PARTNERS FOR ADVANCED TRANSPORTATION TECHNOLOGY
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

Modernization of Center-to-Center Data Communication Standards

Task 3713 (65A0761)

Sample Implementation Administration and User Guide

November 30, 2021

This page left blank intentionally

Sample Implementation Administration and User Guide

Author

Brian Peterson
Software Engineering Manager
California PATH
University of California, Berkeley

This page left blank intentionally

TABLE OF CONTENTS

This page left blank intentionally

LIST OF FIGURES

This page left blank intentionally

LIST OF TABLES

This page left blank intentionally

1. INTRODUCTION

This document provides a guide to the reference sample application developed under the Traffic Management Data Dictionary (TMDD) modernization contract. The application provides a sample implementation of a SOAP implementation using the XML Schema Definition (XSD) and Web Service Description Language (WSDL) developed under the contract. In addition, it provides a sample implementation of a Kafka/JSON based messaging implementation. Both implementations utilize a sample set of data captured during the I-210 Connected Corridors contract. The application is intended for use by experienced Information Technology (IT) personnel as an example of both SOAP and Kafka implementations of a Center-to-Center (C2C) information exchange system.

1.1. INTENDED AUDIENCE

The primary audience for this document includes:

- The Caltrans Division of Research, Innovation, and System Information (DRISI).
- Caltrans Operations personnel involved in specifying, procuring, and implementation of systems requiring C2C communications
- Caltrans Information Technology Personnel that may be tasked with installing this application

The document is intended primarily to provide IT personnel with the information required to install and operate the application, and to understand the basics of its design. It assumes that those who are installing and operating the application have a fundamental understanding of the following:

- Information technology systems and their implementation and operation, especially large-scale implementations
- Database technology, design, and administration. A basic understanding of Mongodb and Postgres is helpful.
- Messaging technology and the Kafka environment. Experience and understanding of the Confluent Kafka stack is helpful.
- Docker and the use of containers and container stacks, docker networking, and the use of docker compose.
- Basic programming experience and the use of Integrated Development Environment tools such as Intellij. Java experience is helpful.

A basic understanding of IT systems is sufficient to install and run the application and the document is written so that an individual with such understanding without specific experience in these areas should be able to install and operate the system. Additional experience will allow a more advanced usage of the system, including understanding, possible modification, and configuration of the system to independently use either the owner or external center with a third party implementation of either a SOAP or Kafka/Mongo service.

1.2. DOCUMENT ORGANIZATION

The remainder of this document is organized as follows:

- **Section 2** provides the requirements for installation
- **Section 3** provides a short overview of the application design
- **Section 4** provides installation instructions
- **Section 5** provides operation instructions

2. INSTALLATION REQUIREMENTS

2.1. HARDWARE REQUIREMENTS

The application was built for operation as a containerized application using Docker. It was built and tested with the following hardware:

- Apple Macbook Pro 16" 2019 with:
  2.3 GHz 8-Core Intel Core i9 Processor
  32 GB 2667 MHz DDR4 Memory
  macOS Catalina Version 10.15.7

Recommended minimum storage available: 60GB

As it is a containerized application, similar hardware and OS environments (Windows or Linux) meeting similar processor and memory performance should work, but may require some modification to the docker configuration and networking. It is configured to be installed on a single computer, but can be reconfigured to run on multiple computers.

2.2. REQUIRED SOFTWARE

The following software is required for installation and operation of the application:

Docker desktop for your operating system (Docker Desktop for Mac v 3.5.2.18 was used)
Robo-3T v1.4.4 or similar Mongodb management tool
IntelliJ IDEA v2020.3.3 or other Java based Integrated Development Environment (IDE)
DataGrip v2021.1 or similar Postgres management tool
Google Chrome or other internet browser software

Refer to the installation instructions for each tool and install on the target machine(s).

All other software utilized is installed within the containers that are deployed and no user installation is required.

This page left blank intentionally

3. APPLICATION DESIGN

3.1. INTRODUCTION

Center to center (C2C) communications occur between an owner center and an external center. Owner and external centers are generally traffic management centers or other producers or consumers of traffic information. C2C communications enable these centers to exchange information related to traffic management, such as inventories and status of infrastructure elements like sensors, intersection signals, or changeable message signs or request execution of control commands affecting those infrastructure elements. An owner center generally refers to the center that owns, manages, or controls those infrastructure elements, and an external center refers to the center that is requesting information related to those infrastructure elements.

The application provides the ability to exchange a limited set of the messages implemented within the Traffic Management Data Dictionary, v 3.3d with modifications recommended within this report. The sample data exchanged is based on data collected over a 24-hour period within the Connected Corridors program. The data is not modified to represent the recommendations made in this report and is exactly as it was collected within the Connected Corridors program. Two methods of data exchange are available within the application. First, a SOAP web services-based exchange is available, implemented using a modified version of a test harness developed for the Connected Corridors program. A Kafka-based messaging exchange is also available, implemented using a Confluent-based Kafka messaging stack.

3.2. DESIGN

The design of the application provides for a single owner center and a separate, single external center. Figure 1 illustrates the application design. Within this illustration, the owner center is shown within the tan colored box on the left side of the diagram, and the external center shown on the right tan colored box. Both the owner center and the external center are each docker stacks with multiple containers within each stack. Each white box within the owner center and external center represents one or more containers that house the services required to run the application.

The owner center on the left side of the figure is a docker stack composed of services and containers represented in Figure 1 and Table 3-1.

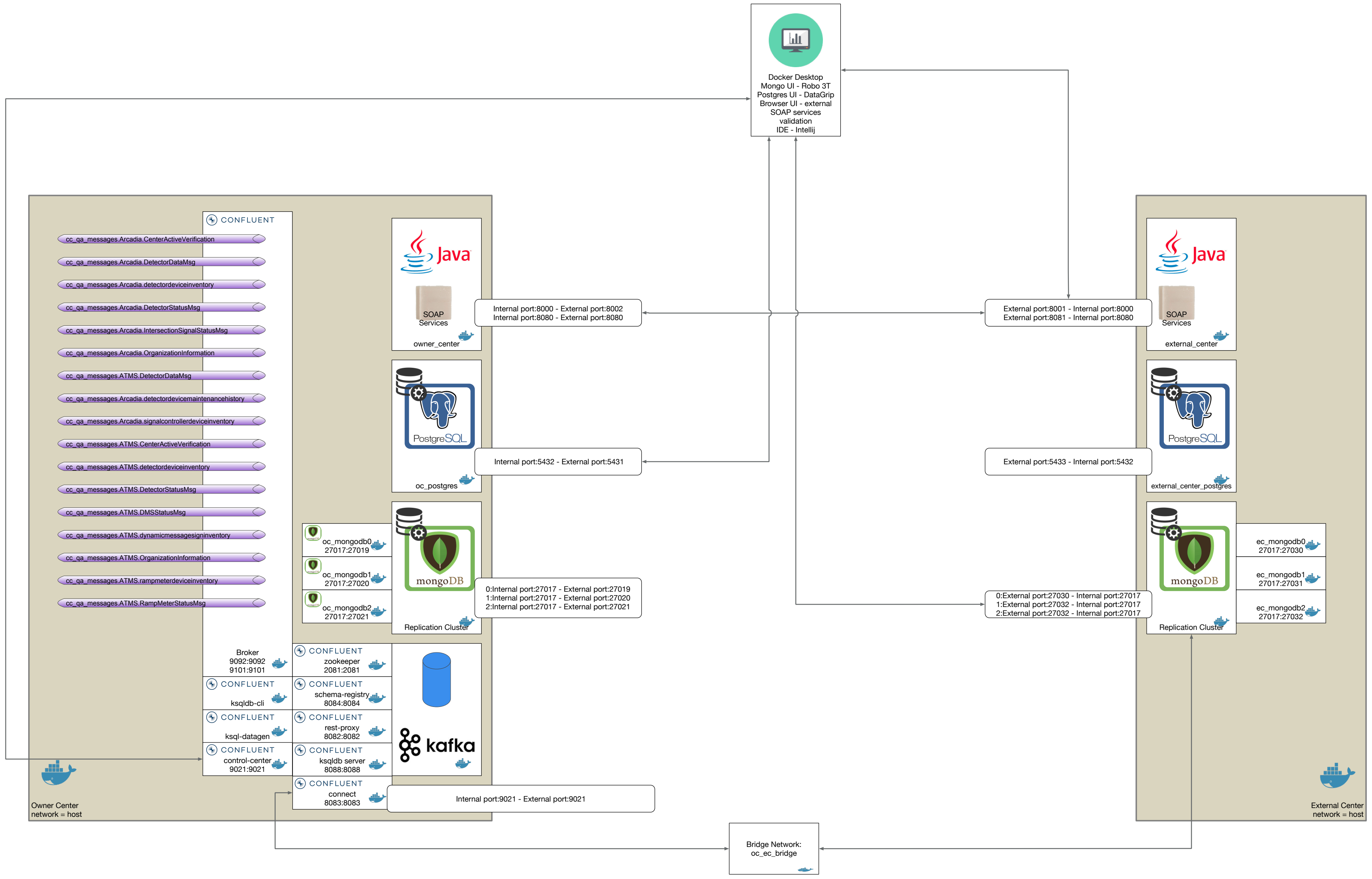

Docker Desktop
Mongo UI - Robo 3T
Postgres UI - DataGrip
Browser UI - external
SOAP services
validation
IDE - Intellij
CONFLUENT
cc_qa_messages.Arcadia.CenterActiveVerification
cc_qa_messages.Arcadia.DetectorDataMsg
cc_qa_messages.Arcadia.detectordeviceinventory
cc_qa_messages.Arcadia.DetectorStatusMsg
cc_qa_messages.Arcadia.IntersectionSignalStatusMsg
cc_qa_messages.Arcadia.OrganizationInformation
cc_qa_messages.ATMS.DetectorDataMsg
cc_qa_messages.Arcadia.detectordevicemaintenancehistory
cc_qa_messages.Arcadia.signalcontrollerdeviceinventory
cc_qa_messages.ATMS.CenterActiveVerification
cc_qa_messages.ATMS.detectordeviceinventory
cc_qa_messages.ATMS.DetectorStatusMsg
cc_qa_messages.ATMS.DMSStatusMsg
cc_qa_messages.ATMS.dynamicmessagesigninventory
cc_qa_messages.ATMS.OrganizationInformation
cc_qa_messages.ATMS.rampmeterdeviceinventory
cc_qa_messages.ATMS.RampMeterStatusMsg
Java
SOAP Services
owner_center
Internal port:8000 - External port:8002
Internal port:8080 - External port:8080
External port:8001 - Internal port:8000
External port:8081 - Internal port:8080
Java
SOAP Services
external_center
PostgreSQL
oc_postgres
Internal port:5432 - External port:5431
External port:5433 - Internal port:5432
PostgreSQL
external_center_postgres
oc_mongodb0
27017:27019
oc_mongodb1
27017:27020
oc_mongodb2
27017:27021
mongoDB
Replication Cluster
0:Internal port:27017 - External port:27019
1:Internal port:27017 - External port:27020
2:Internal port:27017 - External port:27021
0:External port:27030 - Internal port:27017
1:External port:27032 - Internal port:27017
2:External port:27032 - Internal port:27017
mongoDB
Replication Cluster
ec_mongodb0
27017:27030
ec_mongodb1
27017:27031
ec_mongodb2
27017:27032
Broker
9092:9092
9101:9101
CONFLUENT
zookeeper
2081:2081
CONFLUENT
ksqldb-cli
CONFLUENT
schema-registry
8084:8084
CONFLUENT
ksql-datagen
CONFLUENT
rest-proxy
8082:8082
CONFLUENT
control-center
9021:9021
CONFLUENT
ksqldb server
8088:8088
kafka
CONFLUENT
connect
8083:8083
Internal port:9021 - External port:9021
Owner Center
network = host
External Center
network = host
Bridge Network:
oc_ec_bridge

Figure 1 - Application Design

Table 3-1 - Owner Center Docker Stack Elements

Owner Center Docker Stack Elements		
Service	Container(s)	Description
Owner center SOAP service	owner_center	Apache Tomcat container with TMDD SOAP Services owner center application
Owner center Postgres database	oc_postgres	Postgres database containing owner center data (XML formatted)
Owner center Mongodb database (3 containers)	oc_mongodb0	Owner center Mongodb replication cluster containing owner center data (json formatted)
	oc_mongodb1	
	oc_mongodb2	
Kafka (up to 9 containers)	broker	Kafka broker (not clustered)
	zookeeper	Zookeeper cluster manager
	schema-registry (optional)	Kafka schema registry for message validation
	rest-proxy	Kafka control interface
	connect	Kafka connect service (utilized for mongodb-kafka interface)
	ksqldb server (optional	Kafka ksqldb service
	ksqldb-cli (optional)	Kafka ksqldb command line interface
	ksqldatagen (optional)	Kafka random data generator
	control-center	Kafka management user interface

The external center on the right side of the figure is a docker stack composed of the following services and containers:

Table 3-2 - External Center Docker Stack Elements

External Center Docker Stack Elements		
Service	Container(s)	Description
External center SOAP service	external_center	Apache Tomcat container with TMDD SOAP services external center application

External center Postgres database	external_center_postgres	Postgres database containing external center data (XML formatted)
External center Mongodb database (3 containers)	ec_mongodb0	External center Mongodb replication cluster for receiving owner center data (json formatted)
	ec_mongodb1	
	ec_mongodb2	

In addition to the two docker stacks and their containers, at the top of the diagram are the tools used by an administrator or user to manage and operate the owner and external center components. These are installed on the user or administrator's local machine, which can be the same as the owner and external host machine. Users may substitute other similar tools based on personal preference. Those used and tested include the following:

Table 3-3 - C2C Application Management Components

C2C Application Management Components	
Service	Description
Docker Desktop	Used to manage docker deployment, images, and containers
Google Chrome or other internet browser	Used to access control center and SOAP exchange validation report
Robo 3T	Open source Mongodb management tool
DataGrip	Commercial interface for Postgres management (Required only for development – not for operation)
Intellij	Community or Commercial integrated development environment (Required only for development – not for operation)

The design provides for the following:

- Isolation of SOAP and Kafka based implementations
- Automated application deployment and shutdown
- Control of individual message streams for each data message type
- Exchange of messages for the following:
    - Organization
    - Center Active Verification
    - Ramp Meters
    - Intersection Signals
    - Sensors
    - Dynamic Message Signs
- Exchange of different information types, such as:
    - Inventories
    - Status

  - Data
  - Maintenance History

Each of the two Docker stacks (owner center and external center) contain their own isolated default networks which allow the containers within the stacks to communicate. In addition, the two SOAP service containers (external_center and owner_center) communicate on the docker host network. The owner centers connect container communicates with the external center mongodb containers via a bridge network between the two Docker stacks, the oc_ec_bridge.

The next section of this document will describe installation and opportunities to configure the application to run in different environments if desired.

This page left blank intentionally

4. INSTALLATION INSTRUCTIONS

These installation instructions assume installation of the application on a single host. Installation on multiple hosts may require changes to the installation parameters. The instructions will identify installation elements that will require changes in order to install on multiple hosts.

4.1. INSTALLATION PREREQUISITES

Install the non-containerized local components on the host of the application. This includes an internet browser such as Google Chrome, Docker, a mongodb management application such as Robo 3T, a postgres management application such as DataGrip (optional – required for development environments only), and a developer IDE such as Intellij (optional – required for developer environments only). For non-development installations, any text editor may be used instead of the developer IDE to make file edits required for operation. The application is developed using Java, so any IDE selected should be capable of Java based development. The version of the Java Development Kit (JDK) used in development is OpenJDK v1.8.0_282.

