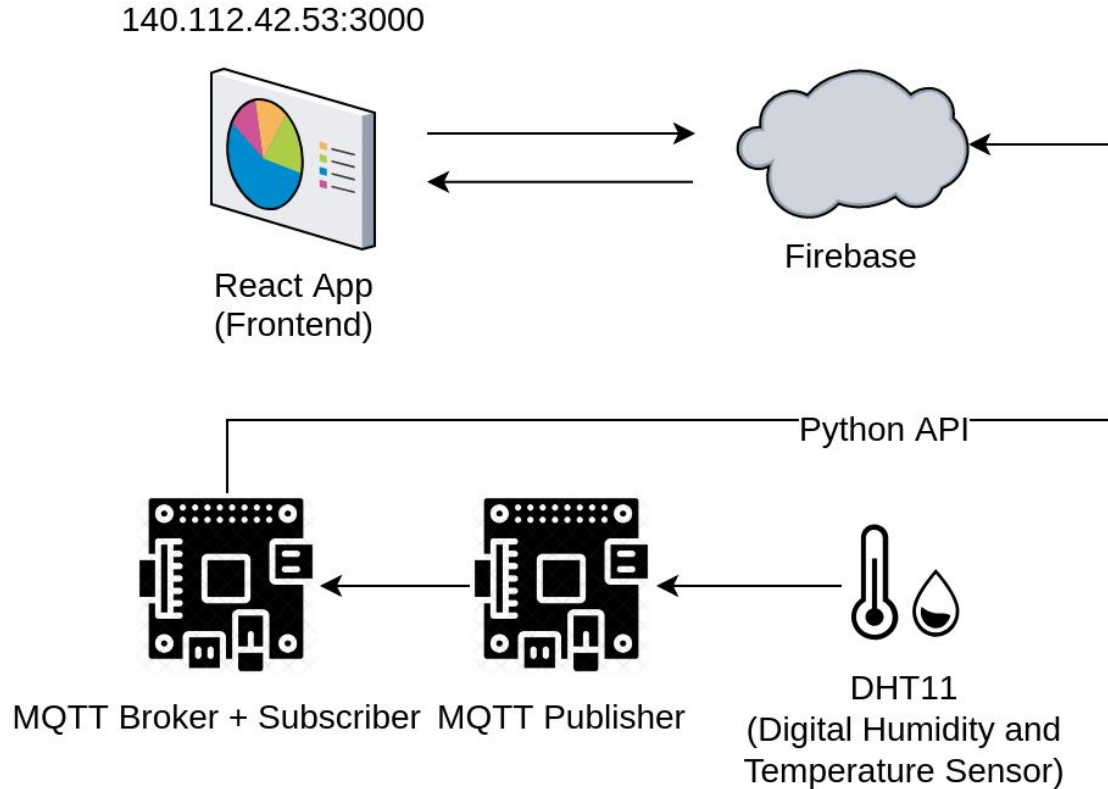


Vision

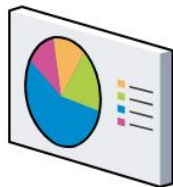


An easy-to-use, configurable visualization dashboard

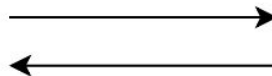
Data Visualization with Firebase



140.112.42.53:3000

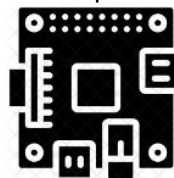


React App
(Frontend)

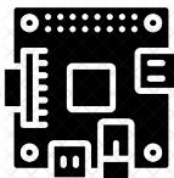


Firebase

Python API



MQTT Broker + Subscriber



MQTT Publisher



DHT11
(Digital Humidity and
Temperature Sensor)

Publisher

```
import paho.mqtt.client as mqtt
client = mqtt.Client()
client.connect('192.168.1.38') # broker ip
client.publish('MyHome/Bedroom/AirConditioning/Temperature', str(temperature))
client.publish('MyHome/Bedroom/AirConditioning/Humidity', str(humidity))
```

Subscriber

```
def on_connect(client, userdata, flags, rc):
    print("Connected")
    client.subscribe("MyHome/Bedroom/AirConditioning/#")

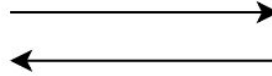
def on_message(client, userdata, message):
    #do something here

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect("localhost")
```

140.112.42.53:3000

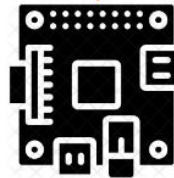


React App
(Frontend)

