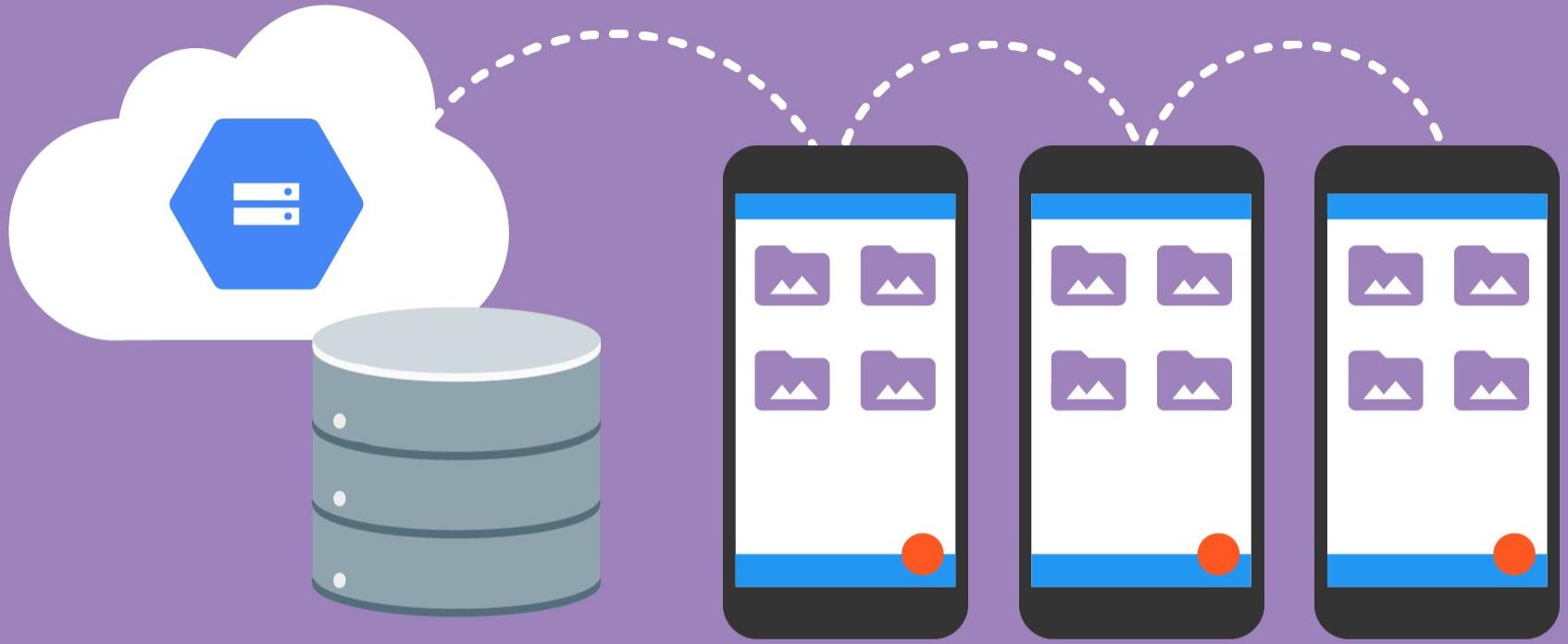
Mobile Application Development

Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE



Storage

Overview (Official)

- Cloud Storage for Firebase lets you upload and share user generated content, such as images and video, which allows you to build rich media content into your apps.
- Your data is stored in a Google Cloud Storage bucket, an exabyte scale object storage solution with high availability and global redundancy.
- Cloud Storage lets you securely upload these files directly from mobile devices and web browsers, handling spotty networks with ease.

