Internet of Things prototyping with Firebase

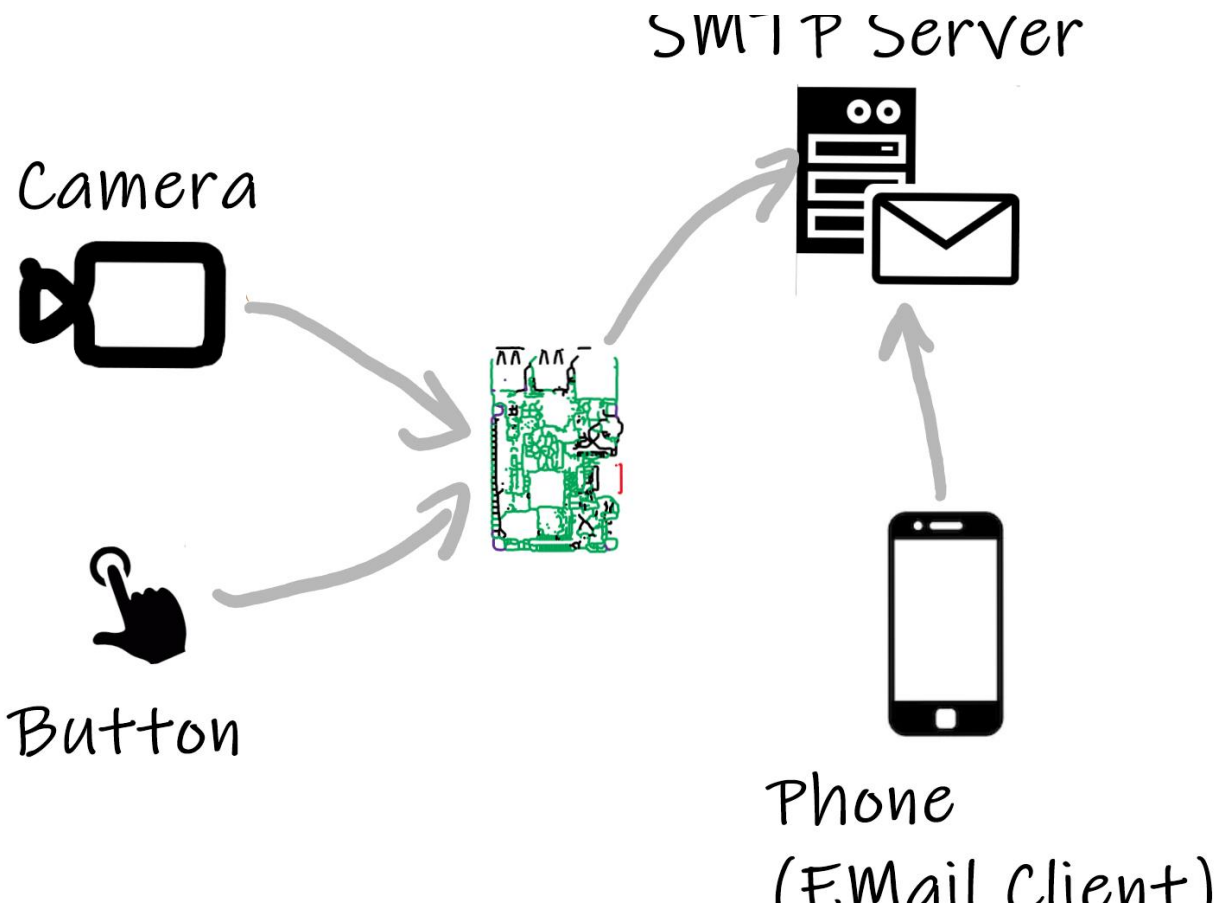
Tutorial

Firebase

- **Firebase** is a Backend-as-a-Service (BaaS)
- Generally used for creating mobile and web applications
- Also handy for IoT
  - Provides persistence, file storage, APIs and SDKs for lots of languages
- One stop shop for all the services we need

