Example with Google Firebase

In this example we will implement a **Firestore** connector (firestore is a realtime nosql database by Google Firebase platform).

The idea is to demonstrate how easy and simple to create a new real-time connector using **btibConnector SDK**.

Data Model

Firebase firestore is a document based database, each document is part of a collection of documents and can contain nested collections as well.

For this example we will use 3 collections:

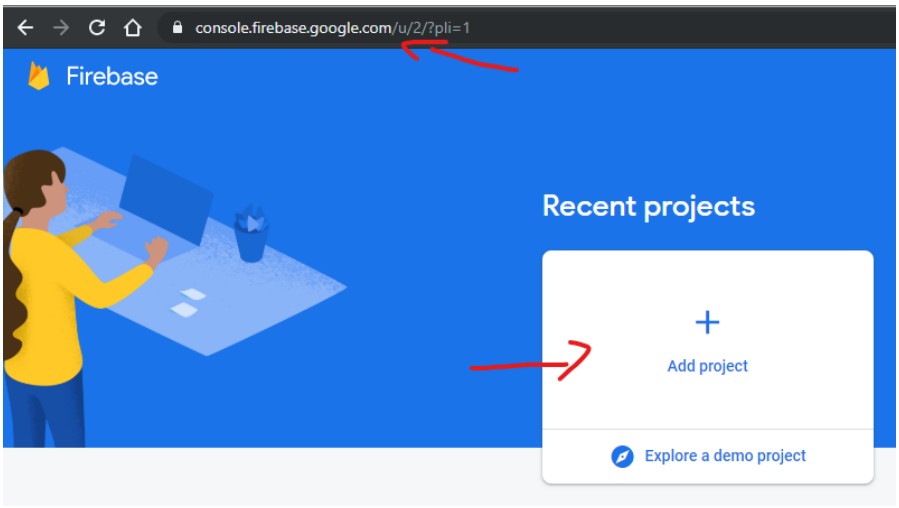
- **devices**: will contains documents that represents a Device (a container of points in Niagara). It will follow this architecture:
- **pointsValues**: The collection that hold the points values.
- **pointsStatuses**: the collection that holds the points statuses;
- **messages**: the collection where we will listen for external messages, for performance purpose we will not listen on the devices/points.
- **alarms**: the collection where the alarm recipient will send alarms data.

Let's get our hands dirty

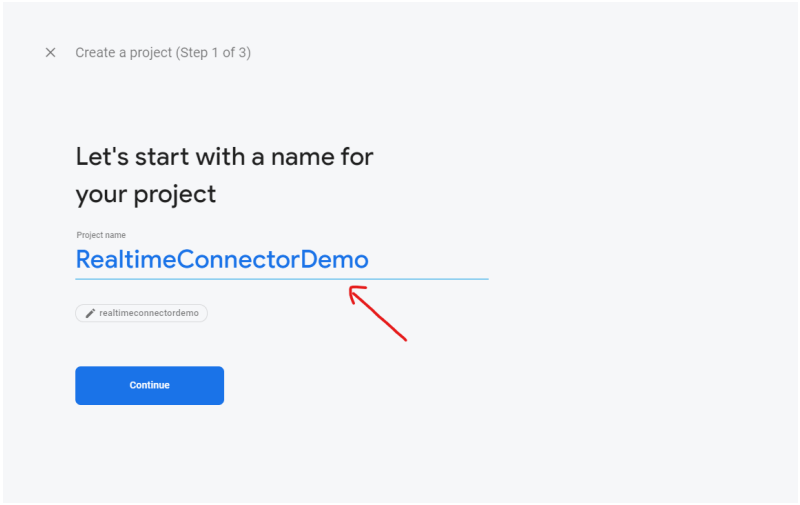
First things first:

Create a firebase project:

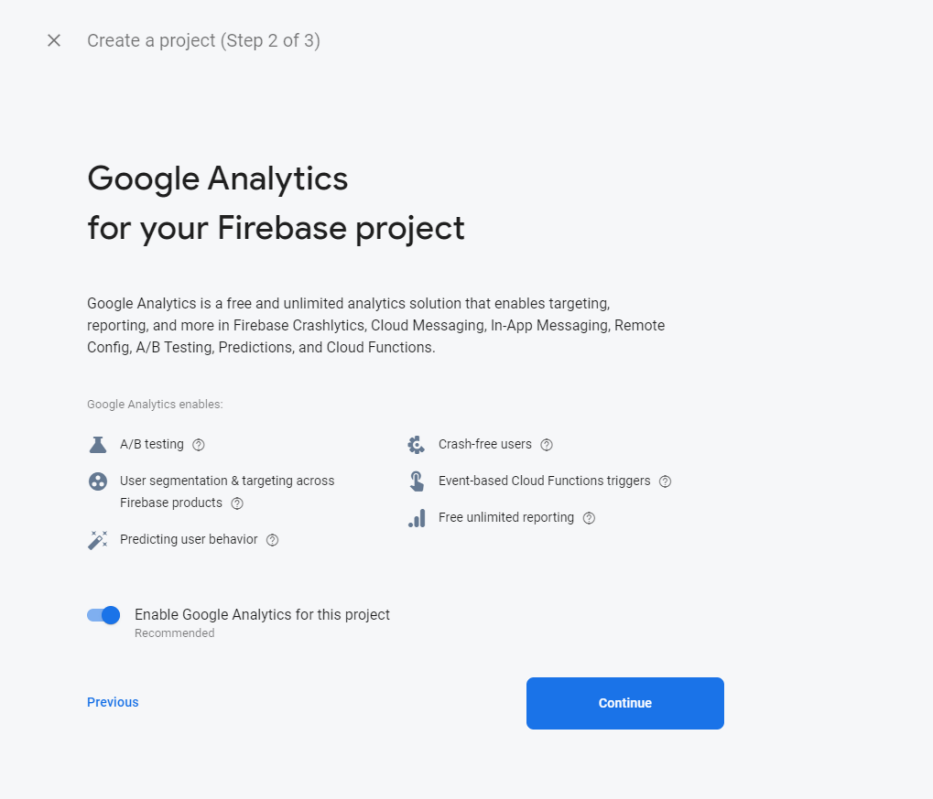
1. Create a new firebase project



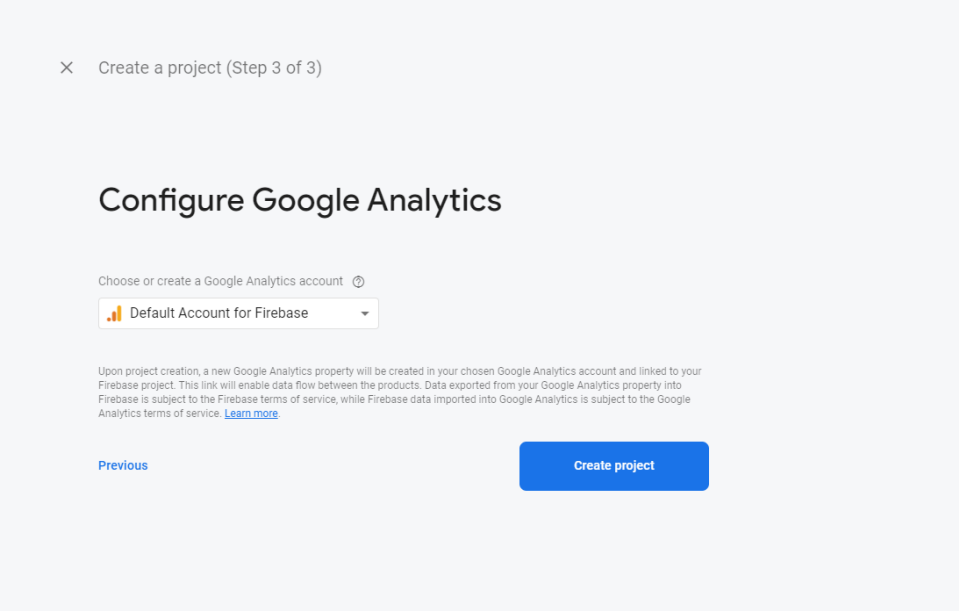
2. Give your project a name then hit **Continue**



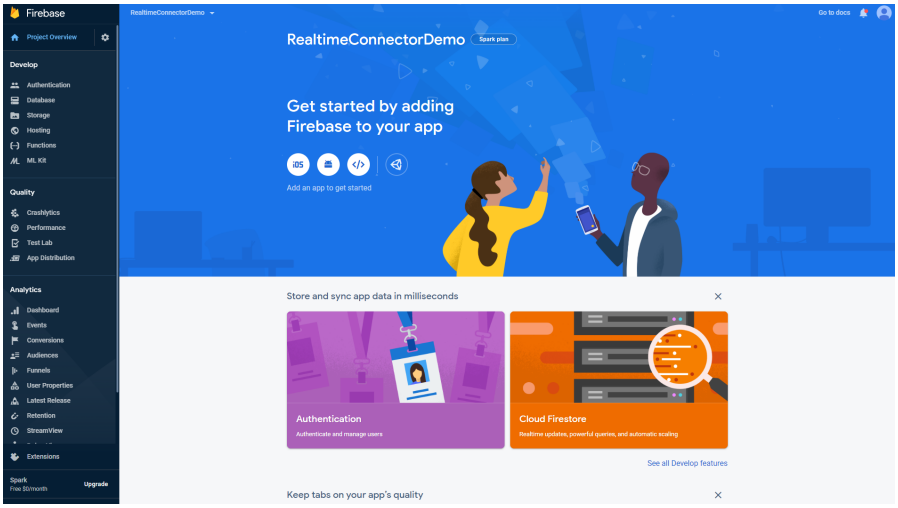
3. Leave the default settings then hit **Continue**.



