both individual accounts\* and service accounts

The screenshot shows the Google Cloud IAM console. On the left, there's a sidebar with 'IAM & Admin' selected. The main area lists two entries:

	User/Service Account	Name	Role
<input type="checkbox"/>	bigquery-user@bt-dev-general.iam.gserviceaccount.com	bigquery-user	BigQuery Data Viewer BigQuery User
<input type="checkbox"/>	caliagusta@developers-townsendandassociates.com	Cali Augusta	BigQuery Data Viewer BigQuery User Service Account Token Creator Service Account User

Two callout boxes are present: one labeled 'Service Account' pointing to the first row, and another labeled 'Individual' pointing to the second row.

\*IAM can also be assigned to groups. Members of that group have the assigned roles.

Google Cloud

## Using developer's personal credentials

- Used during the initial development and testing phase
  - **Not for shared environments such as staging, test, production**
- Allows developer start quickly without the overhead of managing service accounts
  - Developer's account needs IAM roles that reflect the permissions needed by the code to access Google Cloud resources
- Process is initiated by developer logging into Google Cloud from the CLI

Must install the  
Cloud CLI first

# Review: Installing the Google Cloud CLI

- After installation, need to [initialize](#) its settings

The screenshot shows a section titled "Installation instructions" for the Google Cloud CLI. It includes a note about proxy/firewall settings and links for Linux, Debian/Ubuntu, Red Hat/Fedora/CentOS, macOS (selected), and Windows. The main content lists steps for confirming Python version and downloading the install script.

These instructions are for installing the Google Cloud CLI. For information about installing additional components, such as gcloud CLI commands at the alpha or beta release level, see [Managing gcloud CLI components](#).

★ Note: If you are behind a proxy/firewall, see the [proxy settings](#) page for more information on installation.

Linux    Debian/Ubuntu    Red Hat/Fedora/CentOS    **macOS**    Windows

1. Confirm that you have a supported version of Python:
  - To check your current Python version, run `python3 -V` or `python -V`. Supported versions are Python 3 (3.5 to 3.9).
  - For Cloud SDK release version 352.0.0 and above, the main install script offers to install CPython's Python 3.7 on Intel-based Macs.
  - For more information on how to choose and configure your Python interpreter, refer to [gcloud topic startup](#).
2. Download one of the following:

★ Note: To determine your machine hardware name, run `uname -m` from a command line.

[Installing the CLI](#)

Google Cloud

Initializing the gcloud CLI:

<https://cloud.google.com/sdk/docs/initializing>

gcloud command to set configuration:

<https://cloud.google.com/sdk/gcloud/reference/config/set>

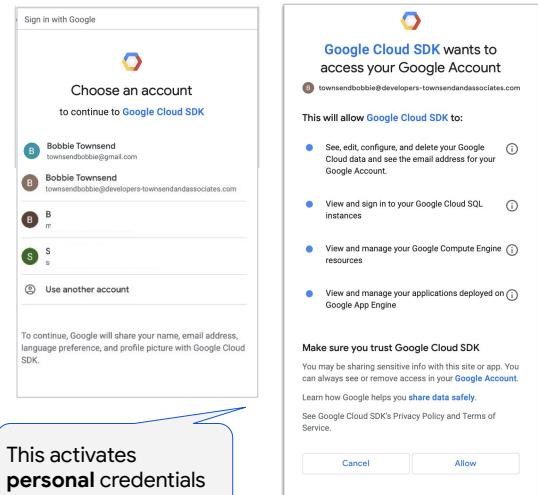
gcloud tool guide:

<https://cloud.google.com/sdk/gcloud/>

After the CLI is installed, you need to do an initial setup. That includes things such as setting a default region and zone.

# Using developer's personal credentials for ADC - step 1

- Developer logs in at the command line using
  - `gcloud auth login`
- Uses OAuth2 for authentication
  - The Google Authorization Server is contacted and responds with a token
    - The token is stored in a SQLite database associated with the CLI
      - `~/.config/gcloud/credentials.db`



Google Cloud

## Using developer's personal credentials for ADC - step 2

- Developer creates a credential file for ADC by running the following in the CLI
  - gcloud auth application-default login
    - Enter **your** credentials in the login screen that appears
- ADC creates a credential file in a “well known location”
  - Linux, macOS:
    - \$HOME/.config/gcloud/application\_default\_credentials.json
  - Windows:
    - %APPDATA%\gcloud\application\_default\_credentials.json

This activates **ADC** credentials using developer's personal login

## ADC credential file contents

```
{  
  "client_id": "764086051850-6qr4p6gpi6hn506pt8ejuq83di341hur.apps.googleusercontent.com",  
  "client_secret": "d-FL95Q19q7MQmFpd7hHD0Ty",  
  "quota_project_id": "bt-dev-general",  
  "refresh_token": "1//0dKjnQWfeqCYRCgYIARAAGA0SNwF-L9Ir2h4ocQGUvH1zA1bWBzqxk2spK3Wn3rJ|",  
  "type": "authorized_user"  
}
```

- client\_id:
  - Identifies the application making a request to the authentication server
- client\_secret:
  - Known only to the application and the authorization server
  - Used to ensure that the client application is the one it claims to be
- refresh\_token:
  - Used to obtain a new access token when the current one expires
  - Automatically handled by ADC
- type
  - “authorized\_user” indicates that these are user credentials

