Collaborative Network for Industry, Manufacturing, Business and Logistics in Europe

D3.5

Distributed Automation: Channel Management for Production Data Sharing

Project Acronym	NIMBLE
Project Title	Collaboration Network for Industry, Manufacturing, Business and Logistics in Europe
Project Number	723810
Work Package	WP3 Core Business Services for the NIMBLE Platform
Lead Beneficiary	
Editor	Benny Mandler (IBM)
Reviewers	W.Behrendt (SRFG), Benny Mandler
Contributors	Johannes Innerbichler (SRFG); Suat Gonul (SRDC), Evgeniy Hvostenko (IBM), W.Behrendt (SRFG)
Dissemination Level	
Contractual Delivery Date	
Actual Delivery Date	
Version	V1.0

Abstract

This deliverable is intended to provide a short high-level view of the design, implementation, and deployment of the Distributed Automation: Channel Management for Production Data Sharing capability, which gained the nick name of data channels within the NIMBLE platform. Data channels refer to the capability of having one company providing another company a controlled glimpse into its internal data in real-time. This capability is made available as the result of a business relation between two entities, following the establishment of an agreed upon business process, via which the extent and characteristics of the data to be shared are agreed. Thus, the business relations include creating a private communication link between the companies and enabling selective data sharing among entities.

NIMBLE in a Nutshell

NIMBLE is the collaboration Network for Industry, Manufacturing, Business and Logistics in Europe. It will develop the infrastructure for a cloud-based, Industry 4.0, Internet-of-Things-enabled B2B platform on which European manufacturing firms can register, publish machine-readable catalogues for products and services, search for suitable supply chain partners, negotiate contracts and supply logistics. In particular, this deliverable focuses on the capability of participating companies to establish private and secure B2B and M2M information exchange channels to optimize business work flows between them. The infrastructure will be developed as open source software under an Apache-type, permissive license. The governance model is a federation of platforms for multi-sided trade, with mandatory interoperation functions and optional added-value business functions that can be provided by third parties. This will foster the growth of a net-centric business ecosystem for sustainable innovation and fair competition as envisaged by the Digital Agenda 2020. Prospective NIMBLE providers can take the open source infrastructure and bundle it with sectorial, regional or functional added value services and launch a new platform in the federation. The project started in October 2016 and will last for 36 months.

Document History

Version	Date	Comments
V0.1	03/04/2018	First version by IBM
V0.2	25/05/2018	Incorporate contributions from SRFG
V0.3	30/05/2018	Input from SRDC
V0.4	15/06/2018	Latest sequence diagrams added
V0.5	22/06/2018	Additions by SRFG/wb; Review by IBM/bm
V0.6	26/06/2018	Final revision by SRFG/wb
V1.0	27/06/2018	Final edits and submission by SRFG/wb

Table of Contents

List of Figures

List of Tables

Acronyms

Table 1: Acronyms table

Acronym	Meaning
API	Application Programming Interface
B2B	Business to business
IoT	Internet of Things
JSON	Java Script Object Notation
JVM	Java virtual machine
NIMBLE	Collaboration Network for Industry, Manufacturing, Business and Logistics in Europe
PaaS	Platform as a Service
REST	Representational State Transfer
UUI	Universally unique identifier

1 Introduction

This document accompanies the demonstration of the data channels capability within the NIMBLE platform, providing the necessary background information including high level design, deployment, and interaction with additional platform components.

The main service provided by the data channels enables one company with a scoped and secure glimpse into internal information of another company, following an agreed upon characteristics of the data to be shared. For example, this capability may provide a buyer of a certain product a glimpse into progress made by the producer of the product in respect to the production of an agreed upon order. The buyer can have a real-time view of the progress made towards completing the order, thus it may be reassured that the delivery date is within reason or be able to identify early signs of delays and be able to devise a "plan B" if necessary.

The design presented is scalable and flexible in the sense that it can support many such data channels in parallel, potentially involving different players, and can sustain a high rate of information flow within these channels.