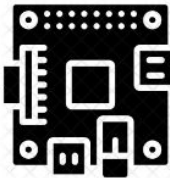


Firebase

Python API



MQTT Broker + Subscriber

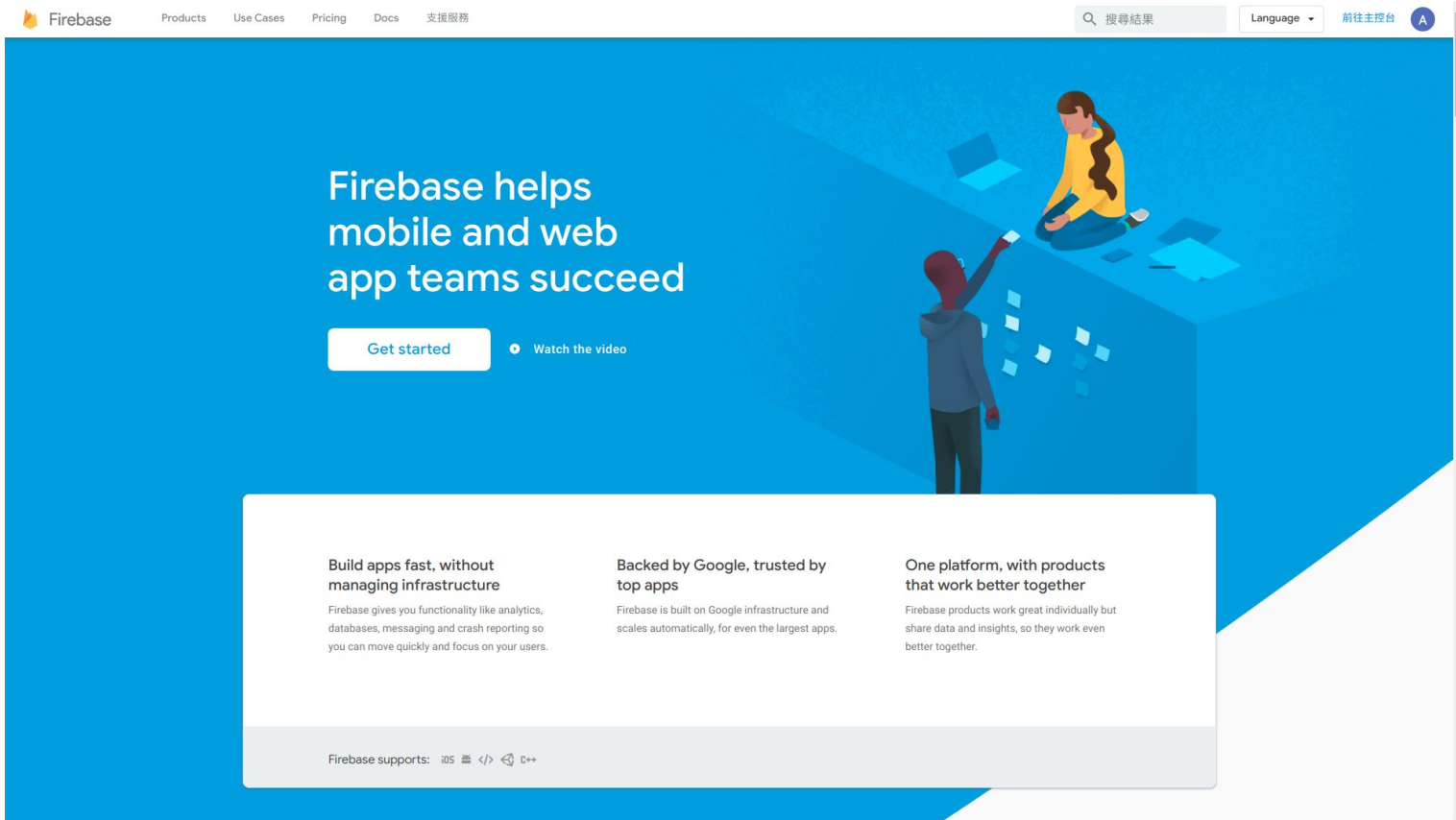


MQTT Publisher



DHT11
(Digital Humidity and
Temperature Sensor)

Firebase



Firebase

The screenshot displays the Firebase console interface. On the left is a dark sidebar menu with the following sections:

- 開發 (Development):** Authentication, Database, Storage, Hosting, Functions, ML Kit.
- 品質 (Quality):** Crashlytics, Performance, Test Lab.
- 數據分析 (Data Analysis):** Dashboard, Events, Conversions, A...
- 拓展 (Extensions):** Predictions, A/B Testing, Cloud Messaging, In-App Messaging.
- Spark:** 免費 每月 \$0 美元, 升級

The main content area has a blue header with the text "vision" and a "Spark 方案" button. Below this, the main heading reads "將 Firebase 新增至應用程式即可開始使用" (Get started by adding Firebase to your app). A subtext explains: "有了我們的 Core SDK，您就能使用大部分的 Firebase 功能，並可透過數據分析查看 iOS 和 Android 應用程式的相關分析資料" (With our Core SDK, you can use most of the Firebase features, and you can view related analytics for your iOS and Android apps through data analysis). Below the text are icons for iOS, Android, and code symbols, followed by the text "首先請新增應用程式" (First, please add the app).

At the bottom, a light gray banner contains the text "在數毫秒內即可儲存及同步處理應用程式資料" (Store and sync app data in milliseconds) and a close button (X). Below the banner are two images: one showing purple lanyards with ID badges, and another showing server racks with a magnifying glass over a data point.

Firebase

The screenshot shows the Firebase console interface. On the left is a dark sidebar with the 'Project Overview' tab selected. Below it are categories: '開發' (Development) with links to Authentication, Database, Storage, Hosting, Functions, and ML Kit; '品質' (Quality) with links to Crashlytics, Performance, and Test Lab; '數據分析' (Data Analysis) with links to Dashboard, Events, Conversions, and Attribution; and '拓展' (Extensions) with links to Predictions, A/B Testing, Cloud Messaging, and In-App Messaging. At the bottom of the sidebar is the 'Spark' section, indicating '免費 每月 \$0 美元' and a '升級' (Upgrade) button.

The main content area has a blue header with the title '設定' (Settings) and a navigation bar with tabs: '一般' (General), 'Cloud Messaging', '整合' (Integrations), '服務帳戶' (Service Accounts), '資料隱私權' (Data Privacy), and '使用者和權限' (Users and Permissions). The '整合' tab is currently selected.

The main area displays a grid of integration cards:

- AdMob**: 放送數百萬 Google 廣告客戶的廣告 (Broadcast millions of Google Advertisers' ads). Must be an AdMob administrator. Link: 連結 → (Connect).
- Google Ads**: 提高安裝次數、取得深入分析資料，並針對特定對象放送廣告活動 (Increase installs, get deep analysis data, and target specific audiences for ad campaigns). Must be a Google Ads administrator. Link: 連結 → (Connect).
- BigQuery** (Sandbox): 無須使用信用卡，即可試用強大的查詢功能 (No need for a credit card to try powerful query capabilities). Link: 連結 → (Connect).
- Display & Video 360**: 瞭解多媒體廣告和搜尋行銷的影響 (Understand the impact of multimedia advertising and search marketing). Link: 瞭解詳情 → (Learn more).
- Google Play**: 追蹤應用程式內購、ANR 事件及當機資料 (Track in-app purchases, ANR events, and crash data). Must be a Google Play owner. Link: 安裝 → (Install).
- Slack**: 針對 Firebase 偵測到的問題，將相關重要快訊傳送給團隊成員 (Send relevant important alerts to team members for issues detected by Firebase). Link: 安裝 → (Install).
- Jira**: 自動在 Jira 中建立問題 (Automatically create issues in Jira). Link: 安裝 → (Install).
- PagerDuty**: 針對 Firebase 偵測到的問題，將相關重要快訊傳送給團隊成員 (Send relevant important alerts to team members for issues detected by Firebase). Link: 安裝 → (Install).

Firebase - Database

The screenshot displays the Firebase Database console interface. On the left is a dark sidebar with the 'Database' option selected. The main content area has a blue header with 'Database' and 'Cloud Firestore' tabs. Below the header, a breadcrumb path shows 'test_data' > 'serial_humidity'. The interface is divided into three columns representing the database hierarchy:

- Column 1 (Project):** Shows the project 'vision-ee232' and a collection 'test_data' which contains sub-collections 'test-ianchen', 'test_data', and 'todos'.
- Column 2 (Collection):** Shows the 'test_data' collection expanded, containing a document 'serial_humidity' and another document 'serial_temperature'.
- Column 3 (Document):** Shows the 'serial_humidity' document expanded, containing a field 'name' with the value 'humidity' and a field 'values' which is an array of four objects. Each object in the array contains 'timestamp' and 'value' fields.