Google Cloud

## Revoke credentials when no longer needed

- IMPORTANT! The local ADC contains your access and refresh tokens.
  - Any user with access to your file system can use those credentials
  - If you no longer need these local credentials, revoke them by using the following
    - `gcloud auth application-default revoke`

## Example: ADC in local environment using developer credentials - Querying BigQuery with Python

- Developer needs `BigQuery User` and `BigQuery Data Viewer` roles
- Developer logs into command line with personal credentials
  - `gcloud auth login`
- Developer creates credentials for ADC
  - `gcloud auth application-default login`

Output of command:

```
Credentials saved to file:  
[$HOME/.config/gcloud/application_default_credentials.json]  
  
These credentials will be used by any library that requests  
Application Default Credentials (ADC).
```

- Developer can now test the code using personal credentials

```
from google.cloud import bigquery  
  
#query a public dataset available in BigQuery  
query = '''  
SELECT  
    year,  
    COUNT(1) as num_babies  
FROM  
    publicdata.samples.natality  
WHERE  
    year > 2000  
GROUP BY  
    year  
'''  
  
#no credentials provided so will use ADC  
client = bigquery.Client()  
print(client.query(query).to_dataframe())
```

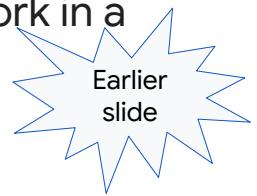
If you don't specify credentials when constructing the client, ADC will look for credentials in the environment.

Google Cloud

# How do Application Default Credentials (ADC) work in a local development environment?

- Three options
  - Each are explained on the following pages

Next topic



1

## Using developer's credentials (easy to get started this way)

Developer needs IAM roles that reflect the permissions needed by the code

```
gcloud auth application-default login
```

2

## Using service account impersonation

Developer needs **Service Account Token Creator** role to impersonate the service account

```
gcloud auth application-default login --impersonate-service-account [SERVICE_ACCT_EMAIL_HERE]
```

Creates an access token which has a limited lifetime

**Best practice**

3

## Using service account key

Developer needs **Service Account User** role plus **Service Account Key Admin** if need to create/download keys

Set the **GOOGLE\_APPLICATION\_CREDENTIALS** environment value to the path of the service account's key key

**AVOID THIS OPTION.** Keys don't expire by default

## Using service account impersonation - step 1

- Create the service account and assign the appropriate IAM roles
  - E.g., **BigQuery User** and **BigQuery Data Viewer**
    - Note: Developer must have **Owner** or **Service Account Admin** role in order to create service accounts and assign roles to them
    - Developer needs **Service Account Token Creator** role to impersonate the service account while creating ADC credentials

The screenshot shows the Google Cloud IAM & Admin interface. On the left, there's a sidebar with options: IAM & Admin (selected), IAM, Identity & Organization, and Policy Troubleshooter. The main area displays a list of users. Two users are listed:

User	E-mail	Role	Cloud IAM Roles
bigquery-user	bigquery-user@bt-dev-general.iam.gserviceaccount.com	bigquery-user	BigQuery Data Viewer BigQuery User
caliagusta	caliagusta@developers-townsendandassociates.com	Cali Agusta	BigQuery Data Viewer BigQuery User Service Account Token Creator Service Account User

Google Cloud

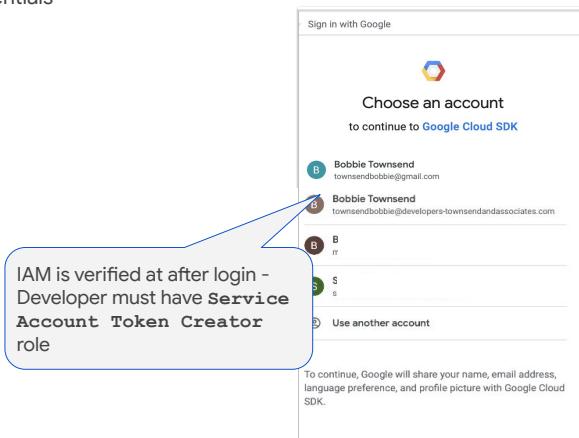
## Impersonate the service account - step 2

- Developer logs into command line with personal credentials
  - `gcloud auth login`
- Developer creates credentials for ADC
  - `gcloud auth application-default login --impersonate-service-account [SERVICE_ACCT_EMAIL_HERE]`
  - Are prompted to login
    - Use personal credentials

Output of command is the same as before

```
Credentials saved to file:  
[$HOME/.config/gcloud/application_default_credentials.json]  
  
These credentials will be used by any library that requests  
Application Default Credentials (ADC).
```

- Developer can now test the code while impersonating the service account



Google Cloud

## Service account impersonation credentials

```
{  
  "delegates": [],  
  "service_account_impersonation_url":  
    "https://iamcredentials.googleapis.com/v1/projects/-/serviceAccounts/cloud-storage-admin@bt-dev-general.iam.gserviceaccount.com:generateAccessToken",  
  "source_credentials": {  
    "client_id": "764086051850-6qr4p6gpi6hn506pt8ejuq83di341hur.apps.googleusercontent.com",  
    "client_secret": "d-FL95Q19q7M0mfPd7hHDDTY",  
    "refresh_token": "1/0d3jkl6HO3w9ICgyIARAAGA0SNwF-L9IrSCIHsMZdeku_yp6qb5-1RusEQu5GIsM6HdPTC0XksRrs0oKXu-aSJ_Qq-UzhHgUC2A",  
    "type": "authorized_user"  
  },  
  "type": "impersonated_service_account"  
}
```