The design builds upon established open source projects, chief among them is Apache Kafka[1], making use of available advanced features in particular streaming[2], while adding the specific capabilities required to realize the data channels. The relevant flow consists of the following stages:

- A negotiation phase in which the scope of the visibility of a company data to another company is agreed upon and registered in the NIMBLE platform in the form of a filter.
- The company providing its data hooks up its data producer through a Kafka topic provided by NIMBLE.
- All incoming messages flow into the streams processing component.
- The streams processing component, based on the characteristic of the incoming message, fetches the relevant filter, if this is the first encounter of that specific filter and caches it internally.
- The correct filter is applied to the incoming messages and only messages that successfully abide by the constraints dictated by the filter are sent to a specific output topic.
- The company receiving the data can access it from the relevant output topic. As it is envisioned that most interactions of end users with the platform would be performed within the scope of NIMBLE platform services, all incoming data will be saved by the platform, and presented upon polling for that information by the receiving company, using the NIMBLE front-end web application. In the future REST endpoints may be established for pushing data into and out of the platform, to hook up to companies' internal systems.

The rest of this deliverable provides the conceptual design of the data channels component, followed by a brief description of the interactions needed to set up a data channel; deployment aspects are discussed next. Last, we elaborate on an example usage of the data filtering capability.

[1] https://kafka.apache.org/

[2] https://kafka.apache.org/documentation/streams/

2 Conceptual design

As described above the main aim of the data channels component is to enable a company (company A) to securely share an agreed upon pre-defined sub-set of its data with another company (company B). To that end the design covers a setup phase followed by a run-time realization.

The set up phase is realized within the NIMBLE platform as a part of the negotiation process (see Figure 1); thus, a part of the business process agreed upon between both companies comprises the characteristics of the data that will be shared by company A and will be made available to company B. The negotiation process may be comprised of several rounds in which both sides attempt to reach mutual agreement.

During the negotiation process for a supply chain contract, the parties may agree that the buyer party will be allowed to monitor parts of the other party's manufacturing process at machine data level, e.g. to monitor a specific quality parameter. For this to happen, at least one local ICT service of the supplier has to be synchronised with NIMBLE. That synchronisation needs to connect the business process (NIMBLE Contract-ID) with the manufacturing process that the supplier has planned to satisfy the order. The agreement that is reached during a NIMBLE business negotiation leads to the definition of a corresponding clause in a formal business contract. The following scenario illustrates this:

Company A is a 3D printer farm with 20 different 3D printers that are online. Company A does business as a supplier with many other manufacturing firms, one of them is Company B who is looking for a metal printer to build a small set of 10 prototypes. In particular, they want to see and record the quality of the print while their artefacts are being printed. Since all of the printers are equipped with high resolution cameras, Company A agrees in the negotiation that Company B will be able to record the printing process exactly when their products are being built. This agreement is being recorded in a clause of the contract about the order for the prototype prints.

We assume that Company A has several metal printers so the 10 prototype items are printed on 3 different printers and thus, the printing process is recorded by 3 different cameras each belonging to one of the printers, according to the production schedule.