The 'values' array data is as follows:

Index	timestamp	value
0	1556814688	38
1	1556814689	37
2	1556814690	38
3		

Firebase Admin SDK

Node.js

Java

Python

Go

C#

```
# Import the Firebase service
from firebase_admin import auth

# Initialize the default app
default_app = firebase_admin.initialize_app(cred)
print(default_app.name) # "[DEFAULT]"

# Retrieve services via the auth package...
# auth.create_custom_token(...)
```



Firebase Admin SDK

The screenshot shows the Firebase Admin SDK configuration page. The left sidebar contains navigation links for Project Overview, Authentication, Database, Storage, Hosting, Functions, ML Kit, Quality (Crashlytics, Performance, Test Lab), Analytics (Dashboard, Events, Conversions, A...), and Expansion (Predictions, A/B Testing, Cloud Messaging, In-App Messaging, Spark). The main content area is titled '設定' (Settings) and includes tabs for 一般 (General), Cloud Messaging, 整合 (Integration), 服務帳戶 (Service Account), 資料隱私權 (Data Privacy), and 使用者和權限 (Users and Permissions). The '服務帳戶' tab is active, showing the 'Firebase Admin SDK' section. It includes a '舊版憑證' (Legacy Certificate) section with a '資料庫密鑰' (Database Key) and a '其他服務帳戶' (Other Service Accounts) section showing '2 個來自 Google Cloud Platform 的服務帳戶' (2 service accounts from Google Cloud Platform). The 'Firebase Admin SDK' section provides a description of the SDK and a link to the '瞭解詳情' (Learn More) page. Below this, the 'Firebase 服務帳戶' (Firebase Service Account) is listed as 'firebase-adminsdk-zo4uo@vision-ee232.iam.gserviceaccount.com'. The 'Admin SDK 設定程式碼片段' (Admin SDK Configuration Code Snippets) section shows code for Node.js, Java, Python, and Go. The Python snippet is highlighted, showing the import of the SDK and the initialization of the app with a service account key. A '產生新的私密金鑰' (Generate New Private Key) button is visible, along with the last download time: '上次下載金鑰時間: 2019年5月9日 下午11:17:24'.

vision

取得說明文件

設定

一般 Cloud Messaging 整合 服務帳戶 資料隱私權 使用者和權限

管理服務帳戶權限

Firebase Admin SDK

舊版憑證

資料庫密鑰

其他服務帳戶

2 個來自 Google Cloud Platform 的服務帳戶

Firebase Admin SDK

您可以利用具有整合功能的 Admin SDK 編寫程式碼，以使用您的 Firebase 服務帳戶驗證多項 Firebase 功能，例如「資料庫」、「儲存空間」和「驗證」。[瞭解詳情](#)

Firebase 服務帳戶

firebase-adminsdk-zo4uo@vision-ee232.iam.gserviceaccount.com

Admin SDK 設定程式碼片段

☐ Node.js ☐ Java ☒ Python ☐ Go

```
import firebase_admin
from firebase_admin import credentials

cred = credentials.Certificate("path/to/serviceAccountKey.json")
firebase_admin.initialize_app(cred)
```

產生新的私密金鑰

上次下載金鑰時間: 2019年5月9日 下午11:17:24

```
cred = credentials.Certificate("./firebase_python_api/vision-admin.json")
firebase_admin.initialize_app(cred)
```

```
db = firestore.client()
doc_ref_temp = db.collection('test_data').document('serial_temperature')
doc_ref_humidity = db.collection('test_data').document('serial_humidity')
```

```
def on_message(client, userdata, message):

    data_type = message.topic.split("/")[-1]
    data = {'timestamp': int(time.time()), 'value': float(message.payload)}
    if data_type == "Humidity":
        doc_ref_humidity.update({'values': ArrayUnion([data])})

    elif data_type == "Temperature":
        doc_ref_temp.update({'values': ArrayUnion([data])})
```

140.112.42.53:3000

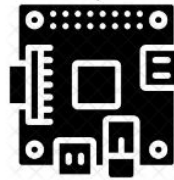


React App
(Frontend)

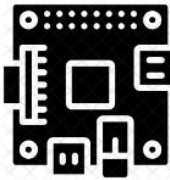


Firebase

Python API



MQTT Broker + Subscriber



MQTT Publisher



DHT11
(Digital Humidity and
Temperature Sensor)

140.112.42.53:3000



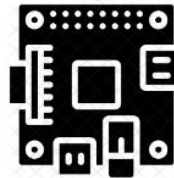
React App
(Frontend)



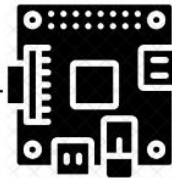
Firebase



Python API



MQTT Broker + Subscriber



MQTT Publisher



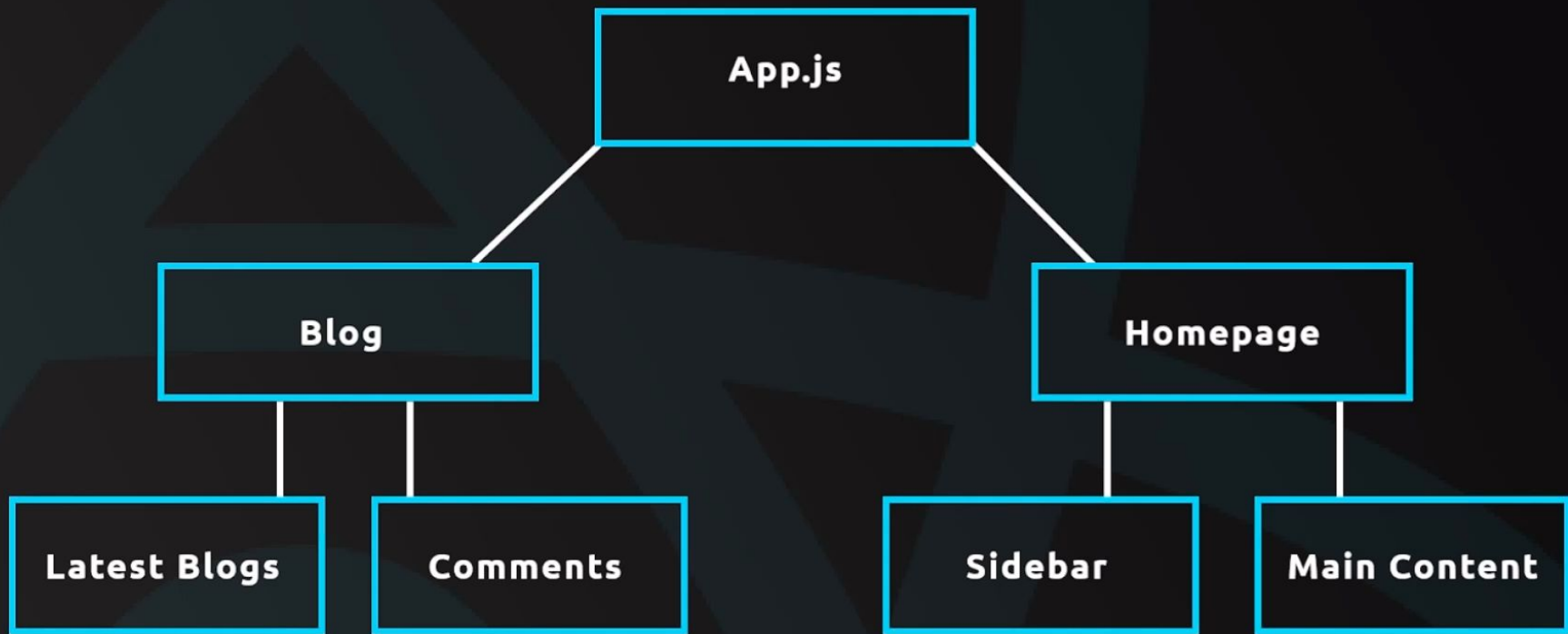
DHT11
(Digital Humidity and
Temperature Sensor)