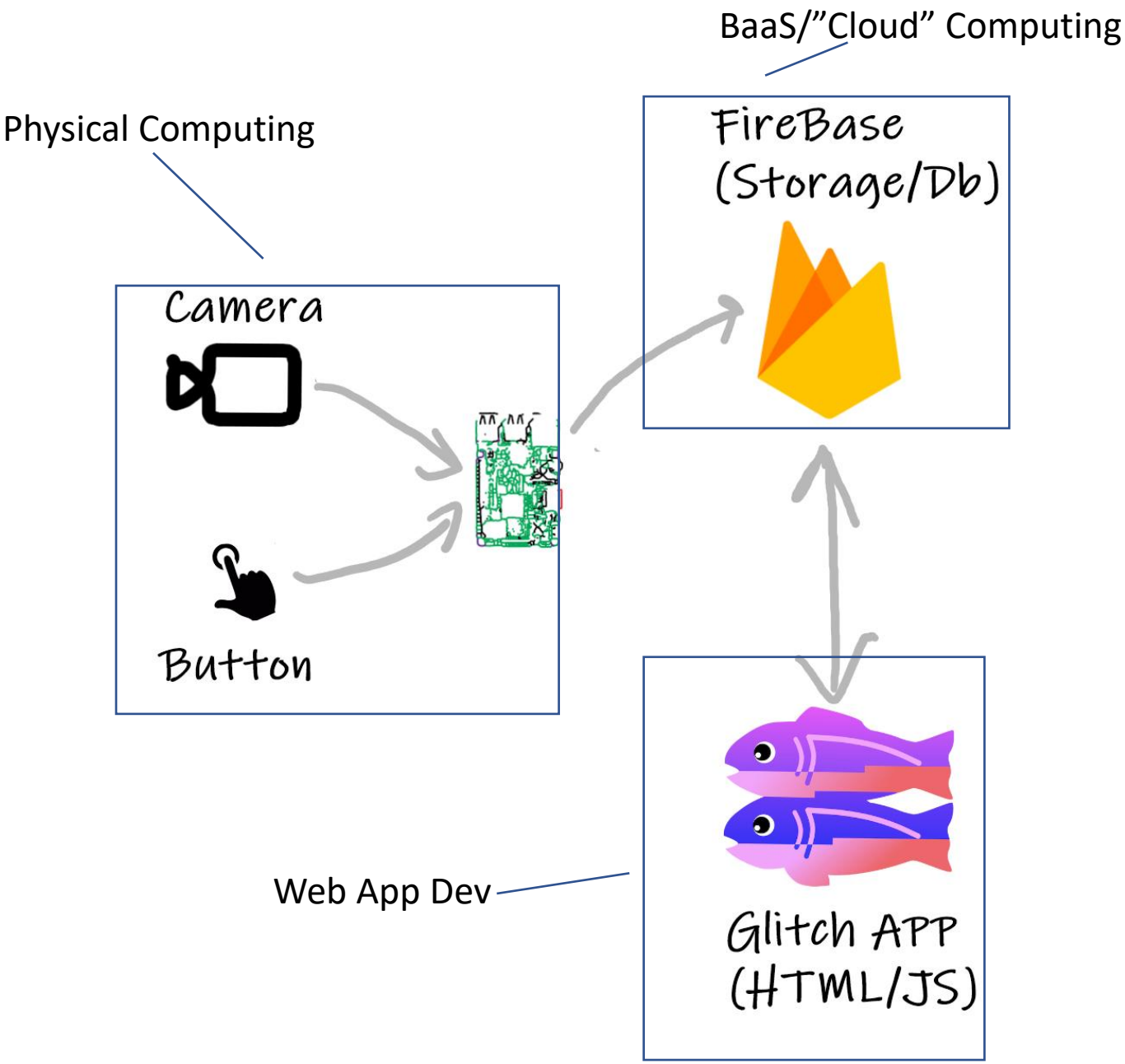
Smart Doorbell – Existing Solution

- Email image of person when press doorbell(button)
- Pros:
  - Everybody has/use email
  - Email inbox acts a record of doorbell event (can retrieve after event/later)
- Cons:
  - Images stored on RPi (not very accessible to other apps/limited space)
  - Email not really “dedicated” client(message could be lost/buried/junk)
  - Rpi app “tightly coupled” to email server. Any change would require code change on RPi
  - Only can be used via email (not very extensible)