- Credentials file stored in the same location as the prior example
- Client Libraries automatically retrieve the credential and refresh the access token as needed
  - *Developers do not need to manage access tokens if using Client Libraries with ADC*

# How do Application Default Credentials (ADC) work in a local development environment?

- Three options
  - Each are explained on the following pages

1

## Using developer's credentials (easy to get started this way)

Developer needs IAM roles that reflect the permissions needed by the code

```
gcloud auth application-default login
```

2

## Using service account impersonation

Developer needs **Service Account Token Creator** role to impersonate the service account

```
gcloud auth application-default login --impersonate-service-account [SERVICE_ACCT_EMAIL_HERE]
```

Creates an access token which has a limited lifetime

**Best practice**

Next topic

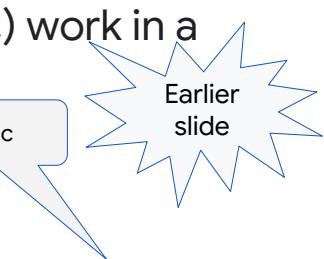
3

## Using service account key

Developer needs **Service Account User** role plus **Service Account Key Admin** if need to create/download keys

Set the **GOOGLE\_APPLICATION\_CREDENTIALS** environment value to the path of the service account's key key

**AVOID THIS OPTION.** Keys don't expire by default



# Service account key management

- Each service account can have one or more public/private RSA key pairs associated with it
  - Can be used by applications running externally to Google Cloud for authentication\*
    - Not best practice, due to key exposure risk**

The screenshot shows the Google Cloud IAM & Admin interface under the Service Accounts section. It lists three service accounts:

Email	Status	Name	Role
bigquery-qwiklab@bt-iam.iam.gserviceaccount.com	✓	bigquery-qwiklab	des Sen Acc Rol Fun lab
bucketadmin@bt-iam.iam.gserviceaccount.com	✓	bucketadmin	full control over Cloud Storage buckets
447159861369-compute@developer.gserviceaccount.com	✓	Compute Engine	No keys

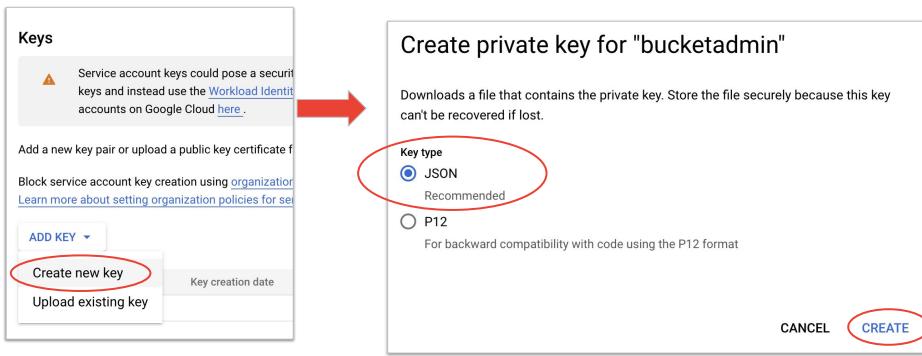
A callout box with a blue border and white text points to the 'Manage keys' button for the first service account. The text inside the box reads: "User must have Service Account Key Admin IAM role in order to do this". A red oval highlights the 'Manage keys' button, and a red arrow points from the text to the button.

\*Applications running within Google Cloud compute services do not need keys

Google Cloud

# Service Account keys must be secured

- When keys are created
  - Public key is kept in Google Cloud
  - Private key is downloaded to the developer's desktop
  - **Customer is responsible for storing the private key securely**



Google Cloud

Create and manage service account keys

<https://cloud.google.com/iam/docs/creating-managing-service-account-keys>

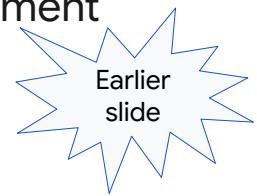
## Why service account keys can be a security issue

- Applications running in Google Cloud rely on SAs for day-to-day operations
  - SAs have more privileged access than end-user identities
    - Provide access to workloads and applications in production environments
  - **Keys by default do not expire** to help ensure operations are not interrupted by an expired key
- Key leakage is similar to a password leak - bad actors can gain access to unauthorized resources
  - For example, developers may accidentally embed SA keys in code and upload it to a code repository
    - Anyone with access to the repository can see the key and potentially use it to access to the services and APIs it is authorized to access
- Options are available to better secure keys
  - Setup [key rotation \(Service account key rotation\)](#) and monitor keys using [cloud asset inventory](#)
    - Send warning notifications to developers when a key is about to expire and it is time to rotate the key
  - Leverage third party tools such [HashiCorp Vault](#) to embed the key rotation process into the developer's workflow
  - New as of July 2023, setup [Service Account key expiry](#)
- Given the security risks associated with SA keys, **best practices is not to use them unless no other Google Cloud options are available**

# Using service account keys

- Create the service account and assign the appropriate IAM roles
  - E.g., **BigQuery User** and **BigQuery Data Viewer**
    - Note: Developer must have **Owner** or **Service Account Admin** role in order to create service accounts and assign roles to them
    - Developer needs **Service Account Key Admin** to create and download service account keys
    - Developer needs **Service Account User** to “act as” the service account
- Create and download a key (or create key on-premise and upload the public key)
- Set the **GOOGLE\_APPLICATION\_CREDENTIALS** environment variable to the path of the JSON file that contains your service account key
  - `export GOOGLE_APPLICATION_CREDENTIALS="PATH_TO_KEY_FILE"`
  - Valid for the existing shell session