Key Features (Official)

- Integrate storage into your apps with a single unified API
- Optimize price/performance across three storage classes with Object Lifecycle Management
- Access data instantly from any storage class
- Designed for secure and durable storage
- Reduce data storage carbon emissions to zero

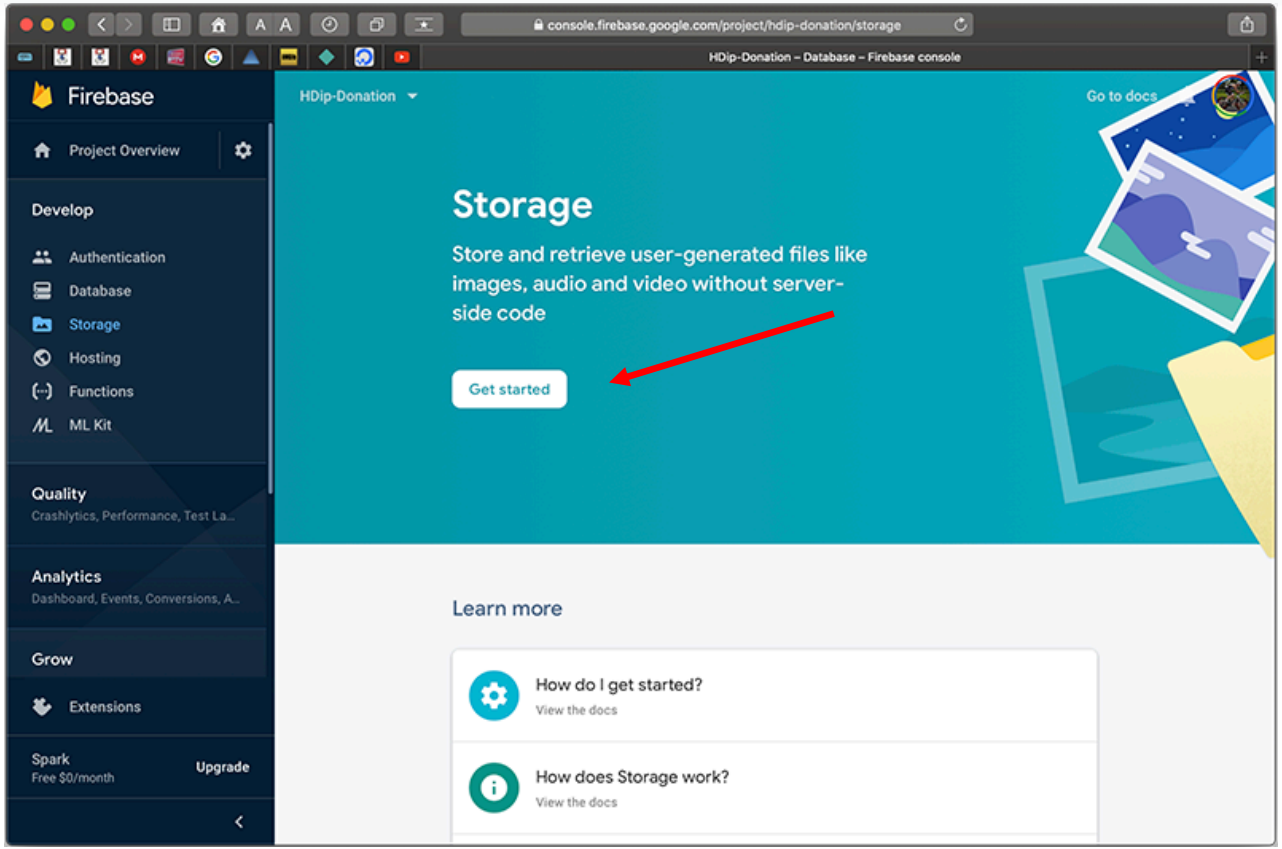
Get Started with Cloud Storage on Android