What is Redux?

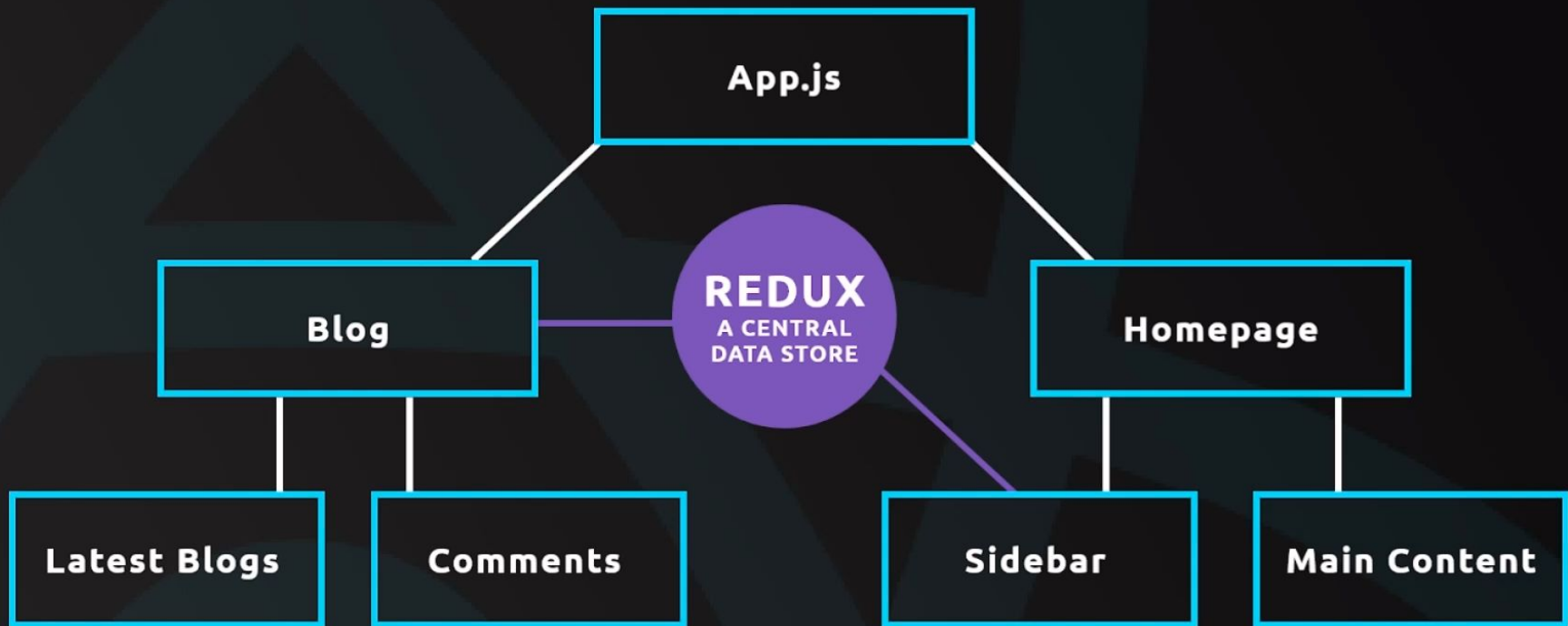


A predictable state container for JavaScript apps.

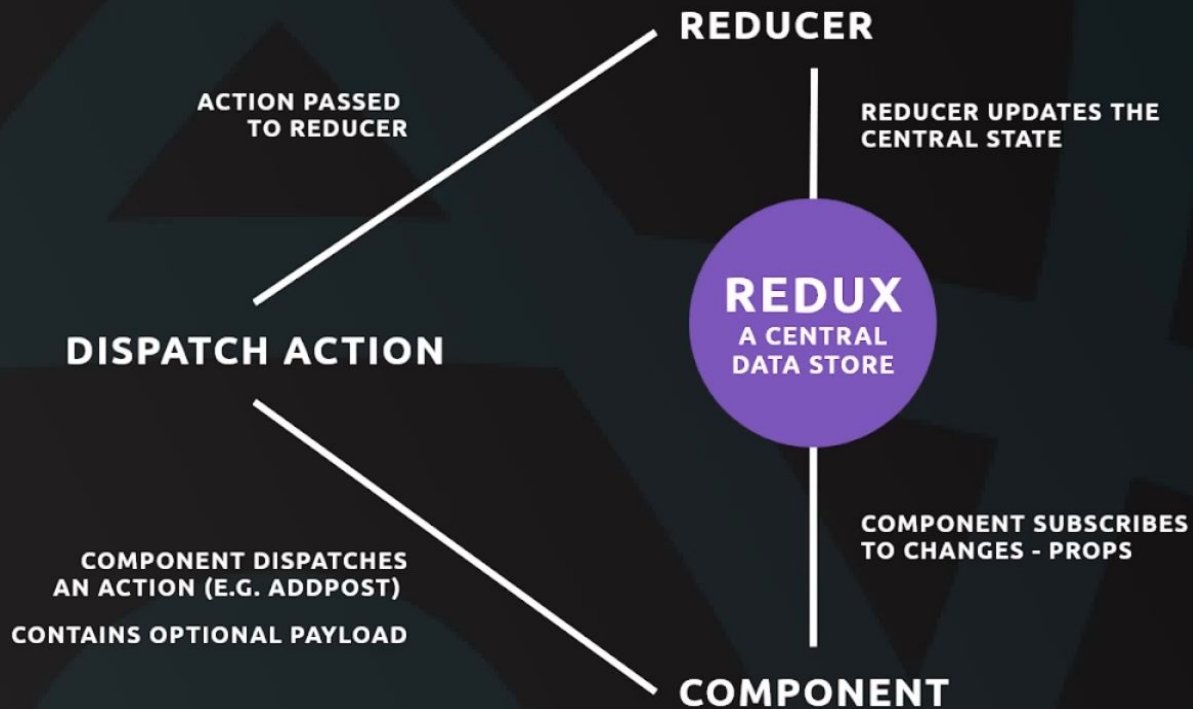
Redux



Redux



Redux



```
1  const { createStore } = Redux;
2
3  ▼ const initState = {
4    todos: [],
5    notes: []
6  }
7
8  ▼ function myreducer(state = initState, action){
9    console.log("action:", action, "state:", state);
10 }
11 const store = createStore(myreducer);
12
13 ▼ store.subscribe(() => {
14   console.log("action occurred");
15 })
16
17 const todoAction = { type: "ADD_TODO", todo: "buy milk"};
18 store.dispatch(todoAction)
```