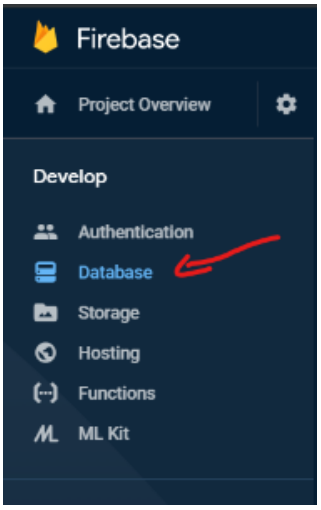
4. Then hit Create project.



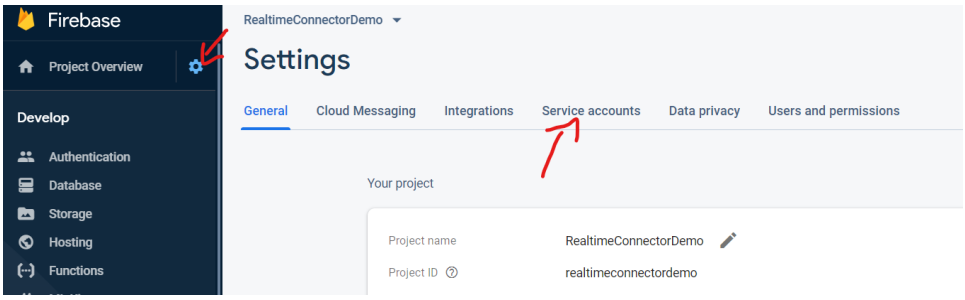
5. Then you should see this page.



6. Go to the database section and enable the firestore database.



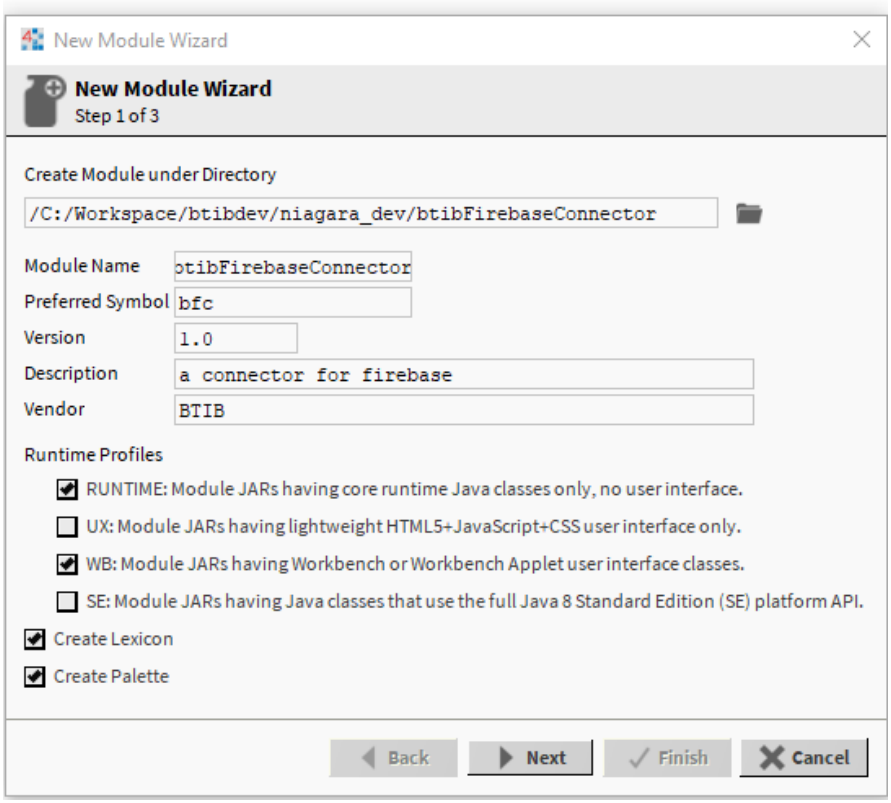
7. Then Go to the **project settings Service Accounts** and generate a new key for the connector.



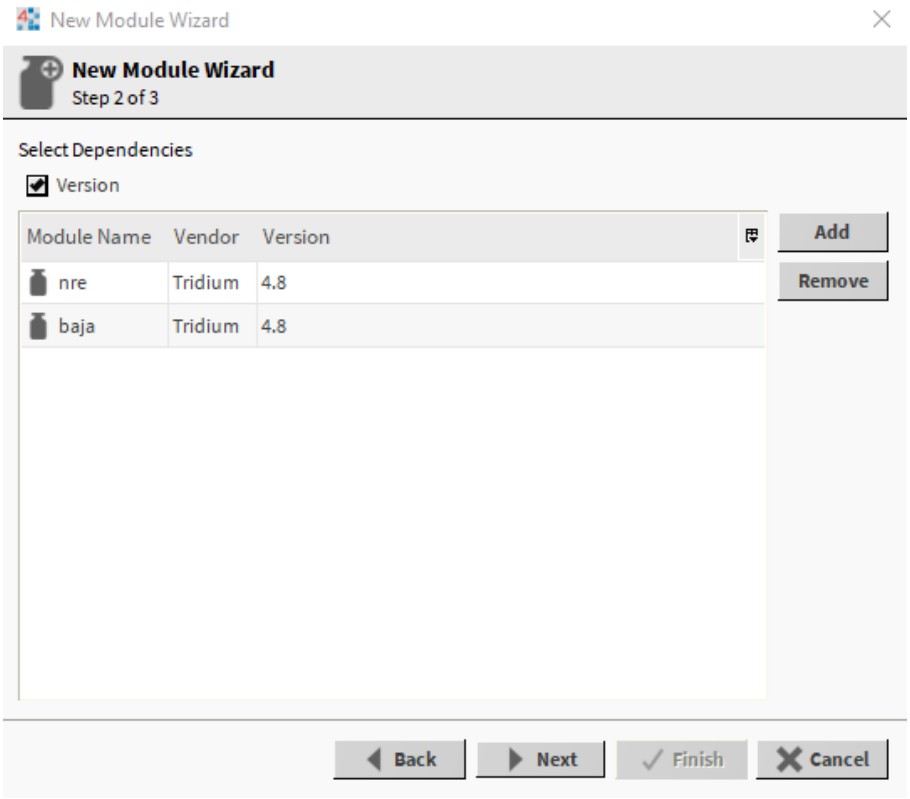
8. A JSON file will be downloaded keep it, we will use it later.

Create a niagara module:

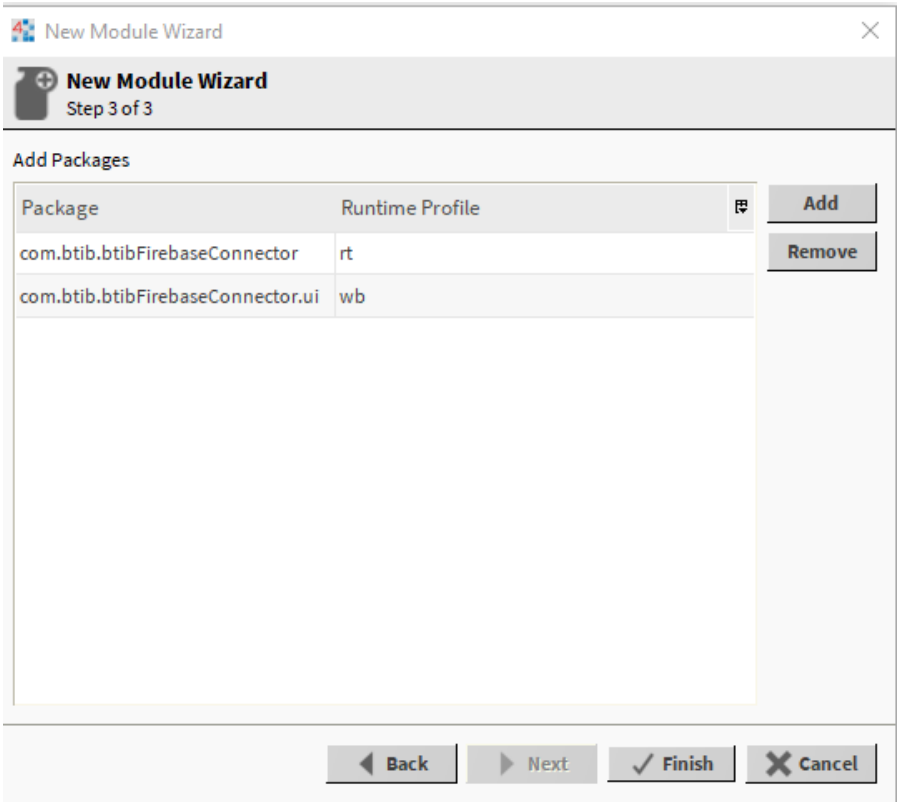
1. Go to the **Workbench Tools Create module** and fill the module information then hit **Next**.