https://firebase.google.com/docs/storage/android/start

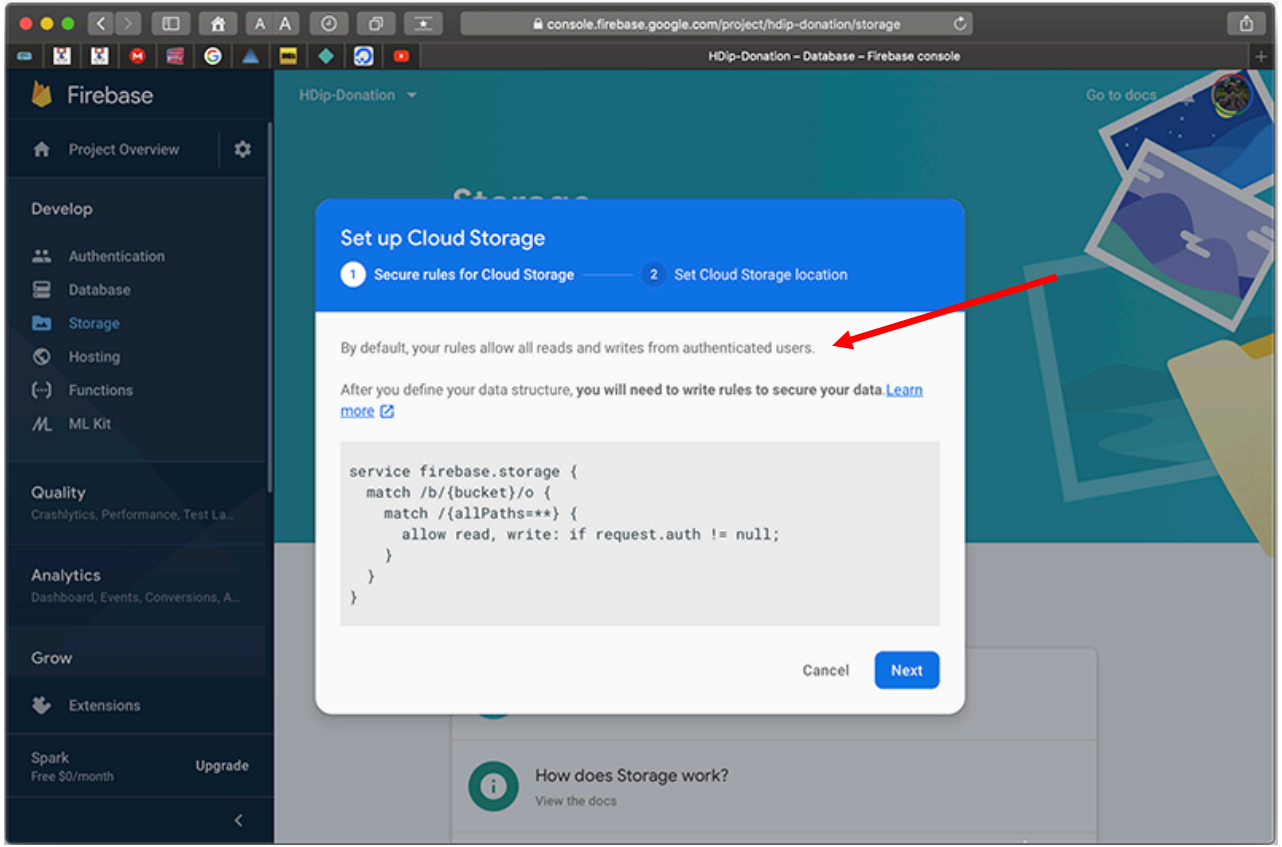
Firebase Cloud Storage

1. Create your Firebase Project (set up in previous labs)

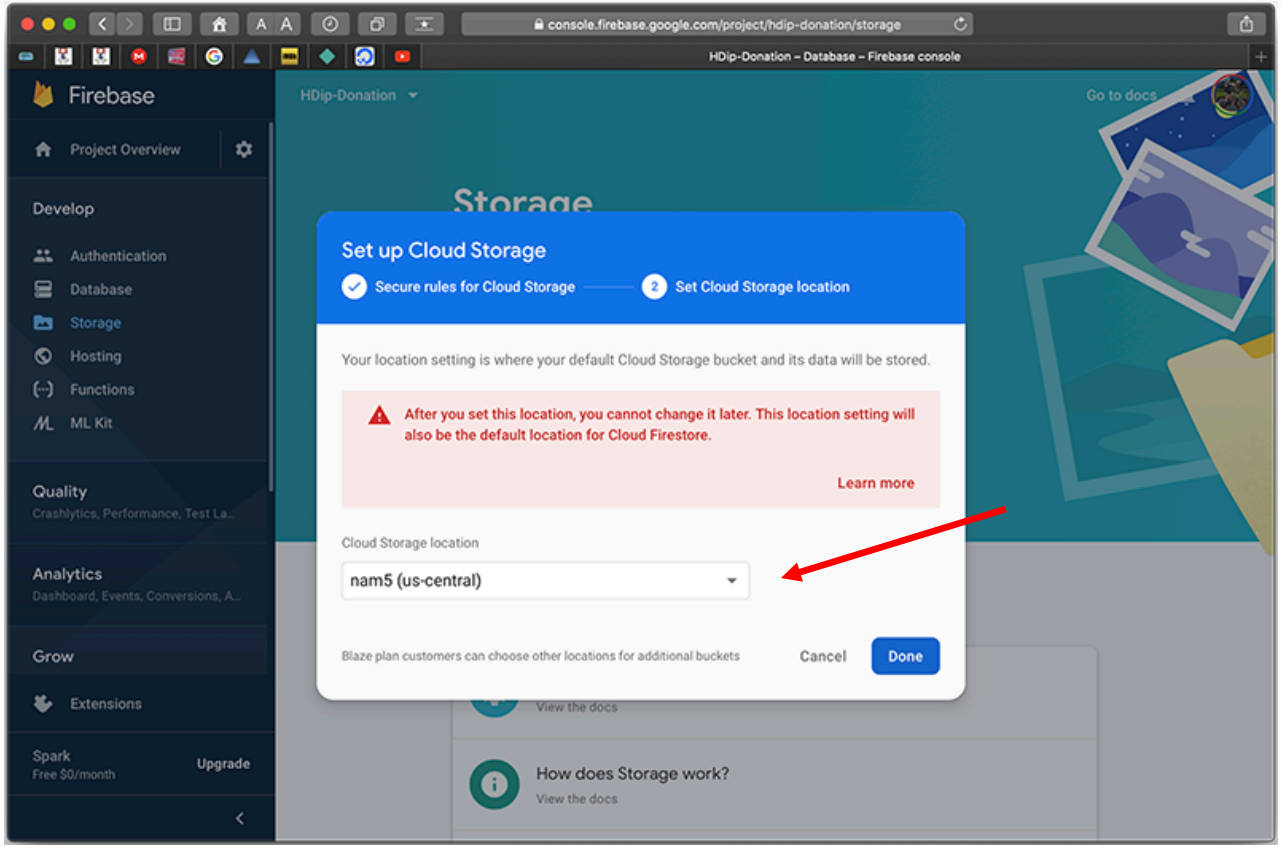
