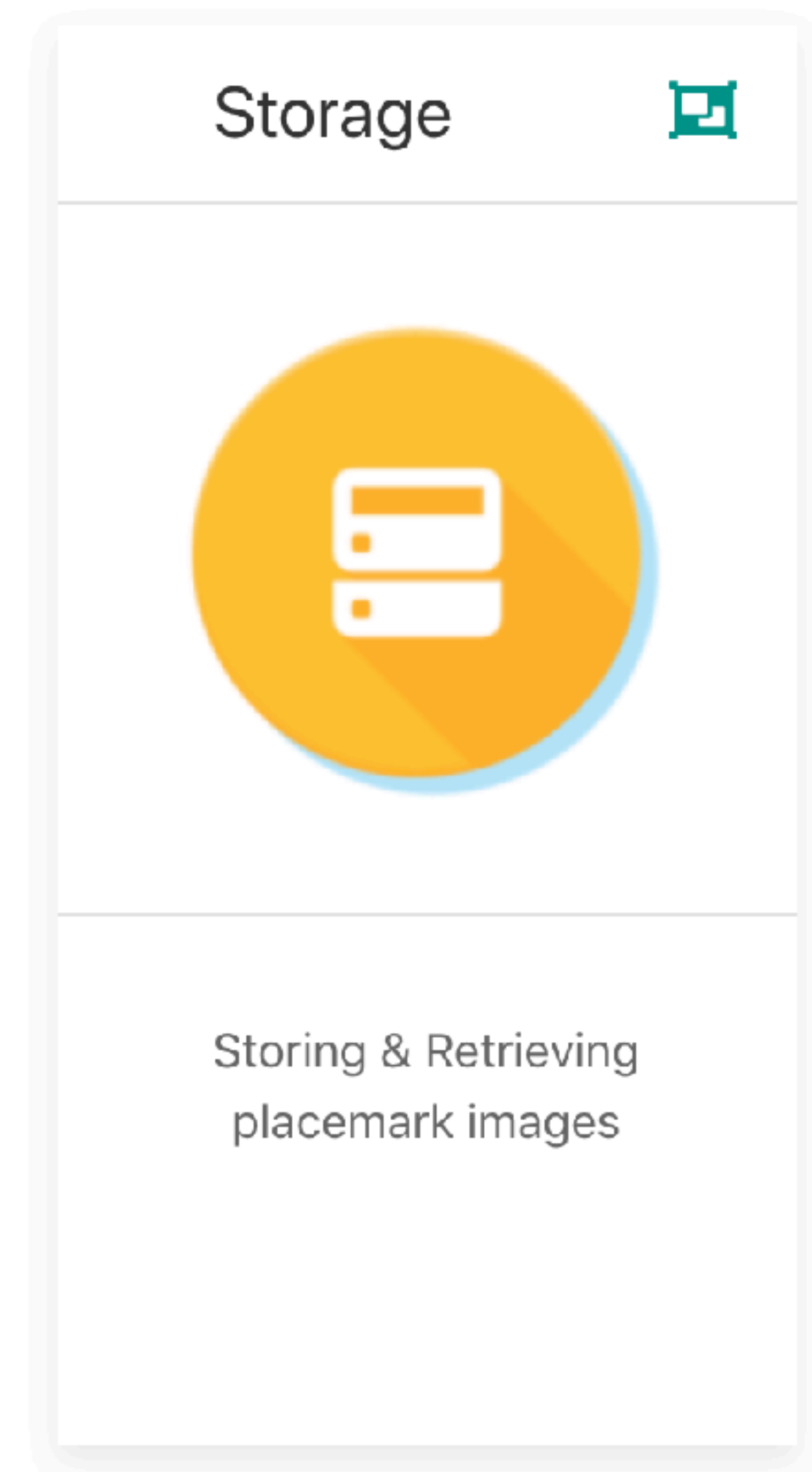


# Firebase Cloud Storage

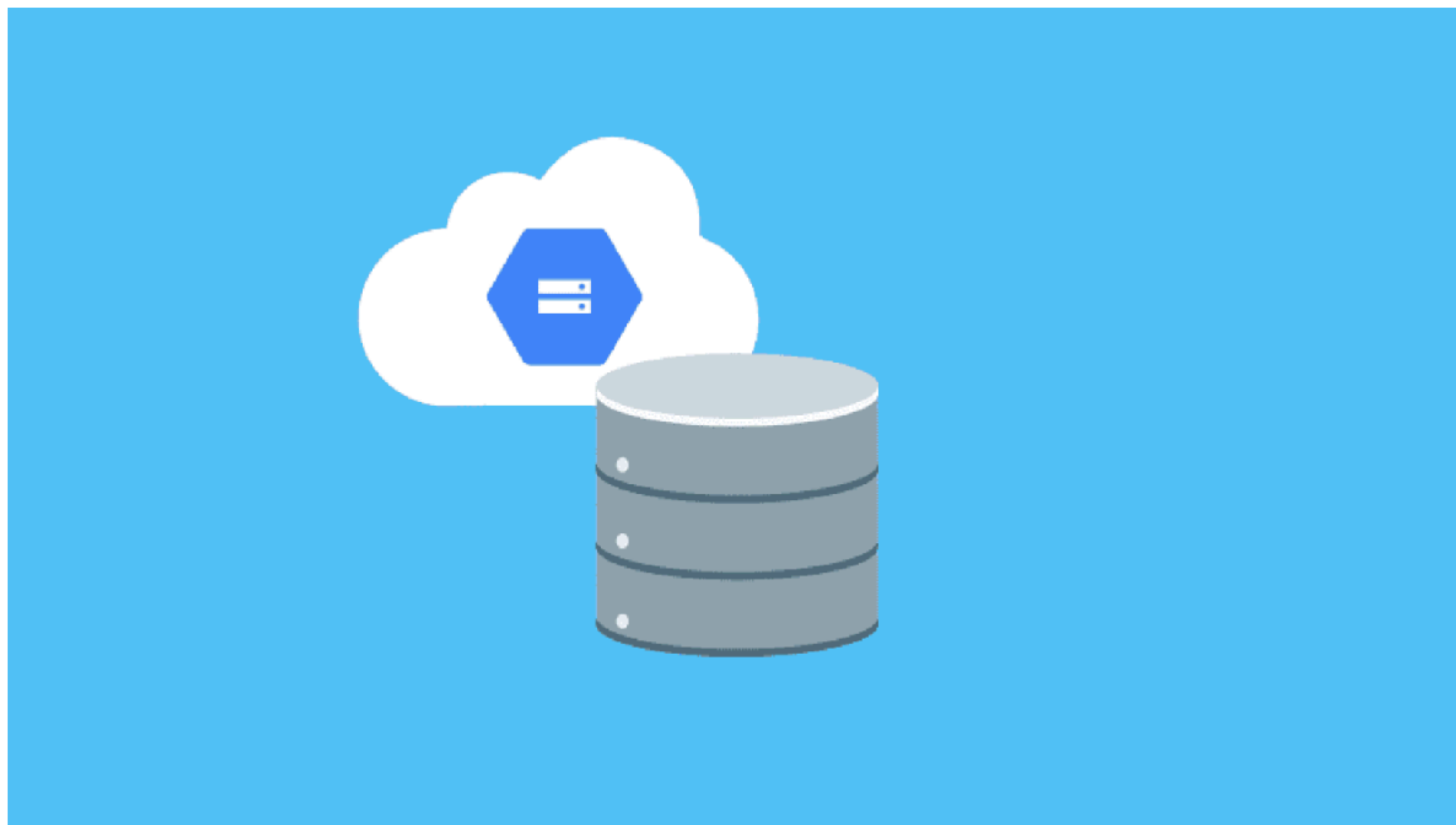


# Store your users' photos and videos

Cloud Storage is designed to help you quickly and easily store and serve user-generated content, such as photos and videos.



