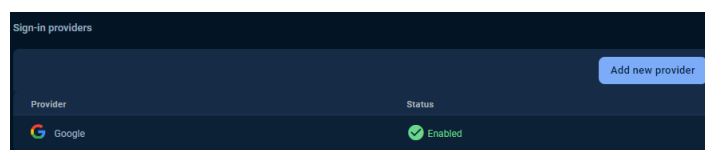


How to Run Node.js on BeagleBone Using Firebase and Connect to Google Cloud

This guide will show you how to set up a Node.js application on BeagleBone Green (BBG) using Firebase for backend services, database management, and user authentication. This setup enables you to monitor and control your BBG remotely by web. By integrating Firebase, you can handle complex backend tasks like authentication and database management efficiently, allowing for real-time operations and secure access control. This streamlined approach ensures that your application is both scalable and secure, using Google's robust infrastructure.


Step 1: Prepare Your Frontend and Authentication Setup

- Implement Google Authentication
 - I. In the Firebase Console, go to the Authentication section and enable Google as a sign-in method.



- II. Integrate Google sign-in in your frontend application using Firebase Authentication SDK
 - III. Direct users to sign in through the frontend, triggering Google's OAuth flow.
- Handle Authentication Tokens
 - I. Upon successful authentication, Firebase returns an authentication token along with a unique user ID.
 - II. Store this token securely in the client's session.
 - III. Use this token for user identification and authorization in all subsequent CRUD operations against your backend or Firebase services.
 - IV. Ensure each request from the client to Firebase or your Node.js backend includes this token to maintain session integrity and security.
 - V. Display the user ID in the user interface, allowing users to see their unique identifier which can be used for tracking activities and personalizing the experience.

 **User ID:** kH36y[REDACTED]LHliar7c9[REDACTED]rE03

 Do not share your ID.

Users can see a unique user id in help of website

Step 2: Set Up Node.js on BeagleBone Green with Firebase SDK

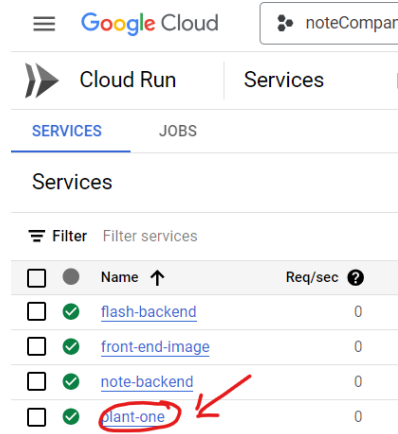
Attention: Make sure you distribute the binary version of the node server code.

- Install Firebase SDK:
 - I. Enable server-side interactions with Firebase services.

```
(bbg) cd /path/to/node/project
(bbg) npm install firebase-admin --save.
```
 - II. Initialize Firebase in Your Node.js Application
download this JSON file from the Firebase Console under Project Settings > Service accounts. Set up the Firebase Admin SDK with

```
var serviceAccount = require("/path/to/service-account-file.json").
```