2. Setup your Storage Bucket



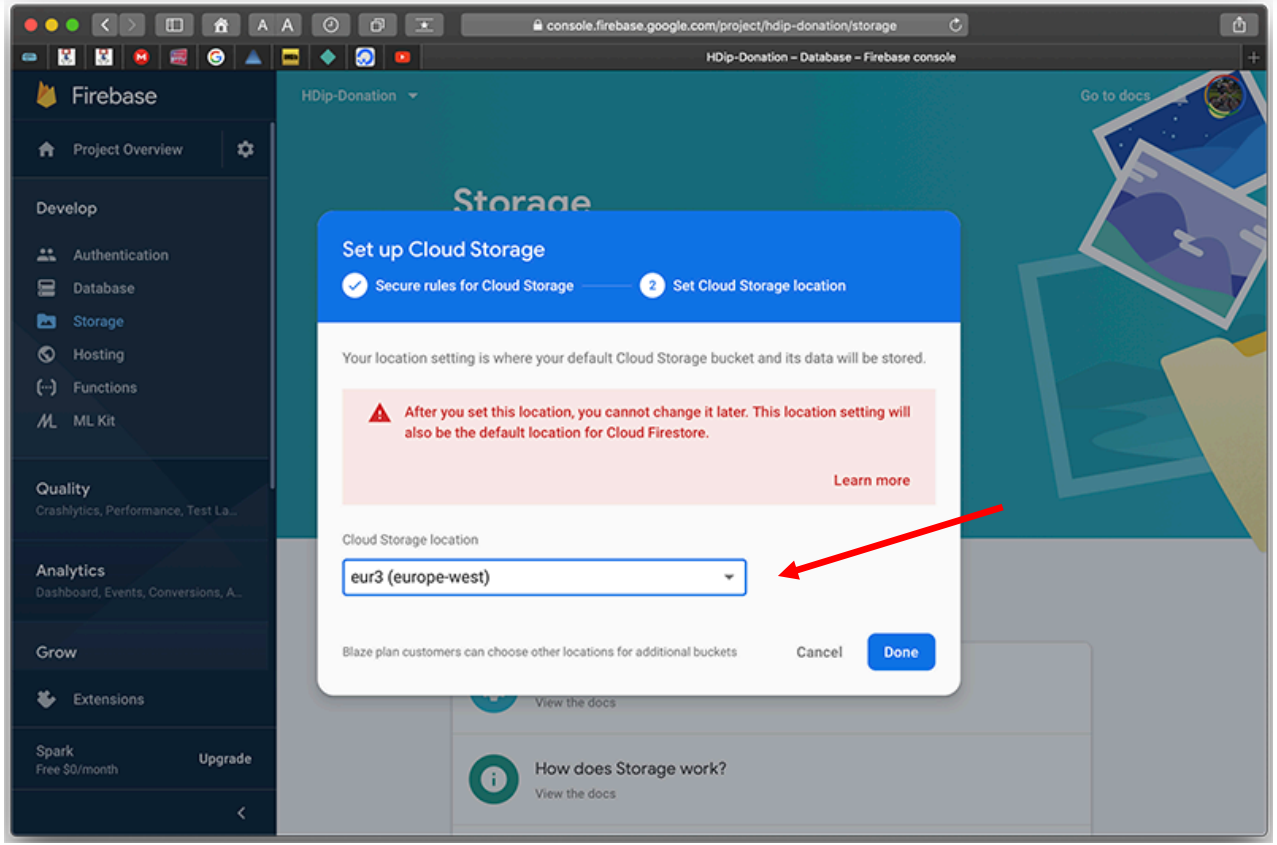
2. Setup your Storage Bucket



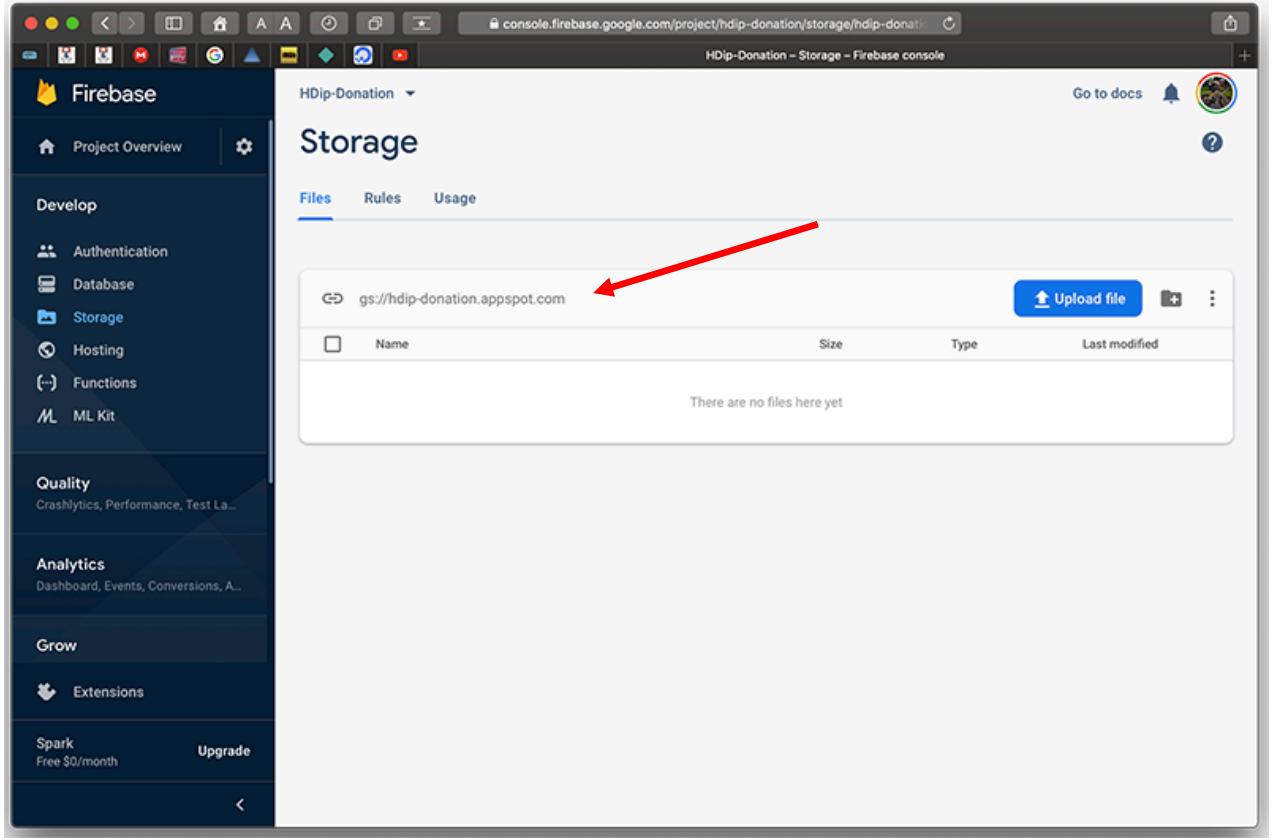
2. Setup your Storage Bucket



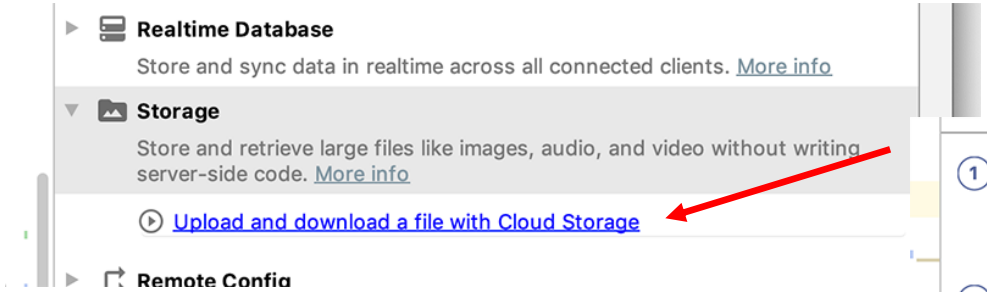
2. Setup your Storage Bucket