4.2. APPLICATION COMPONENTS

The application is delivered within a single zip file. When unzipped, the following high level directory structure is created:

```
/TMDD Modernization
        /tmdd_owner_center
        /tmdd_external_center
        /Mongodata
        /Images
```

tmdd_owner_center contains the code for the owner center, including java code, configuration, and docker instructions for deployment
tmdd_external_center contains the code for the external center, including java code, configuration, and docker instructions for deployment
Mongodata contains the data files for the owner center mongodb database
Images contains docker images for deployment

4.3. INSTALLATION INSTRUCTIONS

To install the application on a single host, complete the following:

4.3.1. OPEN THE JAVA PROJECTS (DEVELOPERS ONLY)

1. Copy or move the /tmdd_owner_center and /tmdd_external_center directories to the directory where your Intellij IDE projects are located. (This step is optional, as generally the project can be opened from any location – just nice from a point of keeping things clean).
2. Open each of the projects within Intellij. You should see something similar to the following:

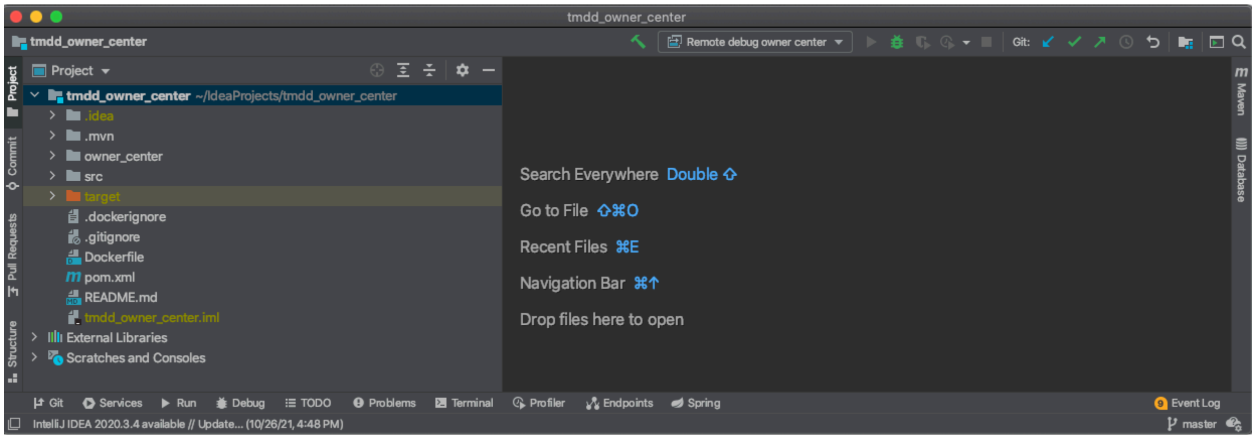


Figure 2 - tmdd_owner_center Opened within Intellij

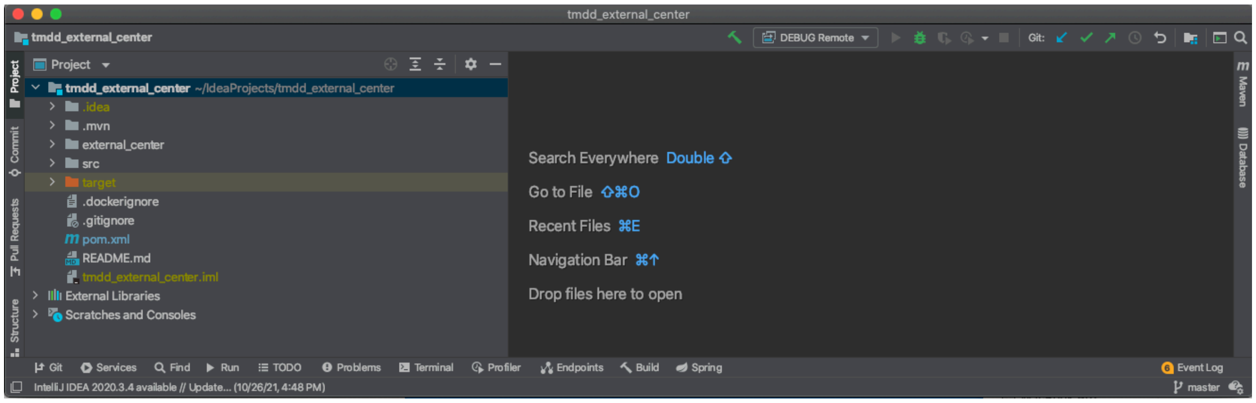


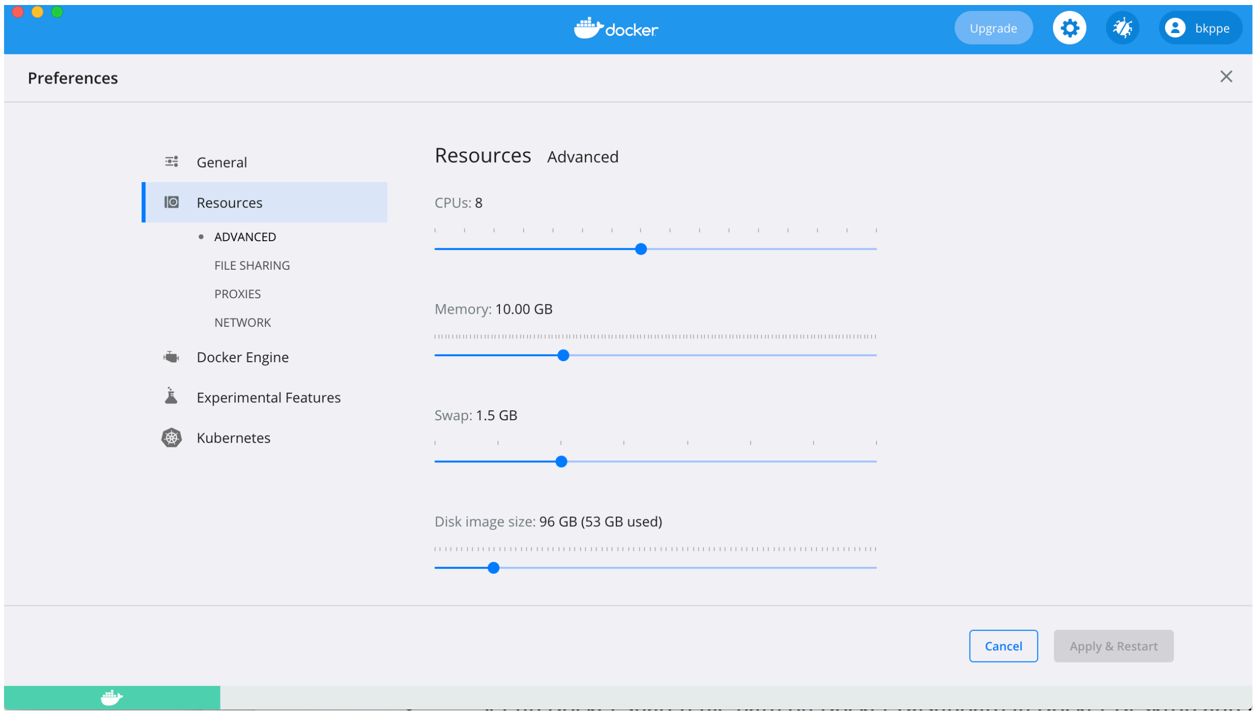
Figure 3 - tmdd_external_center Opened within Intellij

4.3.2. CONFIGURE DOCKER DESKTOP

Docker desktop and your docker environment must be configured to run the application. To configure your docker environment, complete the following:

1. Set up the Docker resources in Docker Desktop.
   a. Open Docker Desktop and click the settings icon on the top right corner of the application. ( )

b. Click Resources.



CPUs: 8

Memory: 10.00 GB

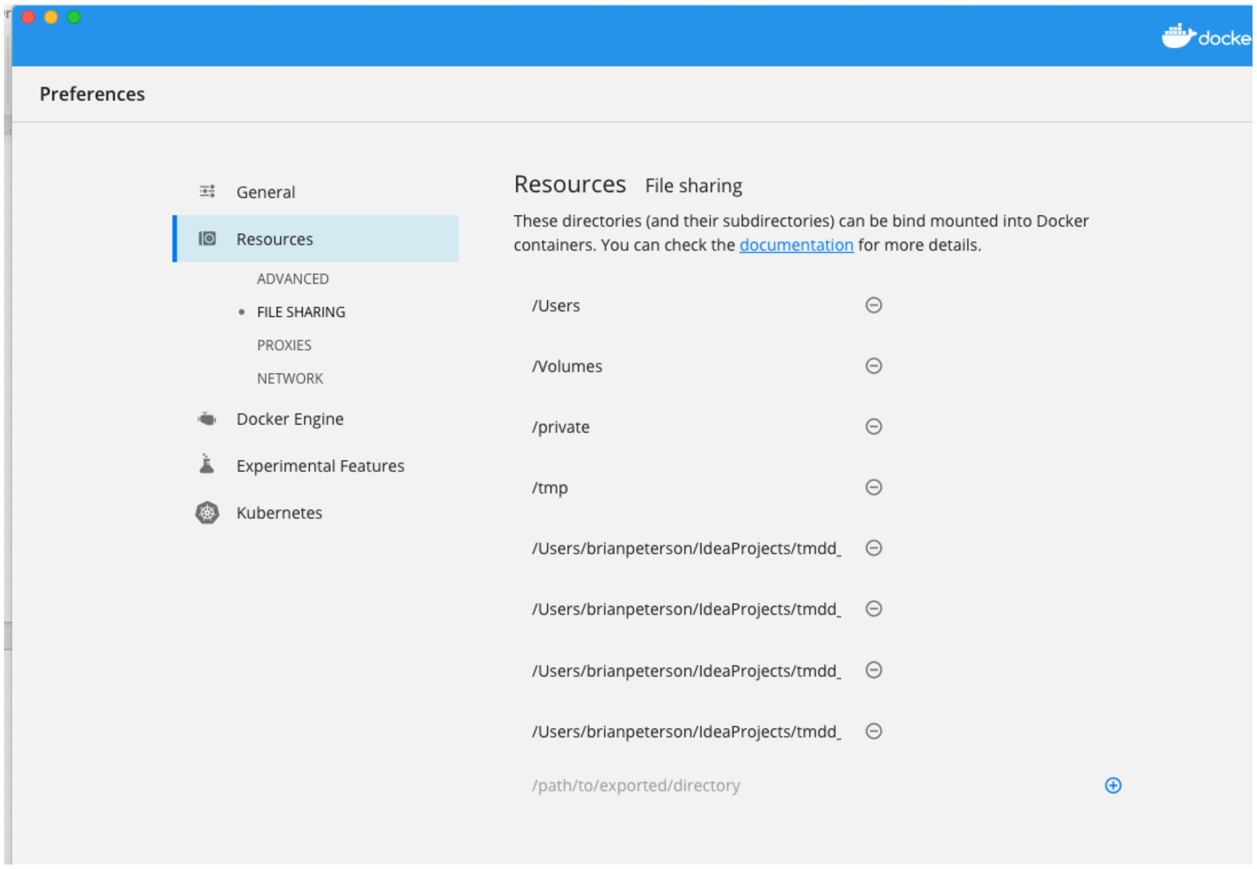
Swap: 1.5 GB

Disk image size: 96 GB (53 GB used)

Figure 4 - Docker Desktop Resources Settings

c. Set the resources as shown in Figure 4. CPUs to a minimum of 8, memory to a minimum of 10GB, swap to a minimum of 1.5 GB, and disk image size to a minimum of 96 GB.

2. Set up Docker file sharing within Docker Desktop.
   a. While you are still in Docker Desktop, click FILE SHARING under the Preferences, Resources menu.



Resources File sharing

These directories (and their subdirectories) can be bind mounted into Docker containers. You can check the documentation for more details.

/Users
/Volumes
/private
/tmp
/Users/brianpeterson/IdeaProjects/tmdd_
/Users/brianpeterson/IdeaProjects/tmdd_
/Users/brianpeterson/IdeaProjects/tmdd_
/Users/brianpeterson/IdeaProjects/tmdd_
/path/to/exported/directory

Figure 5 - Docker Desktop File Sharing Settings

b. Click on the + button on the bottom right to add the full path of each location where the tmdd_owner_center/owner_center/center_service, tmdd_owner_center/owner_center/postgres, tmdd_external_center/external_center/center_service, and tmdd_external_center/external_center/postgres were created in your development environment.

c. Click Apply & Restart in the docker Preferences screen.

3. Install the owner_center Postgres image.
   a. Open a terminal window on the machine in which docker is installed.
   b. Run the following command, substituting the **PATH** with the full path of the directory where the postgres image was unzipped (the images directory in the resulting file structure):
      i. docker load < **PATH**/ocPostgresImage1.1.tar

4.3.3. CONFIGURE THE TMDD_OWNER_CENTER PROJECT

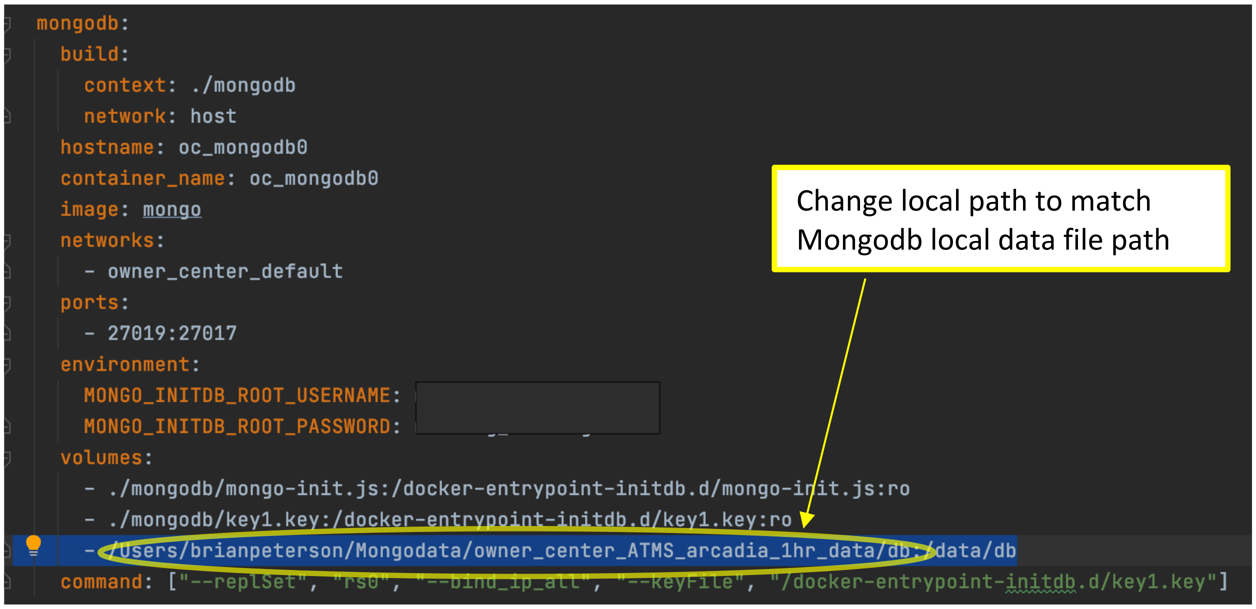
Complete the following within the tmdd_owner_center project to configure it for operation:

1. For the SOAP services, complete the following:

   a. **THIS STEP IS NOT REQUIRED IF STARTING THE INSTALLATION FROM THE ZIP FILE. IT IS ONLY REQUIRED IF STARTING FROM THE FIRST PULL FROM A GIT REPOSITORY.** The target WAR files are included within the zip files. If the source of the tmdd_owner_center project is a git repository, you will need to build the WAR files. To build the WAR files run a maven clean followed by maven package. After the maven package process has successfully completed, copy the following files from the target folder to the owner_center/center_service folder:
      i. TMDDOwnerCenter-Arcadia.war
      ii. TMDDOwnerCenter-ATMS.war
      iii. TMDDOwnerCenter-County.war
      iv. TMDDOwnerCenter-TSMSS.war

2. Configure the Kafka implementation as follows:
   a. In the path where you unzipped the Mongodata folder, create two copies of the directory and contents of Mongodata/owner_center_ATMS_arcadia_1_hr_data/db0, creating the following:
      i. Mongodata/owner_center_ATMS_arcadia_1_hr_data/db0
      ii. Mongodata/owner_center_ATMS_arcadia_1_hr_data/db1
      iii. Mongodata/owner_center_ATMS_arcadia_1_hr_data/db2

