Implementation Guide

Step-by-step instructions for BigQuery-based Rewarded Optimization recipe

Introduction **2**
Objectives 2

Step #2. App : Send Rewarded Funnel events and Virtual Currency events to firebase **3**
spend_virtual_currency example 4
earn_virtual_currency example 4
rewarded_available example (when rewarded ad is available) 5
rewarded_opt_in example - onRewardedAdOpened() 5
rewarded_completed example - onUserEarnedReward(@NonNull reward: RewardItem) 5
rewarded_failed example - onRewardedAdFailedToShow(errorCode: Int) 5
Custom Event Troubleshooting Guide 6

Step #3. Firebase : Create separate ad unit for each of rewarded placement **7**

Step #4. Firebase : BigQuery Export **8**
Link BigQuery 8

Step #5. Install the Firebase Extensions and Setup Log Sink **9**
Installing Rewarded Ad Funnel Extension 9
Set up a Log sink 10
Using the extension 15

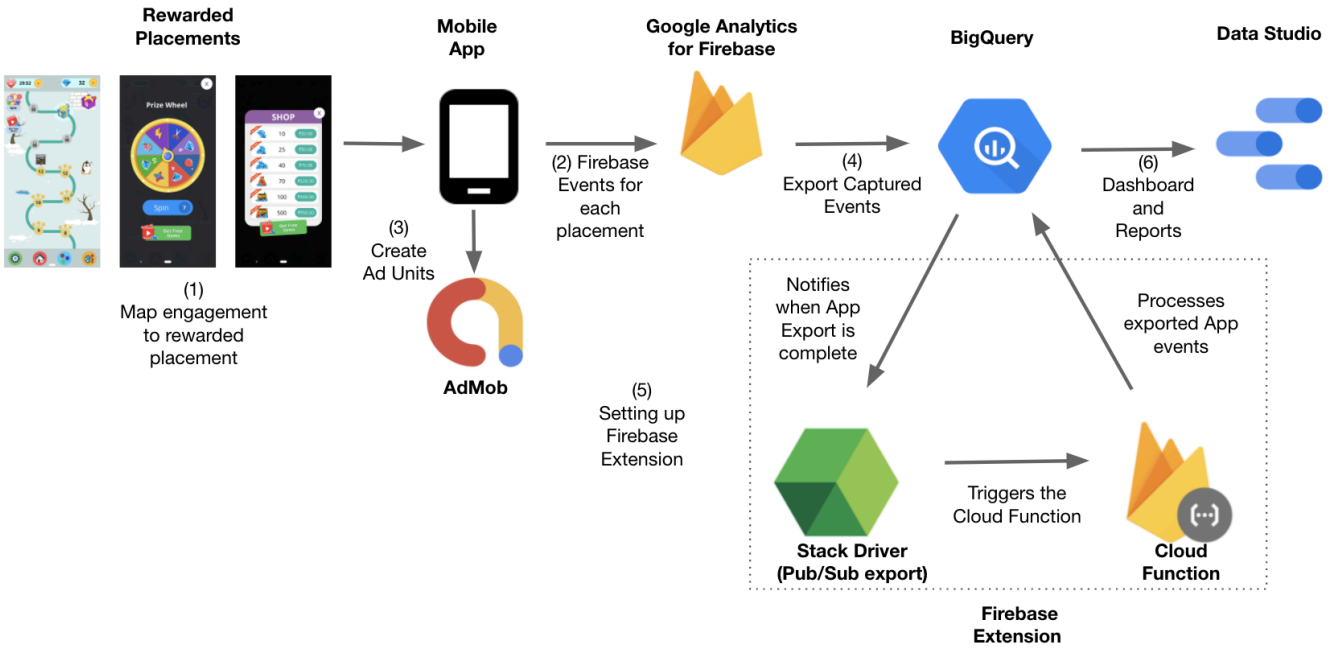
Step #6. DataStudio : Dashboard and Reports **16**
Start from the Template 16
Configure virtual_currency_events_YYYYMMDD data source 16
Configure rewarded_ad_events_YYYYMMDD data source 19
Configure events_YYYYMMDD data source 21
Generate the report 21

Introduction

This document is a step-by-step implementation guide to set up the BigQuery, Firebase, Cloud and Data Studio based recipe to evaluate user engagement and revenue attribution for every rewarded placement.

Objectives

- Understand the performance of different rewarded placements within the app
- Optimize each rewarded placement by analysing the views, opt-in and completions for every rewarded placement
- Make data based decisions to changes in game design such as offering users higher chances to clear levels or lifting frequency caps etc
- Identify low performing rewarded ad placements and improve them by making design changes or improving their placements
- Improve specific features or events within the app.
  For e.g. improve spin wheel design or reward offering if metrics are poor



Step #1. Map existing engagement components to the rewarded placement
Step #2. Send Rewarded Funnel events and Virtual Currency events from your application to firebase
Step #3. Create separate ad unit for each of the rewarded placements
Step #4. Export Firebase data into BigQuery
Step #5. Install Firebase extension in your Firebase Project & Setup Log Sink
Step #6. Visualize and analyse data on DataStudio

Step #1. App : Map existing engagement components to the rewarded placement

Step #2. App : Send Rewarded Funnel events and Virtual Currency events to Firebase

Developers need to set up Google Analytics for Firebase SDK in their app and log rewarded funnel events and virtual currency events and its parameters accordingly.

The recipe requires developers to log six app events; spend_virtual_currency, earn_virtual_currency, rewarded_available, rewarded_opt_in, rewarded_completed and rewarded_failed. Details for each app event are described in the table below.

Note that all event parameters are **mandatory.** Failing to provide appropriate values on each parameter may cause the analysis not to work as expected.

Event name	Parameter name	Type
spend_virtual_currency	virtual_currency_name	Text
	value	Number
	item_name	Text
earn_virtual_currency	virtual_currency_name	Text
	value	Number
	placement	Text
	source	Text (Enum) (e.g., iap | ad | free)
rewarded_available	placement	Text
	stage (optional)	Number
rewarded_opt_in	placement	Text
	stage (optional)	Number
rewarded_completed	placement	Text
	stage (optional)	Number
rewarded_failed	placement	Text
	stage (optional)	Number

Developers need to add some code to track rewarded funnel events and virtual currency events in their apps.

The following code snippet shows how developers can log rewarded funnel events and virtual currency events on Unity with Firebase Unity SDK.

Firebase Custom Event Help Guide :
Android | iOS | Unity

spend_virtual_currency example

```
// User spent 3 stars to buy an item named treasure_box
FirebaseAnalytics.LogEvent(
    FirebaseAnalytics.EventSpendVirtualCurrency,
    new Parameter[] {
        new Parameter(FirebaseAnalytics.ParameterVirtualCurrencyName,
"star"),
        new Parameter(FirebaseAnalytics.ParameterValue, 3),
        new Parameter(FirebaseAnalytics.ParameterItemName,
"treasure_box")
    }
);
```

earn_virtual_currency example

```
// User has acquired 5 starts by watching an ad in daily_login
placement.
FirebaseAnalytics.LogEvent(
    FirebaseAnalytics.EventEarnVirtualCurrency,
    new Parameter[] {
        new Parameter(FirebaseAnalytics.ParameterVirtualCurrencyName,
"star"),
        new Parameter(FirebaseAnalytics.ParameterValue, 5),
        new Parameter("placement", "daily_login"),
        new Parameter("source", "ad")
    }
);
```

rewarded_available example (when rewarded ad is available)

Fire the event when rewarded is available to be viewed by User. Do not fire this event when you request a Rewarded Ad.

```
FirebaseAnalytics.LogEvent(
    FirebaseAnalytics.rewarded_available,new Parameter[] {
        new Parameter("placement", "map"),
        new Parameter("stage", "1") //current user stage
    });
```

rewarded_opt_in example - onRewardedAdOpened()

Fire the event when User clicks on rewarded Ad to avail reward. You can fire this event in onRewardedAdOpened() method.

```
FirebaseAnalytics.LogEvent(
    FirebaseAnalytics.rewarded_opt_in,new Parameter[] {
        new Parameter("placement", "map"),
        new Parameter("stage", "1") //current user stage
    });
```

rewarded_completed example - onUserEarnedReward(@NonNull reward: RewardItem)

Fire the event when the user is done watching rewarded Ad and have finally received the rewards. You can fire the event in onUserEarnedReward() method.

```
FirebaseAnalytics.LogEvent(
    FirebaseAnalytics.rewarded_completed,new Parameter[] {
        new Parameter("placement", "map"),
        new Parameter("stage", "1") //current user stage
    });
```

rewarded_failed example - onRewardedAdFailedToShow(errorCode: Int)

Fire the event if the rewarded Ad fails to load. Do not fire the event in case of no match from the Ad server. You can fire the event in onRewardedAdFailedToShow() method.

```
FirebaseAnalytics.LogEvent(
    FirebaseAnalytics.rewarded_failed,new Parameter[] {
        new Parameter("placement", "map"),
        new Parameter("stage", "1") //current user stage
    });
```

Custom Event Troubleshooting Guide

DebugView enables you to see the raw event data logged by your app on development devices in near real-time. This is very useful for validation purposes during the instrumentation phase of development and can help you discover errors and mistakes in

your analytics implementation and confirm that all events and user properties are being logged correctly.
So You can leverage Firebase debugview in order to verify the custom event implementation and the right set of parameters are flowing to the firebase dashboard or not.