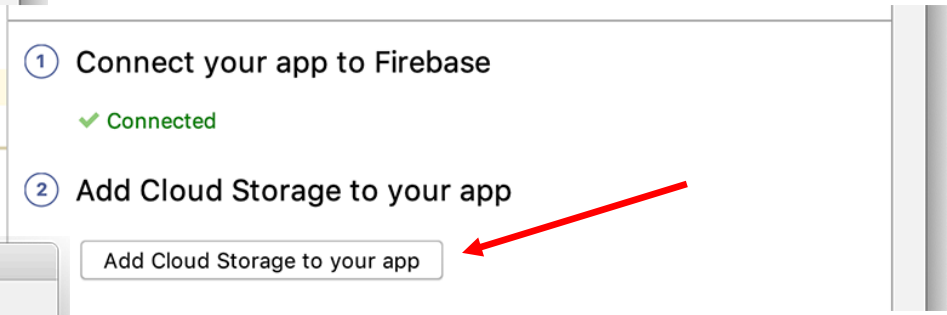


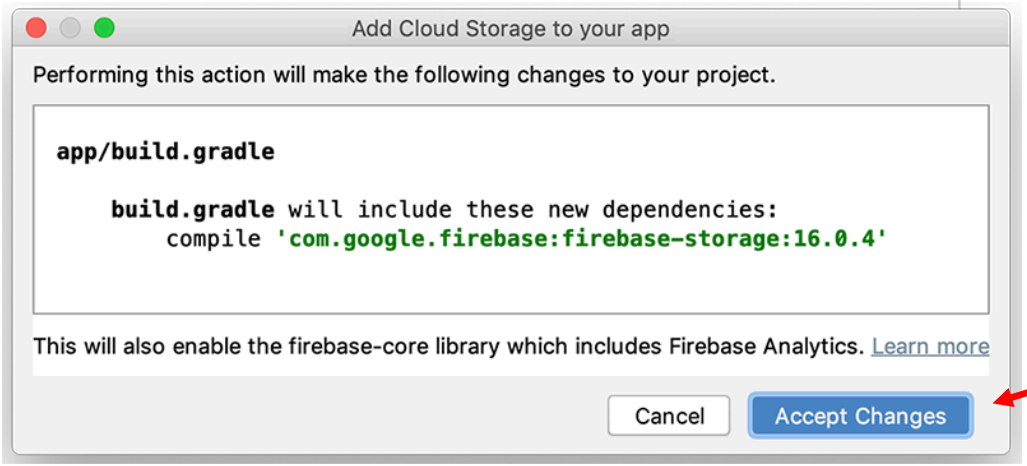
2. Setup your Storage Bucket



3. Configure Your App








```
implementation
'com.google.firebase:firebase-
storage:19.1.0'
```

3. Configure Your App – IMPORTANT!

```
implementation
'com.google.firebase:firebase-storage:19.1.0'
```

4. Start Uploading/Downloading your Files

https://github.com/firebase/quickstart-android

Implementation Path (Official)