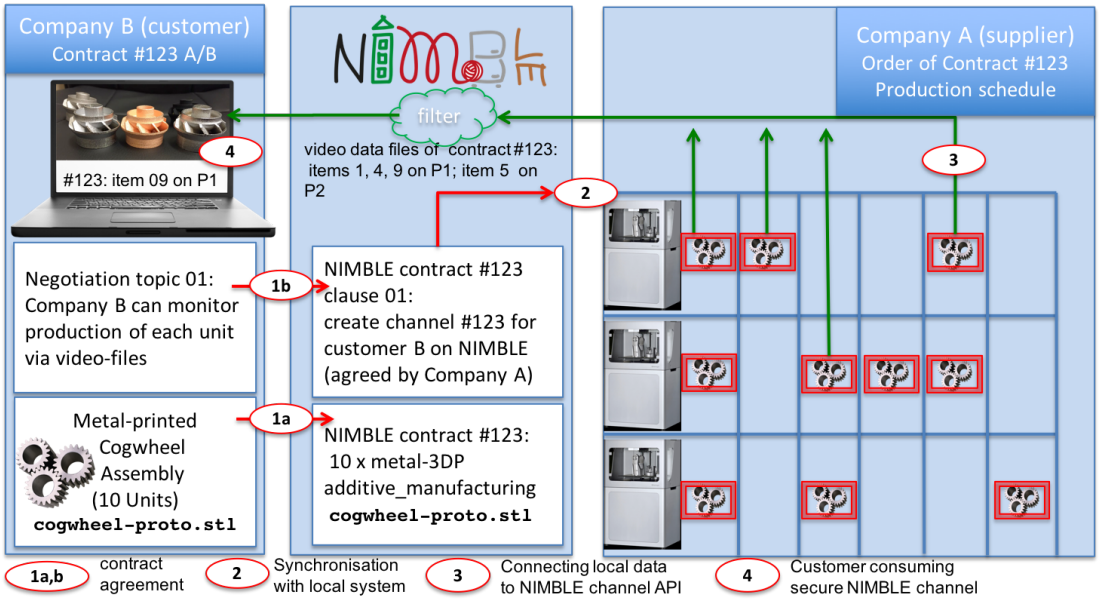


Figure 1: Creating a NIMBLE data channel and synchronizing a local system

As can be seen in Figure 1, in Steps 1a and 1b the partners agree on the terms and on the permission for Company B to monitor the printing process at Company A, for each of the 10 items. As a result of this NIMBLE creates an abstract data channel and makes its endpoint known. Step 2 establishes the connection between the NIMBLE platform and the local system: the local system now connects the contract ID with the local production schedule. In the first instance, we envisage this to be a manual or semi-automatic step, but in the future, this synchronization should be possible automatically. Step 3 establishes the concrete mappings of machines and sensors (in our case, 3D printers and their video cameras) to the previously established NIMBLE channel endpoint. Step 4 is then the consumption of the video data by the customer, according to the previously negotiated agreement. In the figure above we only show 4 of the 10 video objects (green arrows) purely for reasons of better legibility of the diagram.

2.1 Contracts and Clauses

We define a contract to be the result of a negotiation that happened via NIMBLE. A contract is a formal data structure consisting of clauses.

NIMBLE contracts thus contain the promises coming out of the collaboration activities between the trading companies. Contracts can be associated to business processes supported by NIMBLE e.g. the order process and transport execution plan. All the preceding collaboration activities (i.e. the processes executed between the companies) are included as clauses of the contract.

For example, assuming that the companies performed a PPAP and a negotiation process before the order, the order contract would include two Document Clauses referring to the documents exchanged in the PPAP and negotiation processes.

Related to the data channels, the negotiation process includes a term being used through which the buyer company can request data monitoring service for the production processes of the product as the subject of negotiation. As a response to this request, the seller/manufacturer company provides its response indicating the acceptance of the term. So, once an order following a negotiation process with an agreement on the data monitoring service provision leads instantiation of a data channel between the trading companies as elaborated in Section 3.

```
Clause =
  c_clause(cls_001, //ID of the clause within the contract
       from_date (?StartDate), // validity of the clause
       to_date (?EndDate), // end of validity of the clause
           can_monitor(
              user(?MonitoringPerson_ID, ?MonitoringFirm_ID), // this user
              company(?MonitoredFirm_ID), // in that firm
              machine(?MonitoredMachine_ID), // from that machine
              sensors(?MonitoredSensorsList), // data from that sensor
              when( // on condition that
                machined_part(?MP_ID), // the machined part on that machine
                of_order(?OrderOfMachinedPart) // belongs to this order
  ))).
```

The above design is written as a logic programming predicate (upper case terms are variables) and can be translated into an equivalent JSON structure, for implementation.

At the time of agreeing on the contract, some elements of this structure may not be known and fixed yet. For example, if there are several metal printers available then any one of them may be chosen at the time when the order will actually be processed. Therefore, it is also not possible to set at this stage the individual camera that will do the recording, because that too, will depend on the chosen printer and on the concrete production schedule.

In the future, we also want to let users negotiate on the specific data sources related to a production process of an order. This will be enabled by first letting users to register their IoT data sources on NIMBLE. For example, considering the example above users will be able to register a metal printer with a set of sensors. Then, the registered data sources will be provided as negotiable terms e.g. to share the camera data of the printer but not the data of its humidity sensor.

The technical concept how NIMBLE and a local manufacturing system can be connected for the data exchange, is elaborated in the next section.

2.2 Technical Concept

A successful completion of the negotiation phase triggers the creation of a data channel between company A and company B. The creation of the data channel includes the definition of the topic through which the data will flow into the platform, the creation and storage of a filter to be applied to the incoming data providing the rules restricting the data that will be made available to company B. Finally, the output topic through which only allowed data shall flow is defined, enabling company B to subscribe to the messages flow. As mentioned above, when accessing all the capabilities via the NIMBLE platform, data flowing to the output topic will be stored in a DB and be made available to company B upon polling from the platform's front-end.