      This creates a set of data for each of the Mongodb instances within the owner center's replication cluster.
   b. In the file tmdd_owner_center/owner_center/docker-compose.yml, change the path for the volume mounts for each of the Mongodb instances to the full path where you created the three directories holding the mongo data in step a above. Figure 6 highlights the line where you change the full path for the mongodb0 data within the docker-compose.yml file. Make sure to leave as is the end of the line ":/data/db" which references the path used within the container itself that is mapped to your local path. Repeat the procedure for the mongodb1 and mongodb2 data paths you created in step a above.



```
mongodb:
  build:
    context: ./mongodb
    network: host
  hostname: oc_mongodb0
  container_name: oc_mongodb0
  image: mongo
  networks:
    - owner_center_default
  ports:
    - 27019:27017
  environment:
    MONGO_INITDB_ROOT_USERNAME: [redacted]
    MONGO_INITDB_ROOT_PASSWORD:
  volumes:
    - ./mongodb/mongo-init.js:/docker-entrypoint-initdb.d/mongo-init.js:ro
    - ./mongodb/key1.key:/docker-entrypoint-initdb.d/key1.key:ro
    - /Users/brianpeterson/Mongodata/owner_center_ATMS_arcadia_1hr_data/db:/data/db
  command: ["--replSet", "rs0", "--bind_ip_all", "--keyFile", "/docker-entrypoint-initdb.d/key1.key"]
```

Change local path to match Mongodb local data file path

Figure 6 - Owner Center docker-compose.yml Modification for Mongodb

4.3.4. CONFIGURE THE EXTERNAL CENTER PROJECT

Complete the following within the tmdd_external_center project to configure it for operation:

- For the SOAP services, complete the following:
  a. **THIS STEP IS NOT REQUIRED IF STARTING THE INSTALLATION FROM THE ZIP FILE. IT IS ONLY REQUIRED IF STARTING FROM THE FIRST PULL FROM A GIT REPOSITORY.** The target WAR files are included within the zip files. If the source of the tmdd_external_center project is a git repository, you will need to build the WAR files. To build the WAR files run a maven clean followed by maven package. After the maven package process has successfully completed, copy the following file from the target folder to the owner_center/center_service folder:
     i. TMDDExternalCenter.war
  b. Set the owner center postgres username and password (optional).
     i. In the tmdd_external_center/external_center folder, open the docker-compose.yml file. In the postgres: service definition environment variable for POSTGRES_PASSWORD, set the desired postgres password.
     ii. In the external_center\center_service folder, open the application.properties file set the value for postgres.qa.dataSource.pass to match the password set in step i above.
  c. Configure what data will be exchanged when running the SOAP service as follows:
     i. In the application.properties file opened in step 1.b.above, choose the pipelines to be run by setting the pipeline.configFiles. As delivered, the application is set to run the arcadia pipelines. Also available are the atms,

county, and tsmss pipelines. An example of how to run multiple source pipelines is included in the comment above the pipeline.configFiles.

ii. Set the subscription duration limit by entering the duration in seconds as the value for the key subscription.duration.seconds. The delivered application.properties value is set to 120 seconds. A maximum value is 86400 seconds (24 hours). Note that the subscription will end at midnight of the day for which it was started regardless of the duration entered.

d. Configure the endpoint address for each pipeline SOAP service. These endpoint addresses will point to the owner center SOAP service. These instructions assume you are running both owner center and external center stacks on the same local machine. You will need to adjust if the owner center and external center stacks are running on separate machines.

   i. Identify the local address of the host machine. On a mac based machine, open the network preferences on your local machine (found in the System Preferences application), choose the network you are currently using in the list of available networks, click Advanced, and select the TCP/IP tab. Note or copy the IPv4 address of the machine.

   ii. Use the IPv4 address copied in step i above and paste that address in each of the SOAP endpoint IP addresses for each pipeline sources' endpoint and returnAddress. Replace only the ip address in each line and leave the remaining url elements as is.

   iii. If you are using two machines – one for external center and one for owner center, the IP of the owner center should be used for the "endpoint" address and the IP of the external center should be used for the subReturnAddress element.

e. Configure the details of each pipeline startup. These are contained in each of the pipline start yml files located in the tmdd_external_center/external_center/center_service yml files (e.g. _arcadiaPipelineStart.yml, _atmsPipelineStart.yml, etc.)

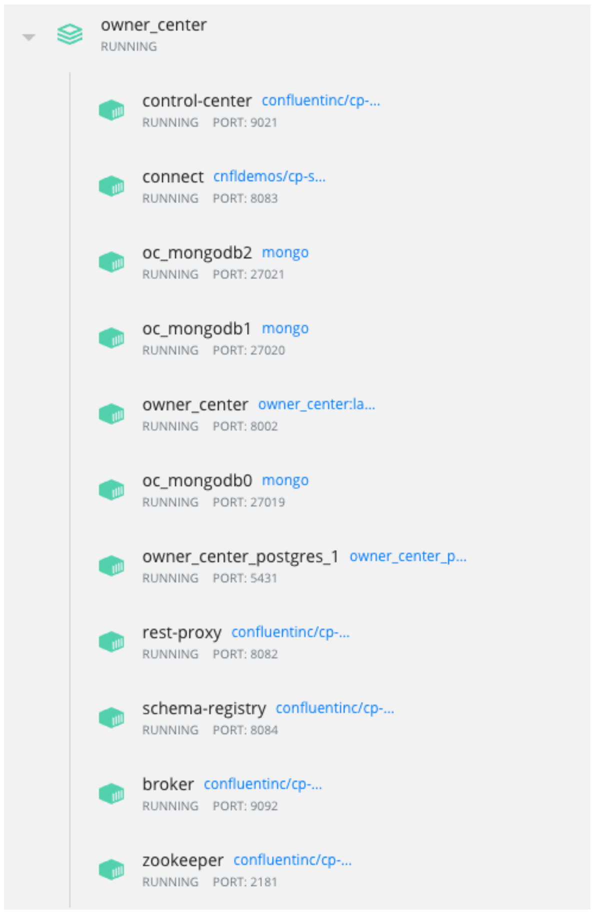
   i. Each of the files are preconfigured to run. The name of the pipeline, its communication type, subtype (if required), and subFrequency (if required) are specified. For each specified, there is data within the postgres database for the run specified. In general, use the following definitions to understand what will run with the specified configuration.

      1. name: the name of the dialog to be run
      2. communication: the type of dialog – sync indicates a request response, async indicates a subscription
      3. subType: if async is specified for the communication parameter, the subType must be specified as onChange for an onchange subscription, periodic for a periodic subscription
      4. subFrequency: if the subType is periodic, subFrequency must be specified in seconds.

Note that it is critical that a supporting set of data be available within the owner_center postgres instance for any desired SOAP dialog and communication parameter combination. That can be determined by reviewing the materialized views within postgres and verifying a set of data exists for the desired dialog and accompanying parameters.

4.3.5. COMPLETE AND VERIFY INSTALLATION

1. (Optional) Setup Mongodb replication set security keys. Each Mongodb replication set (owner center and external center) consist of three individual Mondogb instances. Each instance requires a security key that allows the instance to communicate with the other instances within its replication cluster. A keyfile is provided with the software, however, if a secure keyfile is desired, the provided keyfile can be replaced with a private, secure keyfile. See the instructions for Mongodb keyfile creation in the Mongodb online documentation at https://docs.mongodb.com/manual/tutorial/deploy-replica-set-with-keyfile-access-control/.
2. Change the permissions for each owner center Mongodb keyfile. In a terminal session, run the following command within the directory (3 directories) where the keyfiles are located. They should be located in the tmdd_owner_center/owner_center/mongodb, tmdd_owner_center/owner_center/mongodb1, and tmdd_owner_center/owner_center/mongodb2 directories. Substitute the name of your keyfile for key1.key if you created your own keyfile in step 1 above.
    a. chmod 600 key1.key
3. Repeat step 2 for each external center Mongodb keyfile. Keyfiles are located in the tmdd_external_center/external_center/mongodb, tmdd_external_center/external_center/mongodb1, tmdd_external_center/external_center/mongodb2 directories.
4. Build and start the owner center stack by completing the following:
    a. Open a terminal window and navigate to the directory where you installed the owner center and the docker_compose.yml file exists (tmdd_owner_center/owner_center).
    b. Type `docker–compose up ––build` and press return. Docker will download the required images for installation, build the containers required and start the application. Open docker desktop and when complete, the owner center stack should look similar to Figure 7. Note that it is possible that one or more containers may exit, as occasional timing issues may arise that cause an application to not start. If that occurs, select the container that stopped, and to the right of the container name, click start. This will generally fix the issue.



```
owner_center
RUNNING

control-center  confluentinc/cp-...
RUNNING   PORT: 9021

connect  cnfldemos/cp-s...
RUNNING   PORT: 8083

oc_mongodb2  mongo
RUNNING   PORT: 27021

oc_mongodb1  mongo
RUNNING   PORT: 27020

owner_center  owner_center:la...
RUNNING   PORT: 8002

oc_mongodb0  mongo
RUNNING   PORT: 27019

owner_center_postgres_1  owner_center_p...
RUNNING   PORT: 5431

rest-proxy  confluentinc/cp-...
RUNNING   PORT: 8082

schema-registry  confluentinc/cp-...
RUNNING   PORT: 8084

broker  confluentinc/cp-...
RUNNING   PORT: 9092

zookeeper  confluentinc/cp-...
RUNNING   PORT: 2181
```

Figure 7 - Docker Desktop Display for Running Owner Center

5. Finalize configuration of the owner center Mongodb databases as follows:
    a. Set up your Mongodb client to connect to the Mongodb instances. Follow the instructions of your Mongodb client software to connect to each of the three owner center Mongodb instances. You will want to create two connections to each instance, one logged in as admin and a second logged in as cc_qa_messages. The connection information for each is as follows (it is recommended that you change the password after connecting – see Mongodb documentation for instructions. If you change the password, update your connection and the Kafka Connect configuration files located in the tmdd_owner_center/owner_center/connect directory to reflect the password change):
        i. oc_mongodb0 – admin
            1. address: localhost
            2. port: 27019
            3. database: admin
            4. username: user_admin
            5. password: Hummingbird (it is recommended that you change the password after start – see Mongodb documentation for

instructions. If you change the password, update your connection and the Kafka Connect configuration files located in the tmdd_owner_center/owner_center/connect directory)

ii. oc_mongodb0 – cc_qa_messages
   1. address: localhost
   2. port: 27019
   3. database: cc_qa_messages
   4. username: db_owner
   5. password: TurkeyVulture (it is recommended that you change the password after start – see Mongodb documentation for instructions. If you change the password, update your connection and the Kafka Connect configuration files located in the tmdd_owner_center/owner_center/connect directory)

iii. oc_mongodb1 – admin (same as oc_mongodb0-admin with the following changes):
   1. port:27020

iv. oc_mongodb1 – cc_qa_messages (same as oc_mongodb0-cc_qa_messages with the following changes):
   1. port: 27020

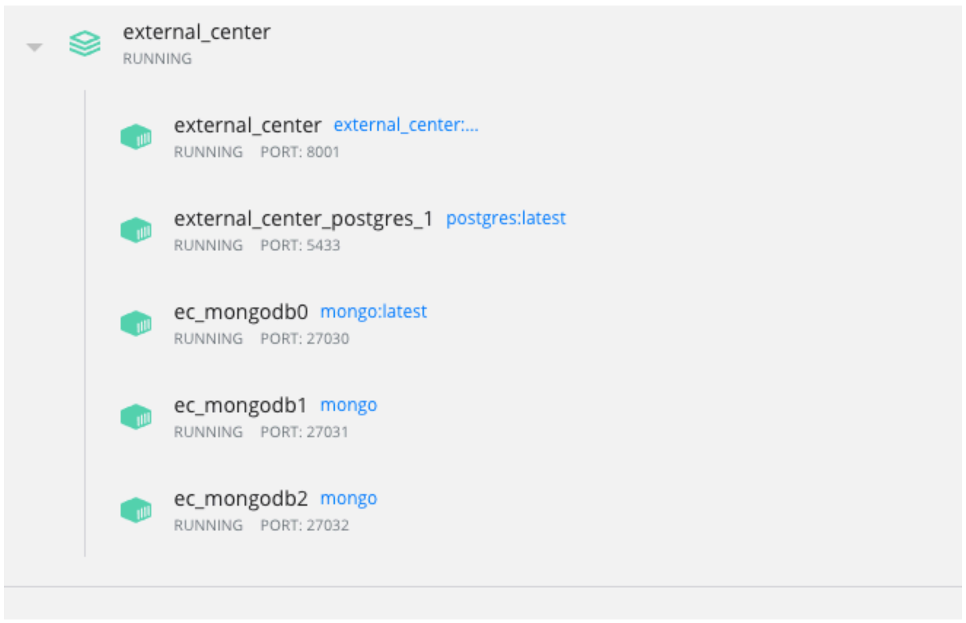
v. oc_mongodb2 – admin (same as oc_mongodb0-admin with the following changes):
   1. port:27021

vi. oc_mongodb2 – cc_qa_messages (same as oc_mongodb0-cc_qa_messages with the following changes):
   1. port: 27021

b. Verify the connection to each instance. Using your Mongodb client, create a shell within the oc_mongdb0-admin connection you just verified. In Robo 3T you can do this by right clicking on the connection in the left pane and selecting Open Shell.
c. Within the shell, type rs.initiate() and execute the command (in Robo 3T, click the green arrow in the top toolbar. In the results pane, expand the results and verify the initiation of the cluster was successful.
d. Replace the rs.initiate command with the following: rs.add('oc_mongodb1') and execute the command. Expand the results and verify the command was successful.
e. Repeat the rs.initiate command for oc_mongodb2.
f. Replace the rs.add command with rs.status(). Execute the command and verify the replication cluster is operating. oc_mongodb0 should be the primary node in the cluster, oc_mongodb1 and oc_mongodb2 should be secondary nodes. (Note, it is possible during shutdown and subsequent restart of the owner center docker stack that one of the secondary nodes becomes primary. This is normal, but may cause confusion when subsequently executing commands for the Mongodb via your Mongodb client. If a command fails, verify the primary node

via the rs.status() command. Any command requiring a write or change to the database must be executed through the primary node.

6. Build and start the external center stack by completing the following:
    a. Open a terminal window and navigate to the directory where you installed the external center.
    b. Type `docker-compose up --build` and press return. Docker will download the required images for installation, build the containers required and start the application. Open docker desktop and when complete, the external center stack should look similar to Figure 8. Note that it is possible that one or more containers may exit, as occasional timing issues may arise that cause an application to not start. If that occurs, select the container that stopped, and to the right of the container name, click start. This will generally fix the issue.


external_center
RUNNING

external_center external_center:...
RUNNING PORT: 8001

external_center_postgres_1 postgres:latest
RUNNING PORT: 5433

ec_mongodb0 mongo:latest
RUNNING PORT: 27030

ec_mongodb1 mongo
RUNNING PORT: 27031

ec_mongodb2 mongo
RUNNING PORT: 27032

Figure 8 - Docker Desktop Display for Running External Center

7. Finalize configuration of the external center Mongodb databases as follows:
    a. Set up your Mongodb client to connect to the Mongodb instances. Follow the instructions of your Mongodb client software to connect to each of the three external center Mongodb instances. As with the owner center cluster, you will want to create two connections to each instance, one logged in as admin and a second logged in as cc_qa_messages. The connection information for each is as follows (it is recommended that you change the password after connecting – see Mongodb documentation for instructions. If you change the password, update your connection and the Kafka Connect configuration files located in the tmdd_owner_center/owner_center/connect directory to reflect the password change):
        i. ec_mongodb0 – admin
            1. address: localhost

2. port: 27030
3. database: admin
4. username: user_admin
5. password: Hummingbird (it is recommended that you change the password after start – see Mongodb documentation for instructions. If you change the password, update your connection and the Kafka Connect configuration files located in the tmdd_owner_center/owner_center/connect directory)

ii. ec_mongodb0 – ec_qa_messages
1. address: localhost
2. port: 27030
3. database: ec_qa_messages
4. username: db_owner
5. password: TurkeyVulture (it is recommended that you change the password after start – see Mongodb documentation for instructions. If you change the password, update your connection and the Kafka Connect configuration files located in the tmdd_owner_center/owner_center/connect directory)

iii. ec_mongodb1 – admin (same as ec_mongodb0-admin with the following changes):
1. port:27031

iv. ec_mongodb1 – ec_qa_messages (same as ec_mongodb0-ec_qa_messages with the following changes):
1. port: 27031

v. ec_mongodb2 – admin (same as ec_mongodb0-admin with the following changes):
1. port:27032

vi. ec_mongodb2 – ec_qa_messages (same as ec_mongodb0-ec_qa_messages with the following changes):
1. port: 27032

b. Verify the connection to each instance. Using your Mongodb client, create a shell within the ec_mongdb0-admin connection you just verified. In Robo 3T you can do this by right clicking on the connection in the left pane and selecting Open Shell.