1	Integrate the Firebase SDKs for Cloud Storage.	Quickly include clients via Gradle, CocoaPods, or a script include.
2	Create a Reference	Reference the path to a file, such as "images/mountains.png", to upload, download, or delete it.
3	Upload or Download	Upload or download to native types in memory or on disk.
4	Secure your Files	Use Firebase Security Rules for Cloud Storage to secure your files.

4. Get a Reference to your Bucket

The first step in accessing your storage bucket is to create an instance of `FirebaseStorage`:

Java Android | Kotlin Android

```
storage = FirebaseStorage.getInstance()
```

You're ready to start using Cloud Storage!

4. Get a Reference to your Bucket

If you want to use a storage bucket other than the default provided above, or use multiple storage buckets in a single app, you can create an instance of `FirebaseStorage` that references your custom bucket:

Java Android | Kotlin Android

```kotlin
// Get a non-default Storage bucket
val storage = FirebaseStorage.getInstance("gs://my-custom-bucket")
```

StorageActivity.kt

4. Uploading Files

To upload a file to Cloud Storage, you first create a reference to the full path of the file, including the file name.

```
// Create a storage reference from our app
val storageRef = storage.reference

// Create a reference to "mountains.jpg"
val mountainsRef = storageRef.child("mountains.jpg")

// Create a reference to 'images/mountains.jpg'
val mountainImagesRef = storageRef.child("images/mountains.jpg")

// While the file names are the same, the references point to different files
mountainsRef.name == mountainImagesRef.name // true
mountainsRef.path == mountainImagesRef.path // false
```

Once you've created an appropriate reference, you then call the `putBytes()`, `putFile()`, or `putStream()` method to upload the file to Cloud Storage.

4. Uploading Files (data in memory)

The `putBytes()` method is the simplest way to upload a file to Cloud Storage. `putBytes()` takes a `byte[]` and returns an `UploadTask` that you can use to manage and monitor the status of the upload.

```
// Get the data from an ImageView as bytes
imageView.isDrawingCacheEnabled = true
imageView.buildDrawingCache()
val bitmap = (imageView.drawable as BitmapDrawable).bitmap
val baos = ByteArrayOutputStream()
bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)
val data = baos.toByteArray()

var uploadTask = mountainsRef.putBytes(data)
uploadTask.addOnFailureListener {
    // Handle unsuccessful uploads
}.addOnSuccessListener {
    // taskSnapshot.metadata contains file metadata such as size, content-type, etc.
    // ...
}
```

Because `putBytes()` accepts a `byte[]`, it requires your app to hold the entire contents of a file in memory at once. Consider using `putStream()` or `putFile()` to use less memory.

4. Uploading Files (from a stream)

The `putStream()` method is the most versatile way to upload a file to Cloud Storage. `putStream()` takes an `InputStream` and returns an `UploadTask` that you can use to manage and monitor the status of the upload.

Java Android | Kotlin Android

```
val stream = FileInputStream(File("path/to/images/rivers.jpg"))

uploadTask = mountainsRef.putStream(stream)
uploadTask.addOnFailureListener {
    // Handle unsuccessful uploads
}.addOnSuccessListener {
    // taskSnapshot.metadata contains file metadata such as size, content-type, etc.
    // ...
}
```

4. Uploading Files (from a local file)

You can upload local files on the device, such as photos and videos from the camera, with the `putFile()` method. `putFile()` takes a `File` and returns an `UploadTask` which you can use to manage and monitor the status of the upload.

```
var file = Uri.fromFile(File("path/to/images/rivers.jpg"))
val riversRef = storageRef.child("images/${file.lastPathSegment}")
uploadTask = riversRef.putFile(file)

// Register observers to listen for when the download is done or if it fails
uploadTask.addOnFailureListener {
    // Handle unsuccessful uploads
}.addOnSuccessListener {
    // taskSnapshot.metadata contains file metadata such as size, content-type, etc.
    // ...
}
```