Predictable

Redux attempts to make state mutations predictable by imposing certain restrictions on how and when updates can happen. These restrictions are reflected in the three principles of Redux:

- **Single source of truth:** The state of your whole application is stored as a tree of plain objects and arrays within a single **store**. (How much you put in the store is up to you - not all data needs to live there.)
- **State is read-only:** State updates are caused by *dispatching* an **action**, which is a plain object describing what happened. The rest of the app is not allowed to modify the state tree directly.
- **Changes are made with pure functions:** All state updates are performed by pure functions called **reducers**, which are `(state, action) => newState`

```
1 import React, {Component} from 'react'
2 import {connect} from 'react-redux'
3
4 class Home extends Component {
5   render(){
6     // do somethings here
7
8     handleClick = () =>{
9       this.props.addTo(this.state.content);
10    }
11
12    const {todos} = this.props
13  }
14 }
15 const mapStateToProps = (state) =>{
16   return todos: state.todos
17 }
18
19 const mapDispatchToProps = (dispatch) =>{
20   return addTo: (content) => {dispatch({type: 'ADD_TODO', todos:content})}
21 }
22
23 export default connect(mapStateToProps, mapDispatchToProps)(Home)
```

```
import { createStore, combineReducers, compose } from 'redux'
import { reduxFirestore, firestoreReducer } from 'redux-firestore'
import firebase from 'firebase/app'
import 'firebase/auth'
import 'firebase/database'
import 'firebase/firestore'

const firebaseConfig = {} // from Firebase Console
const rfConfig = {} // optional redux-firestore Config Options

// Initialize firebase instance
firebase.initializeApp(firebaseConfig)
// Initialize Cloud Firestore through Firebase
firebase.firestore();

// Add reduxFirestore store enhancer to store creator
const createStoreWithFirebase = compose(
  reduxFirestore(firebase, rfConfig), // firebase instance as first argument, rfConfig as optional second
)(createStore)

// Add Firebase to reducers
const rootReducer = combineReducers({
  firestore: firestoreReducer
})

// Create store with reducers and initial state
const initialState = {}
const store = createStoreWithFirebase(rootReducer, initialState)
```


Types of Queries

Each of these functions take a queryOptions object with options as described in the [Query Options section of this README](#). Some simple query options examples are used here for better comprehension.

get

```
props.store.firestore.get({ collection: 'cities' }),  
// store.firestore.get({ collection: 'cities', doc: 'SF' }), // doc
```

onSnapshot/setListener

```
store.firestore.onSnapshot({ collection: 'cities' }),  
// store.firestore.setListener({ collection: 'cities' }), // alias  
// store.firestore.setListener({ collection: 'cities', doc: 'SF' }), // doc
```

setListeners

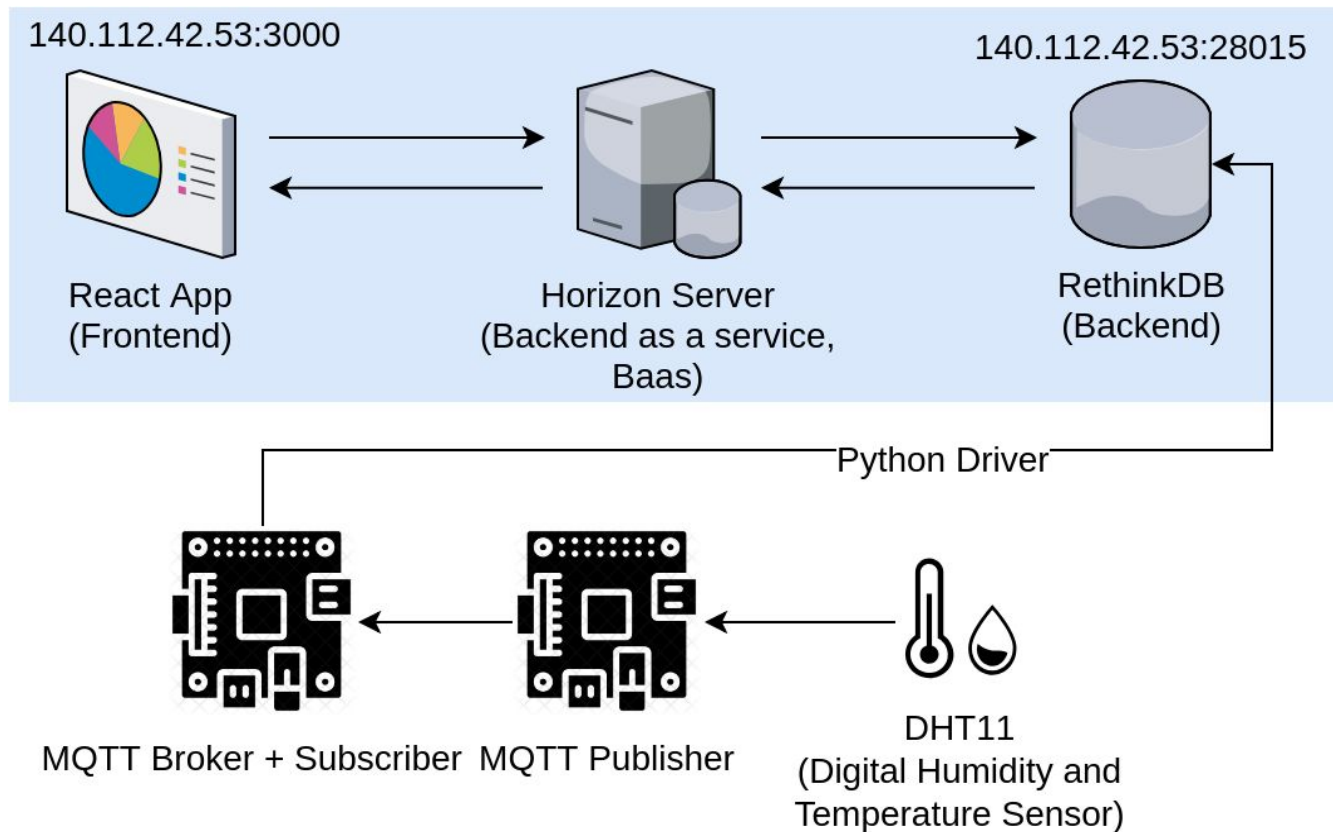
```
store.firestore.setListeners([  
  { collection: 'cities' },  
  { collection: 'users' },  
]),
```