Smart Doorbell: New Solution

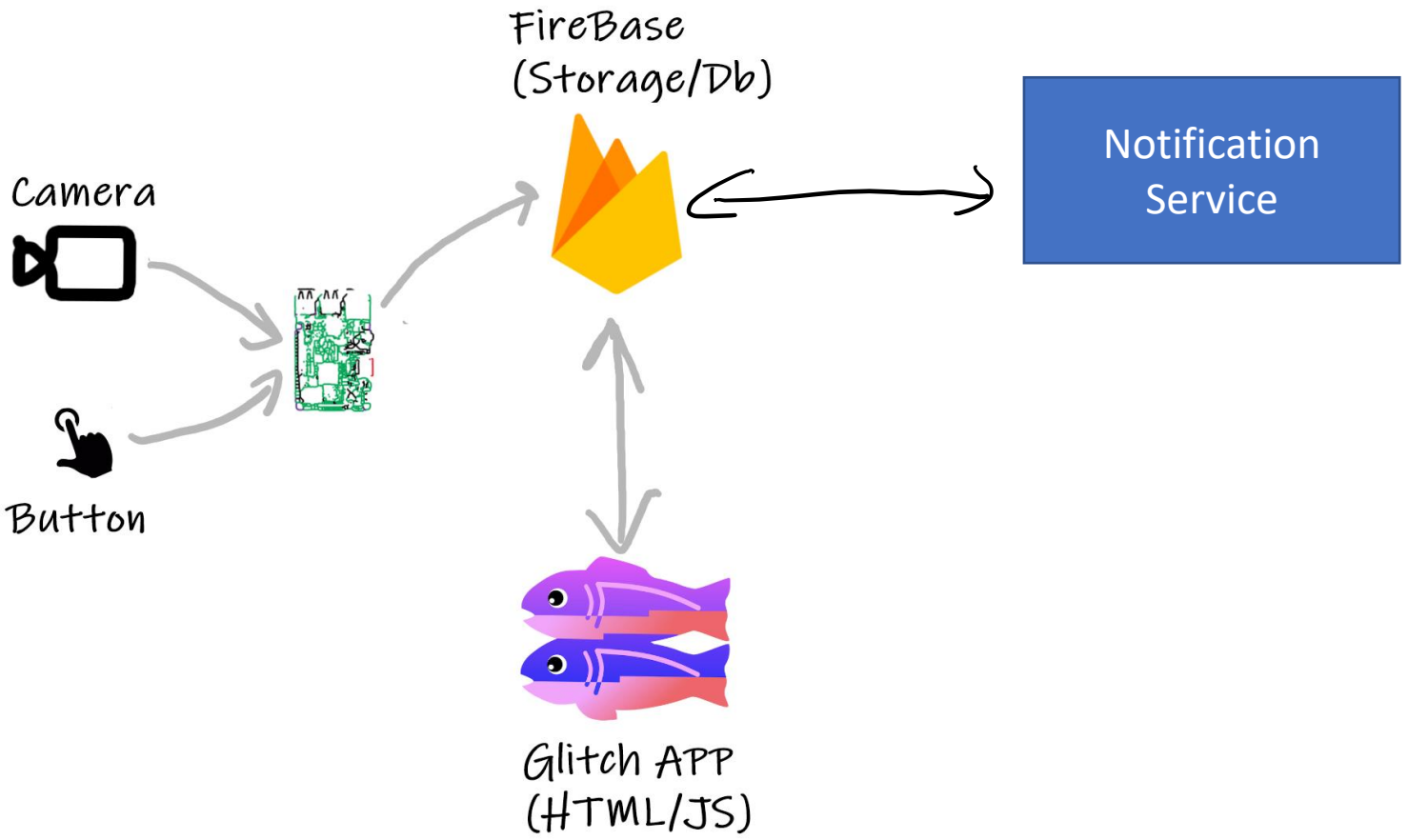
- Use “Event-based” architecture
- Take a photo when the button is pressed
  - Physical Computing
- Store image files in accessible file store
  - Firebase Storage
- Publish “image” events when they happen
  - Firebase Realtime DB
- View in connected app
  - Glitch Web app