# Summary - ADC in the local development environment



Developer logs into the CLI  
`gcloud auth login`

ADC can be setup 3 ways

1

## Using developer's credentials (easy to get started this way)

Developer needs IAM roles that reflect the permissions needed by the code

```
gcloud auth application-default  
login
```

USED WHEN GETTING STARTED AND DEVELOPER'S ACCOUNT HAS ROLES NEEDED BY THE APPLICATION

2

## Using service account impersonation

Developer needs **Service Account Token Creator** role to impersonate the service account

```
gcloud auth application-default  
login --impersonate-service-account  
[SERVICE_ACCT_EMAIL_HERE]
```

Creates an access token which has a limited lifetime

BEST PRACTICE

3

## Using service account key

Developer needs **Service Account User** role plus **Service Account Key Admin** if need to create/download keys

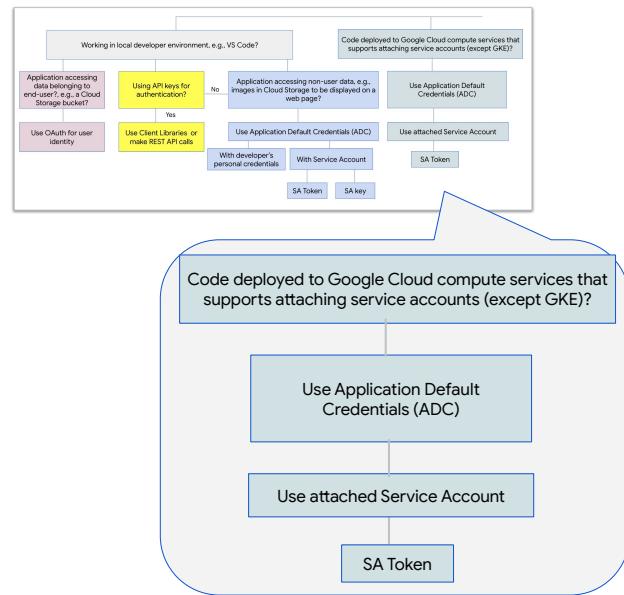
Set the `GOOGLE_APPLICATION_CREDENTIALS` environment value to the path of the service account key

AVOID THIS OPTION. Keys don't expire by default

## ADC searches for credentials in this order

- 1 Checks the `GOOGLE_APPLICATION_CREDENTIALS` environment value to see if it contains a path to a service account key
  - o Uses the key to “act as” the service account
- 2 Checks the “well known path” for credentials created by using either
  - o `gcloud auth application-default login`
    - Uses the credentials of the person logging in at the prompt that is generated
  - o `gcloud auth application-default login --impersonate-service-account [SERVICE_ACCT_EMAIL_HERE]`
    - Uses service account impersonation
- 3 If neither are found, an exception is thrown

# Authentication with Application Default Credentials (ADC)



Google Cloud

# How to provide credentials to ADC

- Method used depends on the environment where your code is running:
  - [Local development environment](#)
  - [Compute Engine or other Google Cloud services that support attaching a service account](#)
  - [Cloud Shell or other Google Cloud cloud-based development environments](#)
  - [Google Kubernetes Engine](#)
  - [On-premises or another cloud provider](#)

Next topic

Crea

## How do Application Default Credentials (ADC) work when running in Google Cloud compute services?

- Code running on the following compute uses a service account as its identity
  - Compute Engine
  - Cloud Run
  - Cloud Functions
  - App Engine
- Google Cloud provide a default service account for these services
  - Using this service account is not recommended as it is highly privileged
    - Violates the principle of least privilege
- **Best practice: Create a custom service account for each use case**

# All Google Cloud compute environments use a metadata server

- Available only from within the instance
  - Is not exposed to the public internet.
- Provides information about the running instance, including
  - ProjectID
  - Service account credentials
  - Zone in which the instance is running
- Client libraries detect when applications run in Google Cloud compute environments
  - Automatically makes a request to the metadata server to fetch the token of the attached service account
  - This token is then used to authenticate API requests made by the client library.
- Do **NOT** set ADC credentials like you did in the local environment or use the `GOOGLE_APPLICATION_CREDENTIALS` environment variable
  - This will override what the Client Libraries automatically do

## Using ADC with Compute Engine

# Retrieving metadata

- Example code to retrieve metadata from a VM after connecting to the instance
  - Always include the header `-H "Metadata-Flavor: Google"` when querying the metadata server to ensure you're interacting with the genuine metadata server

```
import requests

METADATA_URL = "http://metadata.google.internal/computeMetadata/v1/"
HEADERS = {"Metadata-Flavor": "Google"}

response = requests.get(METADATA_URL + "project/project-id", headers=HEADERS)
print("ProjectID: ", response.text)

response = requests.get(METADATA_URL +
"/instance/service-accounts/default/email", headers=HEADERS)
print("ServiceAccount: ", response.text)

response = requests.get(METADATA_URL +
"/instance/service-accounts/default/token", headers=HEADERS)
print("Token: ", response.text)
```

Shown for demo purposes only

Not something a developer typically needs to do

Client Libraries do this automatically

Output

```
ProjectID: bt-dev-general
ServiceAccount: bigquery-user@bt-dev-general.iam.gserviceaccount.com
Token: {"access_token":"ya29.c.rest-of-token-here","expires_in":3230,"token_type":"Bearer"}
```