## Cloud Storage



## Build at Google scale

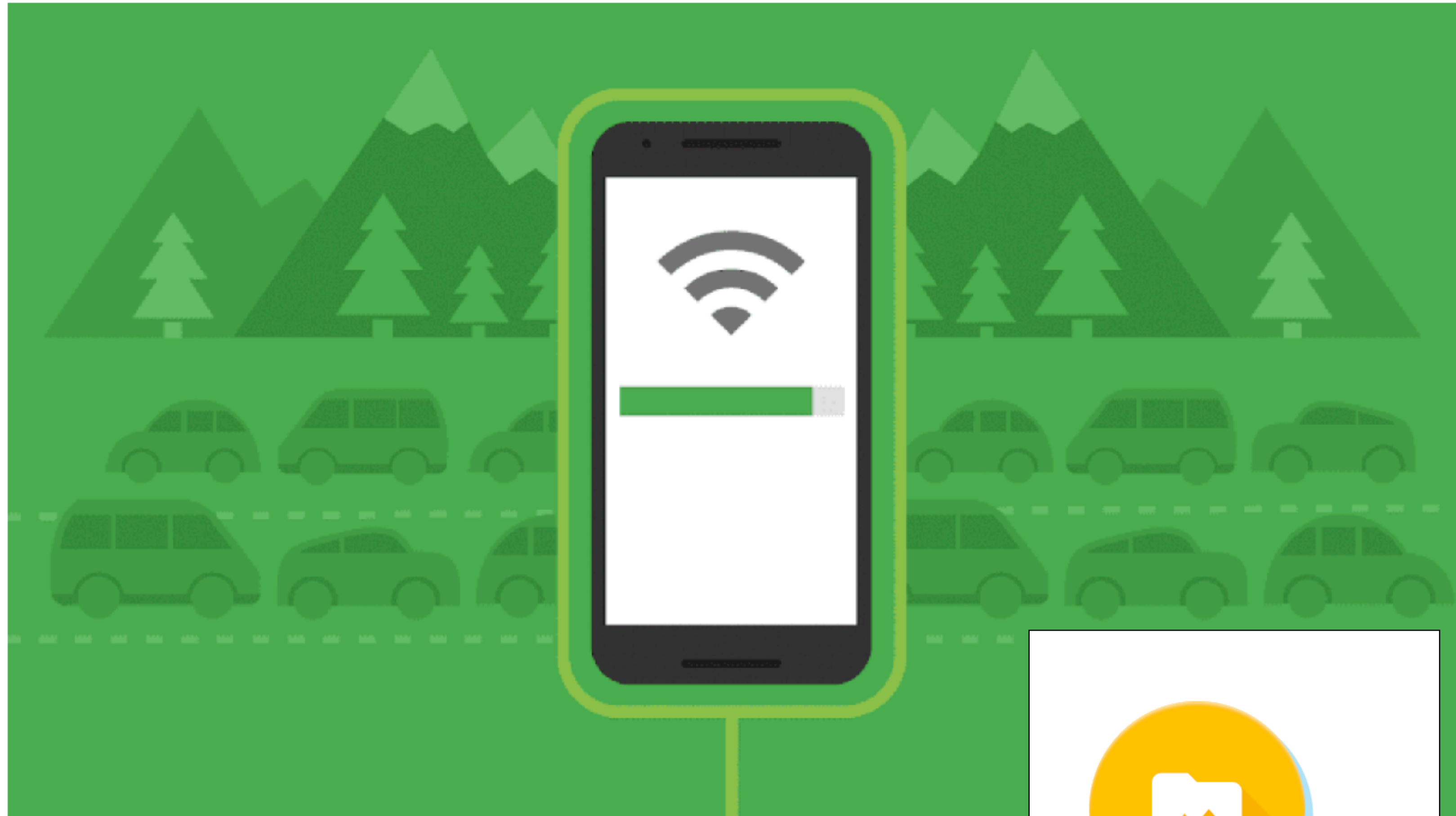
Our infrastructure is built for when your app goes viral. Effortlessly grow from prototype to production using the same technology that powers apps like Spotify and Google Photos.