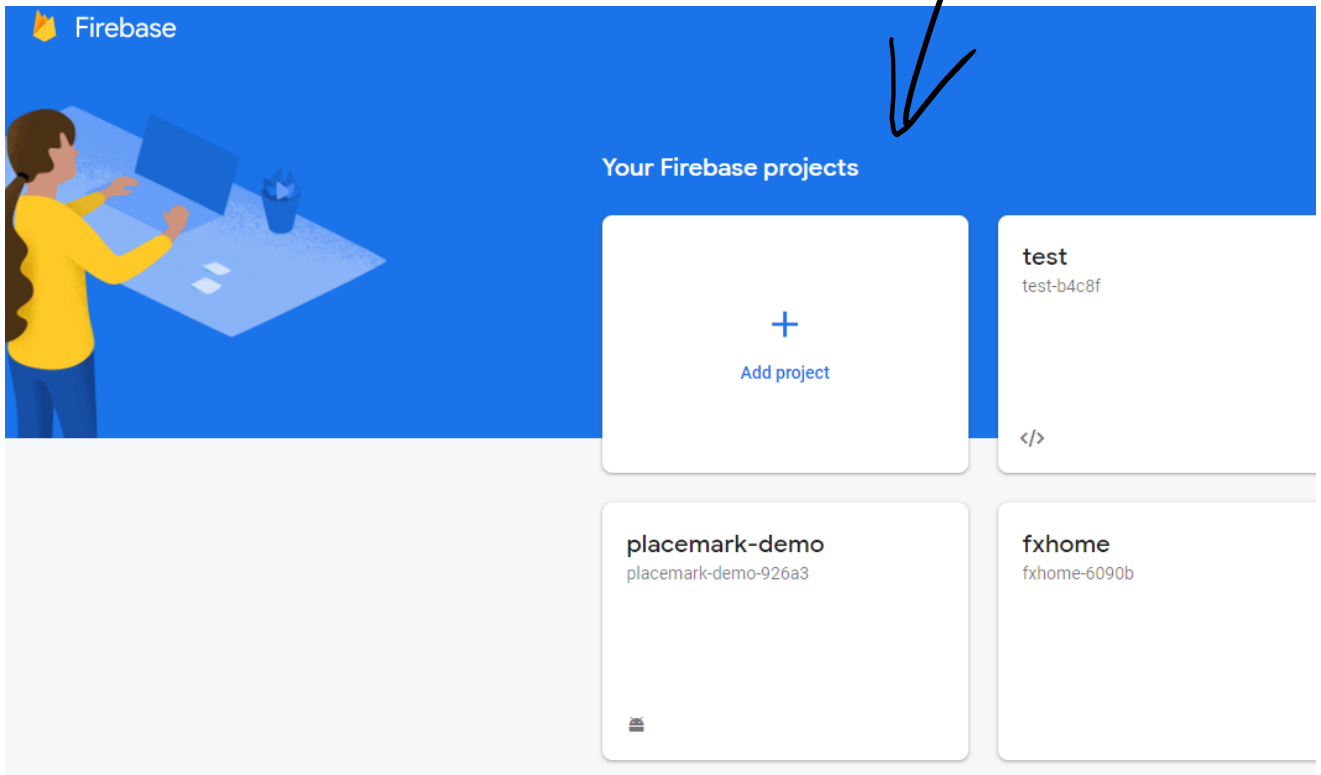
Event-Based Architecture

- Nothing new, but really useful for highly scalable and adaptable applications
  - Scalable – ability to scale up/increase number of users/devices in a solution.
  - Adaptable – ability to change/alter/update a solution, for both small and complex applications
- Consider the following for the smart doorbell application:
  - “I want to increase the number of users/clients from 1 to 10”
    - Email solution: Send email to 10 different emails, need to get 10 email addresses and update program on RPi. 10 emails with attachments stored in different SMTP servers- duplication. NOT VERY SCALABLE.
    - Firebase solution: Give the users access to the Glitch Web App. No change to the RPi code. Just increase in traffic to Firebase (no big deal).
  - “I want to try more functionality, for example send a SMS/Push Notification to the house owner when the button is pressed”
    - Email solution: Update code on RPi to do this as well as the email (SMS/Push Notification/email). Rpi is doing a lot now (not “cohesive”). NOT VERY ADAPTABLE. Best just to get your apps/services to “do one thing and do it well”
    - Firebase solution: Develop a separate SMS service that subscribes/receives for “door event” and sends sms when it occurs. No change to existing code at all! (aside: could use Firebase cloud function for this…).

Event Based Architecture



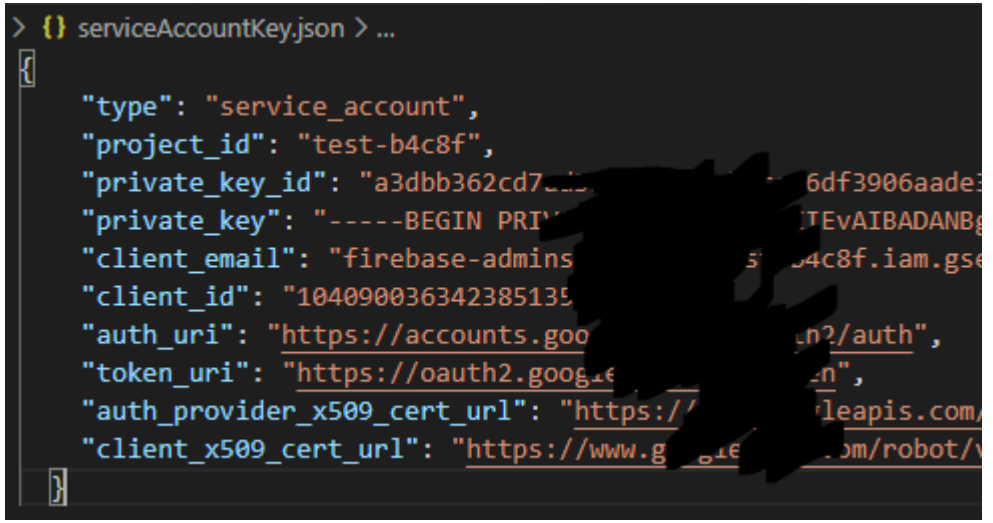
Getting Firebase



- It’s free. You just need a Google ID.
- Go to the **Firebase Console** to start creating “projects” and access Firebase Services
- Create the services you require

Using Firebase

- Firebase software development kit (SDK) allows you to access Firebase programmatically
- SDK available for most mainstream programming languages/Environments
  - Python
  - Javascript
  - Android/iOS
- Must have authentication details to access services
  - For example, serviceAccountKey.json
  - Need to “Register an app” with Firebase first.
  - **Download this from Firebase for your programming language.**