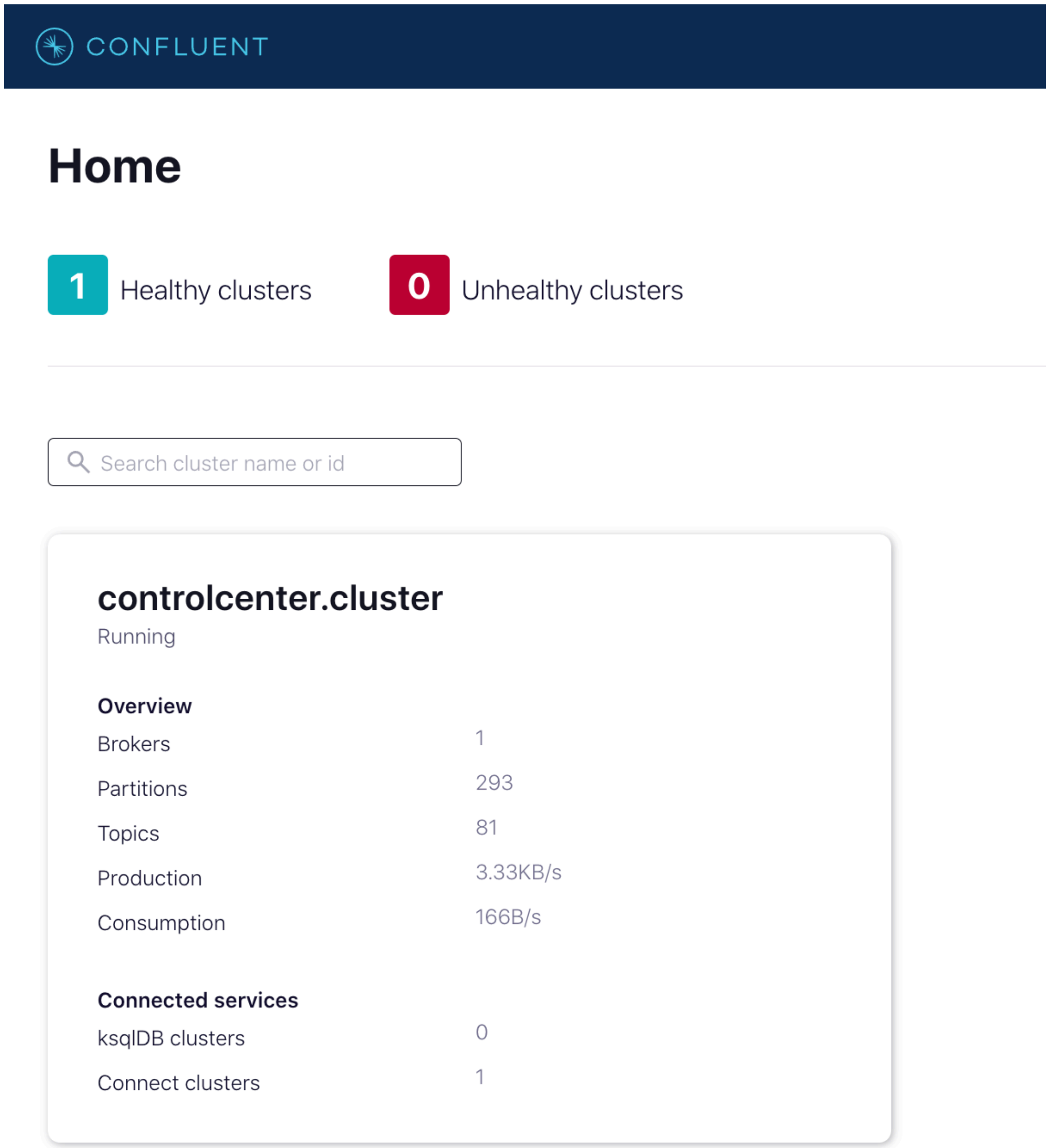
c. Within the shell, type rs.initiate() and execute the command (in Robo 3T, click the green arrow in the top toolbar. In the results pane, expand the results and verify the initiation of the cluster was successful.

d. Replace the rs.initiate command with the following: rs.add('ec_mongodb1') and execute the command. Expand the results and verify the command was successful.

e. Repeat the rs.initiate command for ec_mongodb2.

f. Replace the rs.add command with rs.status(). Execute the command and verify the replication cluster is operating. ec_mongodb0 should be the primary node in the cluster, ec_mongodb1 and ec_mongodb2 should be secondary nodes. (Note, it is possible during shutdown and subsequent restart of the owner center docker stack that one of the secondary nodes becomes primary. This is normal, but may cause confusion when subsequently executing commands for the Mongodb via your Mongodb client. If a command fails, verify the primary node via the rs.status() command. Any command requiring a write or change to the database must be executed through the primary node.

8. Finalize configuration of the Kafka Connect instance as follows:
    a. Open the Kafka Control Center. On the local machine it will be located at http://localhost:9021. You should see something similar to Figure 9.



controlcenter.cluster
Running

Overview

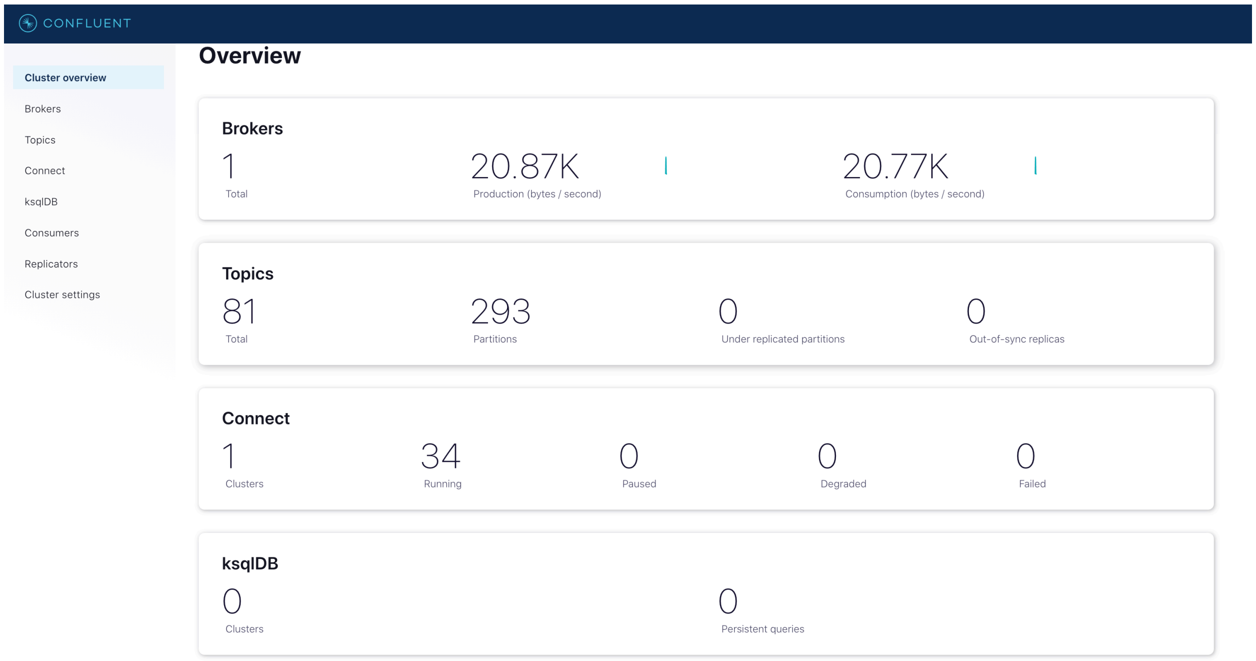
Brokers	1
Partitions	293
Topics	81
Production	3.33KB/s
Consumption	166B/s

Connected services

ksqlDB clusters	0
Connect clusters	1

Figure 9 - Kafka Control Center Home

b. Click on the controlcenter.cluster tile. You should see something similar to Figure 10. Do not be concerned if the number of topics is different. However, there should be at least 1 broker and 1 connect cluster.



Overview

Brokers
1 Total
20.87K Production (bytes / second)
20.77K Consumption (bytes / second)

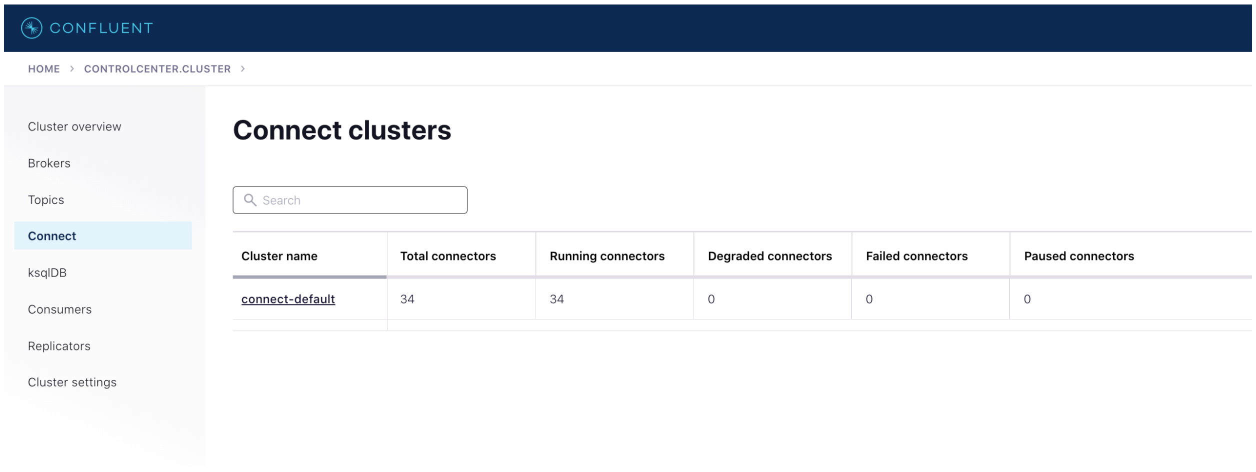
Topics
81 Total
293 Partitions
0 Under replicated partitions
0 Out-of-sync replicas

Connect
1 Clusters
34 Running
0 Paused
0 Degraded
0 Failed

ksqlDB
0 Clusters
0 Persistent queries

Figure 10 - Kafka Cluster Overview Screen

c. Click on the Connect cluster tile. You should see something similar to Figure 11. The number of total connectors may be different.

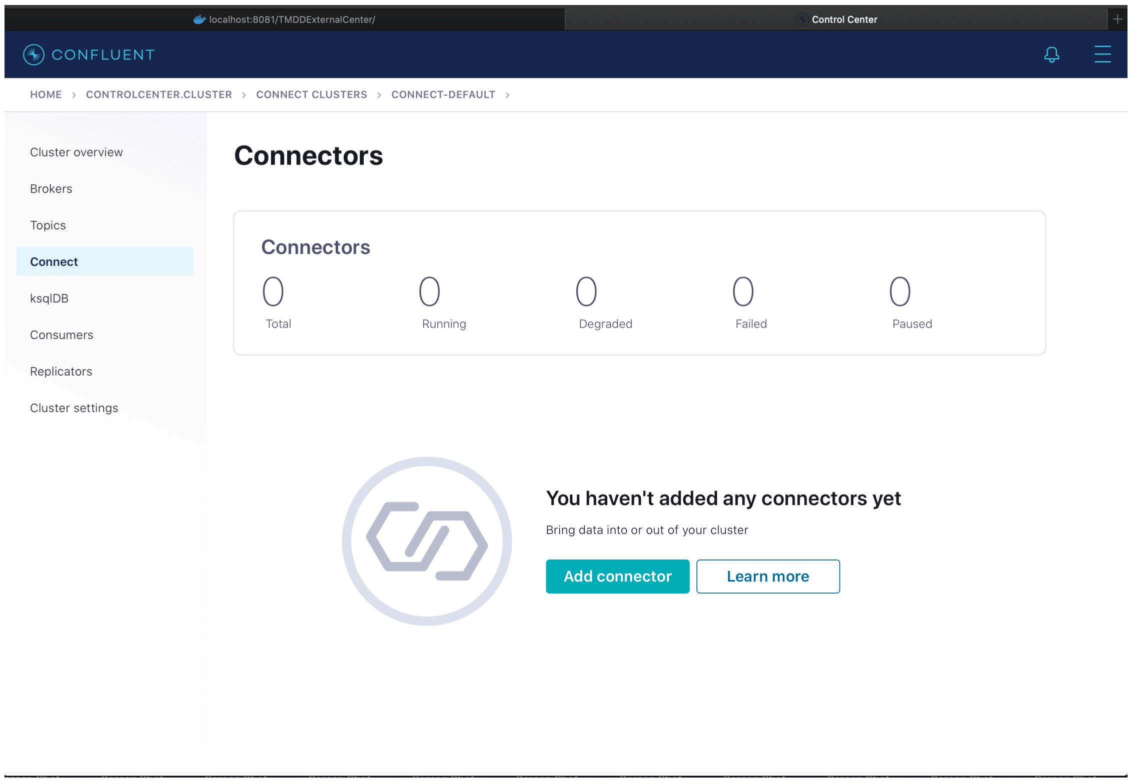


Connect clusters

Cluster name	Total connectors	Running connectors	Degraded connectors	Failed connectors	Paused connectors
connect-default	34	34	0	0	0

Figure 11 - Kafka Connect Cluster Screen

d. Click on the connect-default cluster in the list of Connect clusters. You should see something similar to Figure 13. The specific connectors listed and the number of connectors may be different in your screen. If this is the first install, there will be 0 connectors.