At run-time the design is comprised of an entry point, in which company A hooks up its produced data source to the NIMBLE platform, by publishing that information as messages using an agreed upon Kafka topic with an agreed format. Messages consist of a header and a payload. The header contains the information by which the filtering will take place, while the payload is the actual data to be seen by the counterpart, and is opaque as far as the data channels infrastructure is concerned. At the heart of the data channels component is a filter which is applied to all incoming data using the Kafka streams capability. The filtering process identifies the filter that needs to be applied on a per message basis, fetches it from a DB if required (and proceeds to cache it so that it is readily available for future use), and finally applies the rules of the filter to the messages in flight. Messages that successfully passed the filtering process are forwarded to the defined output topic for the specific interaction taking place, thus made available to the receiving company (company B). The process is depicted at a high level in Figure 2.

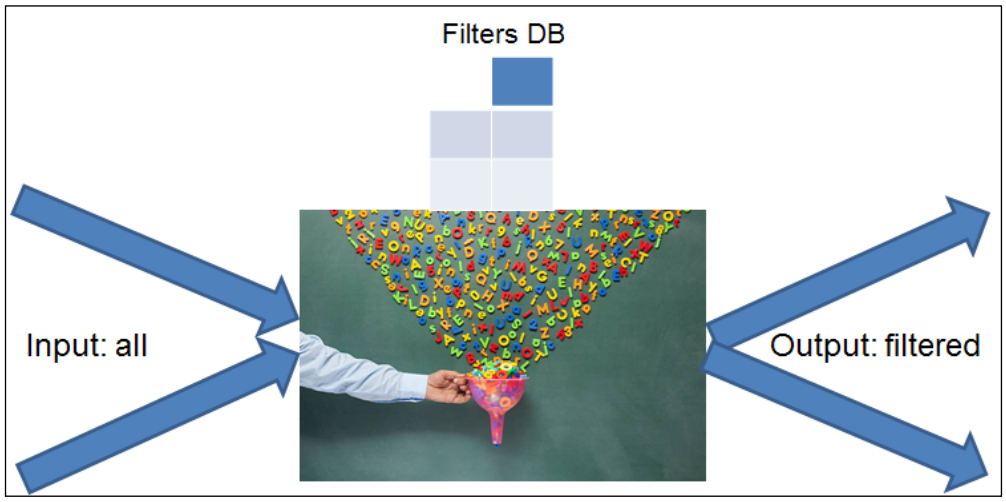


Figure 2: Data channels at run-time

The main component used to realize the data channels is the open source messaging and streaming service Kafka. This service enables the flow of data from producers to consumers in a distributed and asynchronous manner, without the different parties necessarily being aware of each other or the manner in which they can interact with other processes. Moreover, different parties to the interaction do not need to be operational at the same time, but rather the infrastructure will ensure the proper flow of data once entities connect to the system. In addition, the streaming capabilities enable the constant processing of data in real-time as it is being ingested into the system, processing it in a programmatic manner based on the platform needs.

As can be seen in Figure 2, the platform supports several data channels to be run in parallel. Each data channel is identified by a unique identifier that may be comprised of the input topic or added to the messages as a uniquely identifying field. The central streaming capability registers itself as a consumer of every topic which is used as a data channel input, thus all data published reaches the streaming component which implements in turn the filtering capability. Since multiple data channels may share the same topology, the unique identifier mentioned above is used for retrieving and applying the correct filter corresponding to the specific message that is being processed. The specific filter shall determine whether the data in the message should be forwarded further to the end recipient or not. Similarly, several different output topics are defined, and for each data channel the intended recipient of the data is connected to a specific output topic via which only the relevant information will flow. Thus, several such data channels interactions can be supported in parallel by the same infrastructure for different interactions.

The design further calls for a stable storage option for entities that work exclusively via the platform front-end service, and do not introduce additional software processes in their own premises, such as a messaging infrastructure client or a REST endpoint. At this time, we assume that all interactions with the NIMBLE platform are performed via the web based front-end. To that end an internal messages consumer registers itself as a listener to topics which are the output of the filtering process, and proceeds to store each message received in a relational DB. That information is later pulled by the end-users of the platform once they log in and ask for information related to their active data channels. Only the intended recipient will gain access to that information once they have been properly identified, authenticated and authorized by the platform.

2.3 Technical Architecture

The system must be integrated into the existing infrastructure of NIMBLE. Therefore, requirements of the applied architecture are easy integration and modularity. It can thus be adapted to individual use cases, without modules interfering with each other.