Firebase Help :
Android | iOS

Enabling Debug Mode

Android
```
adb shell setprop debug.firebase.analytics.app package_name
```

iOS
```
-FIRDebugEnabled
```

Disabling Debug Mode

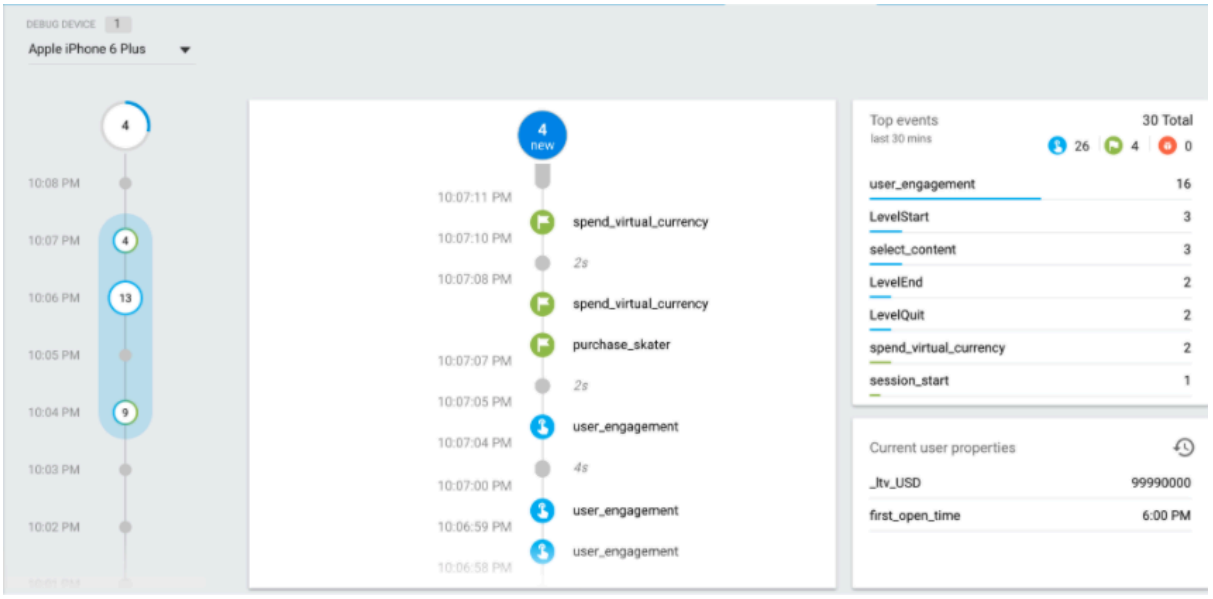
Android
```
adb shell setprop debug.firebase.analytics.app .none.
```

iOS
```
-FIRDebugDisabled
```

Complete guide for enabling debug mode

Reporting

Once you have enabled Debug mode on your development devices, navigate to DebugView by selecting the arrow next to StreamView on the top nav of Google Analytics and selecting DebugView.

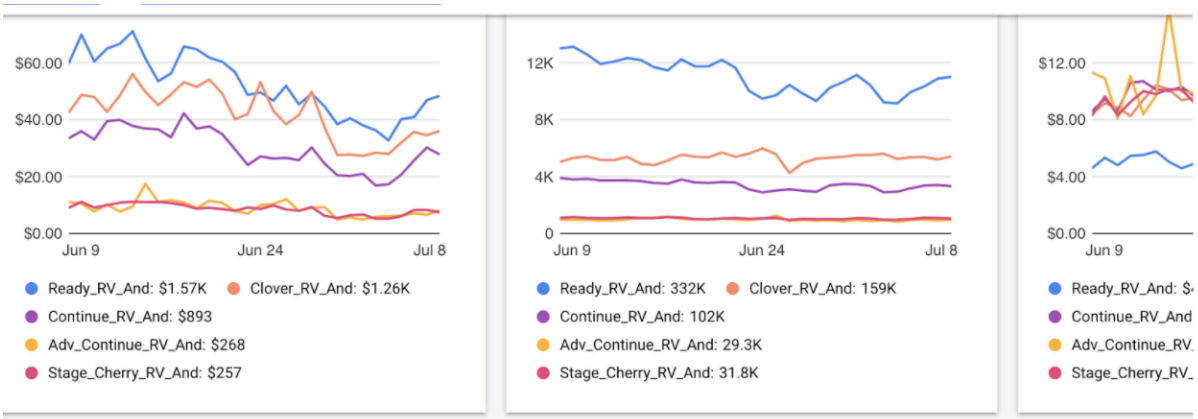


Then, just start using your app to see your app's events being logged in the DebugView report.

Click on the specific custom event to verify parameters flowing along with the event.

Step #3. Firebase : Create separate ad unit for each of rewarded placement

Create a separate ad unit in the AdMob dashboard and map them accordingly with each rewarded placement in the app.
Update the title with a separate ad unit implemented.



AdMob Network

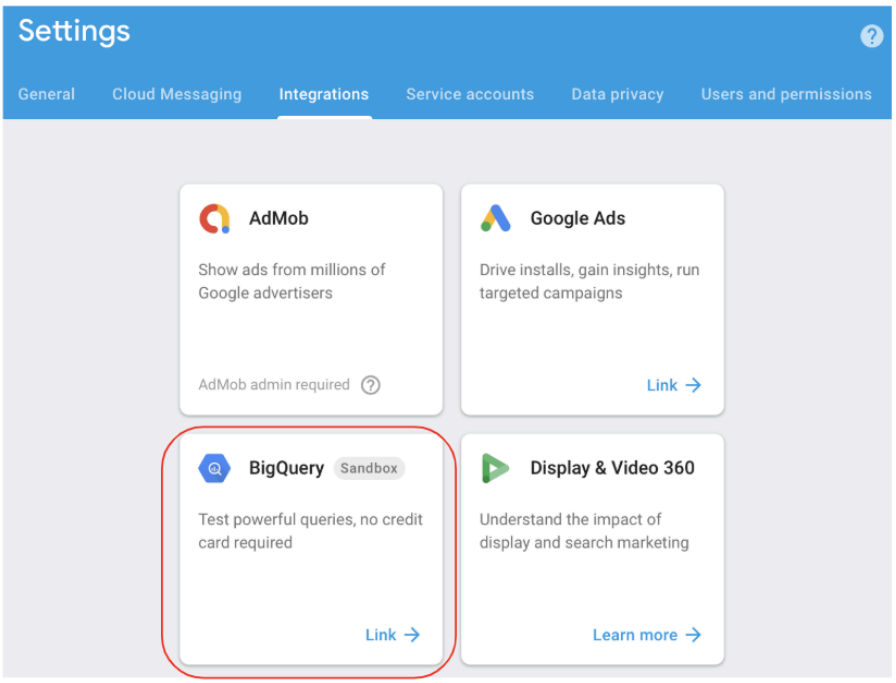
Ad unit	Estimated earnings	Impression RPM	Show rate	Impressions	Impressions CTR	Clicks
Adv_Continue_RV_And	$268.06	$9.15	4.02%	29,294	1.35%	396
Continue_RV_And	$893.12	$8.72	12.52%	102,439	1.86%	1,908
Stage_Cherry_RV_And	$257.01	$8.08	4.34%	31,810	1.16%	368
Clover_RV_And	$1,260.75	$7.91	18.01%	159,301	1.54%	2,450
Ready_RV_And	$1,571.56	$4.74	31.75%	331,773	1.42%	4,713

Step #4. Firebase : BigQuery Export

Link BigQuery

To Export Firebase data into BigQuery, you need to link your Firebase project to BigQuery. Once that done, raw event data including first_open, in_app_purchase, paid_ad_impression and other automatically collected events will be collected and stored in your BigQuery table.