Connectors

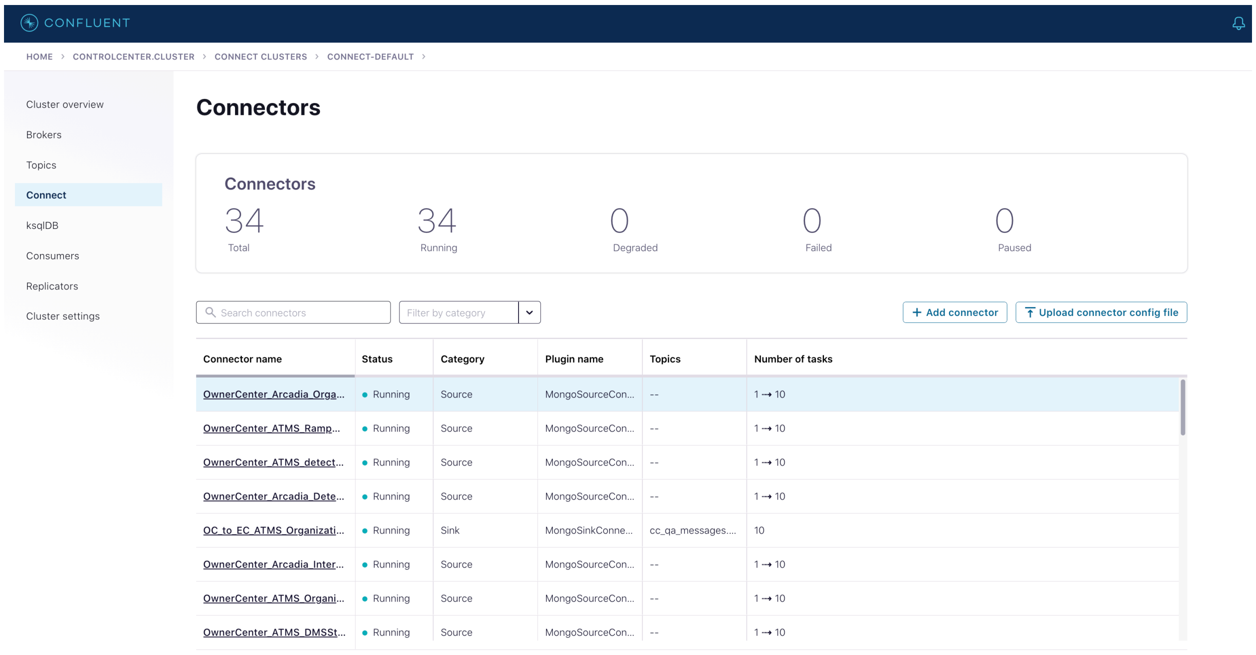
Connectors

0	0	0	0	0
Total	Running	Degraded	Failed	Paused

You haven't added any connectors yet

Bring data into or out of your cluster

Figure 12 - Kafka Connectors Screen (No Connectors)



Connectors

Connectors

34	34	0	0	0
Total	Running	Degraded	Failed	Paused

Connector name	Status	Category	Plugin name	Topics	Number of tasks
OwnerCenter_Arcadia_Orga...	● Running	Source	MongoSourceCon...	--	1 → 10
OwnerCenter_ATMS_Ramp...	● Running	Source	MongoSourceCon...	--	1 → 10
OwnerCenter_ATMS_detect...	● Running	Source	MongoSourceCon...	--	1 → 10
OwnerCenter_Arcadia_Dete...	● Running	Source	MongoSourceCon...	--	1 → 10
OC_to_EC_ATMS_Organizati...	● Running	Sink	MongoSinkConne...	cc_qa_messages....	10
OwnerCenter_Arcadia_Inter...	● Running	Source	MongoSourceCon...	--	1 → 10
OwnerCenter_ATMS_Organi...	● Running	Source	MongoSourceCon...	--	1 → 10
OwnerCenter_ATMS_DMSSt...	● Running	Source	MongoSourceCon...	--	1 → 10

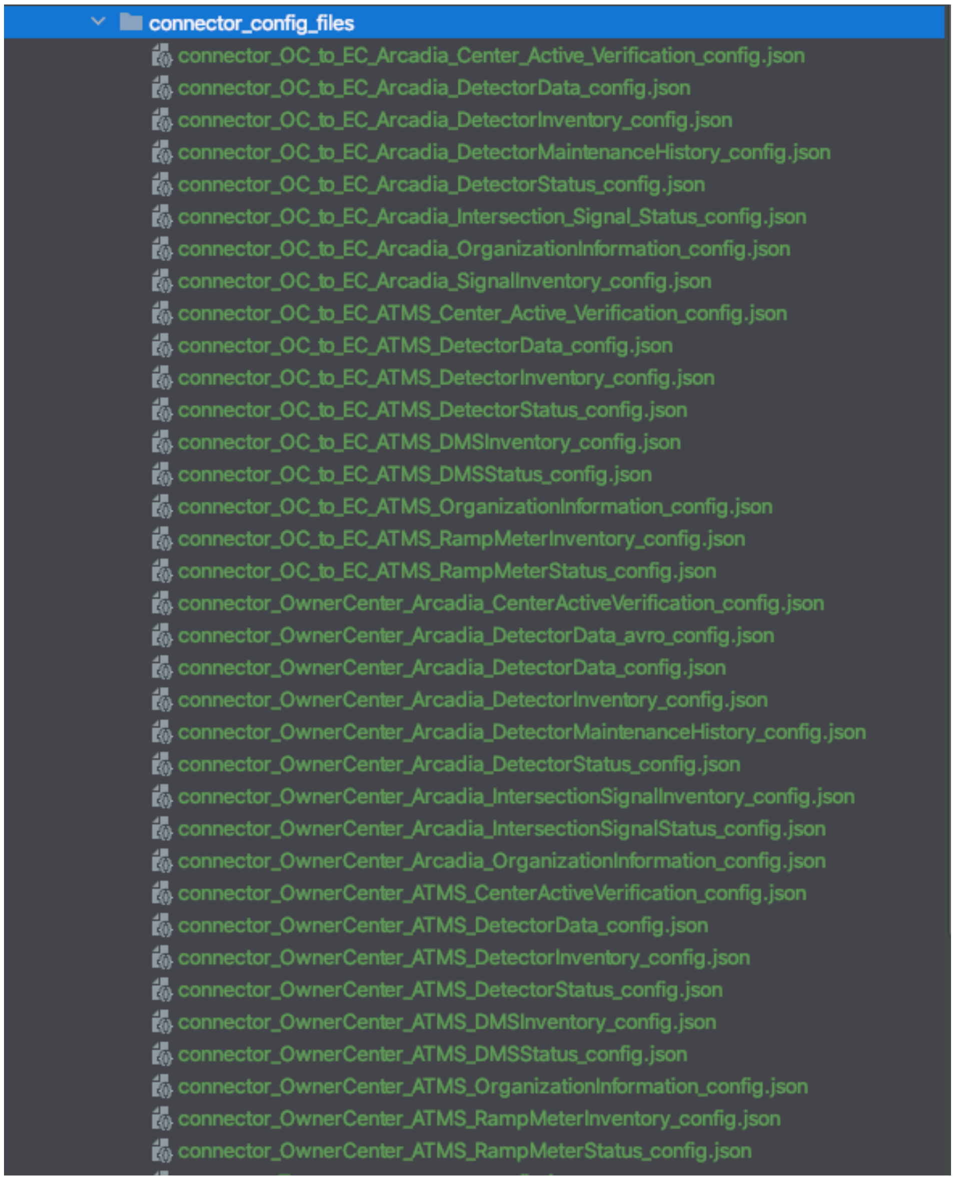
Figure 13 - Kafka Connectors Screen with Connectors

e. Connectors required for a specific pipeline are provided in pairs.

   The first connector in a pair reads from the owner center Mongodb database for a specific data source (Arcadia or ATMS) and for a specific data type (such as intersection signal status). Connectors are configured so that the first time the connector starts, it reads all data from the collection and then monitors the Mongodb collection for any new records. It places each record in the collection in a Kafka topic for that source and data type. If a topic does not yet exist, it creates the topic before populating the topic. The naming standard for this type of connector begins with a prefix of “OwnerCenter”, followed by the source name (“ATMS” or “Arcadia”) and the followed by the data type being transmitted.

   The second connector in the pair reads from the topic and writes the messages from that topic into the external center’s Mongodb database. The external center’s Mongdodb collection that is written to is specific only to the datatype being transmitted, and is not specific to the source (Arcadia or ATMS). The naming standard for this type of connector begins with the prefix “OC_to_EC” followed by the source name and then the data type being transmitted.

   If no connectors exist, or a desired connector does not exist, there are predefined connectors specified within connector config files located within the tmmd_owner_center/owner_center/connect/connector_config_files directory. Figure 14 provides a list of the connector config files that are installed with the application.


connector_config_files

```
connector_OC_to_EC_Arcadia_Center_Active_Verification_config.json
connector_OC_to_EC_Arcadia_DetectorData_config.json
connector_OC_to_EC_Arcadia_DetectorInventory_config.json
connector_OC_to_EC_Arcadia_DetectorMaintenanceHistory_config.json
connector_OC_to_EC_Arcadia_DetectorStatus_config.json
connector_OC_to_EC_Arcadia_Intersection_Signal_Status_config.json
connector_OC_to_EC_Arcadia_OrganizationInformation_config.json
connector_OC_to_EC_Arcadia_SignalInventory_config.json
connector_OC_to_EC_ATMS_Center_Active_Verification_config.json
connector_OC_to_EC_ATMS_DetectorData_config.json
connector_OC_to_EC_ATMS_DetectorInventory_config.json
connector_OC_to_EC_ATMS_DetectorStatus_config.json
connector_OC_to_EC_ATMS_DMSInventory_config.json
connector_OC_to_EC_ATMS_DMSStatus_config.json
connector_OC_to_EC_ATMS_OrganizationInformation_config.json
connector_OC_to_EC_ATMS_RampMeterInventory_config.json
connector_OC_to_EC_ATMS_RampMeterStatus_config.json
connector_OwnerCenter_Arcadia_CenterActiveVerification_config.json
connector_OwnerCenter_Arcadia_DetectorData_avro_config.json
connector_OwnerCenter_Arcadia_DetectorData_config.json
connector_OwnerCenter_Arcadia_DetectorInventory_config.json
connector_OwnerCenter_Arcadia_DetectorMaintenanceHistory_config.json
connector_OwnerCenter_Arcadia_DetectorStatus_config.json
connector_OwnerCenter_Arcadia_IntersectionSignalInventory_config.json
connector_OwnerCenter_Arcadia_IntersectionSignalStatus_config.json
connector_OwnerCenter_Arcadia_OrganizationInformation_config.json
connector_OwnerCenter_ATMS_CenterActiveVerification_config.json
connector_OwnerCenter_ATMS_DetectorData_config.json
connector_OwnerCenter_ATMS_DetectorInventory_config.json
connector_OwnerCenter_ATMS_DetectorStatus_config.json
connector_OwnerCenter_ATMS_DMSInventory_config.json
connector_OwnerCenter_ATMS_DMSStatus_config.json
connector_OwnerCenter_ATMS_OrganizationInformation_config.json
connector_OwnerCenter_ATMS_RampMeterInventory_config.json
connector_OwnerCenter_ATMS_RampMeterStatus_config.json
```

Figure 14 - Provided Connector Config Files

To create a data pipeline for a source/datatype combination, first complete one of the following:

(1) If you have no current connectors, click on the Add Connector button (see Figure 12). A screen similar to Figure 15 will be displayed. Click on the "upload connector config file" button.

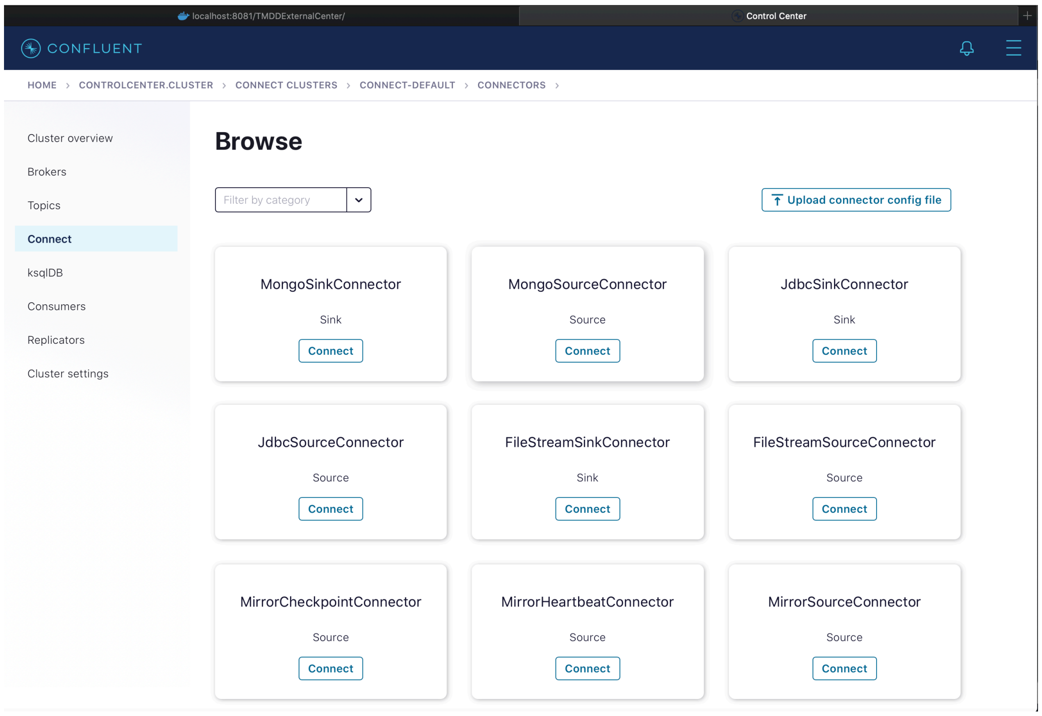
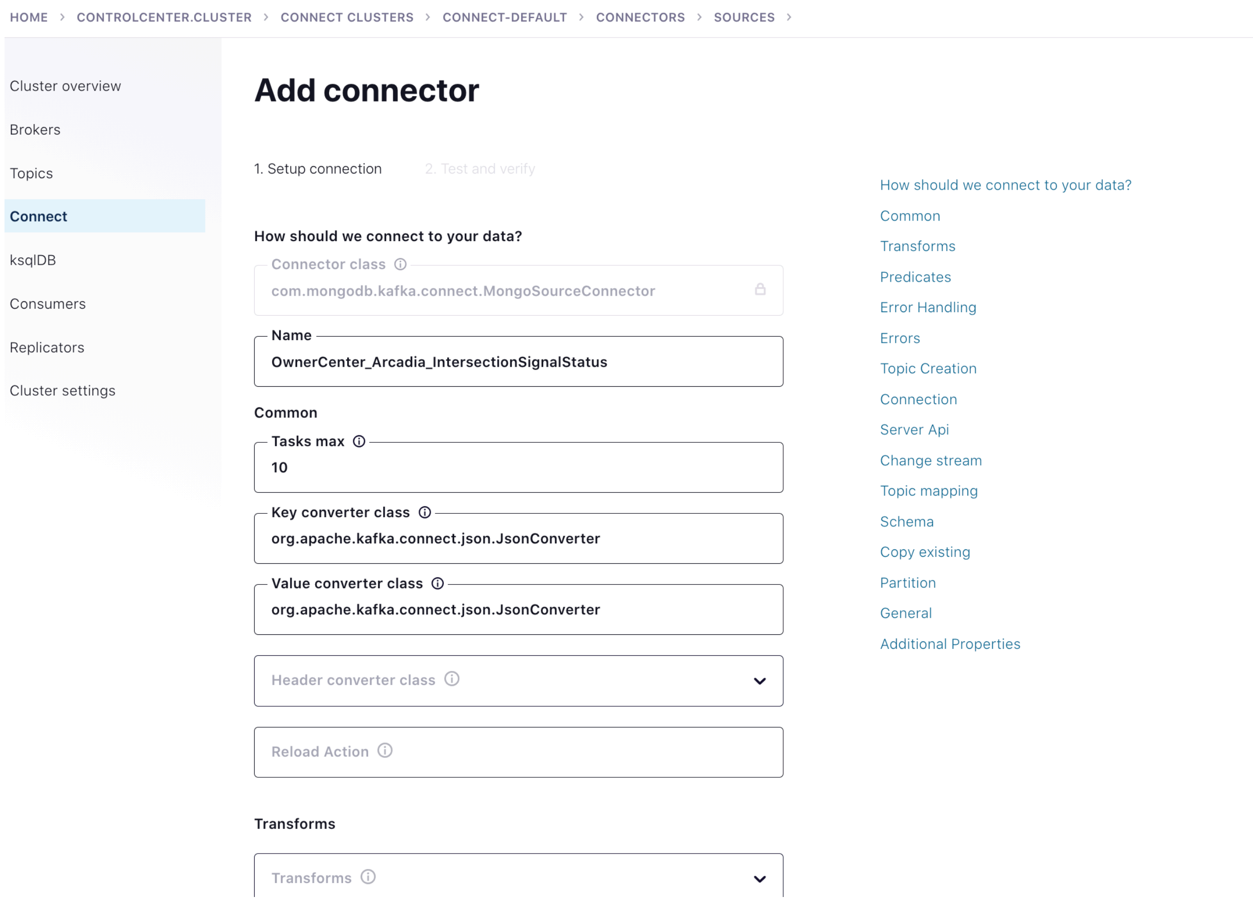


Figure 15 - Add Connector Screen – First Connector

(2) If you have already existing connectors, click on the "upload connector config file" button on the Kafka Control Center connectors screen (see Figure 13).



Add connector

1. Setup connection 2. Test and verify

How should we connect to your data?