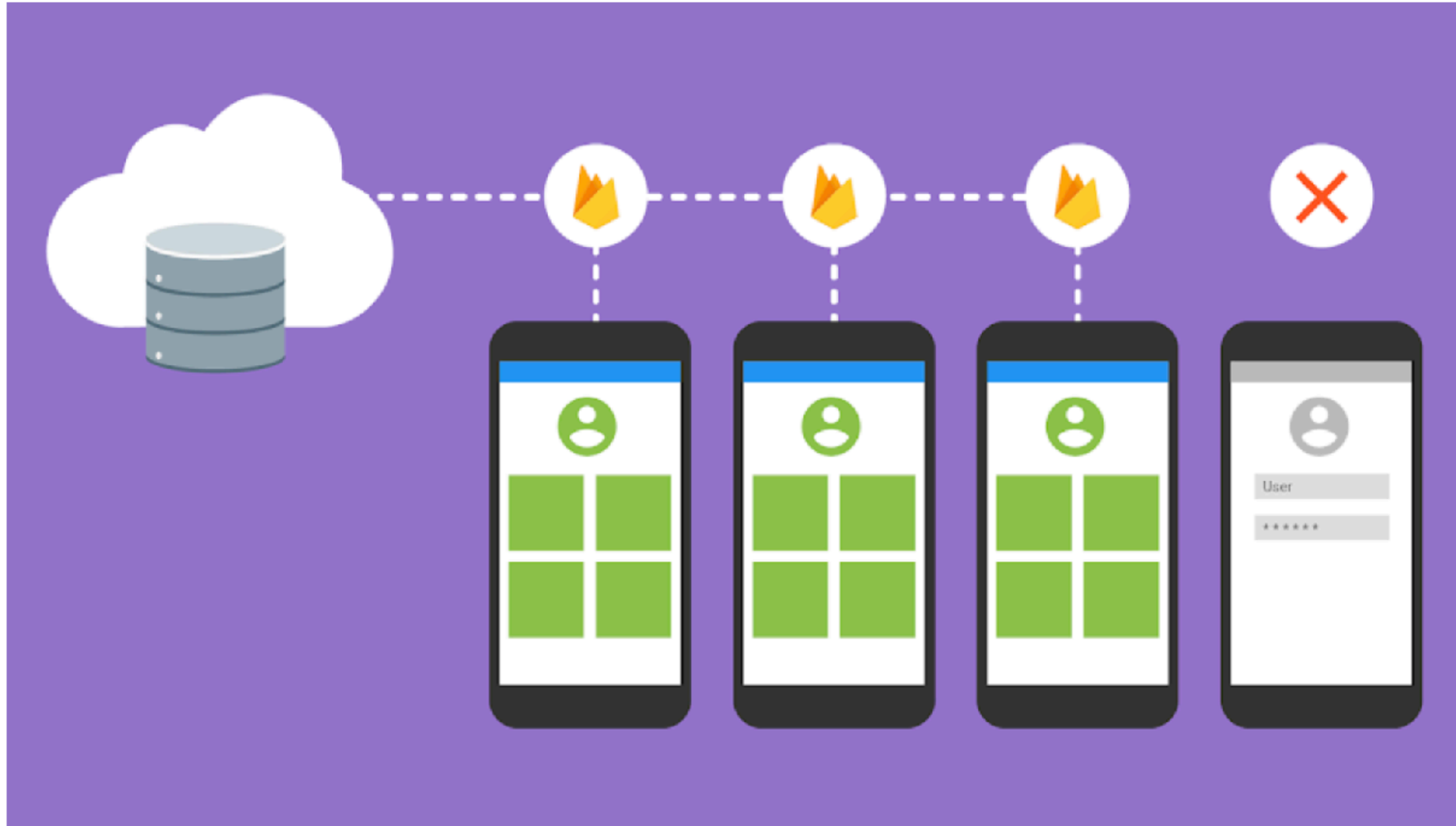
**Cloud Storage**

# Robust uploads and downloads

Your users aren't always online, so we built the Firebase SDK for Cloud Storage with mobile connectivity in mind. It will automatically pause and resume your transfers as the app loses and regains mobile connectivity, saving your users time and bandwidth.



**Cloud Storage**



## Strong user-based security

The Firebase SDK for Cloud Storage integrates with Firebase Authentication to provide simple and intuitive access control. You can use our declarative security model to allow access based on user identity or properties of a file, such as name, size, content type, and other metadata.



**Cloud Storage**



First Introduce Glide Library: <https://github.com/bumptech/glide>

Provides a range of  
image manipulation  
features

*Including loading images  
into ImageView via a URL*

## Glide

maven central 4.10.0 build passing | [View Glide's documentation](#) | [简体中文文档](#) | [Report an issue with Glide](#)

Glide is a fast and efficient open source media management and image loading framework for Android that wraps media decoding, memory and disk caching, and resource pooling into a simple and easy to use interface.