Google Cloud

# Compute Engine metadata server

- Each compute option (GCE, GCR, GCF) stores different data on its metadata server depending on the type of compute
- Documentation regarding Compute Engine's [metadata server](#)
  - [Default values](#) include the name of the attached service account, plus more
- Another way to retrieve Compute Engine metadata
  - SSH into a VM in Google Cloud
    - To see the machine type
      - curl  
"http://metadata.google.internal/computeMetadata/v1/instance/machine-type" -H "Metadata-Flavor: Google"
    - To see the service account
      - curl  
"http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/" -H "Metadata-Flavor: Google"
- The gcloud command is used to access metadata from the developer's CLI
  - gcloud compute instances describe [VM-Name-Here] --zone [ZONE-HERE]

Google Cloud

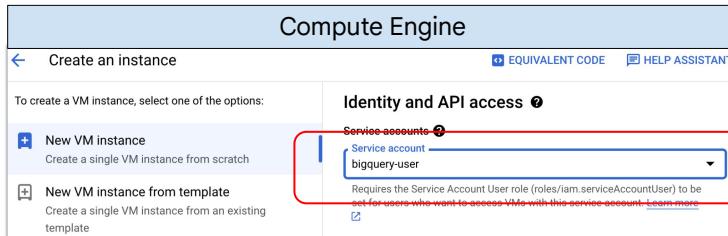
# Compute Engine metadata server

- Each compute option (GCE, GCR, GCF) will store different data on its metadata server depending on the type of compute
- Documentation regarding Compute Engine's [metadata server](#)
  - [Default values](#) include the name of the attached service account, plus more
- Another way to retrieve Compute Engine metadata
  - SSH into a VM in Google Cloud
    - To see the machine type
      - curl  
"http://metadata.google.internal/computeMetadata/v1/instance/machine-type" -H "Metadata-Flavor: Google"
    - To see the service account
      - curl  
"http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/" -H "Metadata-Flavor: Google"
- The gcloud command is used to access metadata from the developer's CLI
  - gcloud compute instances describe [VM-Name-Here] --zone [ZONE-HERE]

Google Cloud

# Using a custom service account with Compute Engine

- Developers must have
  - **Owner or Service Account Admin** role in order to create service accounts and assign roles to them
  - **Service Account User** role to attach a service account to a compute resource



Google Cloud

## Example: ADC on Compute Engine - Querying BigQuery with Python

- Same code that was running in a local development environment
  - Now it is running in a Compute Instance VM
- A service account with the **BigQuery User** and **BigQuery Data Viewer** roles is attached
- ADC queries the VM's metadata server for the attached service account and uses its token for authentication

```
from google.cloud import bigquery

#query a public dataset available in BigQuery
query = """
SELECT
    year,
    COUNT(1) as num_babies
FROM
    publicdata.samples.natality
WHERE
    year > 2000
GROUP BY
    year
"""

#no credentials provided so will use ADC
client = bigquery.Client()
print(client.query(query).to_dataframe())
```

\$ python3 query.py  
 year num\_babies  
 0 2008 4255156  
 1 2001 4031531  
 2 2002 4027376  
 3 2007 4324008  
 4 2006 4273225  
 5 2004 4118907  
 6 2003 4096092  
 7 2005 4145619

Output

DON'T specify credentials when constructing the client. ADC will query the metadata server for the attached service account

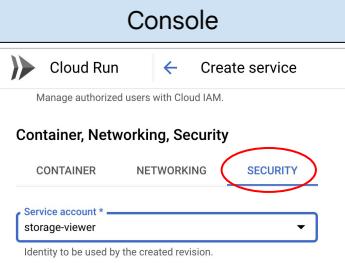
Google Cloud

## Using ADC with Cloud Run

# Using ADC with Cloud Run

- Developers must have
  - Owner or Service Account Admin role in order to create service accounts and assign roles to them
  - Service Account User role to attach a service account to a compute resource
- Set the Cloud Run service's service account using the Google Cloud console or the gcloud CLI

Console



CLI

```
gcloud run deploy [APP-NAME] --image
[IMAGE_URL] --service-account
[SERVICE-ACCOUNT-HERE]

gcloud run services update [APP-NAME]
--service-account [SERVICE-ACCOUNT-HERE]
```

Google Cloud

## Cloud Run metadata server

- Cloud Run's [metadata server](#) is similar in concept to Compute Engine metadata server
  - ADC will automatically query the metadata server to retrieve token associated with the attached service account

# Example: ADC on Cloud Run - Querying BigQuery with Python

- Same code that was running in a local development environment
  - Now it is running as a Cloud Run API
    - Can pass in a year for the SQL query
- Attach a service account with the **BigQuery User** and **BigQuery Data Viewer** roles
- ADC will query the metadata server for the service account token and use that for authentication and authorization

```
import os

from flask import Flask, request, jsonify
from google.cloud import bigquery

app = Flask(__name__)

@app.route('/get_num_babies', methods=['GET'])
def get_num_babies():
    year = request.args.get('year', default=2000, type=int)

    query = f"""
        SELECT
            year,
            COUNT(1) as num_babies
        FROM
            publicdata.samples.natality
        WHERE
            year > {year}
        GROUP BY
            year
        ...
    """

    client = bigquery.Client()
    result = client.query(query).to_dataframe().to_dict(orient='records')
    return jsonify(result)

if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=int(os.environ.get("PORT", 8080)))
```