2. Then **Next** don't worry we will add dependencies later.

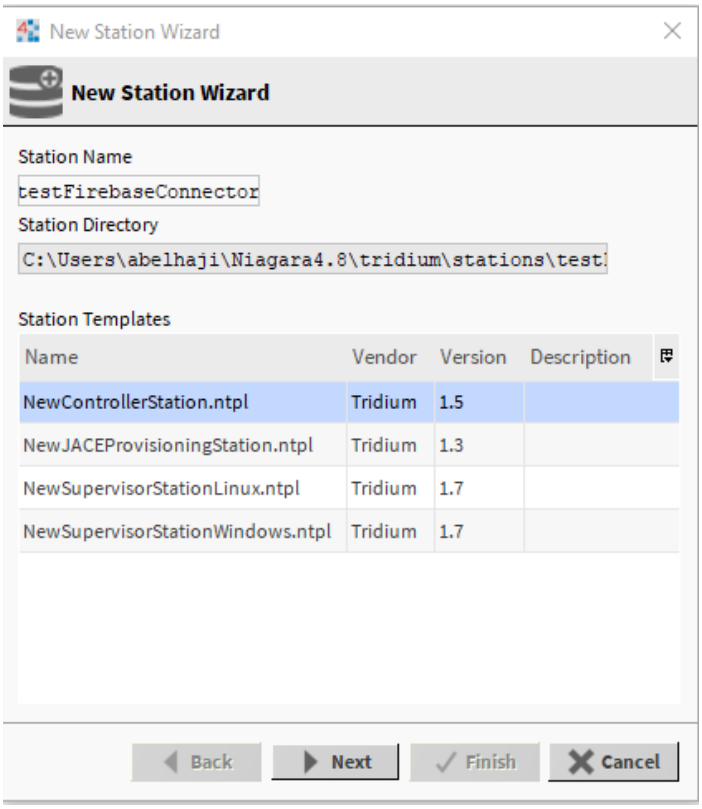


3. Then finish.

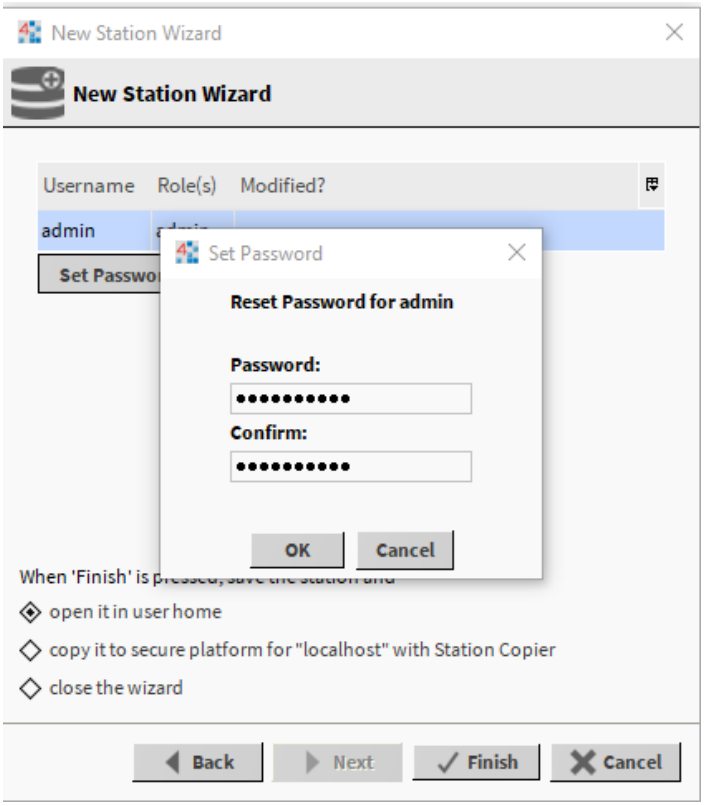


Create a testing station:

1. Go to the **Workbench Tools New Station**, Give your station a name then hit Next**.**



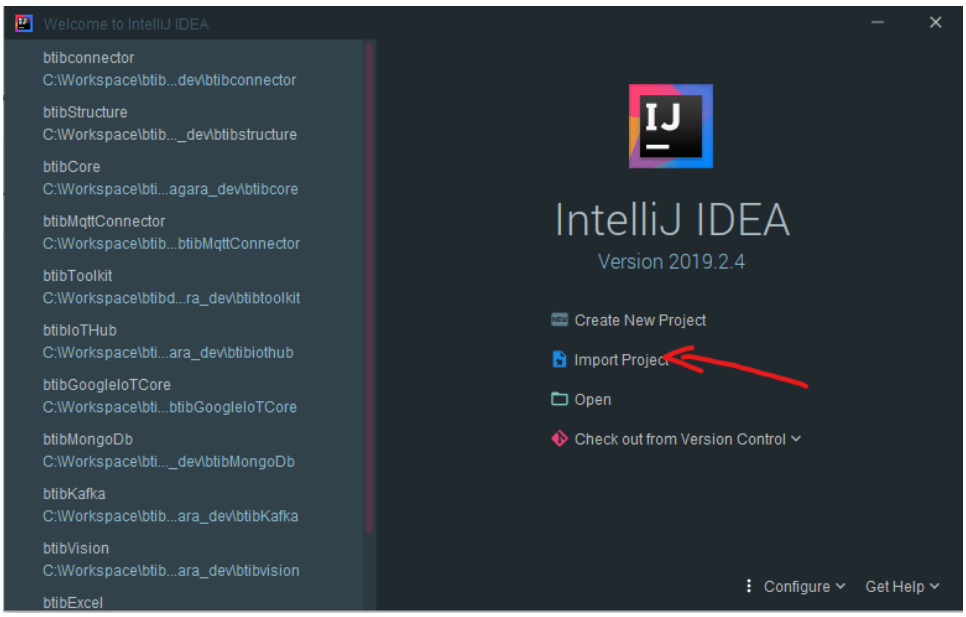
2. Set a password then hit Finish.



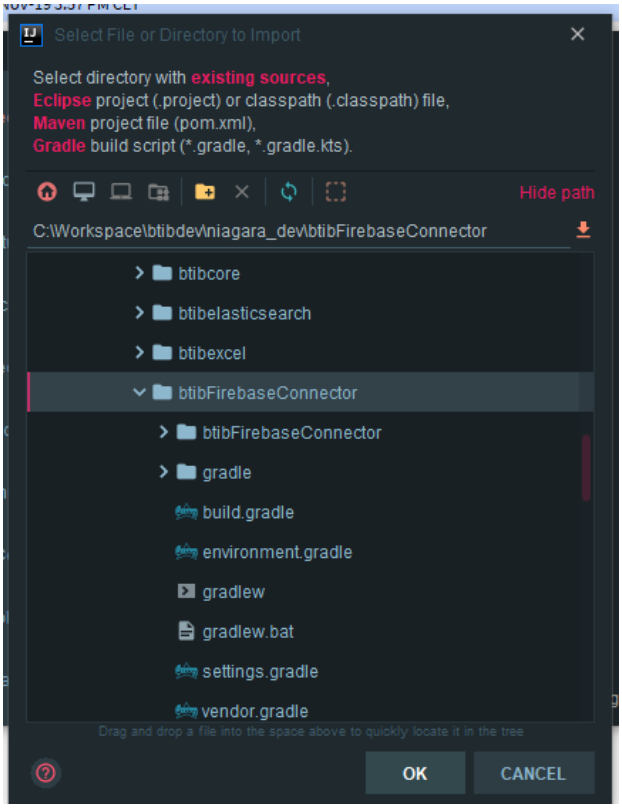
Setup the development environment:

Download and install Inteliij Idea community edition.

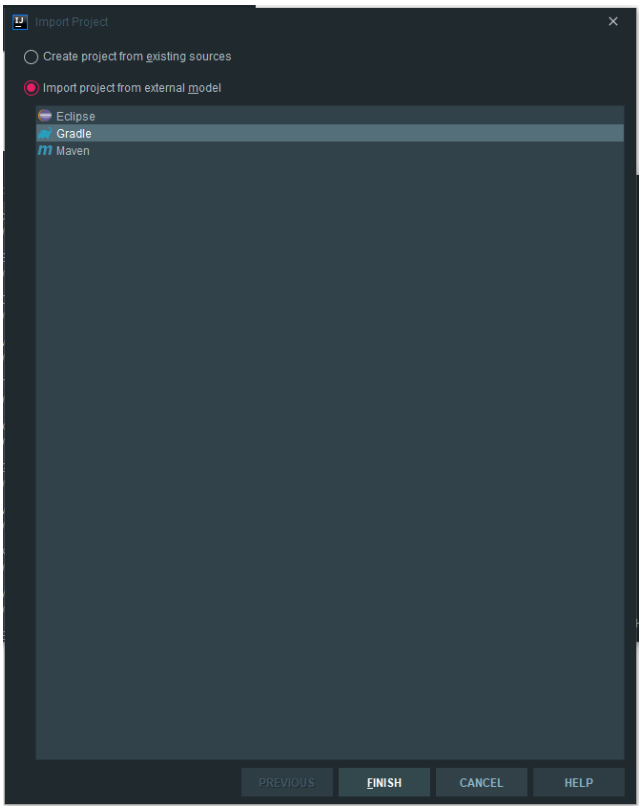
1. Open the module folder you created in **Intellij Idea.**



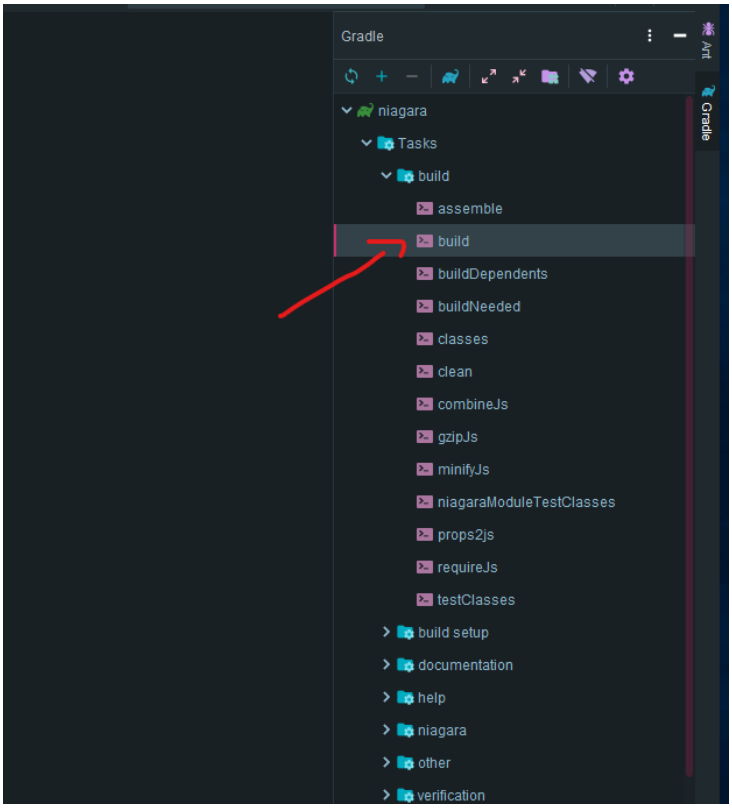
2. Choose the project folder created by the workbench.