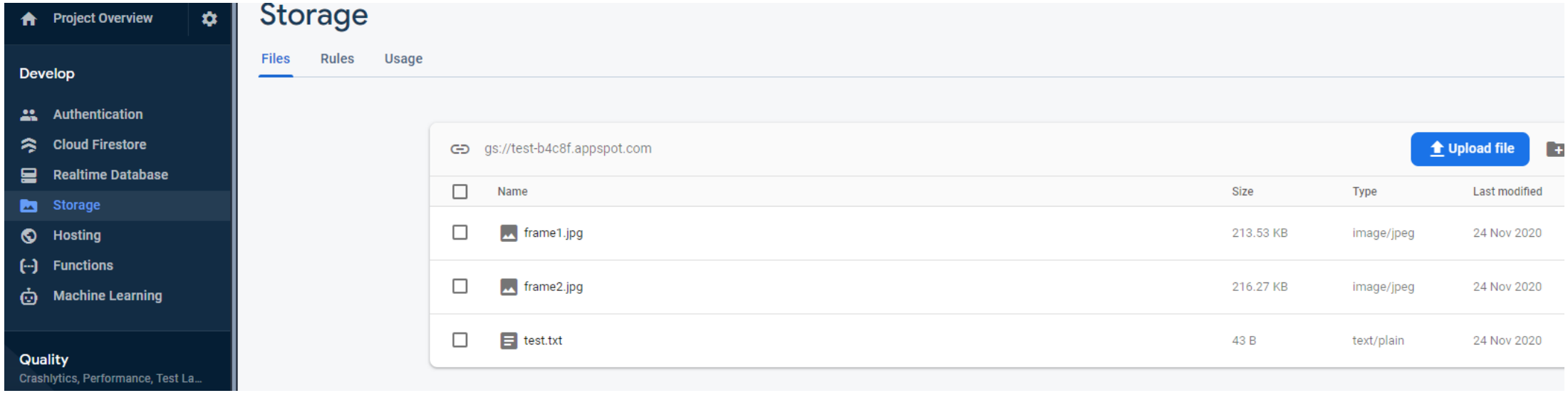

```
cred=credentials.Certificate('./serviceAccountKey.json')
firebase_admin.initialize_app(cred, {
    'storageBucket': 'test-b4c8f.appspot.com',
    'databaseURL':'https://test-b4c8f.firebaseio.com/'
})
```

Firebase Storage

- Use it to store and manage media generated by devices and apps.
- This use case:
  - upload images from RPi
  - Have accessible using public URL
- Stores files in *bucket*