Firebase Help :
Link BigQuery to Firebase
Automatically Collected Events



Step #5. Install the Firebase Extensions and Setup Log Sink

Before installing the extensions, you'll need to install the firebase-tools on your machine. On macOS/Linux environments, sudo may be required.

```
$ npm install -g firebase-tools
```

Installing the extension from local is disabled by default. Enable it by running the following command.

```
$ firebase --open-sesame extdev
```

Next, download the Goose extension file & rewarded ad funnel extension file and unzip to the location wherever you want.

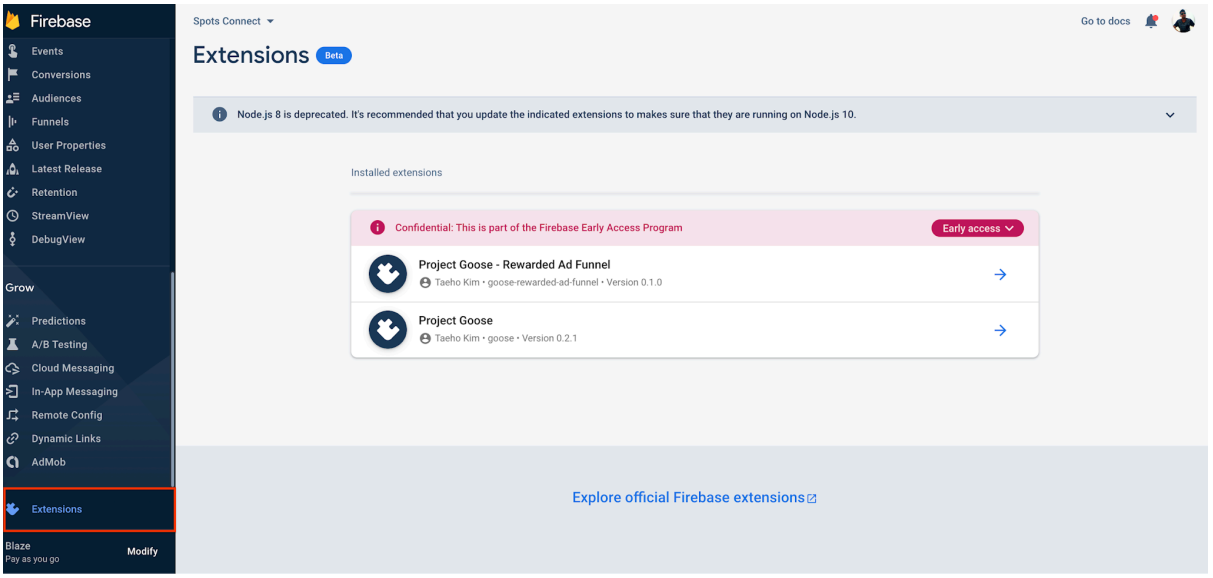
Then, cd to the location where you unarchived the extensions. Run the following commands to install the extension to the Firebase project. Note that it should be run from the extension root directory, where extensions.yaml file exists.

Guide to install Firebase Extensions

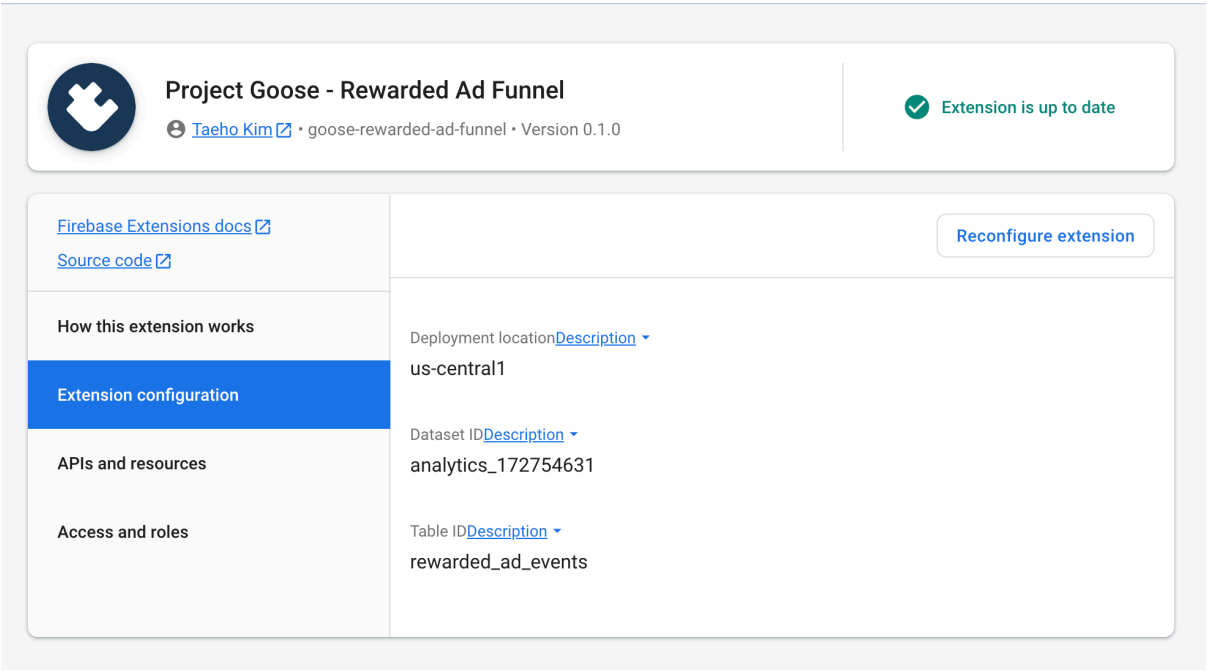
Installing Rewarded Ad Funnel Extension

```
$ npm install
$ npm run build && firebase ext:install . --project=FIREBASE_PROJECT_ID
```

Follow the on-screen instructions to install the extension. Once the extension is installed, you can see the list of installed extensions from **Firebase console > Extensions.**



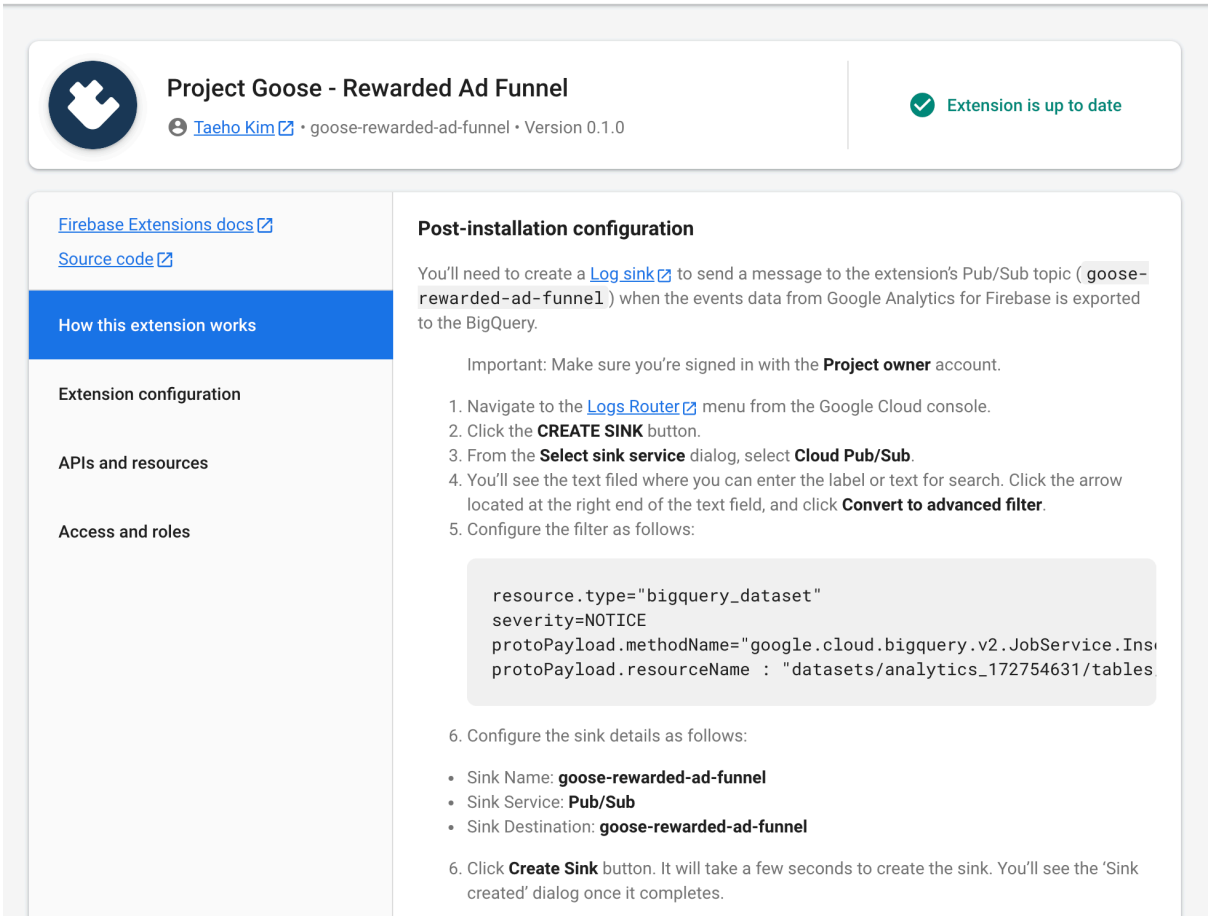
Once you click the **Manage** button, you can see how the extension was configured.