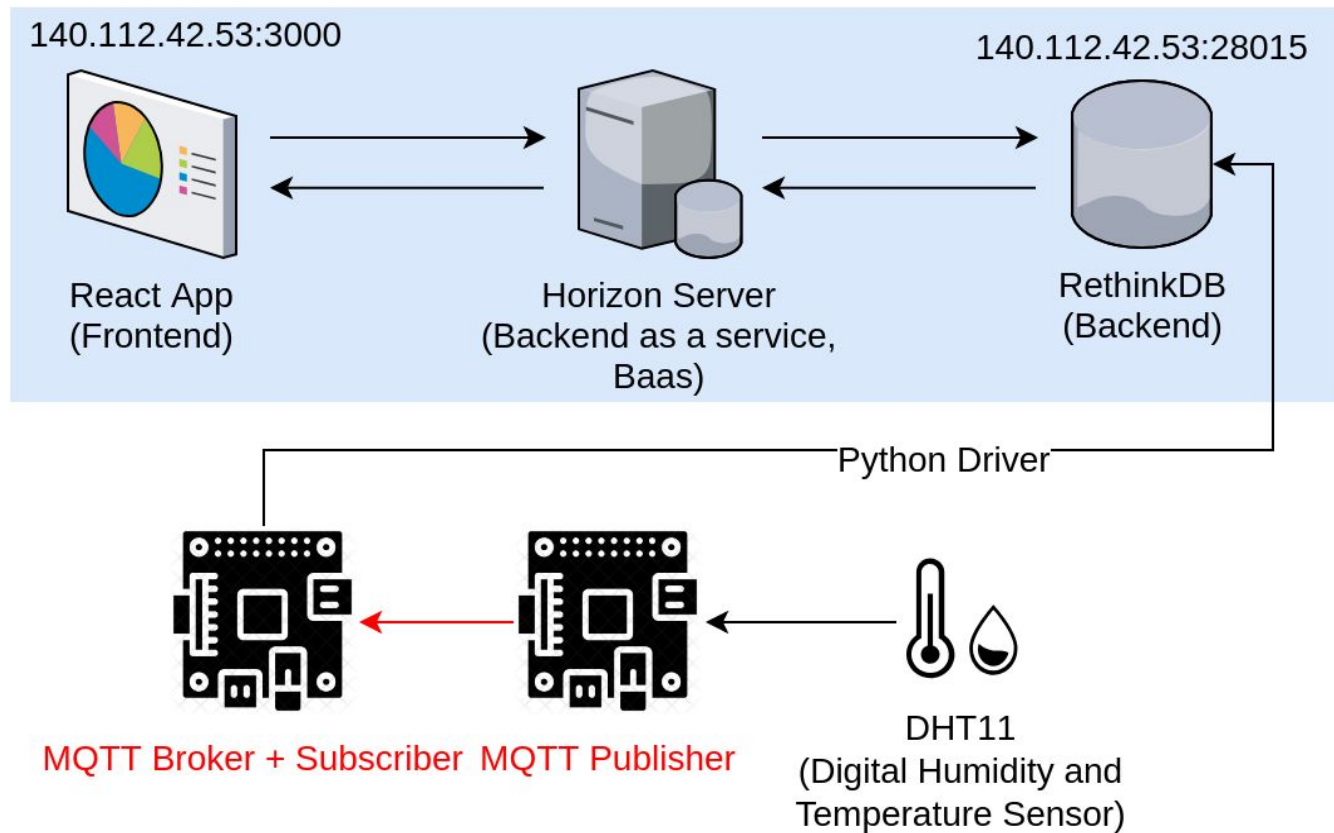
unsetListener / unsetListeners

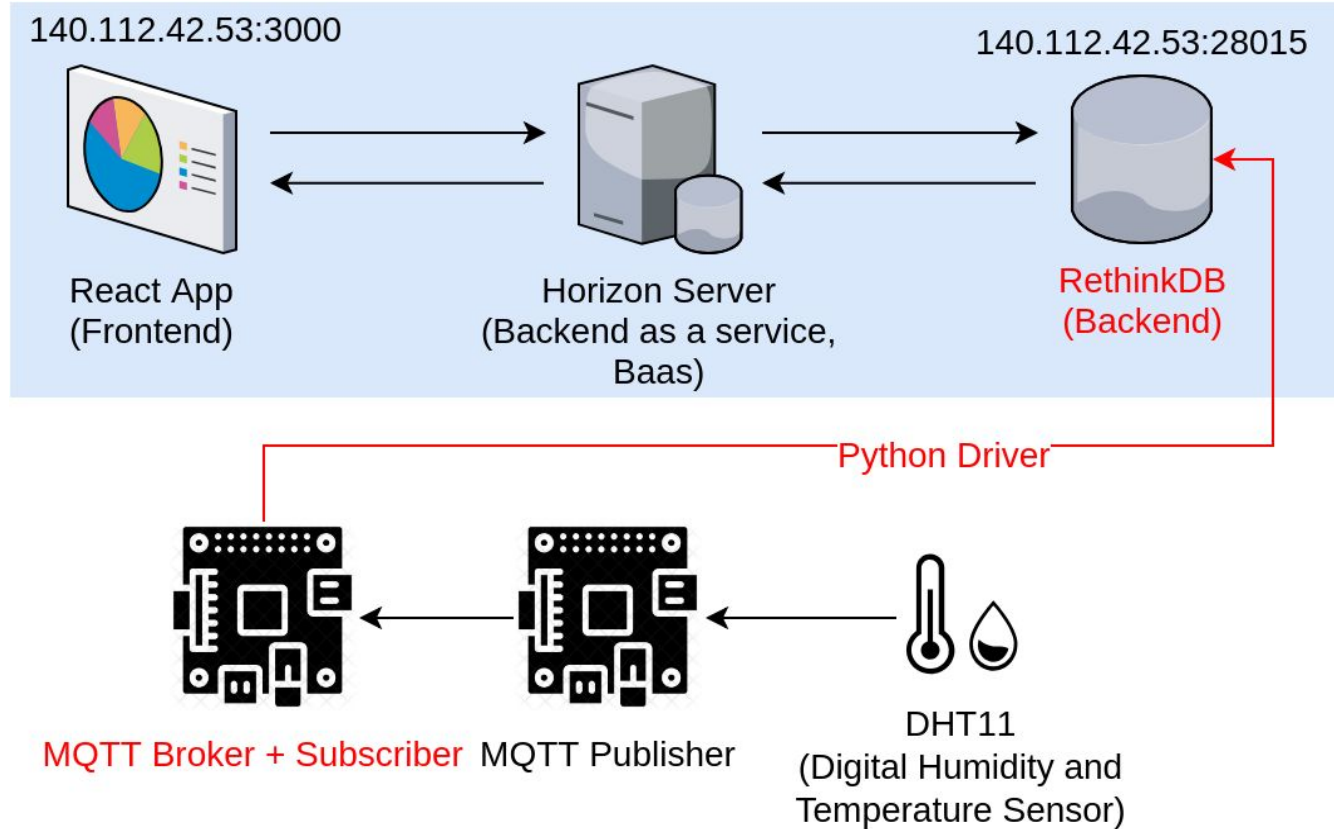
After setting a listener/multiple listeners, you can unset them with the following two functions. In order to unset a specific listener, you must pass the same queryOptions object given to onSnapshot/setListener(s).

```
store.firestore.unsetListener({ collection: 'cities' }),  
// of for any number of listeners at once :  
store.firestore.unsetListeners([query1Options, query2Options]),  
// here query1Options as in { collection: 'cities' } for example
```