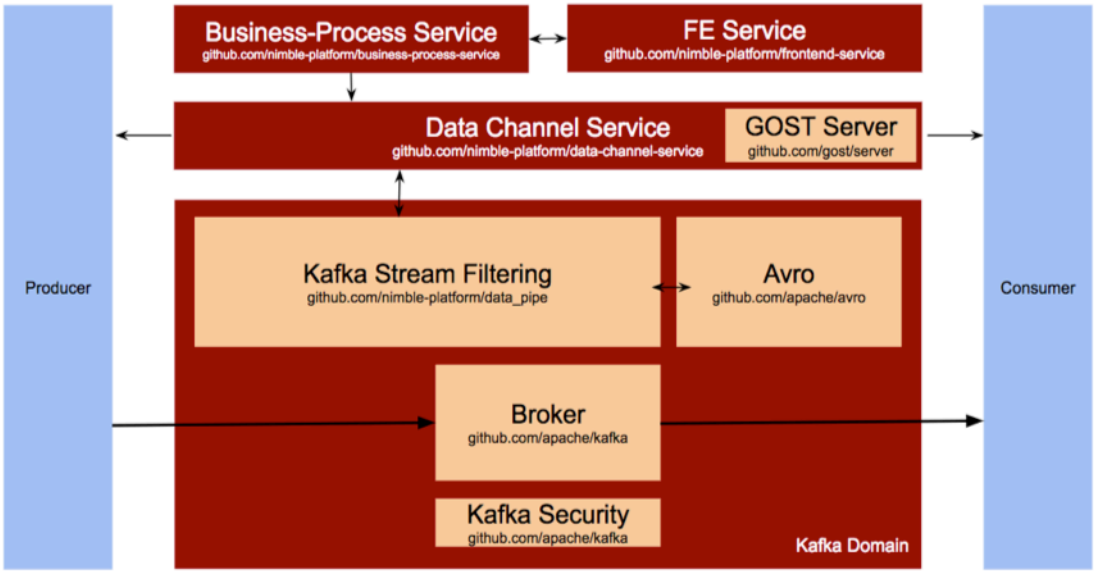


Figure 3: High-level architecture for data channels

The general architecture depicted in Figure 3 consists of three higher level component groups:

- Lightweight Microservices
- Components in the Kafka domain
- Producer and consumer components for sending and receiving messages

The following sections further describe components of each group.

Microservices

The central microservice for managing data channels is the *Data Channel Service*. It provides unified endpoints for opening, configuring and closing single data channels to existing infrastructure services (e.g. *Business-Process Service*). The full-fledged service serves as the bridge to other microservices within the infrastructure components and delegates requests further to its backend services. One such backend service is the GOST[3] server, which allows storing machine and sensor definitions in a SensorThings[4] compliant format. Components in the Kafka domain form a set of additional backend services for handling tasks related to the actual transport of messages. The *Data Channel Service stores* configurations of each data channel and makes it available to other components (e.g. components in the Kafka domain).

Kafka Domain

Kafka-related services are encapsulated in this component group. The *Kafka Broker* is in charge of opening and closing Kafka topics, which can be mapped to individual data channels. Single

[3] https://github.com/gost/server
[4] http://www.opengeospatial.org/standards/sensorthings

messages are received by the *Broker* and processed based on the configuration of the channel. The *Kafka Stream Filtering* component filters and multiplexes incoming messages based on the channel configuration stored in and provided by the *Data Channel Service*. Custom message formats can be defined and stored in the *Avro*[5] component, which provides functionalities for structured data serialization.

Producer and Consumer

Producer and consumer components are deployed and run on the premise of companies and are in charge of sending/receiving messages with a proper format and content (e.g. channel ID). The producer mainly communicates with the *Business-Process Service* for obtaining the business context of each message. These components are not part of the cloud services provided by NIMBLE. However, the platform will provide appropriate tools and software libraries for easing the integration.

3 Detailed flow

3.1 Data channel definition

The starting point for the definition of a data channel is built as a part of the NIMBLE negotiation process: As part of the negotiation, the buyer can request a data monitoring service to be switched on for goods that belong to this specific order. The supplier can grant this right and as a result, NIMBLE provides the customer with a handle to a data channel that is yet to be activated, but is already set up "under the hood". This information now constitutes a contractual clause in the contract between the two companies and it will be executed when the supplier starts manufacturing the agreed goods. The main resulting artefact of this setup process is a filter configuration to be used by NIMBLE that ensures that only data identified as relevant to the business engagement in question is shared and only the intended recipient gains access to that data. Thus, as a part of the negotiation process the parties agree on the characteristics of the information that will be made available to the recipient company.