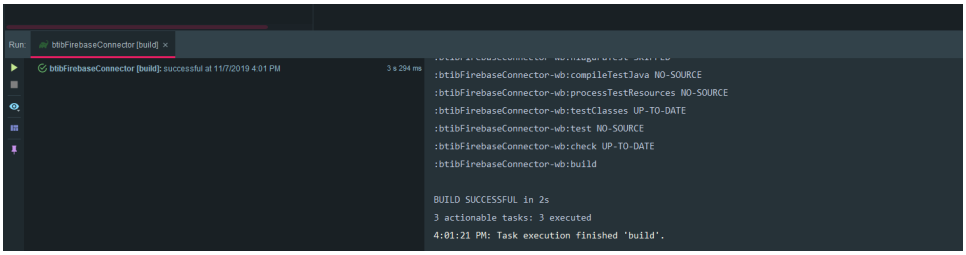
3. Then hit Ok.
4. Choose gradle as project type.



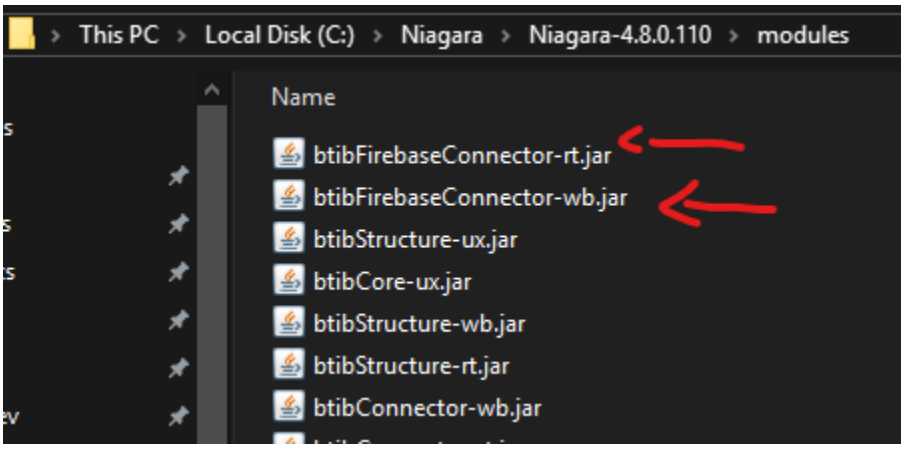
5. Then finish.
6. To test if everything is ok go to the gradle panel on the right then hit build.



7. it should succeed. If not, delete the folder and recreate the module.



8. Go to the **niagara installation folder modules** and you should see your module jars.



Now everything is setup

Start Coding:

Dependencies:

1. open the **<moduleName>-rt <moduleName>-rt.gradle** file and add these dependencies on the dependencies section.

```
// Include resources
jar {
    from("src") {
        include "resource/**/*"
    }
}
dependencies {
   // Niagara
        compile "Tridium:alarm-rt:4.6"
        compile "Tridium:baja:4.6"
        compile "Tridium:control-rt:4.6"
        compile "Tridium:nre:4.6"
        // btib
        compile "BTIB:btibConnector-rt:46"
        compile "BTIB:btibCore-rt:46"
        // external
        uberjar 'com.google.firebase:firebase-admin:6.0.0'
}
```

2. Make sure that your using **uberjar** for external dependencies.

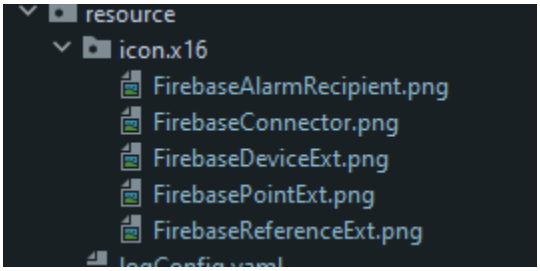
Permissions:

1. Open the **<moduleName>-rt module-permissions.xml** module-permissions.xml.

```
<permissions>
    <niagara-permission-groups type="all">
        <req-permission>
            <name>NETWORK_COMMUNICATION</name>
            <purposeKey>Outside access for the connector</purposeKey>
            <parameters>
                <parameter name="hosts" value="*"/>
                <parameter name="ports" value="*"/>
                <parameter name="proxySelector" value="get"/>
            </parameters>
        </req-permission>
        <req-permission>
            <name>GET_ENVIRONMENT_VARIABLES</name>
            <purposeKey>The firebase needs access to environment variables</purposeKey>
            <parameters>
                <parameter name="variables" value="*"/>
            </parameters>
        </req-permission>
        <req-permission>
            <name>LOAD_LIBRARIES</name>
            <purposeKey>The firebase needs to load native libs for grpc calls</purposeKey>
            <parameters>
                <parameter name="libraries" value="*"/>
            </parameters>
        </req-permission>
        <req-permission>
            <name>MANAGE_EXECUTION</name>
            <purposeKey>The connector at the base of structure must handle its own threads.</purposeKey>
        </req-permission>
        <req-permission>
            <name>REFLECTION</name>
            <purposeKey>Used by firebase SDK to register points and devices</purposeKey>
        </req-permission>
    </niagara-permission-groups>
</permissions>
```

Icons:

1. Copy these icons to the **resource/icon/x16** folder



Connector:

1. Create a data destination enum.

```
@NiagaraType
@NiagaraEnum(
    range = {
        @Range("alarmsCollection"),
        @Range("devicesCollection"),
        @Range("pointsStatusCollection"),
        @Range("pointsValuesCollection")
    }
)
public final class BDataDestinationEnum extends BFrozenEnumBtib
{
    /*+ ------------ BEGIN BAJA AUTO GENERATED CODE ------------ +*/
    /*@ $fr.btib.firebase.connector.BDataDestinationEnum(3747253359)1.0$ @*/
    /* Generated Mon Aug 24 17:20:12 CEST 2020 by Slot-o-Matic (c) Tridium, Inc. 2012 */

    /**
     * Ordinal value for alarmsCollection.
     */
    public static final int ALARMS_COLLECTION = 0;
    /**
     * Ordinal value for devicesCollection.
     */
    public static final int DEVICES_COLLECTION = 1;
    /**
     * Ordinal value for pointsStatusCollection.
     */
    public static final int POINTS_STATUS_COLLECTION = 2;
    /**
     * Ordinal value for pointsValuesCollection.
     */
    public static final int POINTS_VALUES_COLLECTION = 3;

    /**
     * BDataDestinationEnum constant for alarmsCollection.
     */
    public static final BDataDestinationEnum alarmsCollection = new BDataDestinationEnum
(ALARMS_COLLECTION);
    /**
     * BDataDestinationEnum constant for devicesCollection.
     */
    public static final BDataDestinationEnum devicesCollection = new BDataDestinationEnum
(DEVICES_COLLECTION);
    /**
     * BDataDestinationEnum constant for pointsStatusCollection.
     */
    public static final BDataDestinationEnum pointsStatusCollection = new BDataDestinationEnum
(POINTS_STATUS_COLLECTION);
    /**
     * BDataDestinationEnum constant for pointsValuesCollection.
     */
    public static final BDataDestinationEnum pointsValuesCollection = new BDataDestinationEnum
(POINTS_VALUES_COLLECTION);
```

```
    /**
     * Factory method with ordinal.
     */
    public static BDataDestinationEnum make(int ordinal)
    {
        return (BDataDestinationEnum) alarmsCollection.getRange().get(ordinal, false);
    }

    /**
     * Factory method with tag.
     */
    public static BDataDestinationEnum make(String tag)
    {
        return (BDataDestinationEnum) alarmsCollection.getRange().get(tag);
    }

    /**
     * Private constructor.
     */
    private BDataDestinationEnum(int ordinal)
    {
        super(ordinal);
    }

    public static final BDataDestinationEnum DEFAULT = alarmsCollection;

    ////////////////////////////////////////////////////////////////
    // Type
    ////////////////////////////////////////////////////////////////

    @Override
    public Type getType()
    {
        return TYPE;
    }

    public static final Type TYPE = Sys.loadType(BDataDestinationEnum.class);

    /*+ ------------ END BAJA AUTO GENERATED CODE -------------- +*/
}
```

2. Create a new class for the connector that extends the **BRealtimeConnector** class, then run the slotomatic task.
3. Add a property for the firebase credentials, then run slotomatic.

```
@NiagaraType
@NiagaraProperty(
    name = "firebaseKeyJson",
    type = "String",
    defaultValue = ""
)
public class BFirebaseConnector extends BRealtimeConnector
{ ...
```