Data Visualization with RethinkDB









RethinkDB

What is RethinkDB?

- **Open-source** database for building realtime web applications
- **NoSQL** database that stores schemaless JSON documents
- **Distributed** database that is easy to scale
- **High availability** database with automatic failover and robust fault tolerance

RethinkDB client drivers

RethinkDB

[faq](#)[docs](#)[api](#)[community](#)[blog](#)

Getting started

[Thirty-second quickstart](#)[Install the server](#)[Install client drivers](#)[Start a RethinkDB server](#)

The ReQL query language

The RethinkDB data model

ReQL in practice

Administration

Deployment


Troubleshooting


Integration


RethinkDB architecture


Installing RethinkDB client drivers

Official drivers

 JavaScript


 Ruby


 Python


 Java


Current community-supported drivers


These drivers have been updated to use the JSON driver protocol and at least RethinkDB 2.0 ReQL.


 C# bchavez


 C# mfenniak


 C++


 Clojure


 Common Lisp


 Dart


 Delphi


 Elixir


 Erlang


 Go


 Haskell


 JS neumino


 Lua


 Nim

 Perl

 PHP

 R

 Rust

 Swift

Connected to
xeon_e5_1620_q...Issues
No issuesServers
1 connectedTables
8/8 ready

Tables in the cluster

+ Add Database

- Delete selected tables

DATABASE

example_app

+ Add Table

Delete Database

<input type="checkbox"/>	firetony	1 shard, 1 replica	Ready 1/1
<input type="checkbox"/>	hz_collections	1 shard, 1 replica	Ready 1/1
<input type="checkbox"/>	hz_groups	1 shard, 1 replica	Ready 1/1
<input type="checkbox"/>	hz_users_auth	1 shard, 1 replica	Ready 1/1
<input type="checkbox"/>	messages	1 shard, 1 replica	Ready 1/1
<input type="checkbox"/>	tv_shows	1 shard, 1 replica	Ready 1/1
<input type="checkbox"/>	users	1 shard, 1 replica	Ready 1/1

DATABASE

pulse

+ Add Table

Delete Database

<input type="checkbox"/>	pulse	1 shard, 1 replica	Ready 1/1
--------------------------	-------	--------------------	-----------