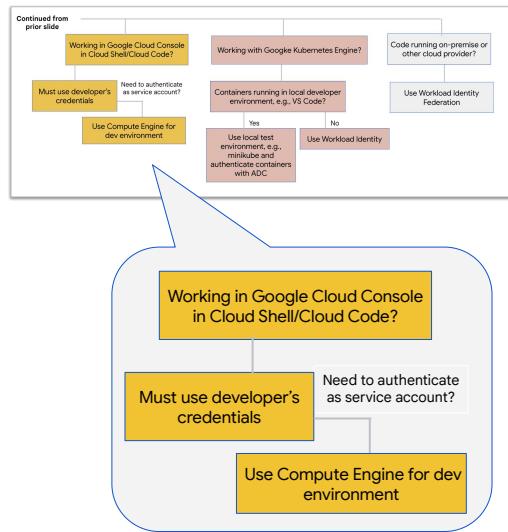
Google Cloud

## Using ADC with Cloud Functions or App Engine

## Using ADC with Cloud Functions or App Engine

- ADC uses the metadata server for Cloud Functions and App Engine in a manner similar to Compute Engine and Cloud Run
  - No further discussion is needed

# Authenticating to Cloud Shell or Cloud Code in Cloud Shell



Google Cloud

# How to provide credentials to ADC

- Method used depends on the environment where your code is running:
  - [Local development environment](#)
  - [Compute Engine or other Google Cloud services that support attaching a service account](#)
  - [Cloud Shell or other Google Cloud cloud-based development environments](#)
  - [Google Kubernetes Engine](#)
  - [On-premises or another cloud provider](#)

Next topic

Crea

# Authenticating to Cloud Shell or Cloud Code in Cloud Shell

- Cloud Shell and Cloud Code in Cloud Shell use the credentials provided upon login
  - Cannot change the ADC credentials in Cloud Shell by using either of these commands
    - gcloud auth application-default login**
    - gcloud auth application-default login --impersonate-service-account [SERVICE\_ACCT\_EMAIL\_HERE]**
- If you need to provide a different user account to ADC, or provide credentials by using a service account,
  - Use a local development environment or a Google Cloud compute resource as your development environment

```

package main

// This file is generated by the command "make generate". It contains
// logic to handle the "get" command. If you want to change how "get"
// works, edit this file directly.
// See https://github.com/mongodb/mongo-go-driver/blob/master/mongo/doc.go#L100
// for more information about the "get" command.

import (
    "context"
    "log"
    "time"

    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
)

var (
    mongoURI = "mongodb://127.0.0.1:27017"
    dbName   = "test"
    collName = "testCollection"
)

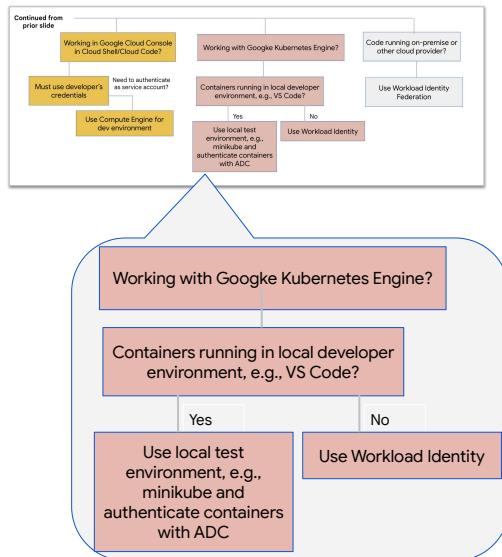
func main() {
    client, err := mongo.Connect(context.TODO(), options.Client().ApplyURI(mongoURI))
    if err != nil {
        log.Fatal(err)
    }
    defer client.Disconnect(context.TODO())
    if err := client.Ping(context.TODO(), options.Ping()); err != nil {
        log.Fatal(err)
    }
    coll := client.Database(dbName).Collection(collName)
    if err := coll.InsertOne(context.TODO(), bson.D{{}}); err != nil {
        log.Fatal(err)
    }
}

```

Cloud Code with Cloud Shell editor

Google Cloud

# Authenticating to a Google Kubernetes Engine environment



Google Cloud

# How to provide credentials to ADC

- Method used depends on the environment where your code is running:
  - [Local development environment](#)
  - [Compute Engine or other Google Cloud services that support attaching a service account](#)
  - [Cloud Shell or other Google Cloud cloud-based development environments](#)
  - [Google Kubernetes Engine\\*](#)
  - [On-premises or another cloud provider](#)

Next topic

\*This discussion assumes a knowledge of Kubernetes

Google Cloud

Crea

# Testing containers locally

- Use a local Kubernetes implementation such as [minikube](#) and the [GCP Auth minikube addon](#)
  - Configure containers to authenticate with ADC
- When containerized applications are running on Google Cloud
  - Use [Workload Identity](#) for GKE

Focus of this section

## Automated Google Cloud Platform Authentication

The `gcp-auth` addon automatically and dynamically configures pods to use your credentials, allowing applications to access Google Cloud services as if they were running within Google Cloud.

The addon defaults to using your environment's [Application Default Credentials](#), which you can configure with `gcloud auth application-default login`. Alternatively, you can specify a JSON credentials file (e.g. service account key) by setting the `GOOGLE_APPLICATION_CREDENTIALS` environment variable to the location of that file.