4. Uploading Files (get your download URL)

After uploading a file, you can get a URL to download the file by calling the `getDownloadUrl()` method on the `StorageReference`:

```
val ref = storageRef.child("images/mountains.jpg")
uploadTask = ref.putFile(file)

val urlTask = uploadTask.continueWithTask { task ->
    if (!task.isSuccessful) {
        task.exception?.let {
            throw it
        }
    }
    ref.downloadUrl
}.addOnCompleteListener { task ->
    if (task.isSuccessful) {
        val downloadUri = task.result
    } else {
        // Handle failures
        // ...
    }
}
```

4. Uploading Files (monitoring progress)

You can add listeners to handle success, failure, progress, or pauses in your upload task:

Listener Type	Typical Usage
`OnProgressListener`	This listener is called periodically as data is transferred and can be used to populate an upload/download indicator.
`OnPausedListener`	This listener is called any time the task is paused.
`OnSuccessListener`	This listener is called when the task has successfully completed.
`OnFailureListener`	This listener is called any time the upload has failed. This can happen due to network timeouts, authorization failures, or if you cancel the task.

`OnFailureListener` is called with an `Exception` instance. Other listeners are called with an `UploadTask.TaskSnapshot` object. This object is an immutable view of the task at the time the event occurred. An `UploadTask.TaskSnapshot` contains the following properties:

4. Downloading Files (get a reference)

You can create a reference by appending child paths to the storage root, or you can create a reference from an existing `gs://` or `https://` URL referencing an object in Cloud Storage.

```
// Create a storage reference from our app
val storageRef = storage.reference

// Create a reference with an initial file path and name
val pathReference = storageRef.child("images/stars.jpg")

// Create a reference to a file from a Google Cloud Storage URI
val gsReference = storage.getReferenceFromUrl("gs://bucket/images/stars.jpg")

// Create a reference from an HTTPS URL
// Note that in the URL, characters are URL escaped!
val httpsReference = storage.getReferenceFromUrl(
        "https://firebasestorage.googleapis.com/b/bucket/o/images%20stars.jpg")
```

4. Downloading Files (into memory)

Download the file to a `byte[]` with the `getBytes()` method. This is the easiest way to download a file, but it must load the entire contents of your file into memory. If you request a file larger than your app's available memory, your app will crash. To protect against memory issues, `getBytes()` takes a maximum amount of bytes to download. Set the maximum size to something you know your app can handle, or use another download method.

```
var islandRef = storageRef.child("images/island.jpg")

val ONE_MEGABYTE: Long = 1024 * 1024
islandRef.getBytes(ONE_MEGABYTE).addOnSuccessListener {
    // Data for "images/island.jpg" is returned, use this as needed
}.addOnFailureListener {
    // Handle any errors
}
```

4. Downloading Files (to a local file)

The `getFile()` method downloads a file directly to a local device. Use this if your users want to have access to the file while offline or to share the file in a different app. `getFile()` returns a `DownloadTask` which you can use to manage your download and monitor the status of the download.

```
islandRef = storageRef.child("images/island.jpg")

val localFile = File.createTempFile("images", "jpg")

islandRef.getFile(localFile).addOnSuccessListener {
    // Local temp file has been created
}.addOnFailureListener {
    // Handle any errors
}
```

4. Downloading Files (via URL)

If you already have download infrastructure based around URLs, or just want a URL to share, you can get the download URL for a file by calling the `getDownloadUrl()` method on a storage reference.

```
storageRef.child("users/me/profile.png").downloadUrl.addOnSuccessListener {
    // Got the download URL for 'users/me/profile.png'
}.addOnFailureListener {
    // Handle any errors
}
```

We'll look at how we work with files (Images) saved to Cloud Storage in the Code walkthrough and the Lab

References

https://console.firebase.google.com/
https://firebase.google.com/
https://github.com/firebase/quickstart-android
https://firebase.google.com/docs/storage/android/start
https://firebase.google.com/docs/storage/android/upload-files


Thanks.