DATABASE

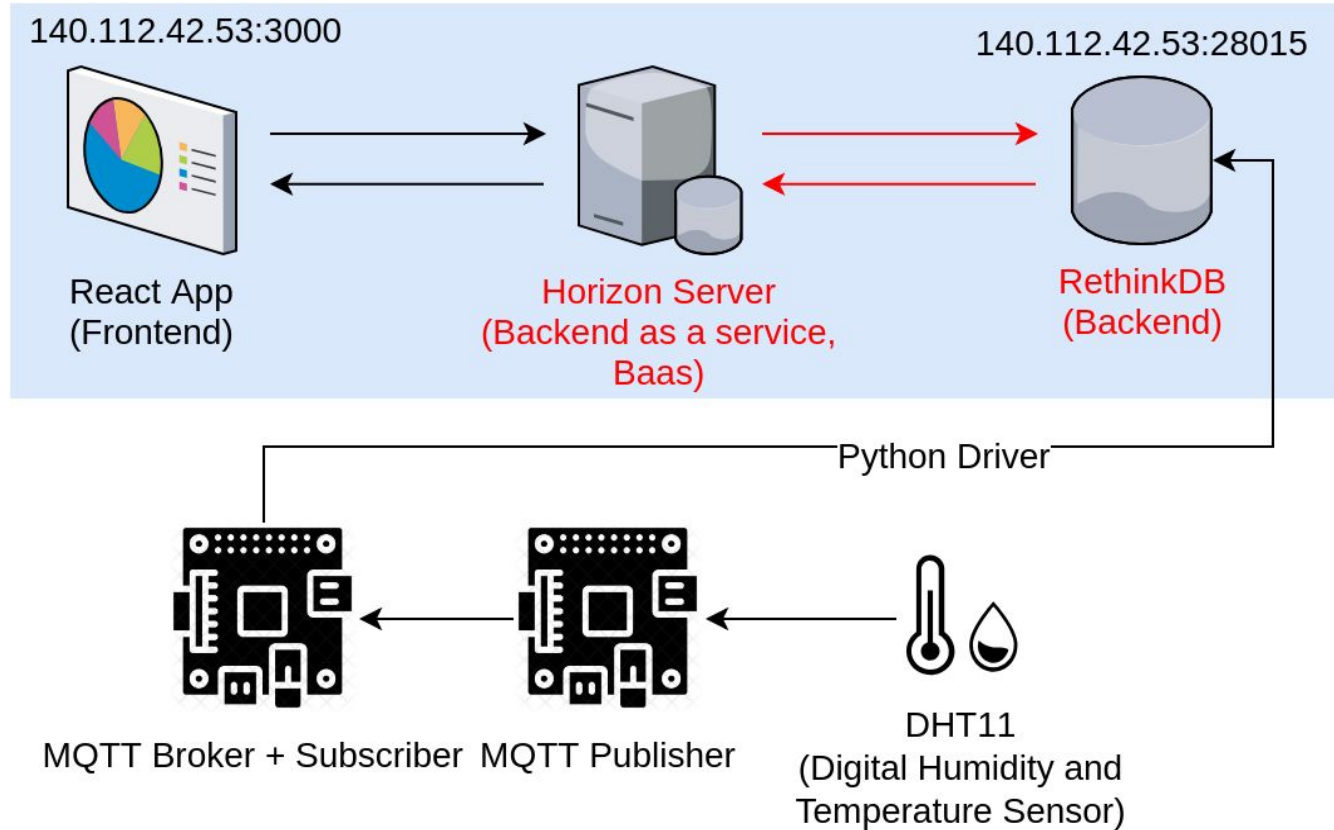
test

+ Add Table

Delete Database

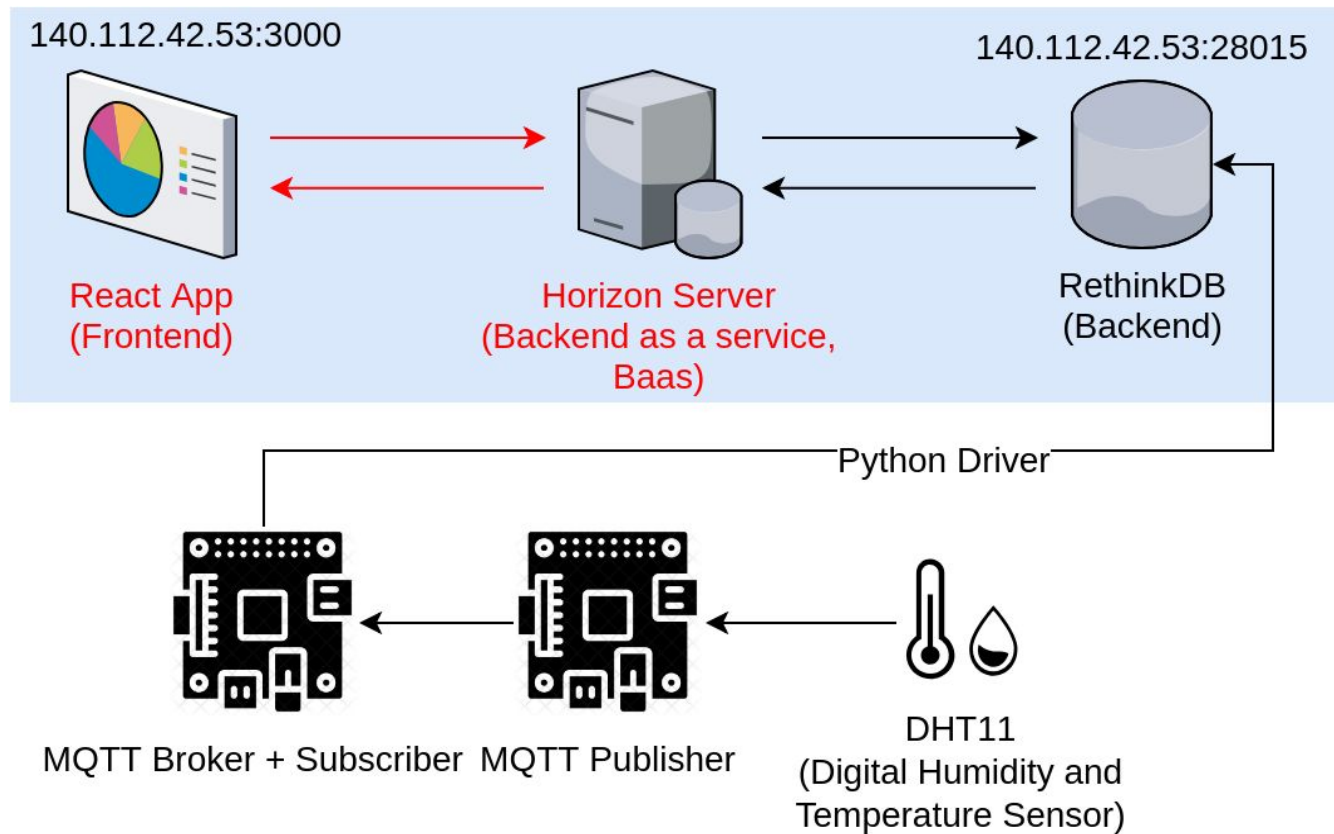
```
rdb = r.RethinkDB()
rdb.connect('140.112.42.53', 28015).repl()

def on_message(client, userdata, message):
    data_type = message.topic.split("/")[-1]
    data = {'timestamp': int(time.time()), 'value': float(message.payload)}
    if data_type == "Humidity":
        rdb.db('example_app').table('firetony').get(1002).update(
            {"values": rdb.row["values"].append(data)}
        ).run()
    elif data_type == "Temperature":
        rdb.db('example_app').table('firetony').get(1003).update(
            {"values": rdb.row["values"].append(data)}
        ).run()
```





- A **backend server** built with Node.js and RethinkDB that supports data persistence, realtime streams, input validation, user authentication, and permissions
- A **JavaScript client library** that developers can use on the frontend to store JSON documents in the database, perform queries, and subscribe to live updates
- A **command-line tool** that can generate project templates, start up a local Horizon development server, and help you deploy your Horizon application to the cloud



Connect to horizon server

```
1 export function connectorEnhancer() {
2   return createStore => (...args) => {
3     const store = createStore(...args)
4     // Initialize Horizon instance
5     const horizon = Horizon({host: host + ":" + port});
6     // Initialize firetony table
7     const serial_data = horizon(table);
8
9     getRethinkdbData(serial_data, store);
10
11     console.log('Listening for changes...')
12     serial_data.watch().subscribe((docs) => { getRethinkdbData(serial_data, store)})
13     return store
14   }
15 }
```

Connect to horizon server

```
1▼ function getRethinkdbData(serial_data, store) {
2    serial_data.fetch().subscribe(
3▼    (items) => {
4        const ret = {}
5▼    items.forEach((item) => {
6        // Each result from the serial_data collection
7        // will pass through this function
8        const { name, ...values } = item;
9        ret[name] = { ...DEFAULT_DATA, ...values }
10    })
11    store.dispatch({type: 'UPDATE', data: ret})
12    },
13▼    (err) => {
14        console.log(err);
15    })
16 }
```

Reference

<https://firebase.google.com/docs/admin/setup>

<https://github.com/Destinia/Vision>

<https://blog.isquaredsoftware.com/presentations/workshops/redux-fundamentals/>

<https://redux.js.org/basics/basic-tutorial>

<https://redux.js.org/advanced/advanced-tutorial>

<https://www.youtube.com/watch?v=OxIDLw0M-m0>

<https://www.youtube.com/watch?v=Oi4v5uxTY5o>

<https://www.rethinkdb.com/docs/install-drivers/>

<https://github.com/rethinkdb/horizon-docs/blob/master/getting-started.md>

<https://github.com/prescottprue/redux-firestore>