- Programming steps:
  1. Authenticate with Firebase
  2. Create reference to a Storage Bucket
  3. Create a “Blob” in the Bucket (blob is an array of byte
  4. Upload File using the blob

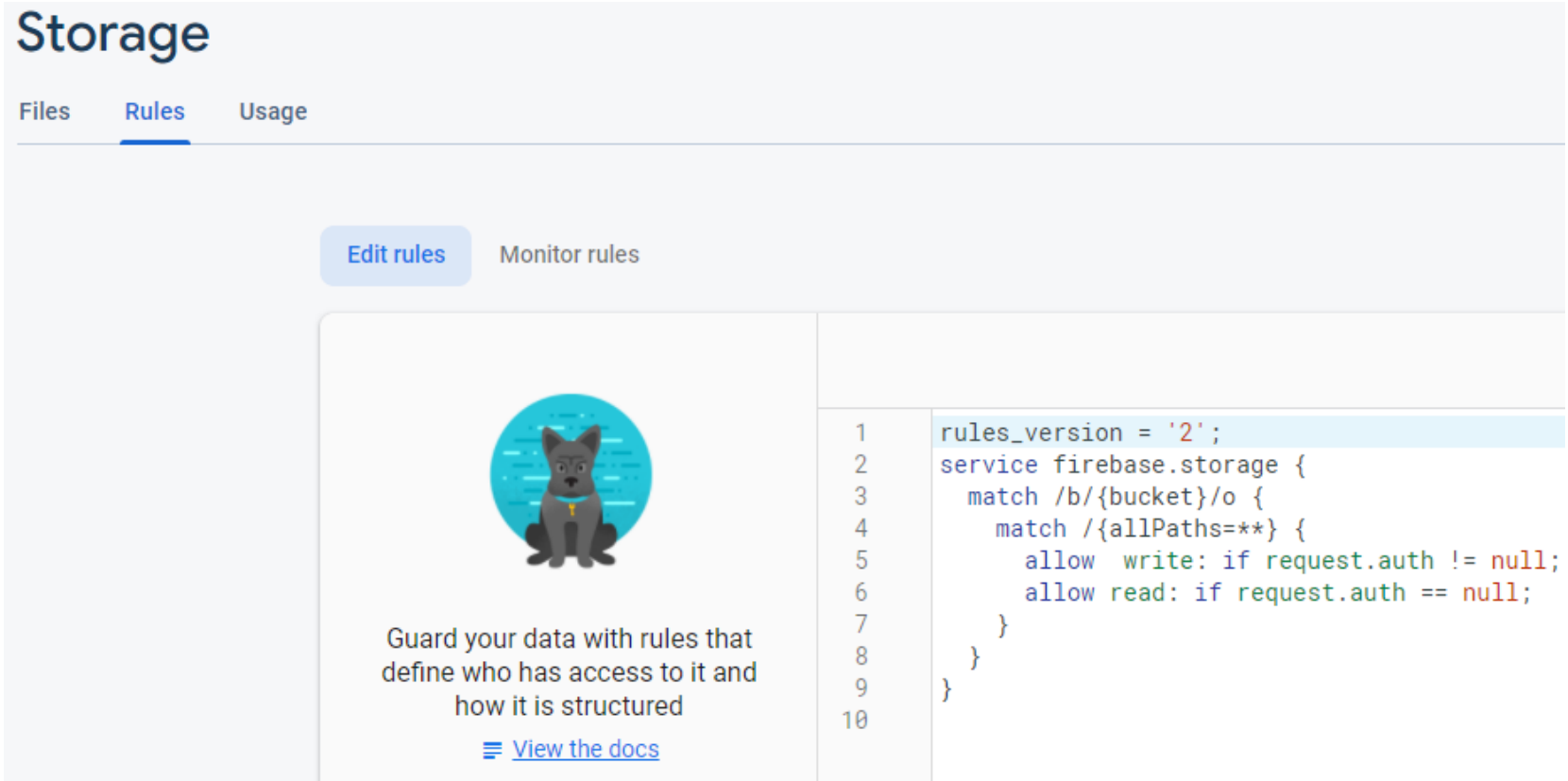
```
from firebase_admin import storage

bucket = storage.bucket()


    filename=os.path.basename(fileLoc)

    # Store File in Fb Bucket
    blob = bucket.blob(filename)
    blob.upload_from_filename(fileLoc)
```