Glide supports fetching, decoding, and displaying video stills, images, and animated GIFs. Glide includes a flexible API that allows developers to plug in to almost any network stack. By default Glide uses a custom `HttpURLConnection` based stack, but also includes utility libraries plug in to Google's Volley project or Square's OkHttp library instead.

Glide's primary focus is on making scrolling any kind of a list of images as smooth and fast as possible, but Glide is also effective for almost any case where you need to fetch, resize, and display a remote image.

### build.gradle

```
...  
implementation 'com.github.bumptech.glide:glide:4.10.0'  
...
```

Replace all calls to setImage:


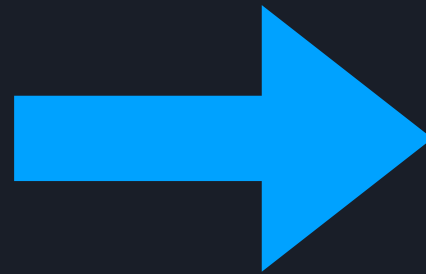
```
placemarkImage.setImageBitmap(readImageFromPath(this, placemark.image))
```

with

```
Glide.with(this).load(placemark.image).into(placemarkImage);
```

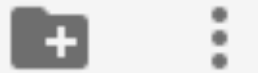
This will continue to render locally stored images  
But will also work with firebase storage hosted images

*Change PlacemarkAdapter, PlacemarkView,  
PlacemarkMapView*

**Develop** **Authentication** **Database** **Storage** **Hosting** **Functions** **ML Kit****Storage**

Store and retrieve user-generated files like images, audio, and video without server-side code

[Learn more](#) [View the docs](#)

**GET STARTED****Storage****FILES****RULES****USAGE** `gs://placemark-test.appspot.com` **UPLOAD FILE****Name****Size****Type****Last modified**

There are no files here yet

# Firestore Console



# Firebase Assistant in Studio

user-generated content.

[Launch in browser](#)

---

① **Connect your app to Firebase**

✓ **Connected**

② **Add Cloud Storage to your app**

[Add Cloud Storage to your app](#)



## Storage

FILES RULES USAGE

gs://placemark-test.appspot.com [UPLOAD FILE](#)

<input type="checkbox"/>	Name	Size	Type	Last modified
There are no files here yet				

**build.gradle**

```
implementation "com.google.firebase:firebase-storage:$firebase_version"
```

user-generated content.

[Launch in browser](#)

---

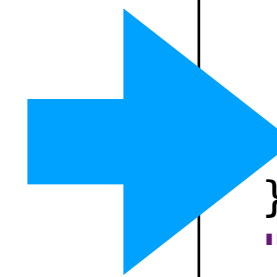
① **Connect your app to Firebase**

✔ Connected

② **Add Cloud Storage to your app**

Add Cloud Storage to your app

google-services.json





```
{
  "project_info": {
    "project_number": "1062442537261",
    "firebase_url": "https://placemark-222108.firebaseio.com",
    "project_id": "placemark-222108",
    "storage_bucket": "placemark-222108.appspot.com"
  },
  "client": [
    {
      "client_info": {
        "mobilesdk_app_id": "1:1062442537261:android:634c4d908a4ce143",
        "android_client_info": {
          "package_name": "org.wit.placemark"
        }
      },
      "oauth_client": [
        {
          "client_id": "10624425372XXXXXXXXXXXXq320l17eu0pdv.apps.googleusercontent.com",
          "client_type": 1,
          "android_info": {
            "package_name": "org.wit.placemark",
            "certificate_hash": "368ead570ae3aa95a69bd78936dc4f2123a7ed96"
          }
        },
        {
          "client_id": "1062442537261-uXXXXXXXXXXXX0avh4p4l5eqc4.apps.googleusercontent.com",
          "client_type": 3
        }
      ],
      "api_key": [
        {
          "current_key": "AIzaSyBXXXXXXXXXXXXXXXX52I95o"
        }
      ],
      "services": {
        "analytics_service": {
          "status": 1
        },
        "appinvite_service": {
          "status": 2,
          "other_platform_oauth_client": [
            {
              "client_id": "106244253XXXXXXXXXXXXhp4l5eqc4.apps.googleusercontent.com",
              "client_type": 3
            }
          ]
        },
        "ads_service": {
          "status": 2
        }
      }
    }
  ],
  "configuration_version": "1"
}
```


# Uploading Image to Storage

gs://placemark-222108.appspot.com > fUZhVTJ1Z...enEzYWLwo2

Upload file

<input type="checkbox"/>	Name	Size	Type	Last modified
<input type="checkbox"/>	 321	221....	image/jpeg	Nov 18, 2...