4. Implement connector methods.
    a. Add these fields

```
/*+ ------------ END BAJA AUTO GENERATED CODE -------------- +*/
// the logger
private static final BtibLogger LOG = BtibLogger.getLogger(TYPE);
// Use the icon on the icon resource
private static final BIcon ICON = BtibIconTool.getComponentIcon(TYPE);
// the firebase url
private static final String DATABASE_URL = "https://realtimeconnectordemo.firebaseio.com";
private static final String DEVICES_COLLECTION = "devices";
private static final String ALARMS_COLLECTION = "alarms";
private static final String POINTS_COLLECTION = "points";
```

```
private static final String MESSAGES_COLLECTION = "messages";
// the workers thread pool
private final ExecutorService executorService = AccessController.doPrivileged
((PrivilegedAction<ExecutorService>) Executors::newCachedThreadPool);
// Devices incoming messages listeners
private final Map<String, BIIncomingMessageListener> listeners = new HashMap<>();
private FirebaseApp firebaseApp = null;
private Firestore firestore = null;
private ListenerRegistration messageListenerRegistration;
```

b. Add private helper methods that we going to use in other implimentations.

```
/**
 * Gets data destination
 *
 * @param slot
 * @return
 */
private String getDestination(String slot)
{
    BDataDestinationEnum tagsDestination = (BDataDestinationEnum) this.getConfigSlot(slot);
    if (tagsDestination == BDataDestinationEnum.pointsValuesCollection)
    {
        return POINTS_VALUES_COLLECTION;
    }
    if (tagsDestination == BDataDestinationEnum.pointsStatusCollection)
    {
        return POINTS_STATUSES_COLLECTION;
    }
    if (tagsDestination == BDataDestinationEnum.devicesCollection)
    {
        return DEVICES_COLLECTION;
    }
    if (tagsDestination == BDataDestinationEnum.alarmsCollection)
    {
        return ALARMS_COLLECTION;
    }
    return "unknown";
}

/**
 * Remove null values
 *
 * @param data
 * @param <T>
 */
private <T> void clearNulls(Map<String, T> data)
{
    List<String> nullKeys = data.entrySet().stream()
        .filter(pair -> pair.getValue() == null)
        .map(Map.Entry::getKey)
        .collect(Collectors.toList());

    nullKeys.forEach(data::remove);
}

/**
 * Create the firestore client   lazily
 *
 * @return
 * @throws Exception
 */
private Firestore getFirestore() throws Exception
{
    if (this.firestore == null)
    {
        if (this.firebaseApp == null)
        {
            this.firebaseApp = this.initializeFirebase();
        }
```

```
        if (this.messageListenerRegistration != null)
        {
            this.messageListenerRegistration.remove();
        }
        this.firestore = AccessController.doPrivileged((PrivilegedAction<Firestore>) () -> {
            try
            {
                return FirestoreClient.getFirestore();
            }
            catch (Exception e)
            {
                e.printStackTrace();
                return null;
            }
        });
        this.listenForMessages();
    }
    return this.firestore;
}

/**
 * Start listening for messages
 */
private void listenForMessages()
{
    this.messageListenerRegistration = this.firestore.collection(MESSAGES_COLLECTION).
addSnapshotListener(new EventListener<QuerySnapshot>()
    {
        @Override
        public void onEvent(@Nullable QuerySnapshot snapshots, @Nullable FirestoreException error)
        {
            if (snapshots != null)
            {
                snapshots.getDocumentChanges().forEach(document -> {
                    try
                    {
                        String jsonMessage = new JSONObject(document.getDocument().getData()).
toString();
                        List<IIncomingMessage> messages = BFirebaseConnector.this.
getMessageExtractor().getMessages(jsonMessage);
                        for (IIncomingMessage message : messages)
                        {
                            if (message.getDeviceId() == null)
                            {
                                BFirebaseConnector.this.getBLog().copy().eventName
("IncomingMessage").failed().message("Device id is required for message: " + message.toString()).
send();
                            }
                            if (BFirebaseConnector.this.listeners.containsKey(message.
getDeviceId()))
                            {
                                BFirebaseConnector.this.listeners.get(message.getDeviceId()).
forEach(listener -> listener.handleIncomingMessage(message));
                            }
                        }
                    }
                    catch (Exception e)
                    {
                        LOG.severe(e.getMessage());
                    }
                });
            }
        }
    });
}

/**
 * Init the firebase application instance
 *
 * @return
 * @throws Exception
```

```
 */
private FirebaseApp initializeFirebase() throws Exception
{
    AtomicReference<Exception> innerException = new AtomicReference<>();
    FirebaseApp app = AccessController.doPrivileged((PrivilegedAction<FirebaseApp>) () -> {
        try
        {
            ByteArrayInputStream credentialsStream = new ByteArrayInputStream(this.
getFirebaseKeyJson().getBytes());
            FirebaseOptions options = new FirebaseOptions.Builder()
                .setCredentials(GoogleCredentials.fromStream(credentialsStream))
                .setDatabaseUrl(DATABASE_URL)
                .build();
            return FirebaseApp.initializeApp(options);
        }
        catch (Exception exception)
        {
            innerException.set(exception);
            return null;
        }
    });
    if (innerException.get() != null)
    {
        throw innerException.get();
    }
    return app;
}

/**
 * Submit the task an returns the future result
 *
 * @param runnableThrowable
 * @return
 */
private CompletableFuture<Void> doAsync(RunnableThrowable runnableThrowable)
{
    CompletableFuture<Void> future = new CompletableFuture<>();
    if (!this.getEnabled())
    {
        future.completeExceptionally(new ConnectorDisabledException());
    }
    this.executorService.submit(() -> {
        try
        {
            AtomicReference<Exception> exceptionAtomicReference = new AtomicReference<>();
            AccessController.doPrivileged((PrivilegedAction<Object>) () -> {
                try
                {
                    runnableThrowable.run();
                }
                catch (Exception e)
                {
                    exceptionAtomicReference.set(e);
                }
                return null;
            });
            if (exceptionAtomicReference.get() != null)
            {
                throw exceptionAtomicReference.get();
            }
            future.complete(null);
        }
        catch (Exception e)
        {
            future.completeExceptionally(e);
        }
    });
    return future;
}
```

c. Implements these methods

```
////////////////////////////////////////////////////////////////
// BRealtimeConnector
////////////////////////////////////////////////////////////////

@Override
public BFrozenEnum getDeviceTagsDestination()
{
    return BDataDestinationEnum.devicesCollection;
}

@Override
public BFrozenEnum getDeviceAlarmDestination()
{
    return BDataDestinationEnum.alarmsCollection;
}

@Override
public BFrozenEnum getPointTagsDestination()
{
    return BDataDestinationEnum.pointsValuesCollection;
}

@Override
public BFrozenEnum getPointStatusDestination()
{
    return BDataDestinationEnum.pointsStatusCollection;
}

@Override
public BFrozenEnum getPointValueDestination()
{
    return BDataDestinationEnum.pointsValuesCollection;
}

@Override
public ExecutorService getRegistryExecutorService()
{
    return this.executorService;
}

@Override
public ExecutorService getConnectionExecutorService()
{
    return this.executorService;
}


////////////////////////////////////////////////////////////////
// BIRealtimeConnector
////////////////////////////////////////////////////////////////

@Override
public Class<? extends BIRealtimeDeviceExtension> getDeviceExtensionClass()
{
    return BFirebaseDeviceExt.class;
}

@Override
public Class<? extends BIRealtimePointExtension> getReferenceExtensionClass()
{
    return BFirebaseReferenceExt.class;
}

@Override
public Class<? extends BIRealtimeDeviceExtension> getAlarmRecipientExtensionClass()
{
    return BFirebaseAlarmRecipient.class;
}
```

```
@Override
public Class<? extends BIRealtimePointExtension> getPointExtensionClass()
{
    return BFirebasePointExt.class;
}
```

d. Implements doPing.

```
@Override
public void doPing()
{
    this.setLastAttempt(BAbsTime.now());
    this.pingService()
        .exceptionally(e -> {
            CompTool.setFault(this, e.getMessage(), e, LOG);
            this.setLastFailure(BAbsTime.now());
            return null;
        })
        .thenRun(() -> {
            CompTool.setOk(this);
            this.setLastSuccess(BAbsTime.now());
        });
}
```

e. Implements pingService

```
@Override
public CompletableFuture<Void> pingService()
{
    return this.doAsync(() -> this.getFirestore().getCollections().forEach(CollectionReference::
getId));
}
```

f. Implements registerDevice.

```
@Override
public CompletableFuture<Void> registerDevice(String deviceId)
{
    return this.doAsync(() -> this.getFirestore().collection(DEVICES_COLLECTION).document
(deviceId).set(new HashMap<>()).get());
}
```

g. Implements registerPoint.

```
@Override
public CompletableFuture<Void> registerPoint(String deviceId, String pointId)
{
    return this.doAsync(() -> {
        this.getFirestore().collection(POINTS_VALUES_COLLECTION).document(pointId).set(new
HashMap<>()).get();
        this.getFirestore().collection(POINTS_STATUSES_COLLECTION).document(pointId).set(new
HashMap<>()).get();
    });
}
```

h. Implements unregisterDevice.

```
@Override
public CompletableFuture<Void> unregisterDevice(String deviceId)
{
    return this.doAsync(() -> {
        this.getFirestore().collection(DEVICES_COLLECTION).document(deviceId).delete().get();
```

```
    });
}
```

i. Implements unregisterPoint.

```
@Override
public CompletableFuture<Void> unregisterPoint(String deviceId, String pointId)
{
    return this.doAsync(() -> {
        this.getFirestore().collection(POINTS_VALUES_COLLECTION).document(pointId).delete().get();
        this.getFirestore().collection(POINTS_STATUSES_COLLECTION).document(pointId).delete().
get();
    });
}
```

j. Implements openConnectionForDevice.

```
@Override
public CompletableFuture<Void> openConnectionForDevice(String deviceId)
{
    return this.doAsync(this::getFirestore);
}
```

k. Implements closeConnectionForDevice.

```
@Override
public CompletableFuture<Void> closeConnectionForDevice(String deviceId)
{
    // Don't need to close the firebase connection
    return CompletableFuture.completedFuture(null);
}
```

l. Implements isDeviceConnected.

```
@Override
public boolean isDeviceConnected(String deviceId)
{
    return this.firestore != null;
}
```

m. Implements sendMessage.

```
@Override
public CompletableFuture<Void> sendMessage(String deviceId, OutgoingPointMessage message)
{
    return this.doAsync(() -> {
        JSONObject jsonObject = new JSONObject(message.getMessageString());
        if (!jsonObject.has(POINT_ID_VARIABLE))
        {
            throw new Exception("the pointId field is required in the json message");
        }
        String pointId = jsonObject.getString(POINT_ID_VARIABLE);

        Map<String, Object> data = new HashMap<>();
        jsonObject.keys().forEachRemaining(key -> data.put((String) key, jsonObject.get((String)
key)));
        this.getFirestore().collection(this.getDestination(POINT_VALUE_DESTINATION_SLOT)).document
(pointId).update(data).get();
    });
}
```