Connector class: com.mongodb.kafka.connect.MongoSourceConnector

Name: OwnerCenter_Arcadia_IntersectionSignalStatus

Common

Tasks max: 10

Key converter class: org.apache.kafka.connect.json.JsonConverter

Value converter class: org.apache.kafka.connect.json.JsonConverter

Header converter class

Reload Action

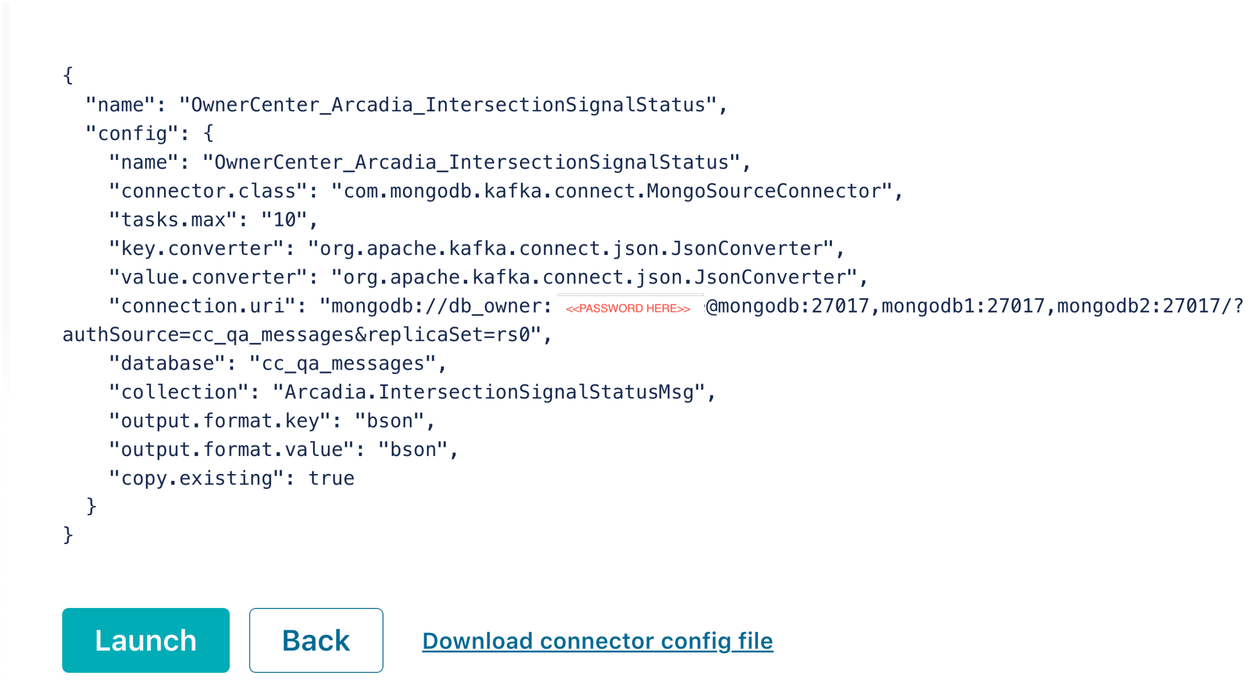
Transforms

Transforms

Figure 16 - Add a Connector

f. In the file dialog, select and open the connector config file starting with "connector_OwnerCenter" that matches the data source and data type for which you wish to create a pipeline. A screen similar to Figure 16 should be displayed.

Note that Kafka connect and control center will not allow duplicate named connectors. If the connector already exists and you attempt to create the connector with the same name, it will display an error. If you want the same connector as one that already exists, you must change the name of the connector in the Name field. Scroll to the bottom and click the Next button. You should see something similar to Figure 17. If you have changed the connector configuration in any way and want to save a configuration file for later use, click download connector file and follow the instructions to save the connector configuration. Click Launch to start the connector. Upon launch Kafka Control Center will return to the list of connectors and you should see the newly created connector in the list. It may begin in a failed state, but should change to a Running status shortly. If you click the Topics link on the Control Center left menu, you should see the new topic being filled with the message(s) from the appropriate Mongodb collection.



```
{
  "name": "OwnerCenter_Arcadia_IntersectionSignalStatus",
  "config": {
    "name": "OwnerCenter_Arcadia_IntersectionSignalStatus",
    "connector.class": "com.mongodb.kafka.connect.MongoSourceConnector",
    "tasks.max": "10",
    "key.converter": "org.apache.kafka.connect.json.JsonConverter",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "connection.uri": "mongodb://db_owner: <<PASSWORD HERE>> @mongodb:27017,mongodb1:27017,mongodb2:27017/?
authSource=cc_qa_messages&replicaSet=rs0",
    "database": "cc_qa_messages",
    "collection": "Arcadia.IntersectionSignalStatusMsg",
    "output.format.key": "bson",
    "output.format.value": "bson",
    "copy.existing": true
  }
}
```

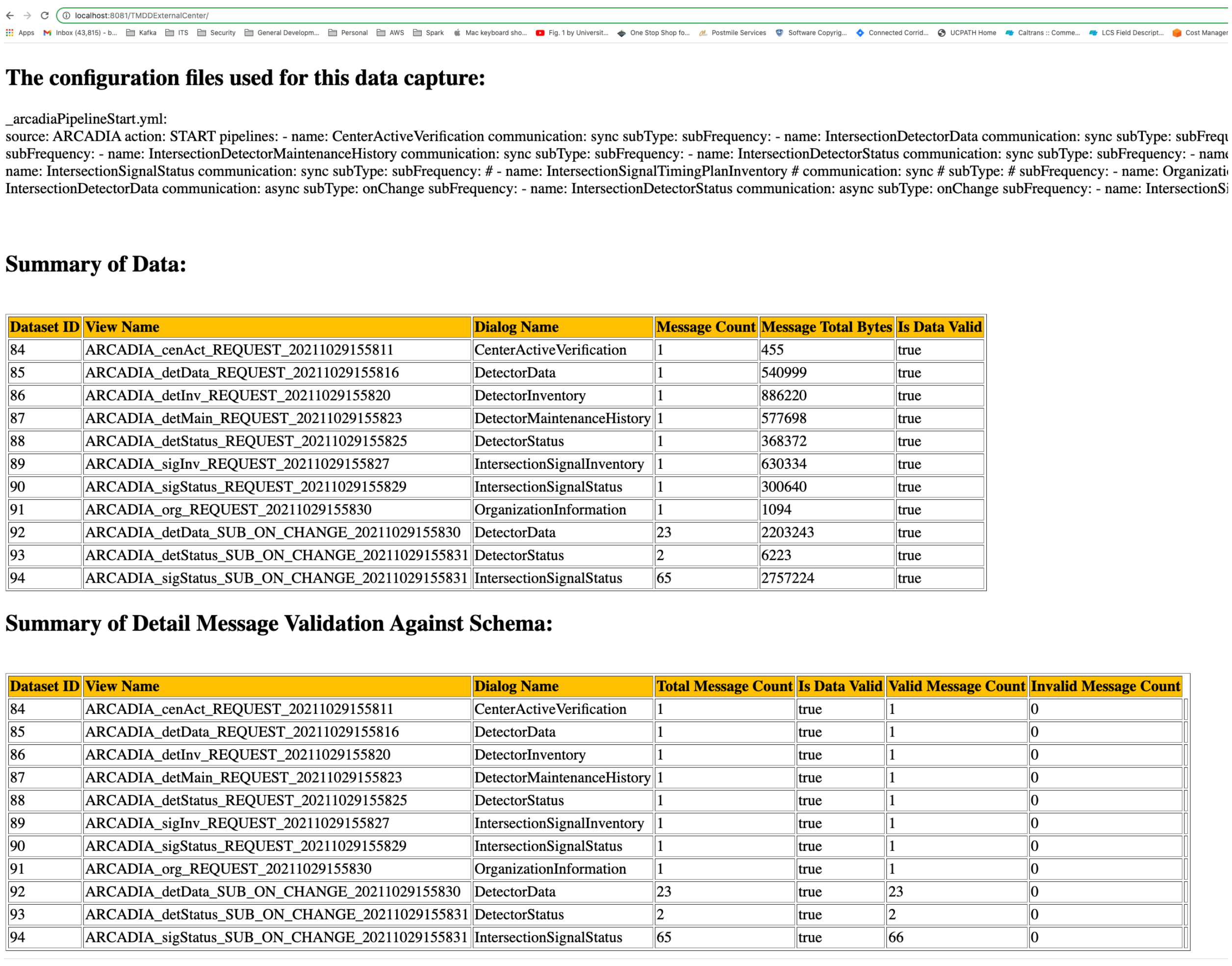
Figure 17 - Add connector Screen 2

g. Repeat steps e and f for the second connector in the pair for your desired data source and data type pipeline. Remember, this connector name will be prefixed with OC_to_EC, and the configuration file will be "connector_OC_to_EC_DATASOURCE_DATATYPE_config.json. Select the file when uploading for the same data source and data type selected for the first connector of the pair.

h. Open your Mongodb client and review the external center target collection for the pipeline created. The names of these collections begin with the prefix "xmlSourced." followed by the data type (such as IntersectionSignalStatus). The collection should start to fill with records from Connect, thereby transferring the data from the owner center to the external center.

i. Repeat steps e through h for each pipeline you want to create.

9. Verify SOAP service operation by opening the data validation report for SOAP services. The report can be located within your browser at the URL http://localhost:8081/TMDDExternalCenter/. A report similar to Figure 18 should be displayed. Note that the message counts for the subscription dialogs may be different, as messages are replayed based on the current time and the collection time, matching what would have been received from the field during the current time period. Request/Response message counts should always be 1 for a successful data exchange.



The configuration files used for this data capture:

_arcadiaPipelineStart.yml:
source: ARCADIA action: START pipelines: - name: CenterActiveVerification communication: sync subType: subFrequency: - name: IntersectionDetectorData communication: sync subType: subFrequ
subFrequency: - name: IntersectionDetectorMaintenanceHistory communication: sync subType: subFrequency: - name: IntersectionDetectorStatus communication: sync subType: subFrequency: - name
name: IntersectionSignalStatus communication: sync subType: subFrequency: # - name: IntersectionSignalTimingPlanInventory # communication: sync # subType: # subFrequency: - name: Organizati
IntersectionDetectorData communication: async subType: onChange subFrequency: - name: IntersectionDetectorStatus communication: async subType: onChange subFrequency: - name: IntersectionSi

Summary of Data:

Dataset ID	View Name	Dialog Name	Message Count	Message Total Bytes	Is Data Valid
84	ARCADIA_cenAct_REQUEST_20211029155811	CenterActiveVerification	1	455	true
85	ARCADIA_detData_REQUEST_20211029155816	DetectorData	1	540999	true
86	ARCADIA_detInv_REQUEST_20211029155820	DetectorInventory	1	886220	true
87	ARCADIA_detMain_REQUEST_20211029155823	DetectorMaintenanceHistory	1	577698	true
88	ARCADIA_detStatus_REQUEST_20211029155825	DetectorStatus	1	368372	true
89	ARCADIA_sigInv_REQUEST_20211029155827	IntersectionSignalInventory	1	630334	true
90	ARCADIA_sigStatus_REQUEST_20211029155829	IntersectionSignalStatus	1	300640	true
91	ARCADIA_org_REQUEST_20211029155830	OrganizationInformation	1	1094	true
92	ARCADIA_detData_SUB_ON_CHANGE_20211029155830	DetectorData	23	2203243	true
93	ARCADIA_detStatus_SUB_ON_CHANGE_20211029155831	DetectorStatus	2	6223	true
94	ARCADIA_sigStatus_SUB_ON_CHANGE_20211029155831	IntersectionSignalStatus	65	2757224	true

Summary of Detail Message Validation Against Schema:

Dataset ID	View Name	Dialog Name	Total Message Count	Is Data Valid	Valid Message Count	Invalid Message Count
84	ARCADIA_cenAct_REQUEST_20211029155811	CenterActiveVerification	1	true	1	0
85	ARCADIA_detData_REQUEST_20211029155816	DetectorData	1	true	1	0
86	ARCADIA_detInv_REQUEST_20211029155820	DetectorInventory	1	true	1	0
87	ARCADIA_detMain_REQUEST_20211029155823	DetectorMaintenanceHistory	1	true	1	0
88	ARCADIA_detStatus_REQUEST_20211029155825	DetectorStatus	1	true	1	0
89	ARCADIA_sigInv_REQUEST_20211029155827	IntersectionSignalInventory	1	true	1	0
90	ARCADIA_sigStatus_REQUEST_20211029155829	IntersectionSignalStatus	1	true	1	0
91	ARCADIA_org_REQUEST_20211029155830	OrganizationInformation	1	true	1	0
92	ARCADIA_detData_SUB_ON_CHANGE_20211029155830	DetectorData	23	true	23	0
93	ARCADIA_detStatus_SUB_ON_CHANGE_20211029155831	DetectorStatus	2	true	2	0
94	ARCADIA_sigStatus_SUB_ON_CHANGE_20211029155831	IntersectionSignalStatus	65	true	66	0

Figure 18 - SOAP Service Validation Report

10. While not required, it is recommended that after first start, you change the owner center and external center postgres instances. Complete the following to change the owner center and external center postgres passwords:
    a. The owner center postgres image is a fully populated database within the image. The external center is a fully structure database (users, roles, tables, etc.) that has not yet be populated with TMDD data. Upon first run of the application however, the data is populated within the external center database.
    b. In the Intellij IDE tmdd_owner_center project, enter the postgres password you choose within the tmdd_owner_center/owner_center/center_service/application.properties file. To do this, change the line within the file starting with postgres.qa.dataSource.pass replacing the password in the file with the desired password.
    c. In the postgres database, using the DataGrip postgres management tool or your preferred postgres management tool, change the password to match the password you set in the application.properties file.
    d. In Docker Desktop, restart the owner_center SOAP services container.

e. Repeat steps a through c for the external center postgres instance. For step b, the location of the application.properties file is tmdd_external_center/external_center/center_service/.
f. Restart the external center SOAP services container in Docker Desktop.
g. Allow the SOAP services to run and upon completion verify success in the SOAP Service Validation Report (Figure 18).

4.3.6. INSTALLATION COMPLETE

Installation is now complete and operation of the services has been completed. Both SOAP and Kafka traffic data services are installed and operation has been verified.

5. OPERATING INSTRUCTIONS

The application, once installed, is for the most part self-operating. The Kafka services will continue to listen for changes in the owner center Mongodb database and upon detecting a change in a collection for which a connector pair is established will transfer the data to the external center Mongodb database. The SOAP services however are generally a one-time operation that can be restarted by stopping and restarting the external center container within Docker Desktop. If changes to the dialogs or sources are desired within the SOAP services, the user can alter the application.properties and individual SOAP pipeline config files prior to restarting the external center SOAP services container as described in the installation instructions.