 321



Name

321

Size

226,346 bytes

Type

image/jpeg

Created

Nov 18, 2018, 12:46:02 PM

Updated

Nov 18, 2018, 12:46:02 PM

File location


Other metadata

gs://placemark-222108.appspot.com

Upload file

<input type="checkbox"/>	Name	Size	Type	Last modified
<input type="checkbox"/>	 9VCoy8TSQCU08vMKAMg1qYfi2RT2/	—	Folder	—
<input type="checkbox"/>	 fUZhVTJ1ZhROOU0I1CenEzYWLwo2/	—	Folder	—
<input type="checkbox"/>	 lh7wuYj3vheuQvAAWejAtSWtH0i1/	—	Folder	—

321



Name  
321

Size  
226,346 bytes

Type  
image/jpeg

Created  
Nov 18, 2018, 12:46:02 PM

Updated  
Nov 18, 2018, 12:46:02 PM

File location

Storage location  
gs://placemark-222108.appspot.com/fUZhVTJ1ZhROOU0l1CenEzYWLwo2/321

Download URL 1 [revoke](#)  
<https://firebasestor...6-9b4c-f0d753351d9c>

[Create new download URL](#)

Other metadata

# Database

Realtime Database

Data Rules Backups Usage

https://placemark-222108.firebaseio.com/

placemark-222108

- users
  - 9VCoy8TSQCU08vMKAMg1qYfi2RT2
    - placemarks
      - LRajxT1JeBmOldBJLx-
      - LRak-ZGQC28xel8Imbh
      - LRalieCFxhl-Nng3nu3
      - LRbdKYneAL81oF5WKEk
        - description: "\"Ecellent Location"
        - fbId: "-LRbdKYneAL81oF5WKEk"
        - id: 0
        - image: "https://firebasestorage.googleapis.com/v0/b/pla..."
        - location
          - lat: 52.245695
          - lng: -7.1391017
          - zoom: 15
          - title: "asd"