Set up a Log sink

Once you click **‘How this extension works’** section, you can see a post-installation step required to complete.

Since the extension will not create a log sink required to trigger the function, you'll need to follow the steps under the **'Set up a Log sink'** section to make the extension run correctly.



The `processRewardedAdEvents` function creates a new table by collecting only rewarded funnel events from the app events exported to the BigQuery database. Therefore, the function needs to be run after the events data is exported to the BigQuery.

To get notified when the export completes, we can leverage the Stackdriver, which is responsible for processing logs from Google Cloud products. In the Stackdriver, you can export specific logs to other channels, including BigQuery, Cloud Storage, and Pub/Sub, by using the Log Sinks.

For each Firebase Extension, one log sink is required to trigger the function via Pub/Sub when the app events are exported to the BigQuery.

Once the app events are exported to the BigQuery, a log similar to the following will be generated.

```
{
  "insertId":"eflcf2e4ahv7",

"logName":"projects/[PROJECT_ID]/logs/cloudaudit.googleapis.com%2Factivi
```

```
ty",
  "protoPayload":{
    "@type":"type.googleapis.com/google.cloud.audit.AuditLog",
    "authenticationInfo":{
      "principalEmail":"firebase-measurement@system.gserviceaccount.com"
    },
    "authorizationInfo":[
      {
        "granted":true,
        "permission":"bigquery.tables.create",
        "resource":"projects/[PROJECT_ID]/datasets/[DATASET_ID]"
      }
    ],
    "metadata":{

"@type":"type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata",
      "tableCreation":{

"jobName":"projects/[PROJECT_ID]/jobs/911435721598[DATASET_ID]0202002071
581109578658882",
        "reason":"JOB",
        "table":{
          "schemaJson":"...",

"tableName":"projects/[PROJECT_ID]/datasets/[DATASET_ID]/tables/[TABLE_I
D]"
        }
      }
    },
    "methodName":"google.cloud.bigquery.v2.JobService.InsertJob",
    "requestMetadata":{

    },

"resourceName":"projects/[PROJECT_ID]/datasets/[DATASET_ID]/tables/[TABL
E_ID]",
    "serviceName":"bigquery.googleapis.com",
    "status":{

    }
  },
  "receiveTimestamp":"2020-02-07T21:06:22.675721915Z",
  "resource":{
    "labels":{
      "dataset_id":"[DATASET_ID]",
      "project_id":"[PROJECT_ID]"
```

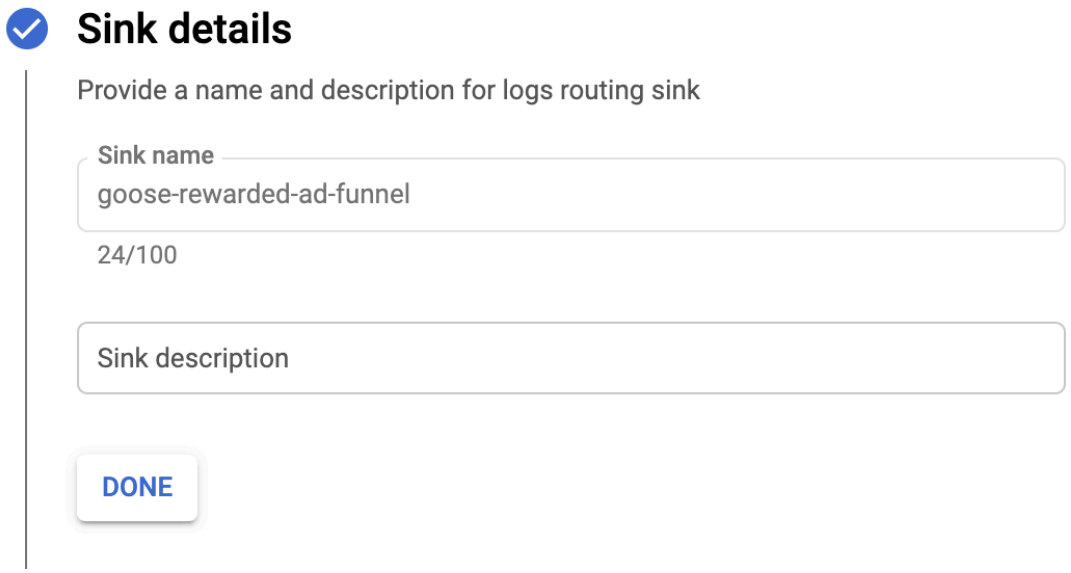
```
    },
    "type":"bigquery_dataset"
  },
  "severity":"NOTICE",
  "timestamp":"2020-02-07T21:06:22.64Z"
}
```


To create a log sink, navigate to the Logs Router menu from the Google Cloud console, and click the **CREATE SINK** button.

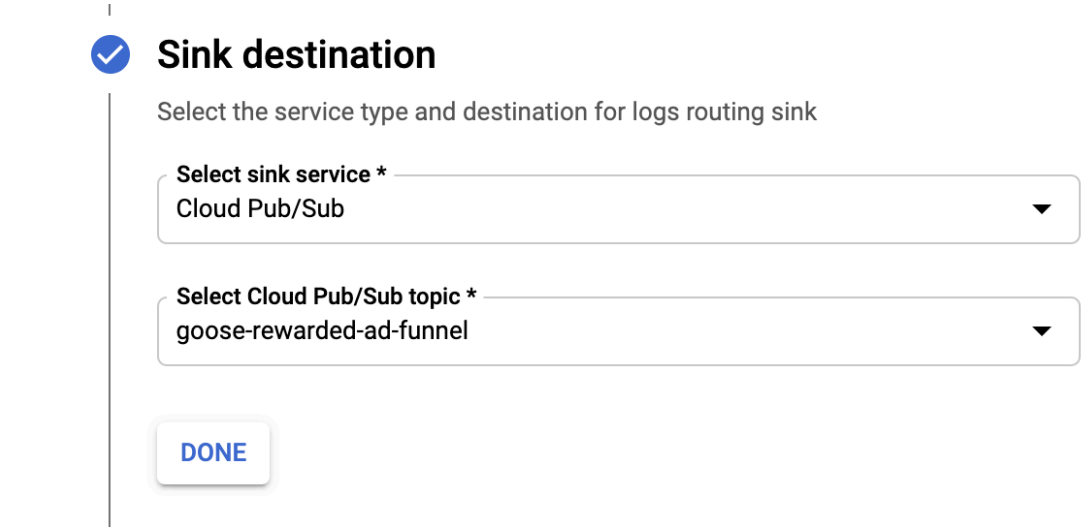
And follow the instructions:

Convert to advanced filter by clicking the **Convert to advanced filter** menu.