These operating instructions will provide instructions for the following:

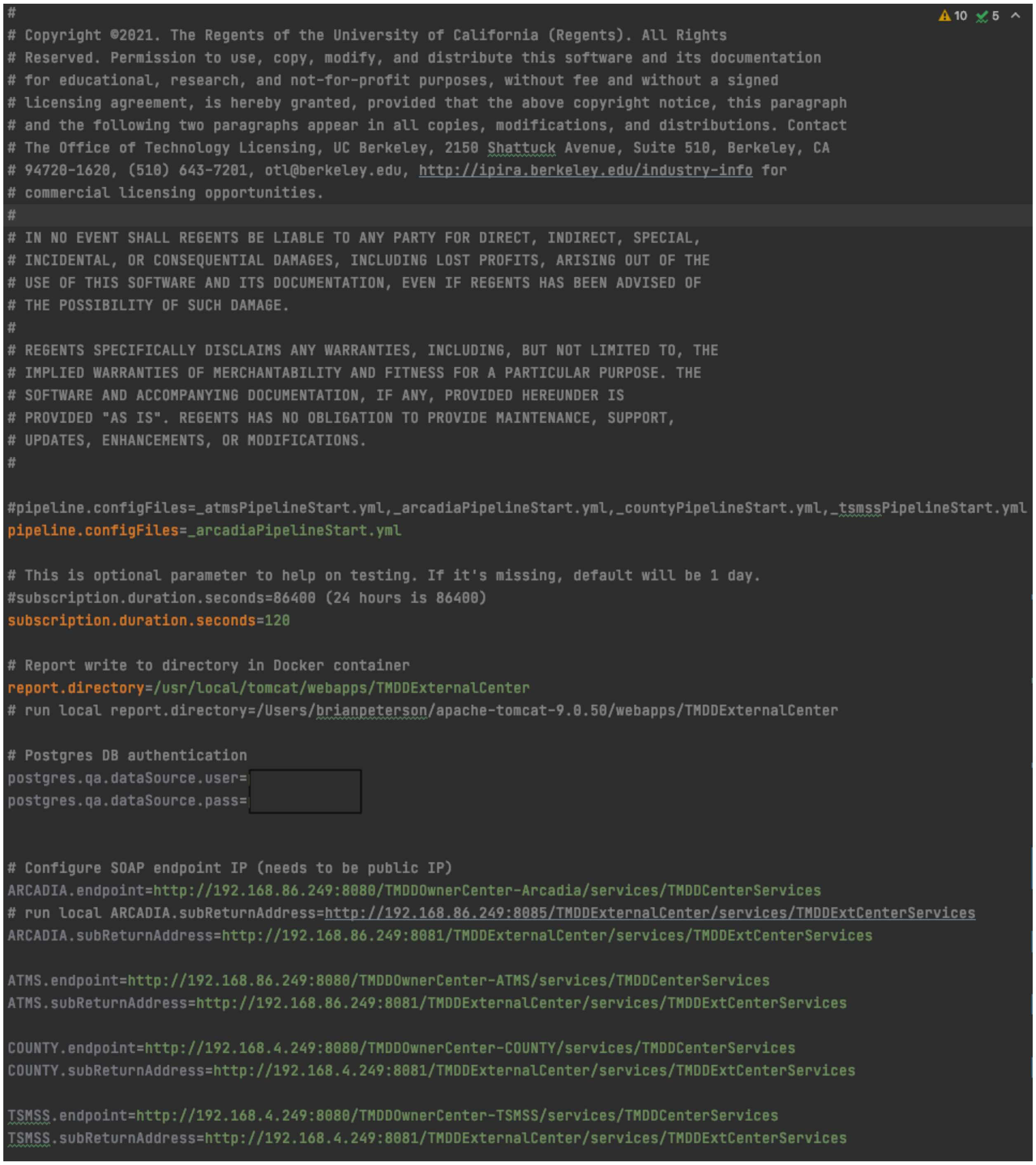
- How to restart the SOAP services pipeline
- How to restart from scratch an existing Kafka pipeline to demonstrate the pipeline with the data contained within the owner center's Mongodb collection for that pipeline.

5.1. RESTART A SOAP SERVICES PIPELINE

The SOAP services pipeline starts under the following conditions

- There is a properly configured owner center application/container available with associated and connected Postgres database with available data
- There is a properly configured external center SOAP application/container available with associated and connected Postgres database

The SOAP services start automatically upon startup of the external center services container. Upon startup, the external center service will retrieve the pipeline configuration to execute (located in the application.properties file). The pipeline configuration includes the list of pipeline config files to execute, subscription duration, report directory, and endpoint of each owner center service to connect to. See Figure 19 for an example application.properties file.



```
#
# Copyright ©2021. The Regents of the University of California (Regents). All Rights
# Reserved. Permission to use, copy, modify, and distribute this software and its documentation
# for educational, research, and not-for-profit purposes, without fee and without a signed
# licensing agreement, is hereby granted, provided that the above copyright notice, this paragraph
# and the following two paragraphs appear in all copies, modifications, and distributions. Contact
# The Office of Technology Licensing, UC Berkeley, 2150 Shattuck Avenue, Suite 510, Berkeley, CA
# 94720-1620, (510) 643-7201, otl@berkeley.edu, http://ipira.berkeley.edu/industry-info for
# commercial licensing opportunities.
#
# IN NO EVENT SHALL REGENTS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL,
# INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE
# USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF REGENTS HAS BEEN ADVISED OF
# THE POSSIBILITY OF SUCH DAMAGE.
#
# REGENTS SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE
# SOFTWARE AND ACCOMPANYING DOCUMENTATION, IF ANY, PROVIDED HEREUNDER IS
# PROVIDED "AS IS". REGENTS HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT,
# UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
#

#pipeline.configFiles=_atmsPipelineStart.yml,_arcadiaPipelineStart.yml,_countyPipelineStart.yml,_tsmssPipelineStart.yml
pipeline.configFiles=_arcadiaPipelineStart.yml

# This is optional parameter to help on testing. If it's missing, default will be 1 day.
#subscription.duration.seconds=86400 (24 hours is 86400)
subscription.duration.seconds=120

# Report write to directory in Docker container
report.directory=/usr/local/tomcat/webapps/TMDDExternalCenter
# run local report.directory=/Users/brianpeterson/apache-tomcat-9.0.50/webapps/TMDDExternalCenter

# Postgres DB authentication
postgres.qa.dataSource.user=[redacted]
postgres.qa.dataSource.pass=


# Configure SOAP endpoint IP (needs to be public IP)
ARCADIA.endpoint=http://192.168.86.249:8080/TMDDOwnerCenter-Arcadia/services/TMDDCenterServices
# run local ARCADIA.subReturnAddress=http://192.168.86.249:8085/TMDDExternalCenter/services/TMDDExtCenterServices
ARCADIA.subReturnAddress=http://192.168.86.249:8081/TMDDExternalCenter/services/TMDDExtCenterServices

ATMS.endpoint=http://192.168.86.249:8080/TMDDOwnerCenter-ATMS/services/TMDDCenterServices
ATMS.subReturnAddress=http://192.168.86.249:8081/TMDDExternalCenter/services/TMDDExtCenterServices

COUNTY.endpoint=http://192.168.4.249:8080/TMDDOwnerCenter-COUNTY/services/TMDDCenterServices
COUNTY.subReturnAddress=http://192.168.4.249:8081/TMDDExternalCenter/services/TMDDExtCenterServices

TSMSS.endpoint=http://192.168.4.249:8080/TMDDOwnerCenter-TSMSS/services/TMDDCenterServices
TSMSS.subReturnAddress=http://192.168.4.249:8081/TMDDExternalCenter/services/TMDDExtCenterServices
```

Figure 19 - External Center application.properties

The pipeline configuration also consists of each source pipeline config file specified at the top of application.properties. These files are located in the tmdd_exernal_center/external_center/center_service directory. See the installation instructions, Section 4.3.4, for instructions on editing the pipeline config files to change the active data pipelines.

Since the pipelines start upon container start, if it is desired to run again, the easiest method is to stop the container in Docker Desktop by clicking the stop icon to the right of the external_center container (see Figure 20).

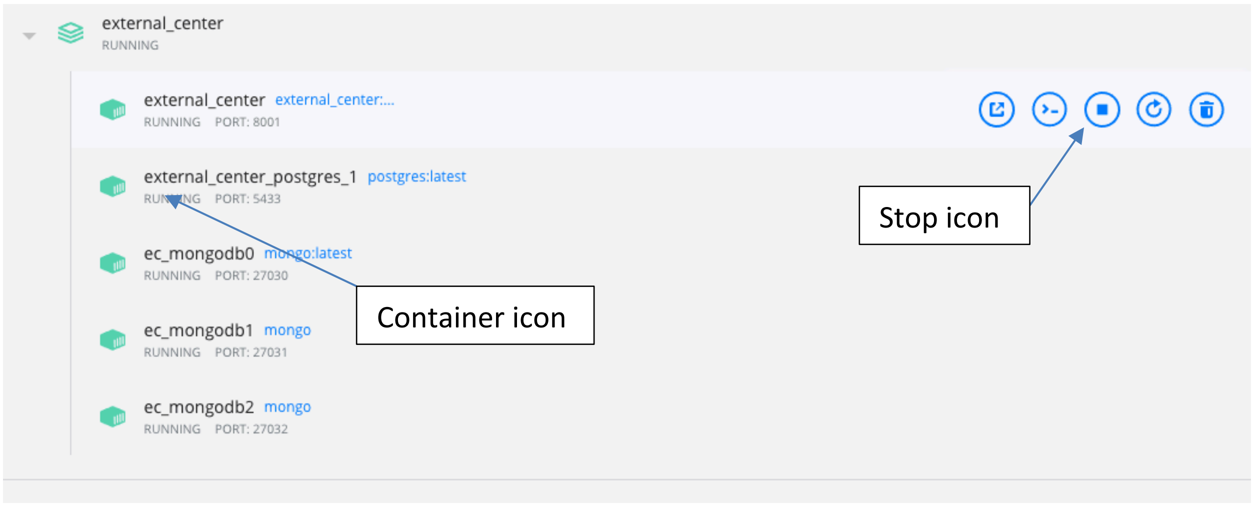
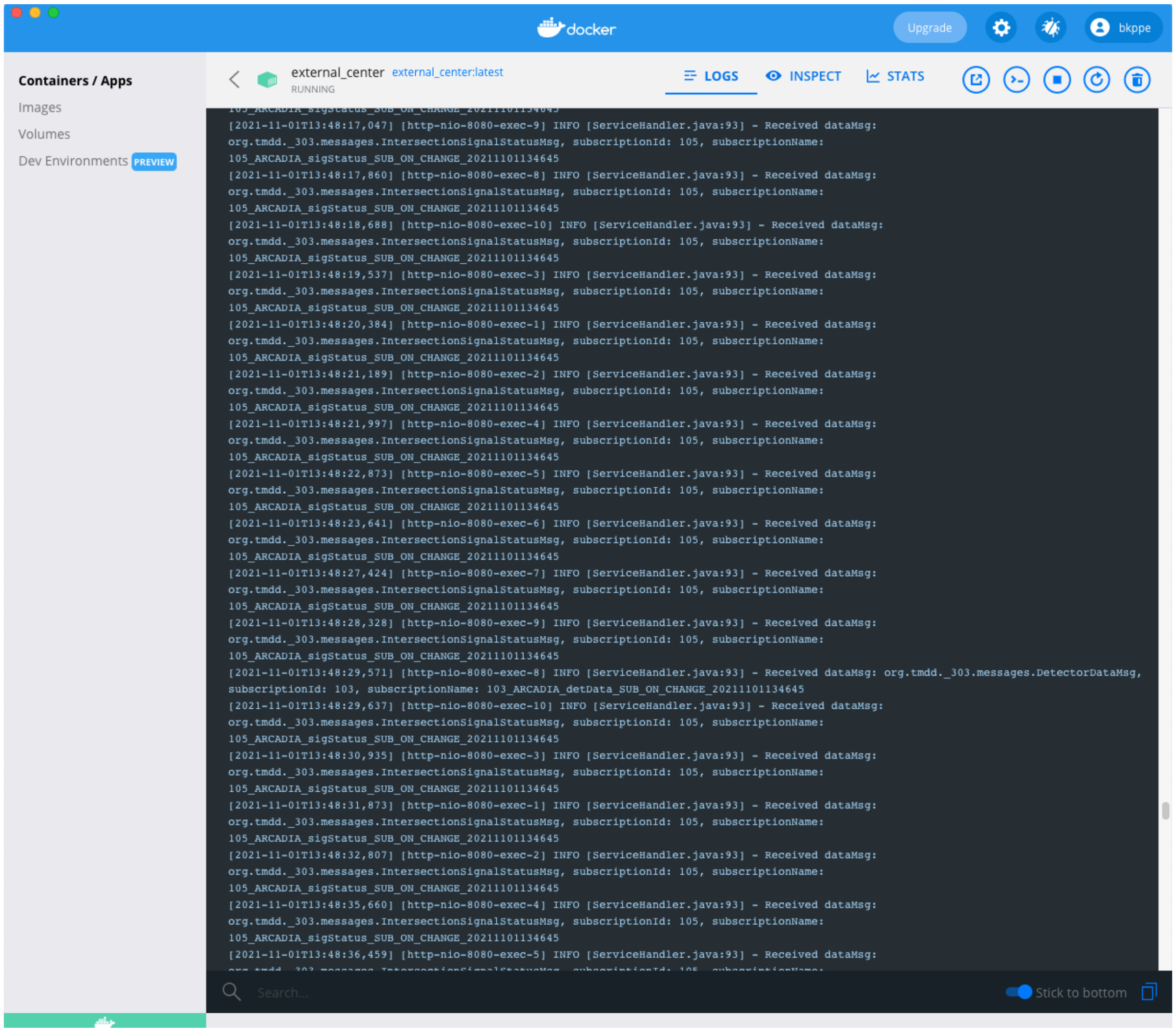


Figure 20 - Stop external center SOAP service container

If desired, alter the pipeline sources to run by altering the pipeline config file list in the application.properties file. If desired, alter which pipelines to run and the type of pipeline (request/response [sync] or subscription[async] and type of subscription and parameters) by changing the pipeline config file for the selected source. The center_service folder is mounted in the container, so the container does not require a rebuild. Simply restart the external center container in Docker desktop by clicking the start icon (replaces the stop icon when the container is stopped). The SOAP pipelines will automatically start as the container starts. You can watch the log to see the log messages for data flow by clicking on the external_center container icon in the Docker Desktop application. See Figure 20 for container icon location and Figure 21 for example logs. The service does take several seconds after container start to begin processing the data.



```
[2021-11-01T13:48:17,047] [http-nio-8080-exec-9] INFO [ServiceHandler.java:93] - Received dataMsg:
org.tmdd._303.messages.IntersectionSignalStatusMsg, subscriptionId: 105, subscriptionName:
105_ARCADIA_sigStatus_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:17,860] [http-nio-8080-exec-8] INFO [ServiceHandler.java:93] - Received dataMsg:
org.tmdd._303.messages.IntersectionSignalStatusMsg, subscriptionId: 105, subscriptionName:
105_ARCADIA_sigStatus_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:18,688] [http-nio-8080-exec-10] INFO [ServiceHandler.java:93] - Received dataMsg:
org.tmdd._303.messages.IntersectionSignalStatusMsg, subscriptionId: 105, subscriptionName:
105_ARCADIA_sigStatus_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:19,537] [http-nio-8080-exec-3] INFO [ServiceHandler.java:93] - Received dataMsg:
org.tmdd._303.messages.IntersectionSignalStatusMsg, subscriptionId: 105, subscriptionName:
105_ARCADIA_sigStatus_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:20,384] [http-nio-8080-exec-1] INFO [ServiceHandler.java:93] - Received dataMsg:
org.tmdd._303.messages.IntersectionSignalStatusMsg, subscriptionId: 105, subscriptionName:
105_ARCADIA_sigStatus_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:21,189] [http-nio-8080-exec-2] INFO [ServiceHandler.java:93] - Received dataMsg:
org.tmdd._303.messages.IntersectionSignalStatusMsg, subscriptionId: 105, subscriptionName:
105_ARCADIA_sigStatus_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:21,997] [http-nio-8080-exec-4] INFO [ServiceHandler.java:93] - Received dataMsg:
org.tmdd._303.messages.IntersectionSignalStatusMsg, subscriptionId: 105, subscriptionName:
105_ARCADIA_sigStatus_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:22,873] [http-nio-8080-exec-5] INFO [ServiceHandler.java:93] - Received dataMsg:
org.tmdd._303.messages.IntersectionSignalStatusMsg, subscriptionId: 105, subscriptionName:
105_ARCADIA_sigStatus_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:23,641] [http-nio-8080-exec-6] INFO [ServiceHandler.java:93] - Received dataMsg:
org.tmdd._303.messages.IntersectionSignalStatusMsg, subscriptionId: 105, subscriptionName:
105_ARCADIA_sigStatus_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:27,424] [http-nio-8080-exec-7] INFO [ServiceHandler.java:93] - Received dataMsg:
org.tmdd._303.messages.IntersectionSignalStatusMsg, subscriptionId: 105, subscriptionName:
105_ARCADIA_sigStatus_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:28,328] [http-nio-8080-exec-9] INFO [ServiceHandler.java:93] - Received dataMsg:
org.tmdd._303.messages.IntersectionSignalStatusMsg, subscriptionId: 105, subscriptionName:
105_ARCADIA_sigStatus_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:29,571] [http-nio-8080-exec-8] INFO [ServiceHandler.java:93] - Received dataMsg: org.tmdd._303.messages.DetectorDataMsg,
subscriptionId: 103, subscriptionName: 103_ARCADIA_detData_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:29,637] [http-nio-8080-exec-10] INFO [ServiceHandler.java:93] - Received dataMsg:
org.tmdd._303.messages.IntersectionSignalStatusMsg, subscriptionId: 105, subscriptionName:
105_ARCADIA_sigStatus_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:30,935] [http-nio-8080-exec-3] INFO [ServiceHandler.java:93] - Received dataMsg:
org.tmdd._303.messages.IntersectionSignalStatusMsg, subscriptionId: 105, subscriptionName:
105_ARCADIA_sigStatus_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:31,873] [http-nio-8080-exec-1] INFO [ServiceHandler.java:93] - Received dataMsg:
org.tmdd._303.messages.IntersectionSignalStatusMsg, subscriptionId: 105, subscriptionName:
105_ARCADIA_sigStatus_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:32,807] [http-nio-8080-exec-2] INFO [ServiceHandler.java:93] - Received dataMsg:
org.tmdd._303.messages.IntersectionSignalStatusMsg, subscriptionId: 105, subscriptionName:
105_ARCADIA_sigStatus_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:35,660] [http-nio-8080-exec-4] INFO [ServiceHandler.java:93] - Received dataMsg:
org.tmdd._303.messages.IntersectionSignalStatusMsg, subscriptionId: 105, subscriptionName:
105_ARCADIA_sigStatus_SUB_ON_CHANGE_20211101134645
[2021-11-01T13:48:36,459] [http-nio-8080-exec-5] INFO [ServiceHandler.java:93] - Received dataMsg:
```

Figure 21 - SOAP Services External Center SOAP Service Logs

Verify that the service completed after the subscription duration has elapsed by reviewing the SOAP services validation report.