fUZhVTJ1ZhROOU0l1CenEzYWLwo2

Store image url in  
Database

```
fun updateImage(placemark: PlacemarkModel) {  
    if (placemark.image != "") {  
  
        // get the full image file name  
  
  
        // create storage object to be uploaded to the cloudstore (node name is id of user)  
  
  
        // read the image into a bitmap object  
  
  
  
        // compress the image into a byte array  
  
  
  
        // put the bytes unto the object and start upload (asynchronous call)  
  
  
        // failure  
  
  
        // success, get full path of uploaded image (asynchronous call)  
        // recover full path  
        // store full path in database  
  
    }  
}
```



```
fun updateImage(placemark: PlacemarkModel) {  
    if (placemark.image != "") {  
  
        // get the full image file name  
        val fileName = File(placemark.image)  
        val imageName = fileName.getName()  
  
        // create storage object to be uploaded to the cloudstore (node name is id of user)  
  
        // read the image into a bitmap object  
  
  
        // compress the image into a byte array  
  
  
        // put the bytes unto the object and start upload (asynchronous call)  
  
        // failure  
  
        // success, get full path of uploaded image (asynchronous call)  
        // recover full path  
        // store full path in database  
  
    }  
}
```

```
fun updateImage(placemark: PlacemarkModel) {  
    if (placemark.image != "") {  
  
        // get the full image file name  
        val fileName = File(placemark.image)  
        val imageName = fileName.getName()  
  
        // create storage object to be uploaded to the cloudstore (node name is id of user)  
        var imageRef = st.child(userId + '/' + imageName)  
  
        // read the image into a bitmap object  
  
  
        // compress the image into a byte array  
  
  
        // put the bytes unto the object and start upload (asynchronous call)  
  
        // failure  
  
        // success, get full path of uploaded image (asynchronous call)  
        // recover full path  
        // store full path in database  
  
    }  
}
```

```
fun updateImage(placemark: PlacemarkModel) {  
    if (placemark.image != "") {  
  
        // get the full image file name  
        val fileName = File(placemark.image)  
        val imageName = fileName.getName()  
  
        // create storage object to be uploaded to the cloudstore (node name is id of user)  
        var imageRef = st.child(userId + '/' + imageName)  
  
        // read the image into a bitmap object  
        val bitmap = readImageFromPath(context, placemark.image)  
  
        bitmap?.let {  
            // compress the image into a byte array  
  
  
  
  
            // put the bytes unto the object and start upload (asynchronous call)  
  
  
            // failure  
  
  
            // success, get full path of uploaded image (asynchronous call)  
            // recover full path  
            // store full path in database  
  
        }  
    }  
}
```

```
fun updateImage(placemark: PlacemarkModel) {  
    if (placemark.image != "") {  
  
        // get the full image file name  
        val fileName = File(placemark.image)  
        val imageName = fileName.getName()  
  
        // create storage object to be uploaded to the cloudstore (node name is id of user)  
        var imageRef = st.child(userId + '/' + imageName)  
  
        // read the image into a bitmap object  
        val bitmap = readImageFromPath(context, placemark.image)  
  
        bitmap?.let {  
            // compress the image into a byte array  
            val baos = ByteArrayOutputStream()  
            bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)  
            val data = baos.toByteArray()  
  
            // put the bytes unto the object and start upload (asynchronous call)  
  
            // failure  
  
            // success, get full path of uploaded image (asynchronous call)  
            // recover full path  
            // store full path in database  
        }  
    }  
}
```

```
fun updateImage(placemark: PlacemarkModel) {  
    if (placemark.image != "") {  
  
        // get the full image file name  
        val fileName = File(placemark.image)  
        val imageName = fileName.getName()  
  
        // create storage object to be uploaded to the cloudstore (node name is id of user)  
        var imageRef = st.child(userId + '/' + imageName)  
  
        // read the image into a bitmap object  
        val bitmap = readImageFromPath(context, placemark.image)  
  
        bitmap?.let {  
            // compress the image into a byte array  
            val baos = ByteArrayOutputStream()  
            bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)  
            val data = baos.toByteArray()  
  
            // put the bytes unto the object and start upload (asynchronous call)  
            val uploadTask = imageRef.putBytes(data)  
            uploadTask.addOnFailureListener {  
                // failure  
  
                // success, get full path of uploaded image (asynchronous call)  
  
                // recover full path  
  
                // store full path in database  
  
            }  
        }  
    }  
}
```



```
fun updateImage(placemark: PlacemarkModel) {  
    if (placemark.image != "") {  
  
        // get the full image file name  
        val fileName = File(placemark.image)  
        val imageName = fileName.getName()  
  
        // create storage object to be uploaded to the cloudstore (node name is id of user)  
        var imageRef = st.child(userId + '/' + imageName)  
  
        // read the image into a bitmap object  
        val bitmap = readImageFromPath(context, placemark.image)  
  
        bitmap?.let {  
            // compress the image into a byte array  
            val baos = ByteArrayOutputStream()  
            bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)  
            val data = baos.toByteArray()  
  
            // put the bytes unto the object and start upload (asynchronous call)  
            val uploadTask = imageRef.putBytes(data)  
            uploadTask.addOnFailureListener {  
                // failure  
                println(it.message)  
            }.addOnSuccessListener { taskSnapshot ->  
                // success, get full path of uploaded image (asynchronous call)  
  
                // recover full path  
  
                // store full path in database  
  
            }  
        }  
    }  
}
```

```

fun updateImage(placemark: PlacemarkModel) {
    if (placemark.image != "") {

        // get the full image file name
        val fileName = File(placemark.image)
        val imageName = fileName.getName()

        // create storage object to be uploaded to the cloudstore (node name is id of user)
        var imageRef = st.child(userId + '/' + imageName)

        // read the image into a bitmap object
        val bitmap = readImageFromPath(context, placemark.image)

        bitmap?.let {
            // compress the image into a byte array
            val baos = ByteArrayOutputStream()
            bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)
            val data = baos.toByteArray()

            // put the bytes unto the object and start upload (asynchronous call)
            val uploadTask = imageRef.putBytes(data)
            uploadTask.addOnFailureListener {
                // failure
                println(it.message)
            }.addOnSuccessListener { taskSnapshot ->
                // success, get full path of uploaded image (asynchronous call)
                taskSnapshot.metadata!!.reference!!.downloadUrl.addOnSuccessListener {
                    // recover full path

                    // store full path in database
                }
            }
        }
    }
}