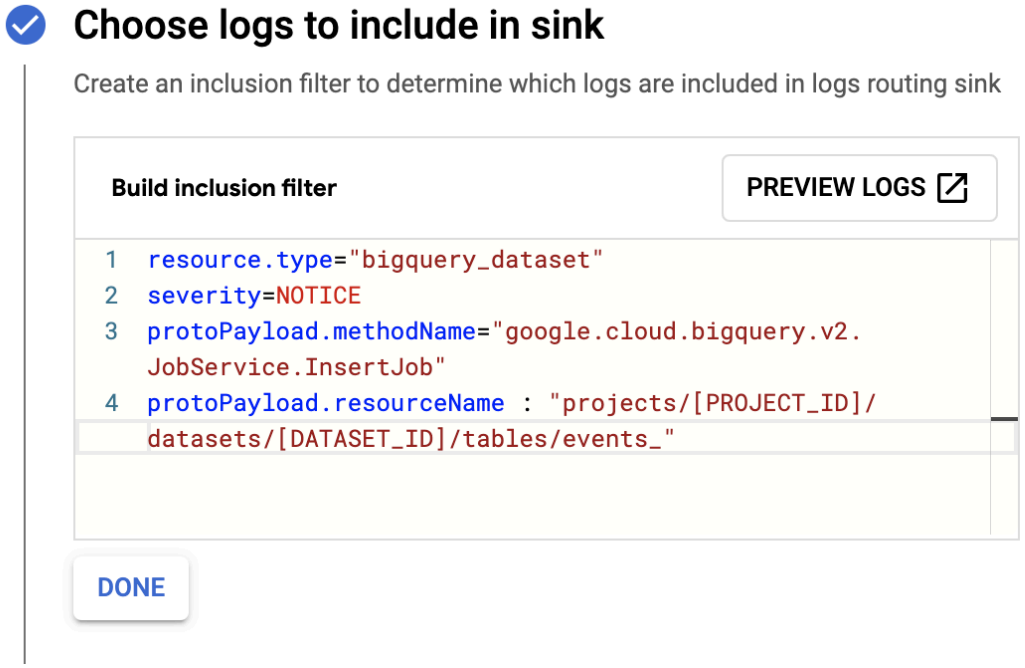
1. Enter Sink Name



2. Select Sink Destination & and Service



3. Choose Logs to include in sink



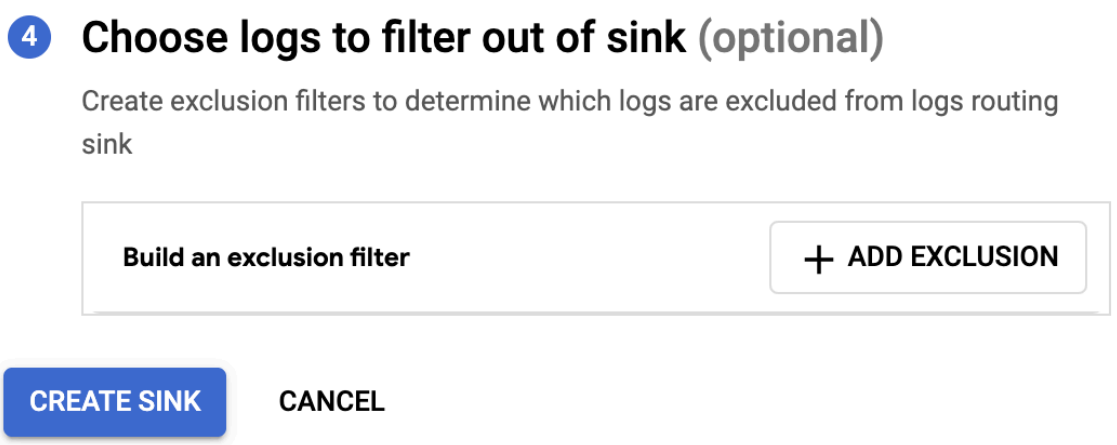
Configure the filter as follows:

```
resource.type="bigquery_dataset"
protoPayload.methodName="google.cloud.bigquery.v2.JobService.InsertJob"
protoPayload.resourceName:"projects/[PROJECT_ID]/datasets/[DATASET_ID]/t
ables/events_"
severity=NOTICE
```

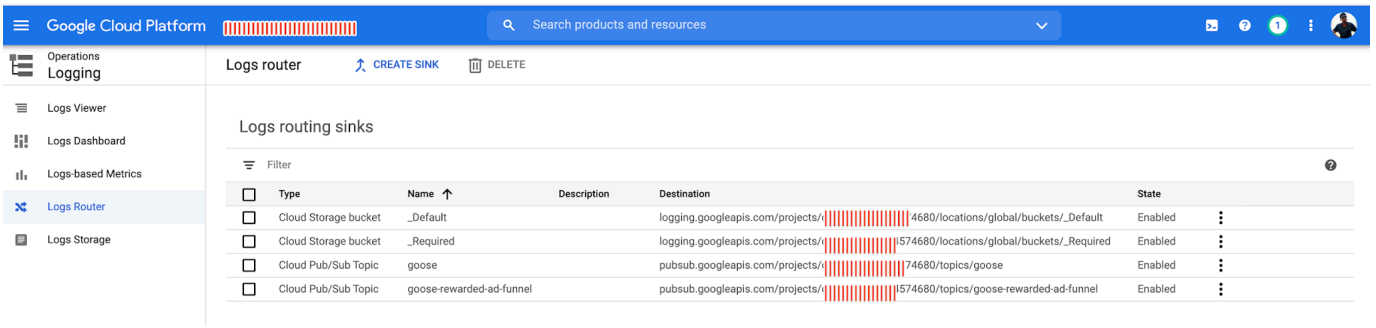
For example, if the project id is `my-awesome-project` and dataset ID is `analytics_123456`, filter configuration will look like:

```
resource.type="bigquery_dataset"
protoPayload.methodName="google.cloud.bigquery.v2.JobService.InsertJob"
protoPayload.resourceName:"projects/my-awesome-project/datasets/analytics_123456/tables/events_"
severity=NOTICE
```

4. Lastly click on “Create Sink”.



Once a log sink is created, you can see the sink from the **Google Cloud console > Logs > Logs Router**.



Follow the same process to set up the goose extension and log sink for virtual currency events.

Using the extension

After you complete the post-installation configuration above, the extension will create new tables (virtual_currency_events_YYYYMMDD) and (rewarded_ad_events_YYYYMMDD) under the dataset analytics_XXXXXXXXX.

Presence of these two tables under the dataset analytics_XXXXXXXXX confirms that the firebase extension and log sink set up is rightly done.

The table stores data specific to virtual currency events and rewarded ad funnel events which are required in the Data Studio reports for data visualization.

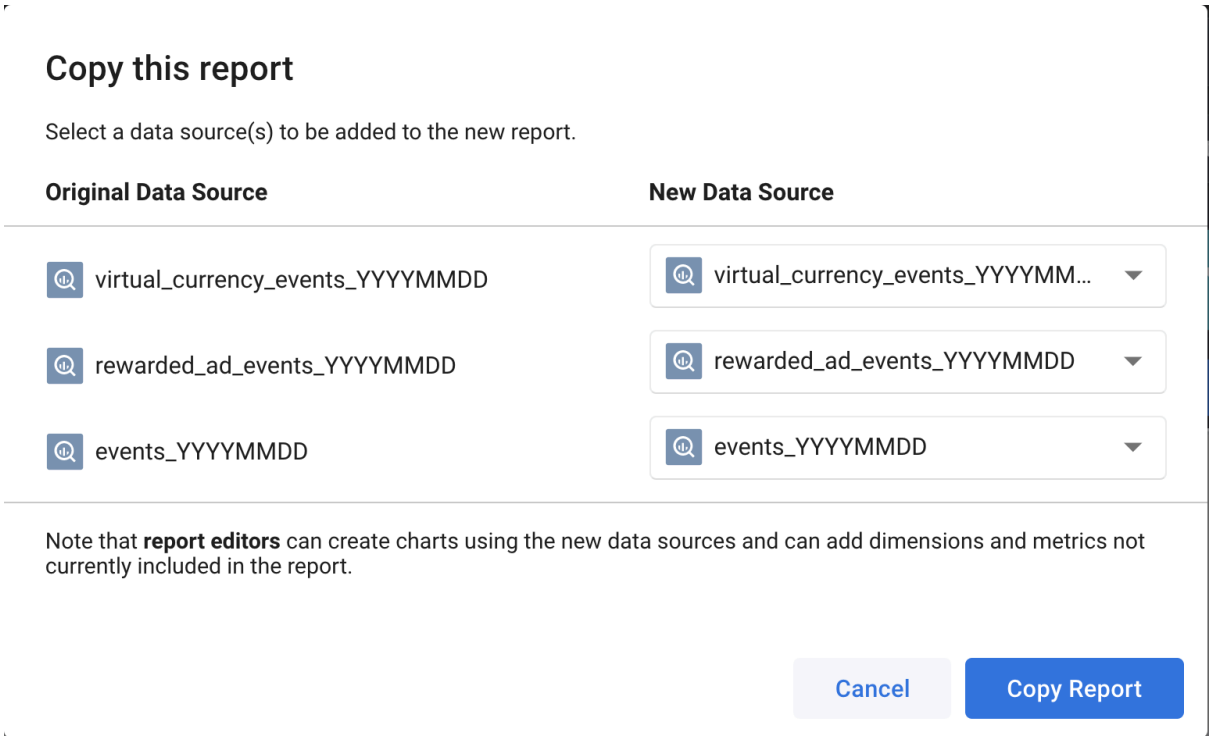
Step #6. DataStudio : Dashboard and Reports

Start from the Template

Open the template and click the ‘USE TEMPLATE’ button in the upper right corner.