The agreement as to the scope of data that will be shared is registered in the NIMBLE platform in the form of a filter. The filter is stored in a relational DB, including an identifier that enables the filtering capability within the streaming process to apply the correct filter to incoming data.

3.2 Data input

The company sharing data hooks up its data producer through a Kafka topic provided by NIMBLE. The source of the information that is to be shared via the NIMBLE platform resides internally within the supplying company. In order for the data to be made available to the consumer company it needs to be transmitted through the platform services, such that the filtering and further information notification can be attained. Thus, the publishing company needs to hook up its internal data source as an input to the data channels. An example would be to hook up a specific internet enabled machine that is used for manufacturing an item that is being ordered by the receiving company as an input source and the filter makes sure that information is shared only from that machine in relevant times in which a specific order is being processed by that machine.

[5] https://avro.apache.org/

Going back to the introductory motivating scenario with the 3D printing farm we established that there is a need to connect sensors that are available locally, at the premises of the manufacturer (i.e. the supplier) with an API that is provided by NIMBLE and furthermore, it is necessary to synchronize the business process that was agreed on, via NIMBLE, with the local manufacturing process that will determine when exactly, each item of the original order will be manufactured and where. For this purpose, we have at present, a semi-automatic step that manages the synchronization. It relies on a local manufacturing execution service that is able to keep track of NIMBLE-generated orders and individual manufacturing items built according to the specified contract. This step allows the supplier to map internal sensor data flows to the NIMBLE API and to send the requested monitoring data to NIMBLE from where it can be consumed by the buyer provided the necessary security credentials are presented.

The following screen shows how the manufacturer can map the internal data sources (sensors) onto the NIMBLE-provided data channel.

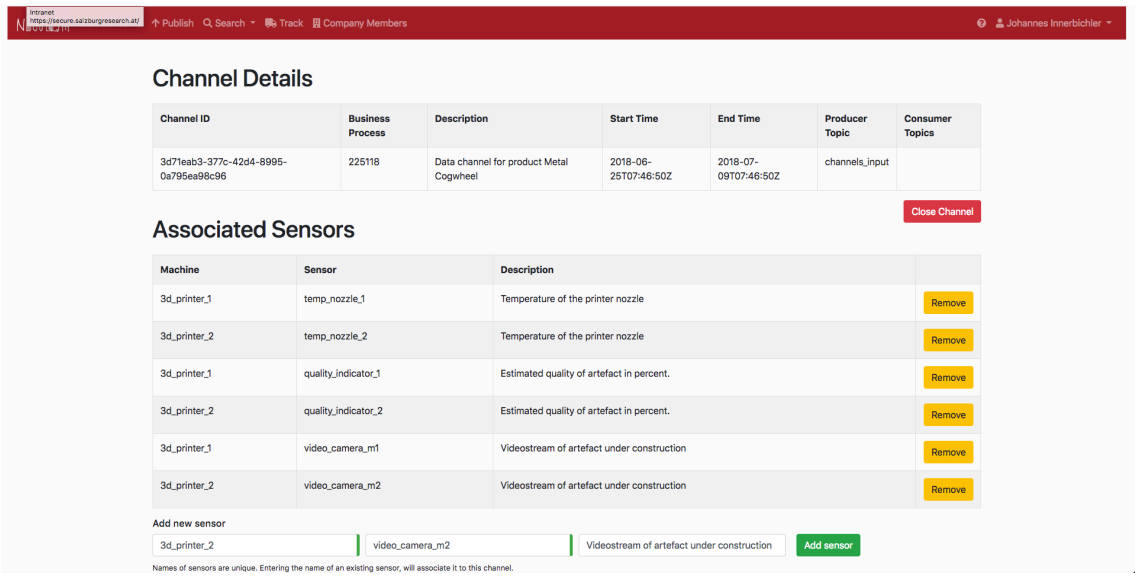


Figure 4: Manufacturer's Setup Screen for an agreed Data Channel

The Channel Details are the presets that NIMBLE created at the time when the contract was agreed. The "Associated Sensors" are those that the supplier can now enable or disable, depending on the terms of the contract. In our example, only the video cameras of the printers will be enabled, but not the temperature sensors because the customer did not request any temperature data.

3.3 Real-time data processing

The streaming and real-time processing of incoming data enable subscribing to topics which supply the data that will be processed by the streaming component, and further establish output topics to which specific results of the internal real-time processing shall be forwarded. Thus, the streaming component is registered as a subscriber to all incoming topics and processes each one according to its specific requirements. Thus, all incoming messages flow into the streams processing component and are further processed via the logic of that component.