n. Implement sendState.

```
@Override
public CompletableFuture<Void> sendState(String deviceId, OutgoingPointMessage message)
{
    return this.doAsync(() -> {
        JSONObject jsonObject = new JSONObject(message.getMessageString());
        if (!jsonObject.has(POINT_ID_VARIABLE))
        {
            throw new Exception("the pointId field is required in the json message");
        }
        String pointId = jsonObject.getString(POINT_ID_VARIABLE);

        Map<String, Object> data = new HashMap<>();
        jsonObject.keys().forEachRemaining(key -> data.put((String) key, jsonObject.get((String)
key)));
        this.getFirestore().collection(this.getDestination(POINT_STATUS_DESTINATION_SLOT)).document
(pointId).update(data).get();
    });
}
```

o. Implelement sendAlarm.

```
@Override
public CompletableFuture<Void> sendAlarm(String deviceId, OutgoingPointMessage message)
{
    return this.doAsync(() -> {
        JSONObject jsonObject = new JSONObject(message.getMessageString());
        Map<String, Object> data = new HashMap<>();
        jsonObject.keys().forEachRemaining(key -> data.put((String) key, jsonObject.get((String)
key)));
        this.getFirestore().collection(getDestination(DEVICE_ALARM_DESTINATION_SLOT)).add(data).
get();
    });
}
```

p. Implements sendTagsForDevice.

```
@Override
public CompletableFuture<Void> sendTagsForDevice(String deviceId, Map<String, String> tags)
{
    return this.doAsync(() -> {
        this.clearNulls(tags);
        Map<String, Object> fields = new HashMap<>();
        fields.put("tags", tags);
        this.getFirestore().collection(DEVICES_COLLECTION).document(deviceId).set(fields).get();
    });
}
```

q. Implements sendTagsForPoint.

```
@Override
public CompletableFuture<Void> sendTagsForPoint(String deviceId, String pointId, Map<String,
String> tags)
{
    return this.doAsync(() -> {
        this.clearNulls(tags);
        Map<String, Object> fields = new HashMap<>();
        fields.put("tags", tags);
        this.getFirestore().collection(this.getDestination(POINT_TAGS_DESTINATION_SLOT)).document
(pointId).update(fields).get();
    });
}
```

r. Implements subscribeToIncomingMessages.

```
@Override
public void subscribeToIncomingMessages(String deviceId, BIIncomingMessageListener
incomingMessageListener) throws Exception
{
    if (!this.listeners.containsKey(deviceId))
    {
        this.listeners.put(deviceId, new HashSet<>());
    }
    this.listeners.get(deviceId).add(incomingMessageListener);
}
```

s. Implements unsubscribeToIncomingMessages.

```
@Override
public void unsubscribeToIncomingMessages(String deviceId, BIIncomingMessageListener
incomingMessageListener)
{
    this.listeners.remove(deviceId);
}
```

t. Implements getLogger.

```
@Override
public BtibLogger getBtibLogger()
{
    return LOG;
}
```

5. Override life cycle methods.
   a. started.

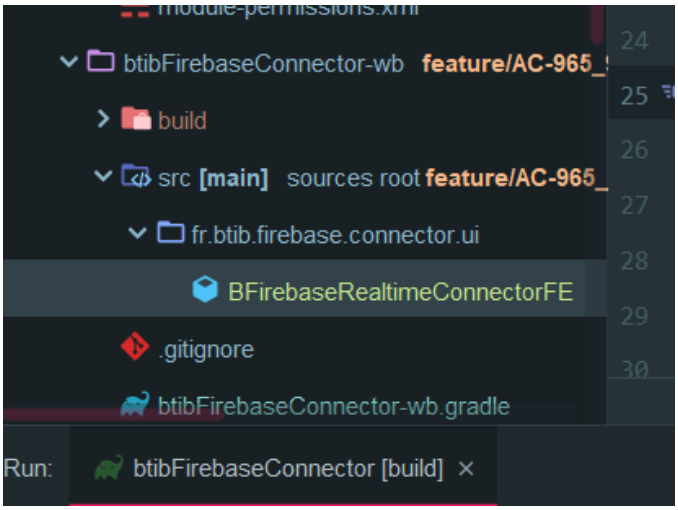
```
@Override
public void started() throws Exception
{
    super.started();
    // check if the station is already started when you drag and drop the component for the first
time
    if (!Sys.isStationStarted())
    {
        // When station is booting
        this.firebaseApp = this.initializeFirebase();
    }
}
```

   b. changed.

```
@Override
public void changed(Property property, Context context)
{
    if (!this.isRunning())
    {
        return;
    }

    if (property == firebaseKeyJson)
    {
        try
        {
            this.firestore = null;
            this.firebaseApp = this.initializeFirebase();
        }
        catch (Exception e)
        {
            e.printStackTrace();
```

```
        }
    }
}
```

6. Implements the device extension.
    a. Create a new class in the wb module.



    b. Add dependencies to the **<moduleName>-wb.gradle**

```
dependencies {
         compile "Tridium:nre:4.8"
        compile "Tridium:baja:4.8"
        compile project(":btibFirebaseConnector-rt")

        // wb
        compile "BTIB:btibConnector-wb:46"
        compile "BTIB:btibCore-wb:46"
        compile "Tridium:bajaui-wb:4.6"
        compile "Tridium:workbench-wb:4.6"
}
```

    c. Implement the Field Editor.

```
@NiagaraType
public class BFirebaseRealtimeConnectorFE extends BRealtimeConnectorFE
{...}
```

    d. Implements isConnectorValid.

```
@Override
public boolean isConnectorValid(BExternalConnector connector)
{
    return super.isConnectorValid(connector) && connector instanceof BFirebaseConnector;
}
```

    e. Create a class that extends BDeviceRealtimeConnectorExt.
    f. Add the filed editor property.

```
@NiagaraProperty(
    name = "connector",
    type = "String",
    defaultValue = "",
    facets = @Facet(name = "BFacets.FIELD_EDITOR", value = "\"btibFirebaseConnector:
BFirebaseRealtimeConnectorFE\"")
```

```
)
@NiagaraType
public class BFirebaseDeviceExt extends BDeviceRealtimeConnectorExt
{...}
```

g. Add the icon.

```
private static final BIcon ICON = BtibIconTool.getComponentIcon(TYPE);

@Override
public BIcon getIcon()
{
    return ICON;
}
```

h. Implements filterConnector.

```
@Override
public boolean filterConnector(BIRealtimeConnector connector)
{
    return connector instanceof BFirebaseConnector;
}
```

7. Implements the point extension.

a. Create a class that extends BPointRealtimeConnectorExt, then run slotomatic.

```
@NiagaraType
public class BFirebasePointExt extends BPointRealtimeConnectorExt
{...}
```

b. Add the icon.

```
private static final BIcon ICON = BtibIconTool.getComponentIcon(TYPE);


@Override
public BIcon getIcon()
{
    return ICON;
}
```

c. Implements

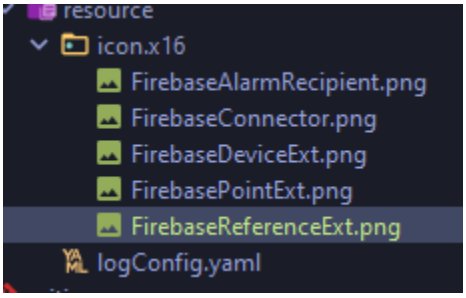
```
@Override
public boolean filterDeviceExt(BIRealtimeDeviceExtension extension)
{
    return extension instanceof BFirebaseDeviceExt;
}
```

8. Implements the reference extension.

```
@NiagaraType
public class BFirebaseReferenceExt extends BRealtimeReferenceExt
{
    /*+ ------------ BEGIN BAJA AUTO GENERATED CODE ------------ +*/
    /*@ $fr.btib.firebase.connector.BFirebaseReferenceExt(2979906276)1.0$ @*/
    /* Generated Tue Sep 08 09:51:48 CEST 2020 by Slot-o-Matic (c) Tridium, Inc. 2012 */

    ////////////////////////////////////////////////////////////////
    // Type
    ////////////////////////////////////////////////////////////////
```

```
    @Override
    public Type getType()
    {
        return TYPE;
    }

    public static final Type TYPE = Sys.loadType(BFirebaseReferenceExt.class);

    /*+ ------------ END BAJA AUTO GENERATED CODE -------------- +*/

    private static final BIcon ICON = BtibIconTool.getComponentIcon(TYPE);

    ////////////////////////////////////////////////////////////////
    // BRealtimeReferenceExt
    ////////////////////////////////////////////////////////////////

    @Override
    public boolean filterDeviceExt(BIRealtimeDeviceExtension extension)
    {
        return extension instanceof BFirebaseDeviceExt;
    }

    ////////////////////////////////////////////////////////////////
    // Setters / Getters
    ////////////////////////////////////////////////////////////////

    @Override
    public BIcon getIcon()
    {
        return ICON;
    }
}
```

9. And the Icon.



10. Implements Alarm Recipient.
    a. Create a new class that extends BRealtimeAlarmRecipient.
    b. Add the connector editor field property then run slotomatic.

```
@NiagaraProperty(
    name = "connector",
    type = "String",
    defaultValue = "",
    facets = @Facet(name = "BFacets.FIELD_EDITOR", value = "\"btibFirebaseConnector:
FirebaseRealtimeConnectorFE\"")
)
@NiagaraType
public class BFirebaseAlarmRecipient extends BRealtimeAlarmRecipient
{...}
```

    c. Implements the logger.

```
private static final BtibLogger LOG = BtibLogger.getLogger(TYPE);


@Override
public BtibLogger getLogger()
```

```
{
    return LOG;
}
```

d. Implements the icon.

```
public static final BIcon ICON = BtibIconTool.getComponentIcon(TYPE);

@Override
public BIcon getIcon()
{
    return ICON;
}
```

e. Implements filterConnector.

```
@Override
public boolean filterConnector(BRealtimeConnector connector)
{
    return connector instanceof BFirebaseConnector;
}
```