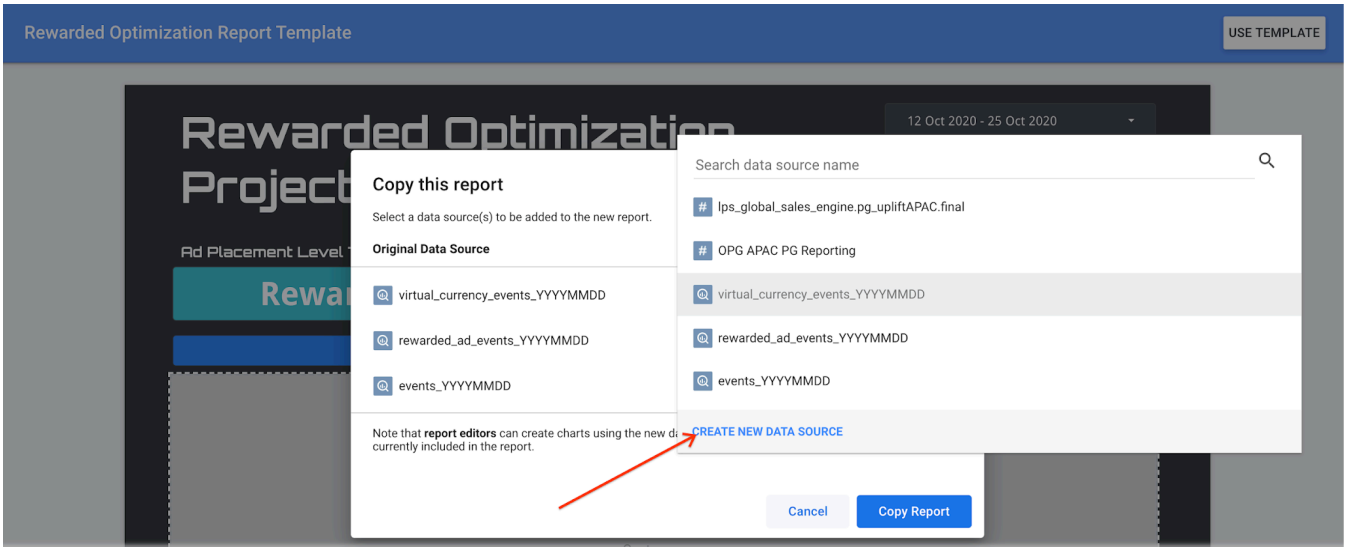
DataStudio Template :
[Link to Template]

Next, you’ll need to configure the data source to see the metrics based on your data. Three data sources need to be set in the report.

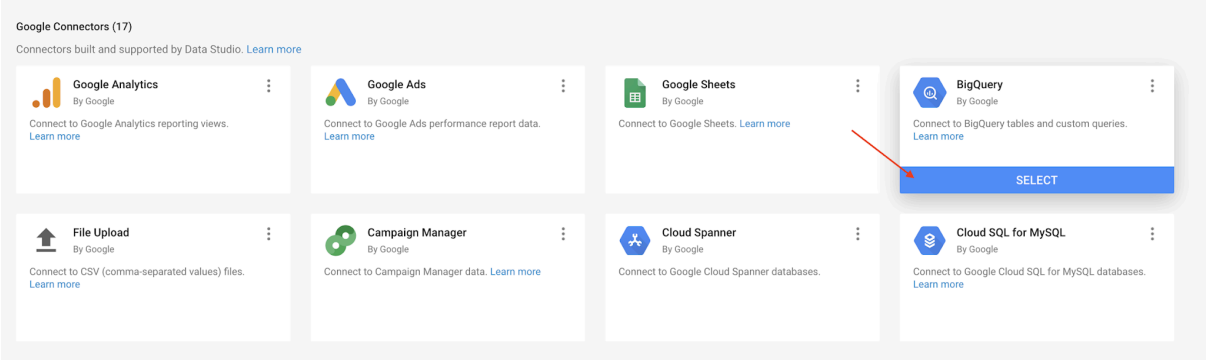


Configure virtual_currency_events_YYYYMMDD data source

Let’s start with the `virtual_currency_events_YYYYMMDD`. Click the combo box next to the data source, then click the **CREATE NEW DATA SOURCE**.



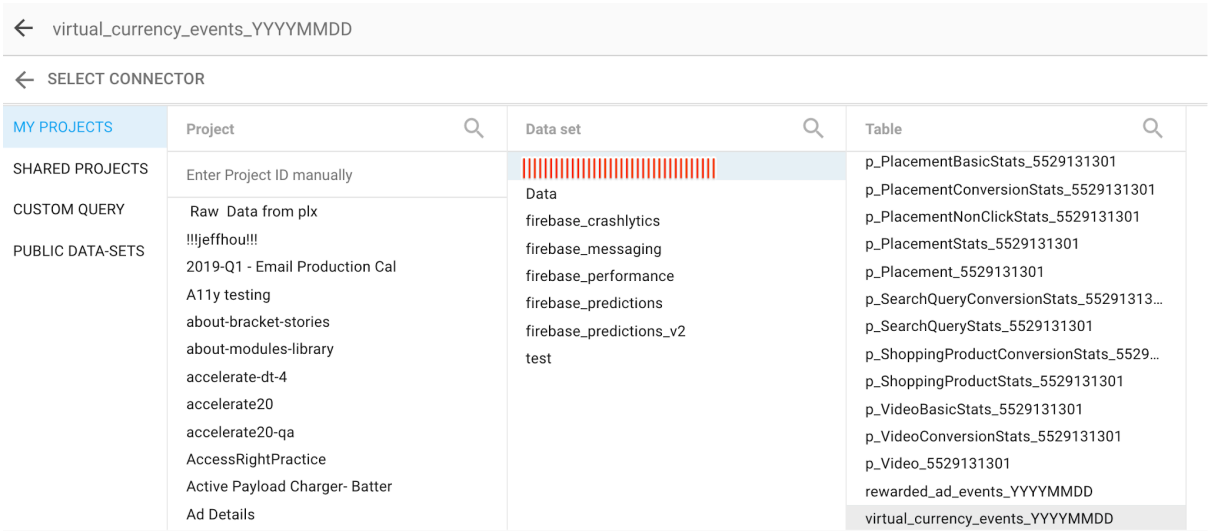
Select the BigQuery from the connectors list.



Click **MY PROJECTS**, then select the project (your Firebase project) from the list, then select the dataset.

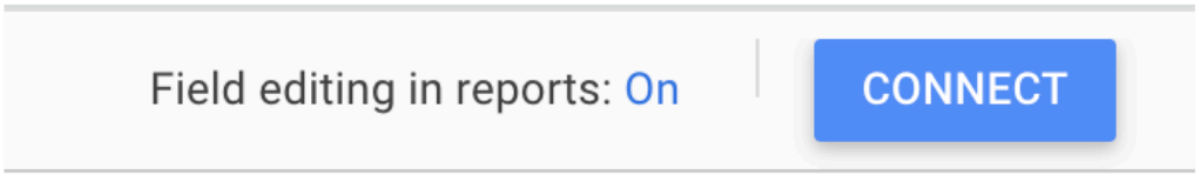
Note: If you can't find the project from the list, check the account that you're currently signed in has access to the project.

Next, select the `virtual_currency_events_YYYYMMDD` table.