Firebase Storage: Files



Firebase Storage: Rules

- By default, authentication required to read and write
- Can change in *Rules* to allow reads/writes

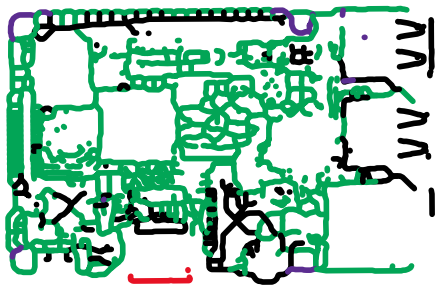


Firebase Realtime DB

- Cloud-based database that uses live connections to the app.
  - The data is stored a json objects and is only intended for text, to allow for fast responses.
- Useful for constantly changing data
  - differs from traditional databases containing persistent data, mostly unaffected by time

Firebase Realtime DB: Pushing Data

- The data is stored as a json object.
  - Each new entry assigned a unique key.
  - Push or Set data from client
- Programming steps:
  1. Authenticate with Firebase
  2. Create reference to “node” in DB
  3. Push JSON data on to DB



Push

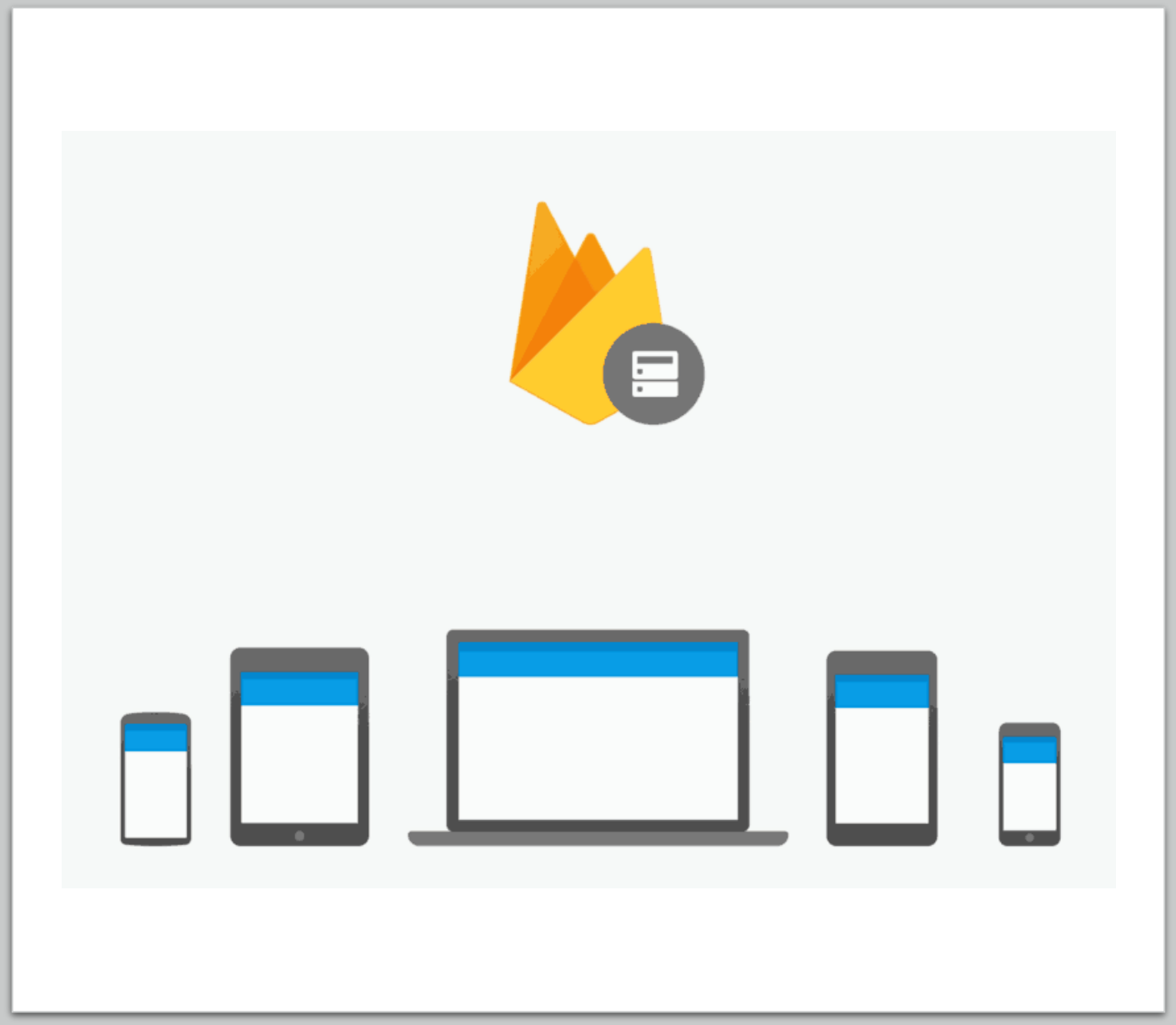
https://test-b4c8f.firebaseio.com/

```
test-b4c8f
  file
    -MMuNqjaCkf5TpjsTtkw
      image: "test.txt"
      timestamp: "12/11/2020 9:00"
    -MMuPlJShYcvV2w2zLvl
      image: "frame1.jpg"
      timestamp: "24/11/2020 13:19:58"
    -MMuSrlKzAmOdOdQkskU
      image: "frame2.jpg"
      timestamp: "24/11/2020 13:35:33"
```

```python
ref = db.reference('/')
home_ref = ref.child('file')

def push_db(fileLoc, time):

    filename=os.path.basename(fileLoc)

    # Push file reference to image in Realtime DB
    home_ref.push({
        'image': filename,
        'timestamp': time}
    )
```

Application: Connecting to DB

- Can create connection to Realtime DB from many other services
  - Uses HTTP Web Sockets
- Every change on the DB is passed to all connected processes/devices
- Use this to notify web app that new image uploaded
  - Or any other app in the future, for example if we develop an Android app tomorrow!