5.2. RESTART A KAFKA PIPELINE

As discussed during the installation procedure, Section 4.3.5, paragraph 8.e, each Kafka pipeline is defined within a pair of Kafka connectors:

- A connector that:
  - Reads all data initially, and then monitors for updates, from the owner center Mongodb collection containing a set of data for a single source
  - Places the data read from Mongodb into a Kafka topic
- A second connector that:

- Pulls the data from the Kafka topic
- Writes the data from the topic to the external center Mongodb replication cluster in a collection specific to the data type being sent from the owner center

As a result, once the initial data is loaded, there will be no further activity once the initial data transfer is complete without one of two actions:

- New data records are added to the owner center source collection for the pipeline
- A new connector is added that will restart the data flow (it is recommended that to keep clutter down in the list of connectors, that any time a connector is added to restart data flow, that the connector be deleted upon completion of data transfer)

In a production system, new data would regularly flow into the Mongodb source collection as data is received from field units. However, in an isolated environment such as this application, there is no external source. Records can be manually added if desired and the user should have an appropriate set of data and consult the Mongodb documentation on how to insert records.

To restart a pipeline with a new connector, complete the following:

1. Open the Mongodb client and delete all documents in the external center Mongodb target collection. The collection name will be prefixed with “xmlSourced.” and will be named based on the data type being transferred within the pipeline. Note that you can only delete the documents from a connection to the cluster or from the primary node of the cluster. If you are using Robo 3T, you can right click on the pipeline target collection and select “Remove All Documents…” as illustrated in Figure 22. The transfer will proceed without this step, but since the documents are duplicates of what was originally transferred, it is easier to verify that the transfer was completed via the record count within the collection. Verify that the collection now has 0 documents using the “Statistics…” option in the collection context menu.

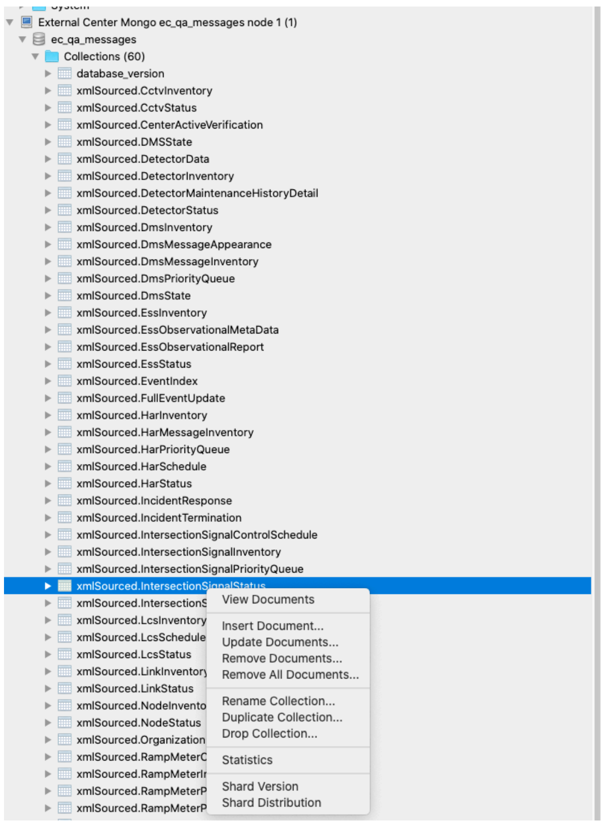
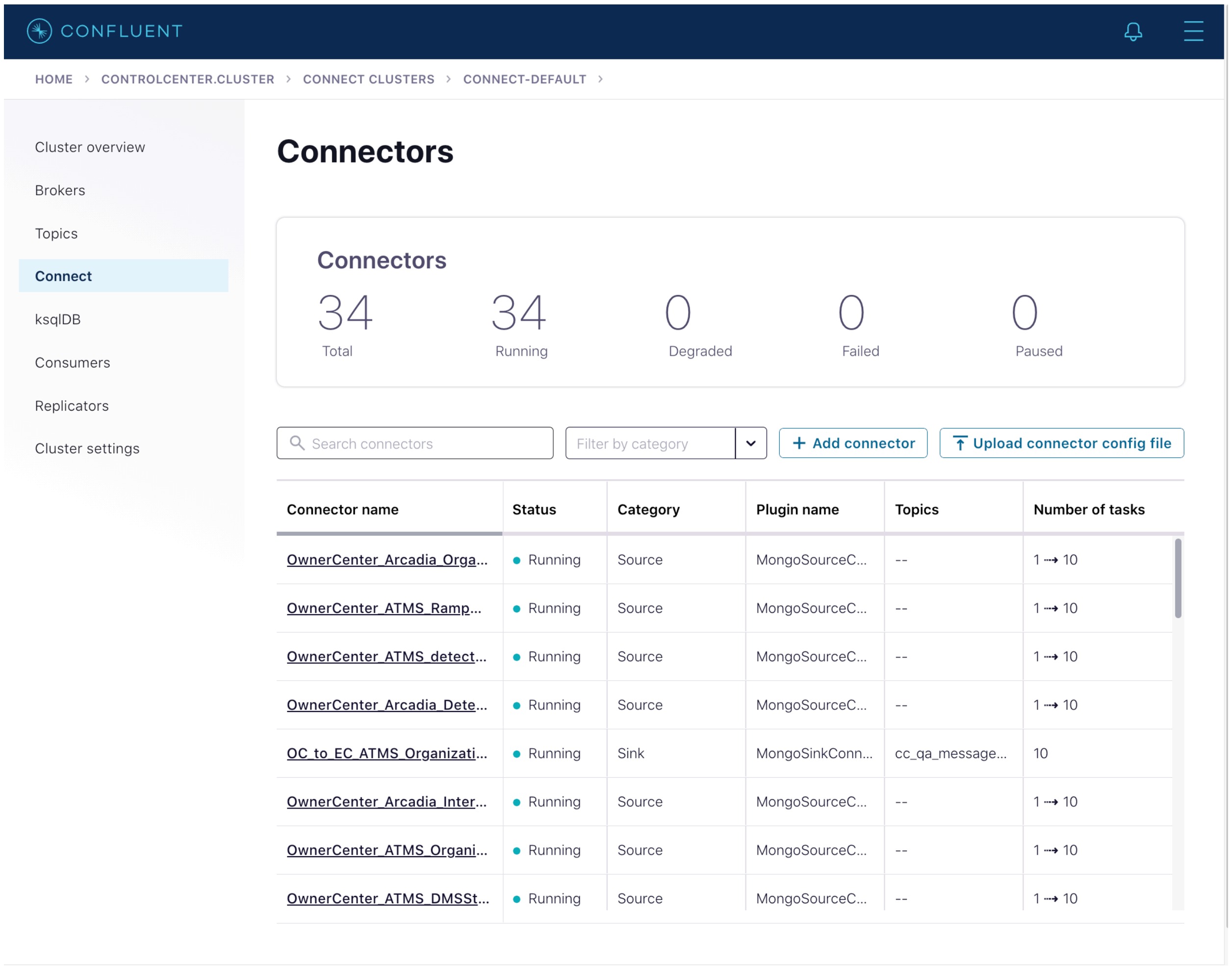


Figure 22 - Mongodb Collection Delete All Documents

2. Open the Kafka Control Center in your browser and navigate to the list of connectors.



Connectors

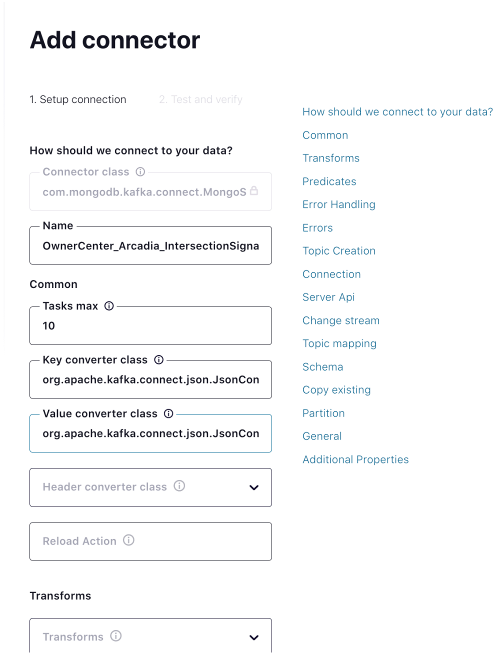
Connectors

34	34	0	0	0
Total	Running	Degraded	Failed	Paused

Connector name	Status	Category	Plugin name	Topics	Number of tasks
OwnerCenter_Arcadia_Orga...	● Running	Source	MongoSourceC...	--	1 → 10
OwnerCenter_ATMS_Ramp...	● Running	Source	MongoSourceC...	--	1 → 10
OwnerCenter_ATMS_detect...	● Running	Source	MongoSourceC...	--	1 → 10
OwnerCenter_Arcadia_Dete...	● Running	Source	MongoSourceC...	--	1 → 10
OC_to_EC_ATMS_Organizati...	● Running	Sink	MongoSinkConn...	cc_qa_message...	10
OwnerCenter_Arcadia_Inter...	● Running	Source	MongoSourceC...	--	1 → 10
OwnerCenter_ATMS_Organi...	● Running	Source	MongoSourceC...	--	1 → 10
OwnerCenter_ATMS_DMSSt...	● Running	Source	MongoSourceC...	--	1 → 10

Figure 23 - List of Kafka Connectors

3. Click on the “Upload connector config file” button. In the file dialog box, navigate to and select the connector configuration file for the pipeline you wish to restart. Select the configuration file for which the name begins with “OwnerCenter” followed by the pipeline source (ATMS or Arcadia) and the data type for the pipeline. The Control Center will load the configuration file and display the Add connector screen populated with the parameters for the selected connector (Figure 23).



Add connector

1. Setup connection   2. Test and verify

How should we connect to your data?

Connector class: com.mongodb.kafka.connect.MongoS

Name: OwnerCenter_Arcadia_IntersectionSigna

Common

Tasks max: 10

Key converter class: org.apache.kafka.connect.json.JsonCon

Value converter class: org.apache.kafka.connect.json.JsonCon

Header converter class

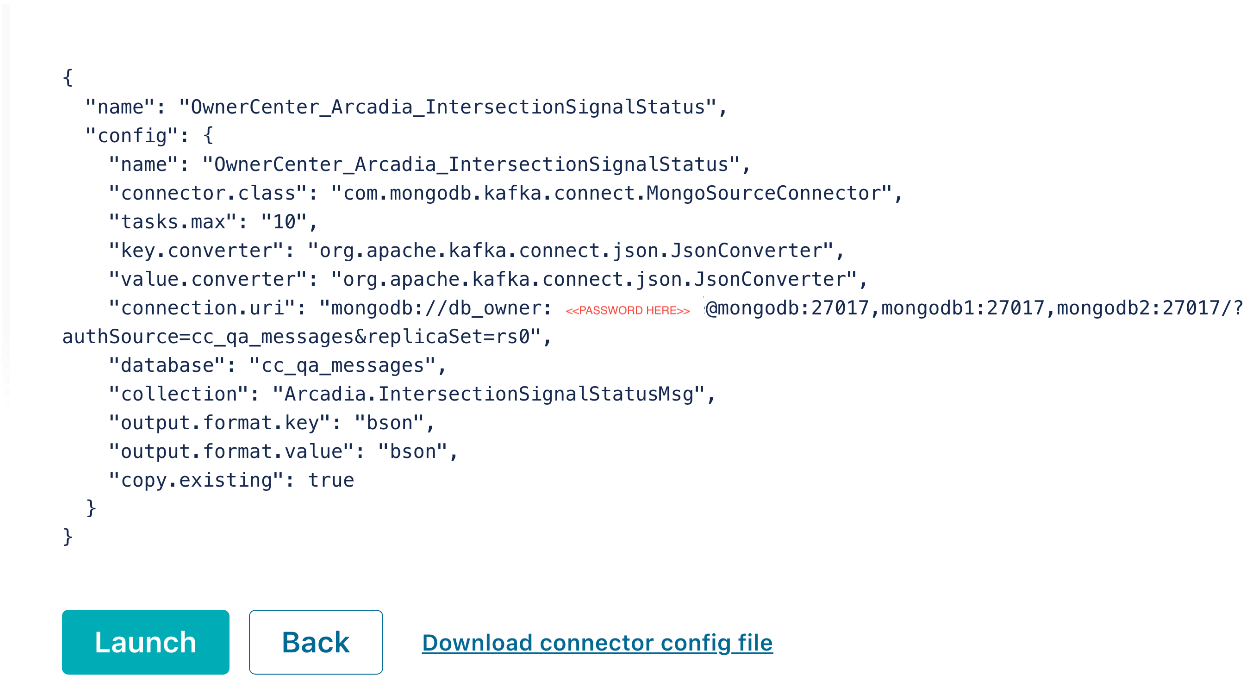
Reload Action

Transforms

Transforms

Figure 24 - Add Connector Screen

4. Change the connector name (Kafka Connect will not allow duplicate connectors). Scroll down to the bottom of the screen and click Next. Control Center will display the pipeline configuration and the Launch button will be displayed (Figure 25). Click Launch to start the connector.



```
{
  "name": "OwnerCenter_Arcadia_IntersectionSignalStatus",
  "config": {
    "name": "OwnerCenter_Arcadia_IntersectionSignalStatus",
    "connector.class": "com.mongodb.kafka.connect.MongoSourceConnector",
    "tasks.max": "10",
    "key.converter": "org.apache.kafka.connect.json.JsonConverter",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "connection.uri": "mongodb://db_owner: <<PASSWORD HERE>> @mongodb:27017,mongodb1:27017,mongodb2:27017/?
authSource=cc_qa_messages&replicaSet=rs0",
    "database": "cc_qa_messages",
    "collection": "Arcadia.IntersectionSignalStatusMsg",
    "output.format.key": "bson",
    "output.format.value": "bson",
    "copy.existing": true
  }
}
```

Figure 25 - Launch Connector Screen

5. Connect will start the connector and Control Center will now list the new connector within the list of connectors. If you see a connector fail status, give the connector a few seconds to see if it starts before troubleshooting.
6. Verify that the data is flowing.
    a. You can verify that the Kafka topic that is being populated is receiving data in the Control Center Topics screen for the topic receiving data (click Topics on the Control Center left menu and then select the topic for the specific source and data type to see the topic screen for that pipeline topic. The production and Consumption graphs should indicate data is flowing.
    b. You can verify the data is being received within the external center Mongodb database collection by returning to the Mongodb client and checking the document count for the target collection. You should see the document count increase from 0 to the number of documents within the owner center source collection.
7. It is recommended that you delete the added connector or the original connector to ensure only one Mongo – Topic connector is running at any time for a specific source and data type.

This page left blank intentionally