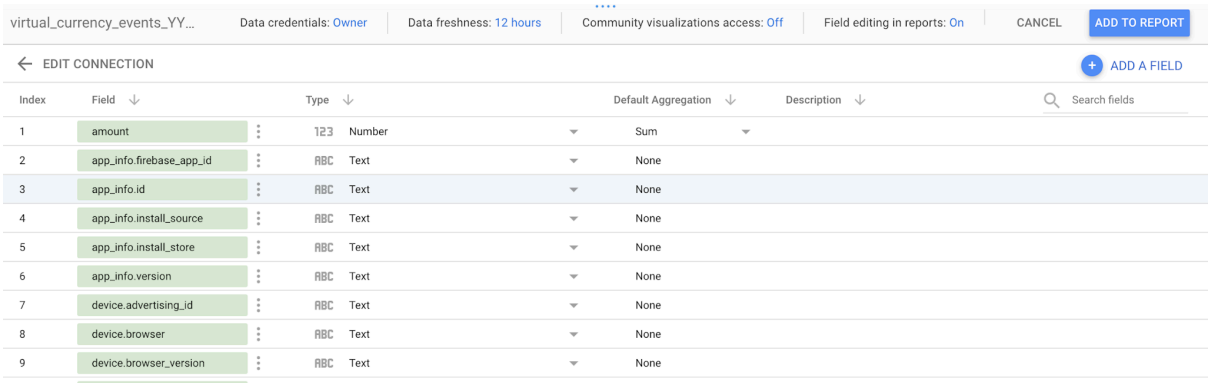


Note: It may take up to 24 hours until the `virtual_currency_events_YYYYMMDD` table is ready since you install the extension.

Click the **CONNECT** button on the top right corner to create a data source connected to the `virtual_currency_events_YYYYMMDD` table.



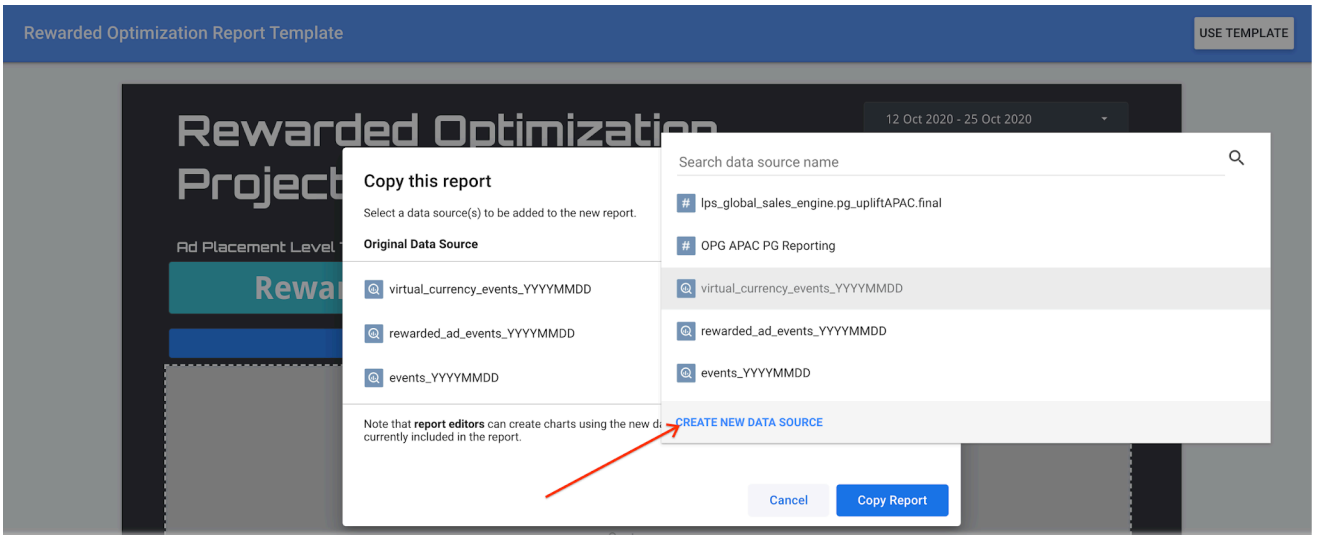
Once the data source is created, you'll see the following screen where you can configure the data source that you've just created.



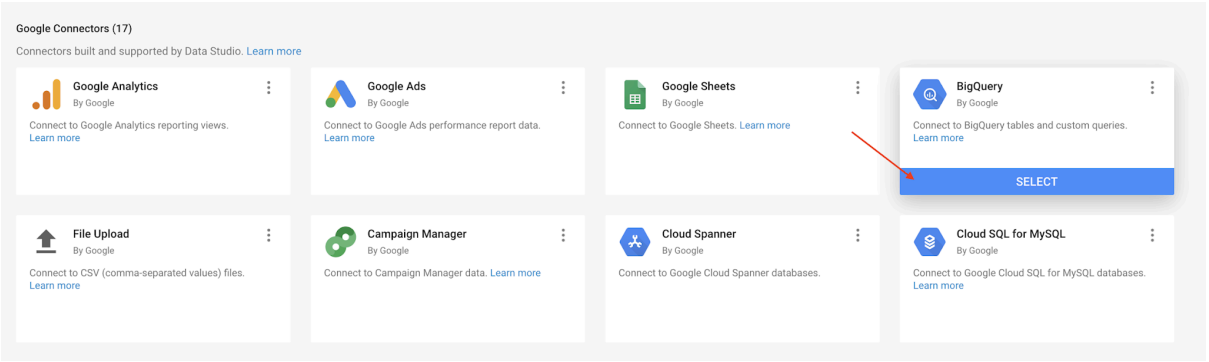
Click the **ADD TO REPORT** button to link the data source to the report.

Configure rewarded_ad_events_YYYYMMDD data source

Click the combo box next to the data source, then click the **CREATE NEW DATA SOURCE**.



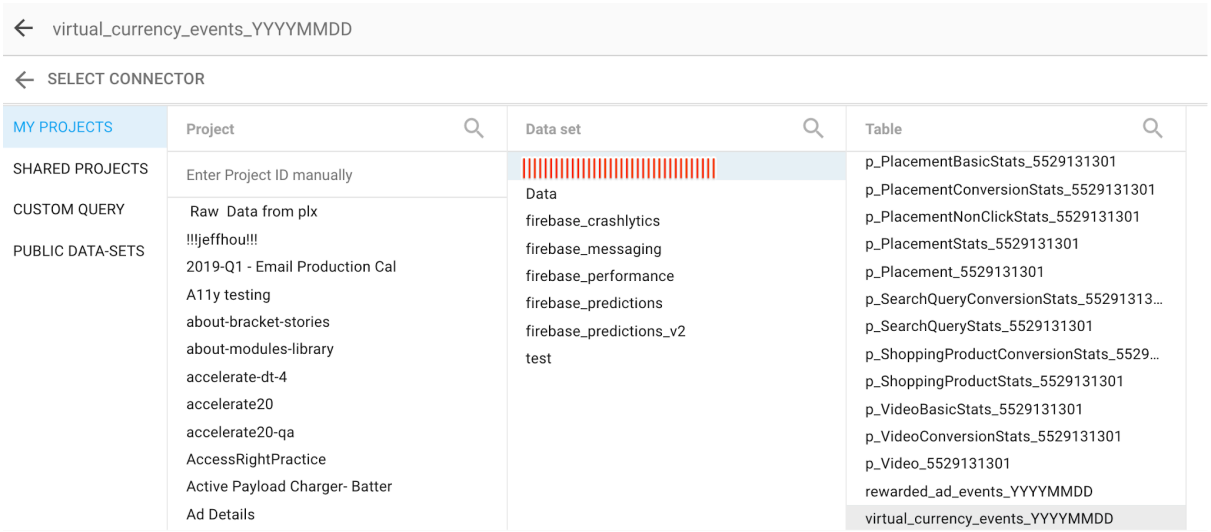
Select the BigQuery from the connectors list.



Click **MY PROJECTS**, then select the project (your Firebase project) from the list, then select the dataset.

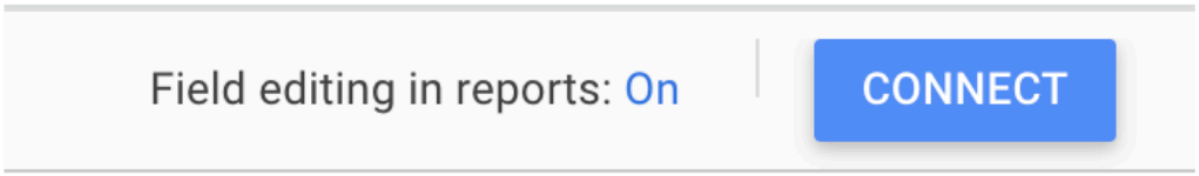
> **Note:** If you can't find the project from the list, check the account that you're currently signed in has access to the project.