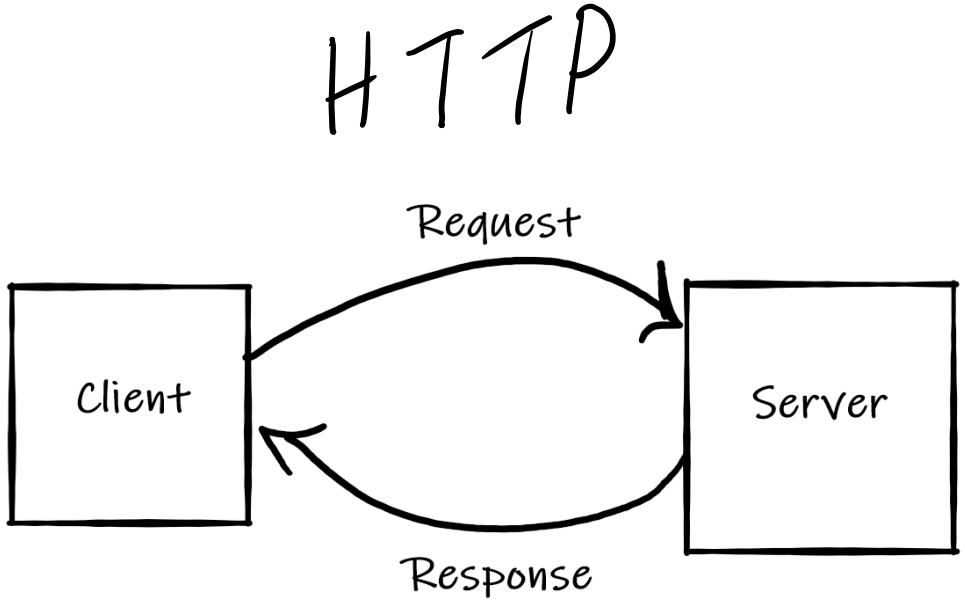


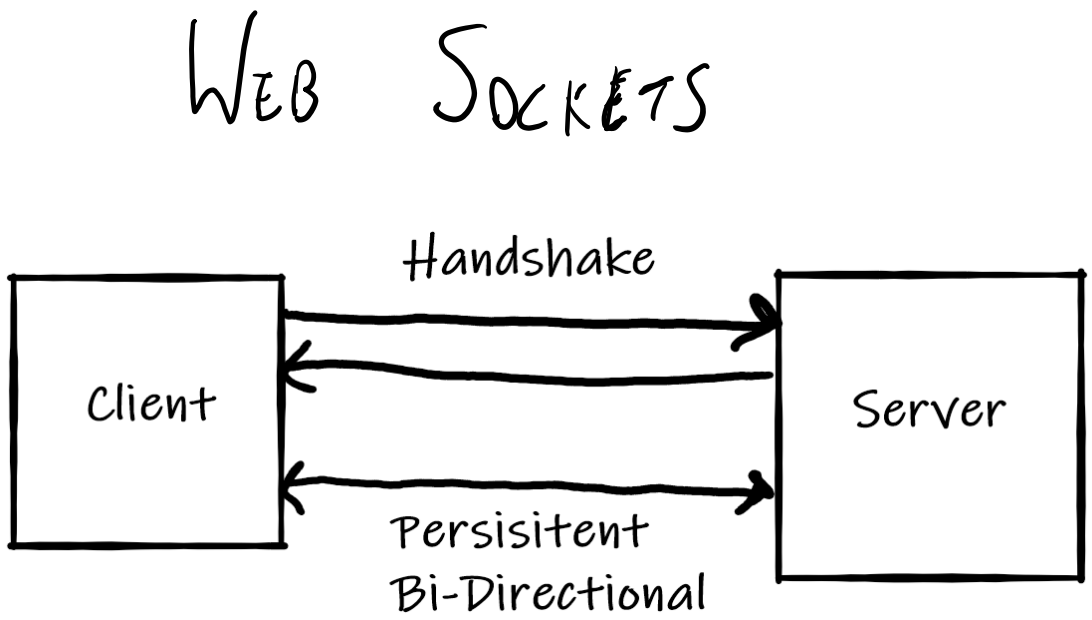
```
firebase.initializeApp(firebaseConfig);

// Get a reference to the file storage service
const storage = firebase.storage();
// Get a reference to the database service
const database = firebase.database();

// Create camera database reference
const camRef = database.ref("file");

// Sync on any updates to the DB. THIS CODE RUNS EVERY TIME AN UPDATE OCCURS ON THE DB.
camRef.limitToLast(1).on("value", function(snapshot) {
  snapshot.forEach(function(childSnapshot) {
    const image = childSnapshot.val()["image"];
    const time = childSnapshot.val()["timestamp"];
    const storageRef = storage.ref(image);

    storageRef
      .getDownloadURL()
      .then(function(url) {
        console.log(url);
        document.getElementById("photo").src = url;
        document.getElementById("time").innerText = time;
      })
      .catch(function(error) {
        console.log(error);
      });
  });
});
```

```
<body>
  <h1>Raspberry Pi Doorbell Cam</h1>

  <p>
    Photo taken at your front door at <span id="time"></span>: <br>
    <img id="photo" />
  </p>


</body>
```

Realtime DB Connection: WebSockets

- Allows real-time interactive communication between the client browser and a server
- Application layer protocol that allows bidirectional data flow.





HTTP vs WebSockets

- **HTTP** is half-duplex, a one-way communication protocol
  - Client Polls only, then server send
- Complex, Inefficient, Wasteful
- It is designed for file transfer, or server any other static resources

- **WebSockets** is TCP based, bi-directional, full-duplex messaging
- Establish connection (Single TCP connection)
  - Send messages in both direction (Bi-directional)
  - Send message independent of each other (Full Duplex)
- End connection

HTTP Limitations

- Many web apps require near realtime response
  - NetFlix, Youtube,...
  - Online stock trading, sensor/device monitoring
- HTTP is not designed for those applications
  - large overhead
  - one-way communication, not efficient
- WebSockets has lower overhead and more suited for these types of apps.

Websocket vs HTTP Example: Header overhead

Assume HTTP header is 871 bytes (some are 2000bytes!)

After initial handshake, WebSockets only use 2 bytes in each message

HTTP

- 1,000 clients polling every second: Network traffic is

(871 x 1,000) = 871,000 bytes = 6,968,000 bits per second (6.6 Mbps)

WebSockets

- 1,000 clients receive 1 message per second: Network traffic is

(2 x 1,000) = 2,000 bytes = 16,000 bits per second (0.015 Mbps)

WebSockets vs MQTT (why not use MQTT)

- Actually you can do MQTT over Websockets if you want.
- Web browsers don’t have MQTT protocol (HTTP and WebSockets)
- Broker has to support Web Sockets