The addon also defaults to using your local gcloud project, which you can configure with `gcloud config set project <project name>`. You can override this by setting the `GOOGLE_CLOUD_PROJECT` environment variable to the name of the desired project.

Once the addon is enabled, pods in your cluster will be configured with environment variables (e.g. `GOOGLE_APPLICATION_DEFAULTS`, `GOOGLE_CLOUD_PROJECT`) that are automatically used by GCP client libraries. Additionally, the addon configures [registry pull secrets](#), allowing your cluster to access the container images hosted in [Artifact Registry](#) and [Google Container Registry](#).

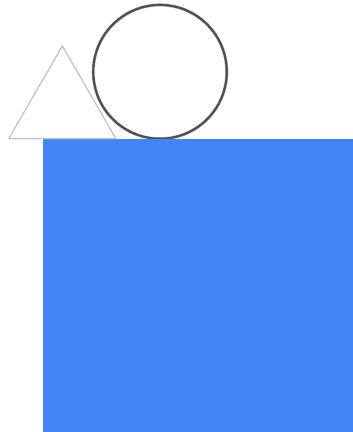
Using the [gcp-auth addon](#) for minikube

[Minikube tutorial](#)

Google Cloud

# Application Authentication

- Overview
- API keys
- Application Default Credentials
- **Workload Identity**
- Workload Identity Federation



Google Cloud

# GKE Workload Identity

- Workload Identity is the recommended way for workloads to access Google Cloud services in a secure way
  - A Kubernetes service account (KSA) in a specific namespace is bound to a Google Cloud service account (GSA)
  - Allows the KSA to act as the GSA for the purpose of accessing Google Cloud services.
- When enabled
  - When a pod uses a KSA to access Google Cloud services, the GKE metadata server issues an identity token.
    - Has the necessary claims to act as the associated Google Cloud service account
    - The token is then used to authenticate the application to the Google Cloud service

# Guide: Workload Identity

Topics covered include

- Workload Identity
- An overview of how to map Kubernetes RBAC service account to Google Cloud service accounts

## Guide: Workload Identity

### What is Workload Identity?

- [Workload Identity](#) allows a Kubernetes service account in your GKE cluster to act as an IAM service account. Pods that use the configured Kubernetes service account automatically authenticate as the IAM service account when accessing Google Cloud APIs. Using Workload Identity allows you to assign distinct, fine-grained identities and authorization for each application in your cluster

### Youtube video: Secure access to GKE workloads with Workload Identity

- In this video, you learn how to map Kubernetes RBAC roles to Google Cloud IAM Service Accounts



<https://www.youtube.com/watch?v=4OzbPaJCUr8>

## Workload Identity

Google Cloud

# Service accounts usage

- A GKE cluster is a collection of compute instance “nodes”
  - **Each node has an attached service account**
    - Used to grant the VM instances permissions to interact with Google Cloud services
      - For example, to pull container images from the Artifact Registry
      - These SAs are not used as part of Workload Identity
    - An **Kubernetes service account (KSA)**
      - An **identity within Kubernetes used to grant permissions to workloads (like pods) running inside the cluster**
        - For example, a pod can use a KSA with the appropriate permissions to list other pods in the same namespace
        - Permissions are managed through Kubernetes' Role-Based Access Control (RBAC)
  - **Workload Identity**
    - **IAM role based service accounts**
    - **KSAs are bound to these accounts**
      - Allows workloads running in GKE (using a specific KSA) to act as the associated Google Cloud service account to access Google Cloud services, e.g., Cloud Storage

# Enabling workload identity

- To get started, enable Workload Identity on clusters and node pools using the Google Cloud CLI or the Google Cloud console.
  - Once enabled at the cluster level is automatically enabled on node pools

```
gcloud container clusters create workload-identity-test \
--zone us-central1-a --machine-type=e2-micro \
--workload-pool=bt-dev-general.svc.id.goog
```

Enabling the workload identity

[Use Workload Identity](#)

Google Cloud

## Demo: Secret Manager access

- [Secret Manager](#) is a service that provides a secure and convenient method for storing API keys, passwords, certificates, and other sensitive data
  - For the purpose of this demo, a secret named **vision-api-key** has already been created

Secret details

SECRET: "vision-api-key"

projects/96147636371/secrets/vision-api-key

OVERVIEW VERSIONS PERMISSIONS LOGS

Versions + NEW VERSION ENABLE DISABLE DESTROY

Filter Enter property name or value

	Version	Aliases	Status	Encryption	Created on
<input type="checkbox"/>	1	-	<span>Enabled</span>	Google-managed	8/28/23, 2:50 PM

No versions selected

[Create a secret](#)

Google Cloud

## Create service accounts with Secret Manager roles

- Create 2 service accounts using IAM
  - One has read-only permission to the **vision-api-key** in Secret Manager
  - One has read-write permission to the **vision-api-key** in Secret Manager

<input type="checkbox"/>	 readonly-secrets@bt-dev-general.iam.gserviceaccount.com	readonly-secrets	Secret Manager Secret Accessor
<input type="checkbox"/>	 readwrite-secrets@bt-dev-general.iam.gserviceaccount.com	readwrite-secrets	Secret Manager Secret Accessor Secret Manager Secret Version Adder