3.4 Data filtering

The streams processing component logic is realized as a dynamic data filter. The basic capability provided by the filter is whether a specific input message is to be passed on to the corresponding recipient or whether it should be dropped, taking into account the established visibility rules between the companies for the current interaction. The generic implementation of the dynamic filter starts by fetching the relevant filter based on a unique characteristic of the specific incoming message which is being processed at the time. Once the filter has been instantiated this component parses the message and applies the specific filtering rules to it. A message that has successfully passed the filtering action is than forwarded to be published on the agreed upon output topic.

Currently the filter format is represented as a list of key-value pairs encoded in a JSON format. The filtering is performed on the header of each incoming message, checking that indeed all the key-value pairs indicated in the filter have corresponding entries in the inspected message.

Specific filters are first inserted into a relational DB. Upon identification of a new interaction by the streaming component, the relevant filter is retrieved by the streaming component and is cached locally for faster future operation.

3.5 Information delivery

The last stage in this process calls for making the relevant information available to the intended recipient. First and foremost, the recipient company can access incoming data by registering itself as a subscriber of the specific output topic. Nevertheless, as it is currently envisioned that most interactions of end-users would be performed within the scope of the NIMBLE platform, data channels support shall be integrated within the NIMBLE web based front-end. Thus, all incoming data will be saved by the platform in a relational DB and presented upon polling for that information by the receiving company, using the NIMBLE front-end web application.

3.6 Data Channel Invocation and Setup

The precise internal procedures for establishing a channel and exchanging messages are shown in Figure 5. It is assumed that necessary configuration parameters are concluded in a preceding business process. The scenario starts at the end of the business process and describes the mechanisms for creating a channel and exchanging messages on it.

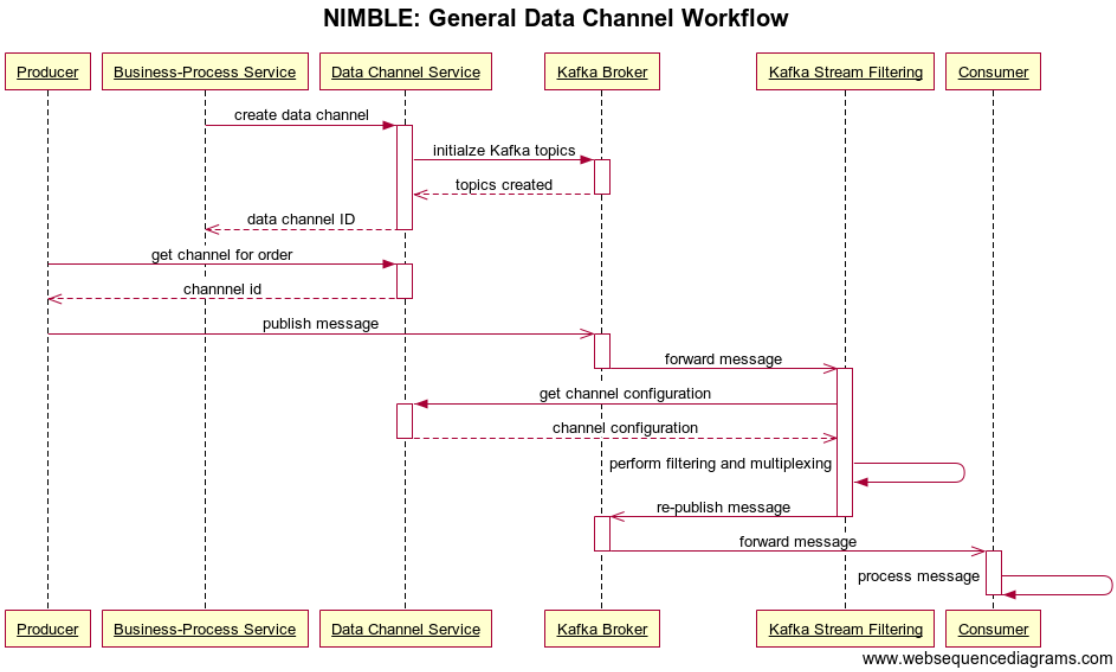


Figure 5: Exemplary workflow of channel interaction.

The following table further describes single interactions and corresponding information being exchanged.