Next, select the `rewarded_ad_events_YYYYMMDD` table.

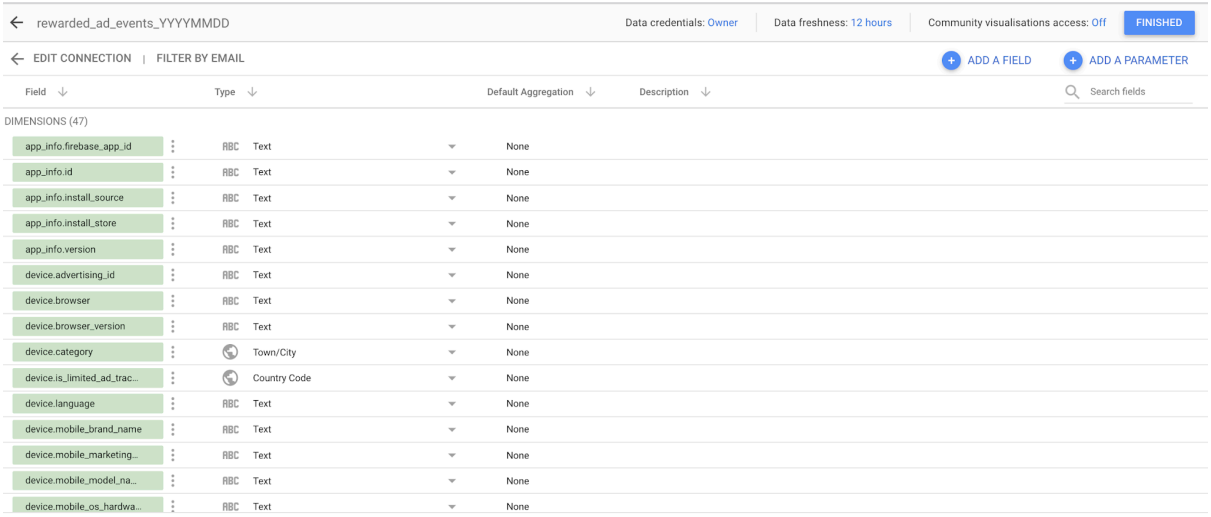


Note: It may take up to 24 hours until the `rewarded_ad_events_YYYYMMDD` table is ready since you install the extension.

Click the **CONNECT** button on the top right corner to create a data source connected to the `rewarded_ad_events_YYYYMMDD` table.



Once the data source is created, you'll see the following screen where you can configure the data source that you've just created.



You will be required to add 3 new calculated field:

1. Click the **ADD A FIELD** button at the top right corner, and create a new calculated field as follows:

- Field name: `rewarded available users`
- Formula: `case when event_name = 'rewarded_available' then user_pseudo_id else null end`

  Click the **DONE** button to add a calculated field to the data source.



2. Click the **ADD A FIELD** button at the top right corner, and create a new calculated field as follows:

- Field name: `rewarded completed users`
- Formula: `case when event_name='rewarded_completed' then user_pseudo_id else null end`

  Click the **DONE** button to add a calculated field to the data source.



3. Click the **ADD A FIELD** button at the top right corner, and create a new calculated field as follows:

- Field name: `rewarded opt-in users`
- Formula: `case when event_name = 'rewarded_opt_in' then user_pseudo_id else null end`

  Click the **DONE** button to add a calculated field to the data source.



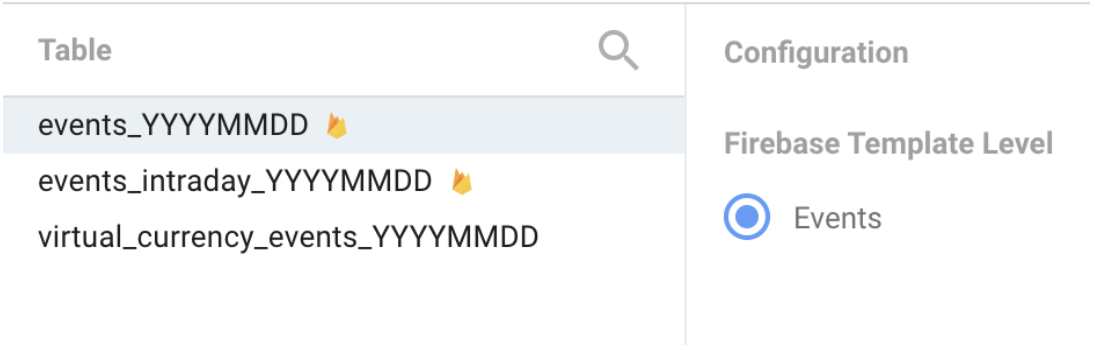
Click the **ADD TO REPORT** button to link the data source to the report.

Configure events_YYYYMMDD data source

Click the combo box next to `events_YYYYMMDD` data source, then click the **CREATE NEW DATA SOURCE**.

Select the BigQuery from the connectors list. Then, click **MY PROJECTS**, and select the project (your Firebase project) from the list and select the dataset.

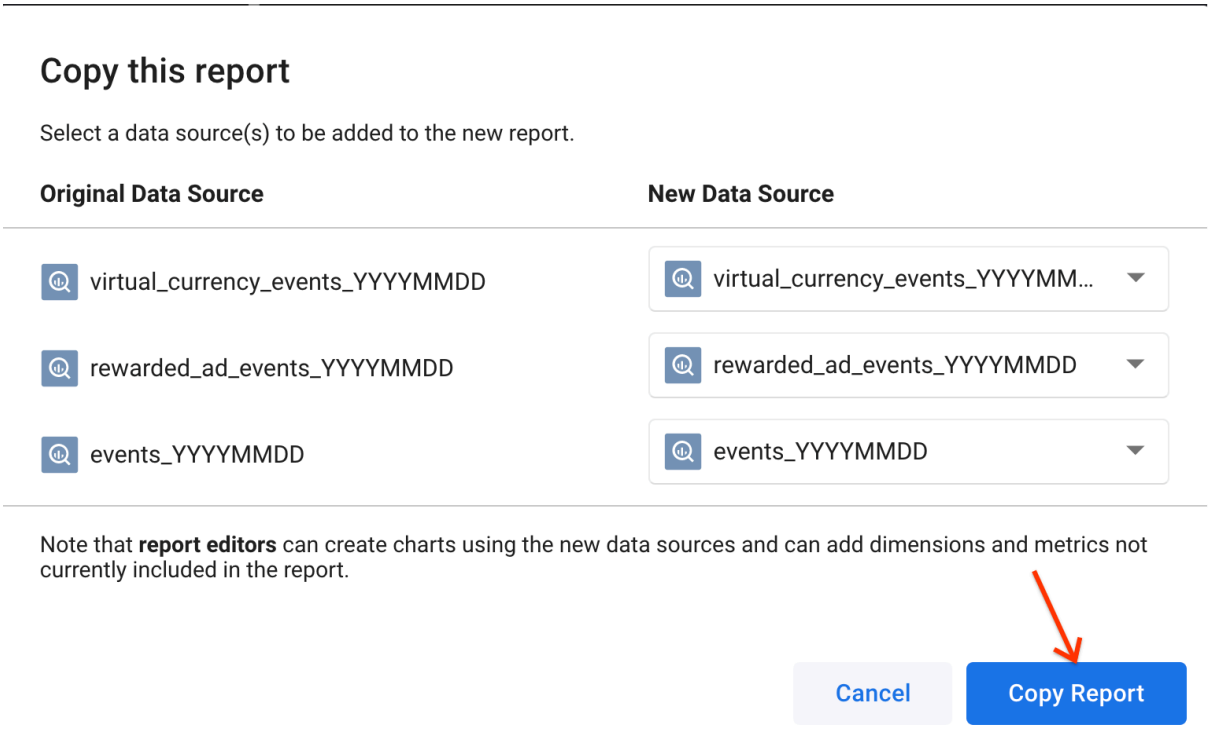
Select `events_YYYYMMDD` table, and select Events as a Firebase Template Level.



Once complete, click the **CONNECT** button on the top right corner. Then, click the **ADD TO REPORT** button to add a data source to the report.

Generate the report

Once you've completed the data source configuration, click the Copy **Report** button to generate a new report.



Once the report is created, you'll see the metrics on the report based on your app's data.