# Create pods with different namespaces and RBAC roles

- Create 2 namespaces

- `kubectl create namespace readonly-ns`
- `kubectl create namespace admin-ns`

*#This Pod uses the readonly-sa service account in the readonly-ns namespace*

```
apiVersion: v1
kind: Pod
metadata:
  name: readonly-test
  namespace: readonly-ns
spec:
  containers:
    - image: google/cloud-sdk:slim
      name: workload-identity-test
      command: ["sleep","infinity"]
  resources:
    requests:
      cpu: "150m"
      memory: "150Mi"
  serviceAccountName: readonly-sa
```

In the  
readonly-ns  
namespace

Kubernetes  
RBAC role

- Create 2 pods

- `kubectl apply -f readonly-pod.yaml`
- `kubectl apply -f readwrite-pod.yaml`

*#This Pod uses the admin-sa service account in the admin-ns namespace.*

```
apiVersion: v1
kind: Pod
metadata:
  name: admin-test
  namespace: admin-ns
spec:
  containers:
    - image: google/cloud-sdk:slim
      name: workload-identity-test
      command: ["sleep","infinity"]
  resources:
    requests:
      cpu: "150m"
      memory: "150Mi"
  serviceAccountName: admin-sa
```

In the  
admin-ns  
namespace

Kubernetes  
RBAC role

[Kubernetes best practices: Organizing with Namespaces](#)

Google Cloud

## Bind the Google Cloud IAM service accounts to the Kubernetes service accounts

```
gcloud iam service-accounts add-iam-policy-binding  
readonly-secrets@bt-dev-general.iam.gserviceaccount.com \\\n    --member=serviceAccount:bt-dev-general.svc.id.goog[readonly-ns/readonly-sa] \\  
    --role='roles/iam.workloadIdentityUser'
```

Google Cloud IAM service account

Google Cloud IAM role

GKE name space and RBAK service account

```
gcloud iam service-accounts add-iam-policy-binding  
readwrite-secrets@bt-dev-general.iam.gserviceaccount.com \\\n    --member=serviceAccount:bt-dev-general.svc.id.goog[admin-ns/admin-sa] \\  
    --role='roles/iam.workloadIdentityUser'
```

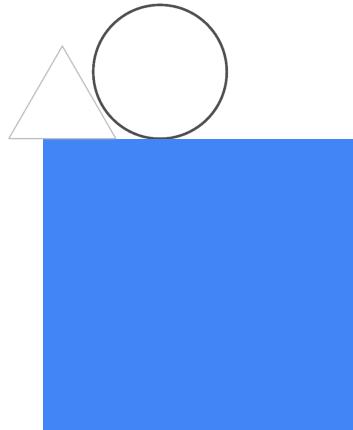
Google Cloud

## Connect to the pods and test access to Secret Manager

- Open a shell in the read-only pod
  - `kubectl exec -it readonly-test --namespace=readonly-ns -- /bin/bash`
- Can I access the secret? (yes)
  - `gcloud secrets versions access 1 --secret=vision-api-key`
- Can I write? (no)
  - `printf "my-second-api-key" | gcloud secrets versions add vision-api-key --data-file=-`
- **Exit** the pod
- Repeat with
  - `kubectl exec -it admin-test --namespace=admin-ns -- /bin/bash`

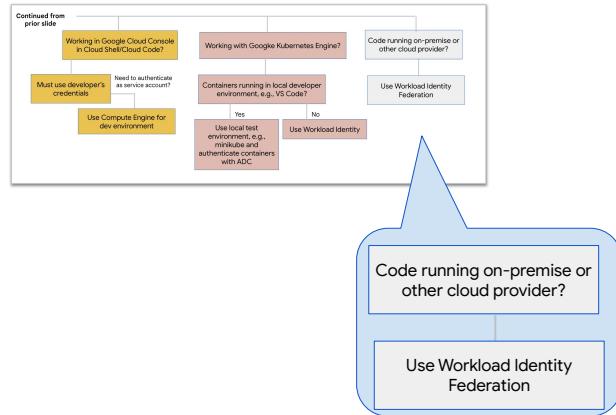
# Application Authentication

- Overview
- API keys
- Application Default Credentials
- Workload Identity
- **Workload Identity Federation**



Google Cloud

# Authenticate code running in on-premises or another cloud provider



Google Cloud

# How to provide credentials to ADC

- Method used depends on the environment where your code is running:
  - [Local development environment](#)
  - [Compute Engine or other Google Cloud services that support attaching a service account](#)
  - [Cloud Shell or other Google Cloud cloud-based development environments](#)
  - [Google Kubernetes Engine](#)
  - [On-premises or another cloud provider](#)

Next topic

Crea

## Applications running on-premise or in other clouds

- Application running outside of Google Cloud need to provide credentials that are recognized by Google Cloud to use Google Cloud services
  - In the past, this was accomplished through the use of downloaded service account keys
- [Workload identity federation](#) is the preferred way to grant on-premises or multi-cloud workloads access to Google Cloud resources
  - Can grant external identities IAM roles, including the ability to impersonate service accounts
    - Service account keys are no longer required

# Guide: Workload Identity Federation

Topics covered include

- Workload Identity Federation
- An example of how to use it in a CI/CD pipeline with GitHub Actions

## Guide: Workload Identity Federation

### Workload Identity Federation

- This [document](#) provides an overview of identity federation for external workloads. Using identity federation, you can grant on-premises or multi-cloud workloads access to Google Cloud resources, without using a service account key.
  - Works with

### Youtube video: What is Workload Identity Federation

- This video provides an overview of Workload Identity Federation



<https://www.youtube.com/watch?v=4vajaxZHNQ8>

## Workload Identity Federation

Google Cloud