```

```
fun updateImage(placemark: PlacemarkModel) {  
    if (placemark.image != "") {  
  
        // get the full image file name  
        val fileName = File(placemark.image)  
        val imageName = fileName.getName()  
  
        // create storage object to be uploaded to the cloudstore (node name is id of user)  
        var imageRef = st.child(userId + '/' + imageName)  
  
        // read the image into a bitmap object  
        val bitmap = readImageFromPath(context, placemark.image)  
  
        bitmap?.let {  
            // compress the image into a byte array  
            val baos = ByteArrayOutputStream()  
            bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)  
            val data = baos.toByteArray()  
  
            // put the bytes unto the object and start upload (asynchronous call)  
            val uploadTask = imageRef.putBytes(data)  
            uploadTask.addOnFailureListener {  
                // failure  
                println(it.message)  
            }.addOnSuccessListener { taskSnapshot ->  
                // success, get full path of uploaded image (asynchronous call)  
                taskSnapshot.metadata!!.reference!!.downloadUrl.addOnSuccessListener {  
                    // recover full path  
                    placemark.image = it.toString()  
                    // store full path in database  
  
                }  
            }  
        }  
    }  
}
```

```

fun updateImage(placemark: PlacemarkModel) {
    if (placemark.image != "") {

        // get the full image file name
        val fileName = File(placemark.image)
        val imageName = fileName.getName()

        // create storage object to be uploaded to the cloudstore (node name is id of user)
        var imageRef = st.child(userId + '/' + imageName)

        // read the image into a bitmap object
        val bitmap = readImageFromPath(context, placemark.image)

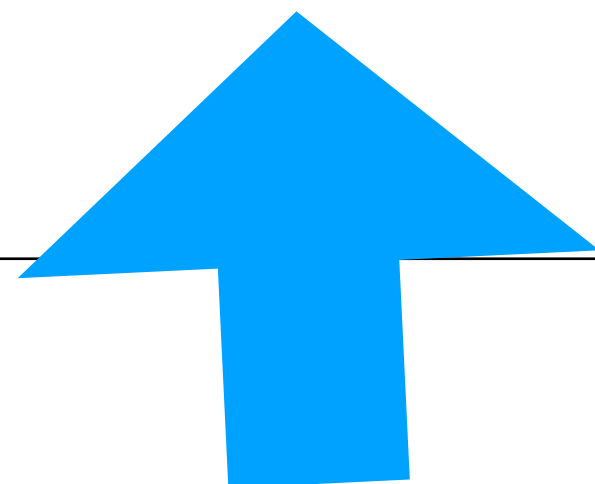
        bitmap?.let {
            // compress the image into a byte array
            val baos = ByteArrayOutputStream()
            bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)
            val data = baos.toByteArray()

            // put the bytes unto the object and start upload (asynchronous call)
            val uploadTask = imageRef.putBytes(data)
            uploadTask.addOnFailureListener {
                // failure
                println(it.message)
            }.addOnSuccessListener { taskSnapshot ->
                // success, get full path of uploaded image (asynchronous call)
                taskSnapshot.metadata!!.reference!!.downloadUrl.addOnSuccessListener {
                    // recover full path
                    placemark.image = it.toString()
                    // store full path in database
                    db.child("users").child(userId).child("placemarks").child(placemark.fbId).setValue(placemark)
                }
            }
        }
    }
}

```

# update()

```
override fun update(placemark: PlacemarkModel) {  
    var foundPlacemark: PlacemarkModel? = placemarks.find { p -> p.fbId == placemark.fbId }  
    if (foundPlacemark != null) {  
        foundPlacemark.title = placemark.title  
        foundPlacemark.description = placemark.description  
        foundPlacemark.image = placemark.image  
        foundPlacemark.location = placemark.location  
    }  
  
    db.child("users").child(userId).child("placemarks").child(placemark.fbId).setValue(placemark)  
    if ((placemark.image.length) > 0 && (placemark.image[0] != 'h')) {  
        updateImage(placemark)  
    }  
}
```



Upload Image whenever update requested