11. Edit the *module.palette* file, add the components to the palette.

```
<?xml version="1.0" encoding="UTF-8"?>
<bajaObjectGraph version="1.0">
    <p m="b=baja" t="b:UnrestrictedFolder">
        <!-- Connector -->
        <p n="FirebaseConnector" m="bfc=btibFirebaseConnector" t="bfc:FirebaseConnector"/>
        <!-- Alarm -->
        <p n="Alarm" t="b:UnrestrictedFolder">
            <p n="FirebaseAlarmRecipient" t="bfc:FirebaseAlarmRecipient"/>
        </p>
        <!-- Extensions -->
        <p n="Extensions" t="b:UnrestrictedFolder">
            <p n="FirebaseDeviceExt" t="bfc:FirebaseDeviceExt"/>
            <p n="FirebasePointExt" t="bfc:FirebasePointExt"/>
                        <p n="FirebaseReferenceExt" t="bfc:FirebaseReferenceExt"/>
        </p>
    </p>
</bajaObjectGraph>
```

12. Enable the log, create a yaml file named **logConfig.yaml** in the resource folder.

```
btibFirebaseConnector:
  - BFirebaseAlarmRecipient
  - BFirebaseConnector
  - BFirebaseDeviceExt
  - BFirebasePointExt
```

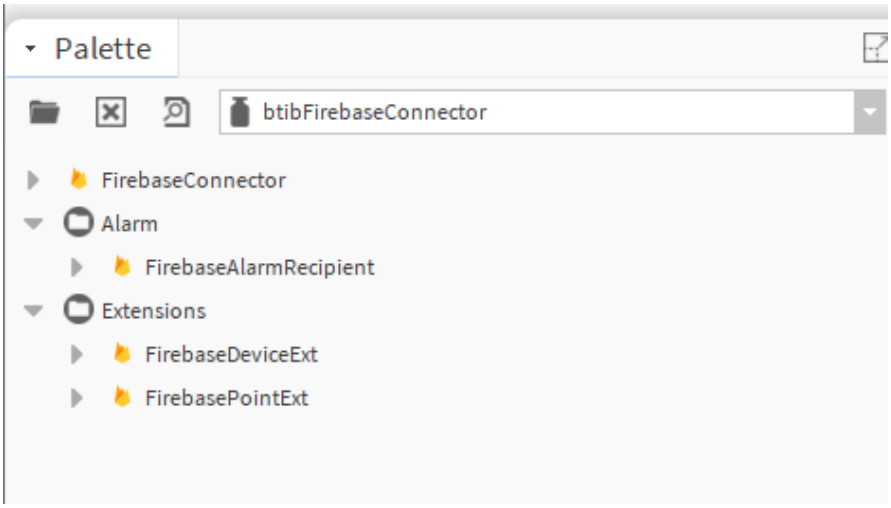
Test:

Build the module and sign the jars using your certificate.

1. Start the station we created earlier, Go to the Niagara installation folder/bin, open the command line and run station.exe <stationName>.

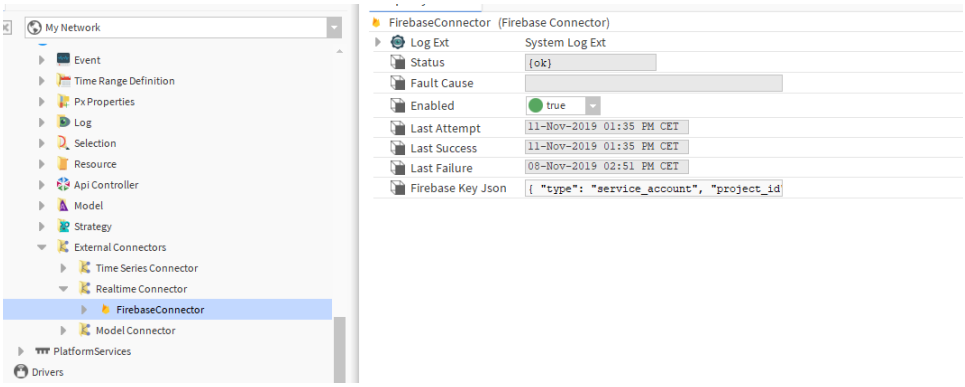
```
C:\Niagara\Niagara-4.8.0.110\bin> .\station.exe testFirebaseConnector
```

2. Start the workbench, then go to the palette, choose your module you should see the components we created.

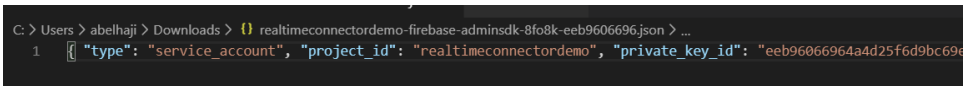


.

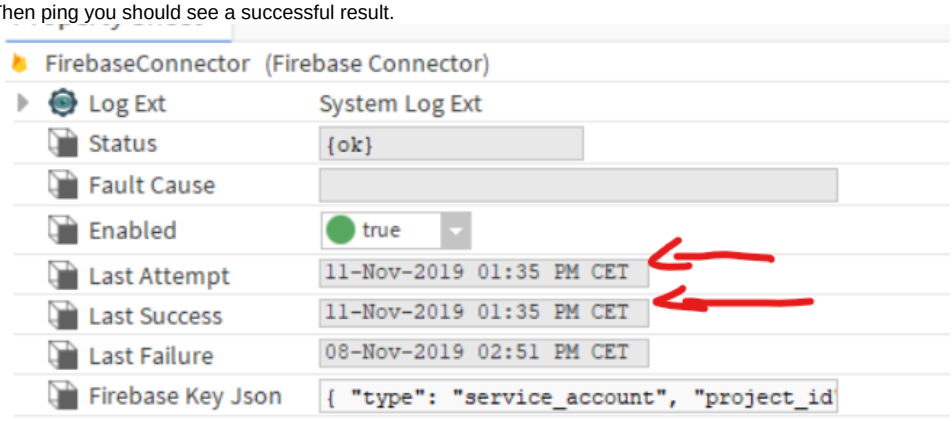
3. Drag and drop the connector to the **btibService External Connectors Realtime Connectors** folder



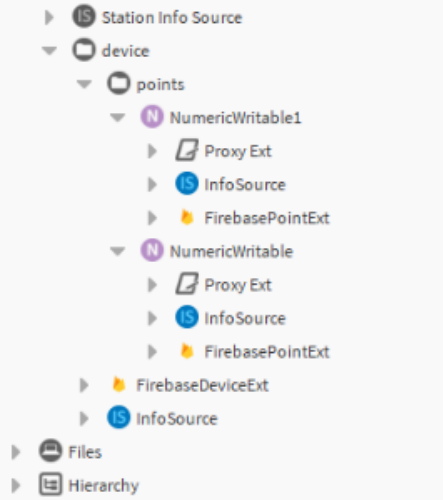
4. On the firebase Key JSON property copy and paste the content of the json file we downloaded it earlier. Make sure you removed the line breaks before.



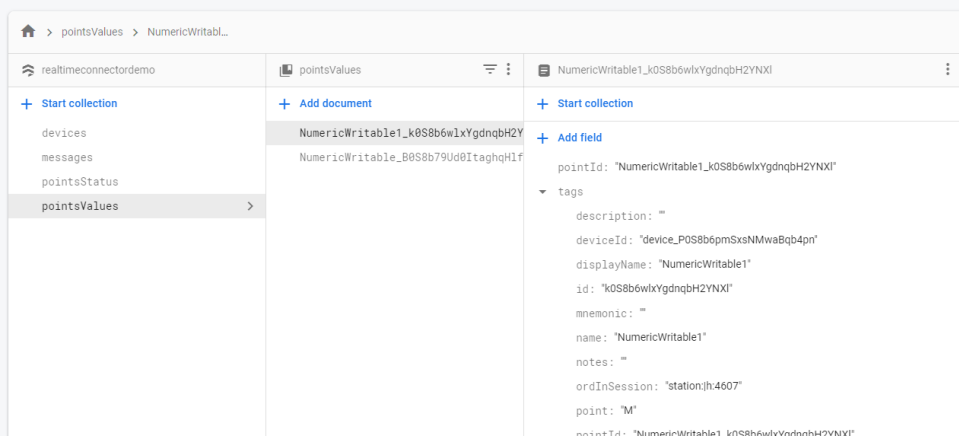
5. Then ping you should see a successful result.