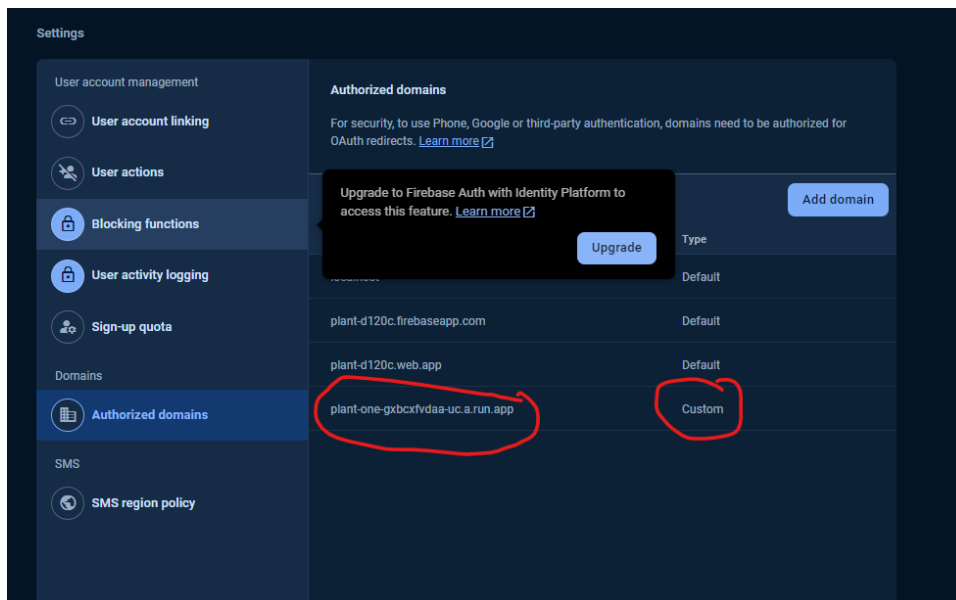

- V. Install Google Cloud SDK Shell and login to the account of you GCP.
- VI. Run (\$) `docker build -t gcr.io/your-project-id` on `GCP/your-app-name: tag` .
- VII. Run (\$) `gcloud auth configure-docker`
- VIII. Deploy Your Container Image to Cloud Run
 (\$) `gcloud run deploy your-service-name --image gcr.io/your-project-id/your-app-name:tag --platform managed --region your-region --allow-unauthenticated`



you can also push you image to Cloud repo and manually in Cloud Run create a service.

Then select the container image.

The google run now creates a domain after deploying. You should add it to firebase domains table to allow authentication.



Troubleshooting Guide

This section provides practical solutions to common problems that may arise during the setup and operation of your Node.js application on BeagleBone Green, using Firebase for backend services, and deploying to Google Cloud Run.

Trouble 1: Firebase Authentication Not Working

Symptoms:

Users cannot sign in.

Authentication token is not being issued.

Solutions:

Check Firebase Console Settings:

Ensure that Google authentication is enabled in the Firebase Console under the Authentication section.

Verify that the correct project is selected and that you have not exceeded any quota limits.

Validate OAuth Configuration:

Double-check the OAuth consent screen configuration in Google Cloud Console. Make sure the correct scopes are authorized and the app is published if required.

Inspect Network Requests:

Use browser developer tools to inspect authentication requests and responses. Look for errors in the network tab.

Check SDK Initialization:

Ensure that the Firebase SDK is correctly initialized with the right project credentials. Verify that the configuration matches the one in your Firebase Console.

Trouble 2: CRUD Operations Fail or Return Incorrect Data

Symptoms:

CRUD operations do not reflect the expected changes in the database.

Data retrieved does not match what is expected.

Solutions:

Permissions and Rules:

Check Firestore or Firebase Realtime Database rules in the Firebase Console. Ensure rules allow for the intended operations based on user authentication status.

Logging and Error Handling:

Implement detailed logging around CRUD operations. Capture and log any errors returned by Firebase operations.

Use try-catch blocks in your Node.js code to handle and debug errors effectively.

Verify User IDs:

Ensure that the user ID used in CRUD operations matches the user ID from the authentication token.

Confirm that this ID is correctly passed from the frontend to the backend.

Trouble 3: Inter-Server Communication Issues

Symptoms:

Node.js and C++ servers are not syncing data correctly.

Data corruption or loss during read/write operations.

Solutions:

Check Lock File Mechanism:

Verify that the lock file is being created and deleted correctly. Ensure that both servers respect the lock file protocol.

Add logging around lock file creation and deletion to track its state.

File Access Permissions:

Confirm that both Node.js and C++ applications have the necessary read/write permissions for the shared file and the directory it resides in.

Handling Simultaneous Access:

Implement a retry mechanism in both Node.js and C++ code to handle cases where the file is locked.

Set up a reasonable retry interval and maximum retry count.

Trouble 4: Deployment Issues on Cloud Run

Symptoms:

Deployment fails.

The service does not start or responds with errors.

Solutions:

Container Issues:

Ensure that your Dockerfile is correctly set up and all necessary files are included in the build context. Check the Docker build logs for errors. Ensure that your `node_modules` directory is not causing issues; consider using a `.dockerignore` file to exclude it from the build context.

Google Cloud SDK and Configuration:

Make sure the Google Cloud SDK is correctly installed and authenticated. Run `gcloud auth list` to see active accounts.

Verify that the project ID and region are correctly specified in your deployment commands.

Service Configuration:

Check the logs in Cloud Run to diagnose startup failures. Use `gcloud run logs read --service your-service-name` to view logs.

Ensure that the Cloud Run service has the necessary permissions, especially if it interacts with other Google Cloud resources.