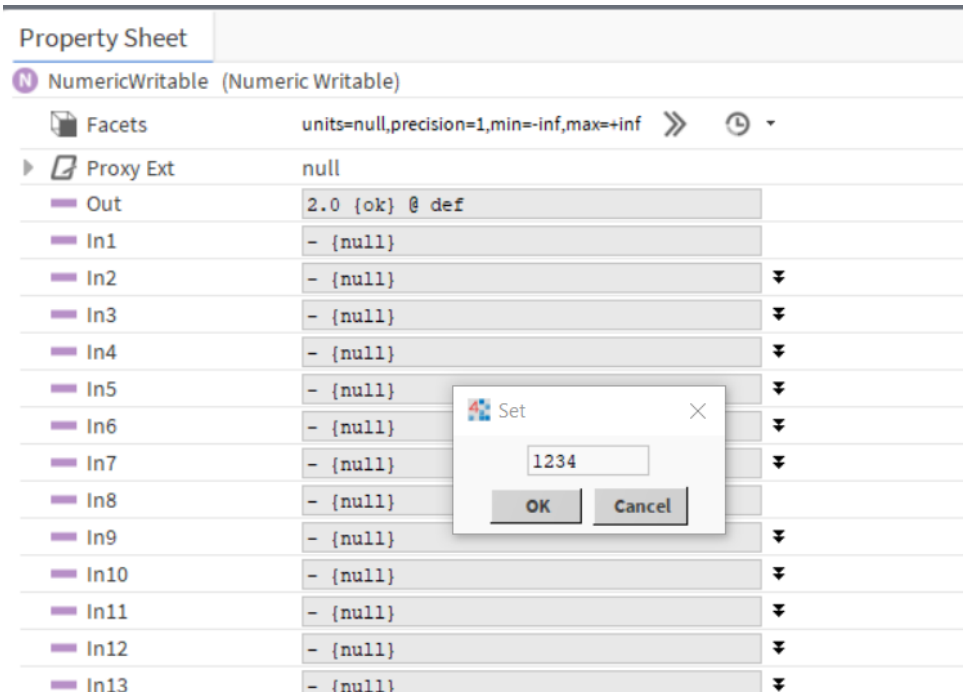
6. Now go to the **Drivers** and create a network then a device and a point.



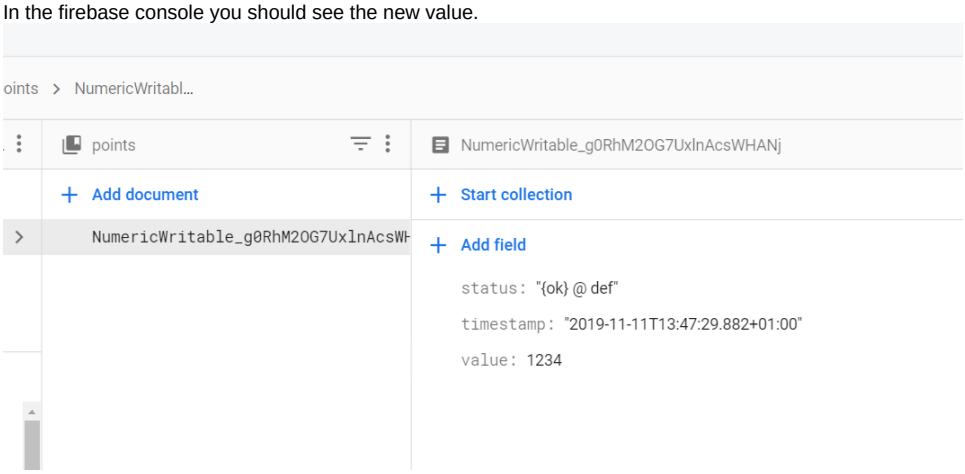
7. Then add the device extension and the point extension.
8. Go to firebase console you should see the device the point inside including the tags.



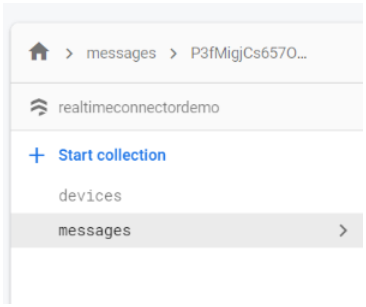
9. Now go to the point and change its value.



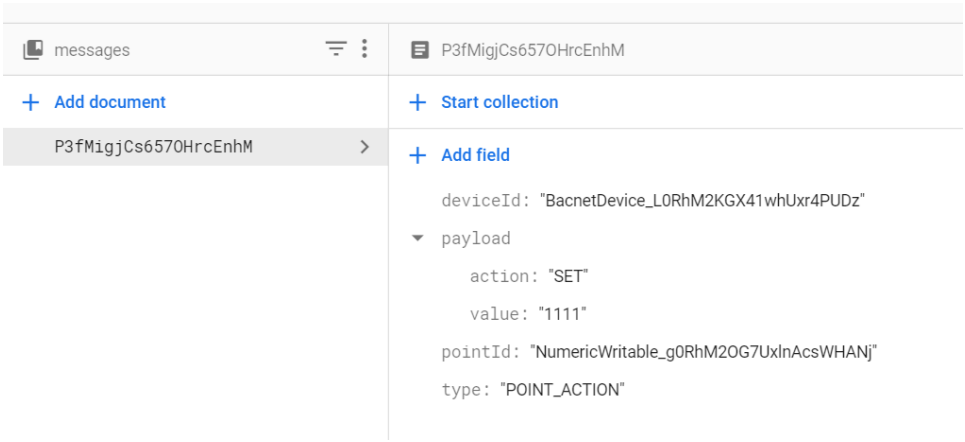
10. In the firebase console you should see the new value.



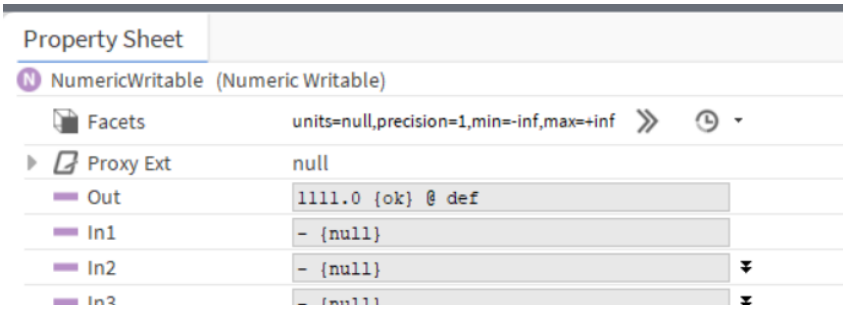
11. Now let's test the external messages, go to the firebase and create the messages collection.



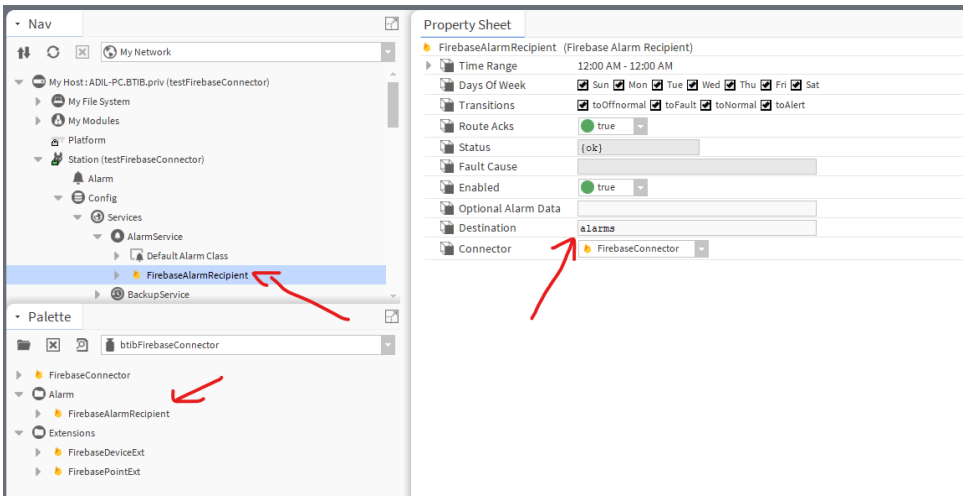
12. Add a new document with the message type (check the connectors documentation for more info).



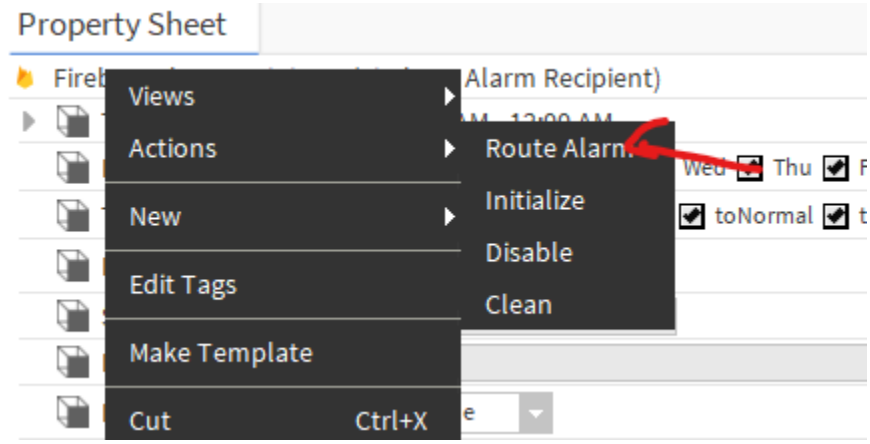
13. Go back to the workbench and you should see the value populated.



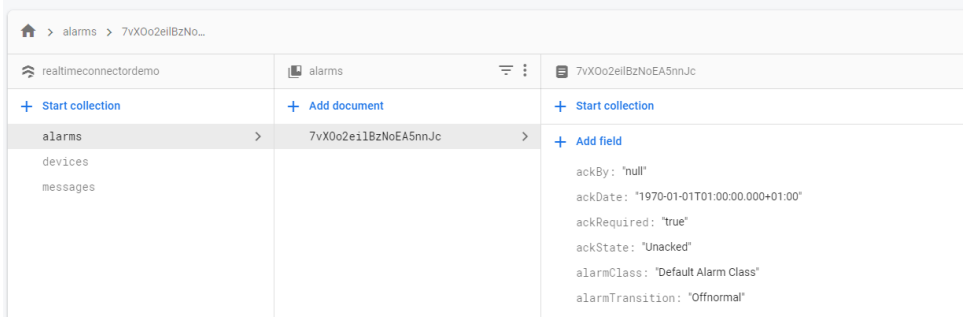
14. Go to the alarm service and add the recipient.



15. Choose the name of the destination collection.
16. Then route a test alarm.



17. Go to firebase and you should see your alarm.



Full source code https://github.com/VayanData/btibFirebaseConnector_n4