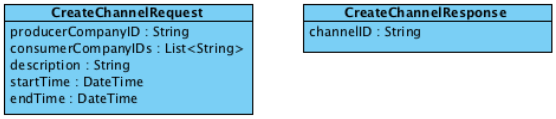
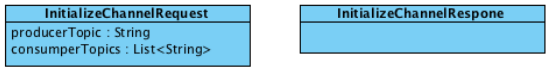
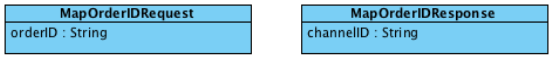
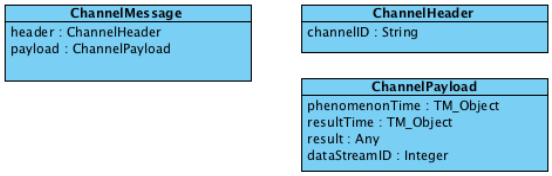
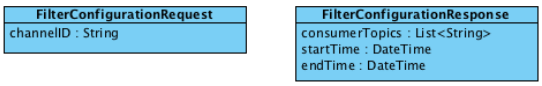
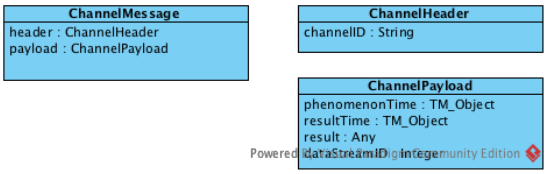
Step	Involved entities	Description	Exchanged information
Create channel	Business-Process Service Data Channel Service	The Business-Process Service triggers the creation of a data channel with proper parameters. The configuration of the channel is stored in the Data Channel Service.	
Initialize Channel	Data Channel Service Kafka Broker	Necessary Kafka topics are set up and initialized via the Kafka Broker.	
Map Order ID	Producer Data Channel Service	The producer obtains the proper business context by mapping the order ID to a specific data channel ID.	
Publish Message	Producer Kafka Broker	Messages are published in an appropriate format (i.e. header and payload). The header of each message contains the channel ID, which is used for further filtering and multiplexing.	
Message Filtering	Kafka Broker Kafka Stream Filtering	The Kafka Broker receives incoming messages and performs proper filtering based on the configuration of the channel.	
Message Forwarding	Kafka Broker Consumer	After the appropriate consumers (e.g. companies) were defined, the message is forwarded accordingly to the output topic defined.	

Table 2: Details on exchanged requests in the exemplary scenario.

4 Interaction with the NIMBLE platform

The main interaction of the data channels component is with the NIMBLE platform front-end. The creation and initiation of a new data channel is performed through the front-end as the culmination of a business process negotiation among different parties. Data reception is made accessible via the dashboard corresponding to the entity authorized to receive the filtered information.

At the back-end this component makes use of NIMBLE wide cloud services, chief among them is an instance of MessageHub[6] (messaging as a service hosted by the IBM cloud, BlueMix[7]), and an instance of PostgreSQL DB[8] which is hosted as a cloud service by BlueMix as well.

The data channels input and output, governing its interaction with other components, is made through the topics supported by the messaging service. The input point into the data channels is via a topic to which the data channel component is subscribed to receive all messages published on that topic. As an output another topic is being used to which the data channels component publishes all messages that have been approved by the corresponding filter. A company sharing data attaches the data to be shared to the corresponding input topic of the data channels component.

The corresponding filters are stored in a relational DB and fetched and cached within the data channels component upon demand. Furthermore, the DB is used to store messages published by the data channels, to be presented to the corresponding user upon polling for that information via the front-end service.

5 Deployment

The data channels component is deployed as a service within the NIMBLE Kubernetes cluster hosting the entire NIMBLE platform (see Figure 6). The entry point of the cluster is accessible via the following URL: https://nimble-platform.uk-south.containers.mybluemix.net/. The messaging service used is an instance of MessageHub named Message Hub-Nimble which is accessible via a replicated set of servers accessible via the following URLs:

```
kafka[01..05]-prod02.messagehub.services.eu-gb.bluemix.net:9093
```

The DB used is an instance of PostgreSQL hosted by BlueMix, named NIMBLE-Main-Prod, which is accessible at the following location:

```
sl-eu-lon-2-portal.5.dblayer.com:21113/compose
```

[6] https://developer.ibm.com/messaging/message-hub/

[7] https://www.ibm.com/cloud/

[8] https://console.bluemix.net/catalog/services/compose-for-postgresql

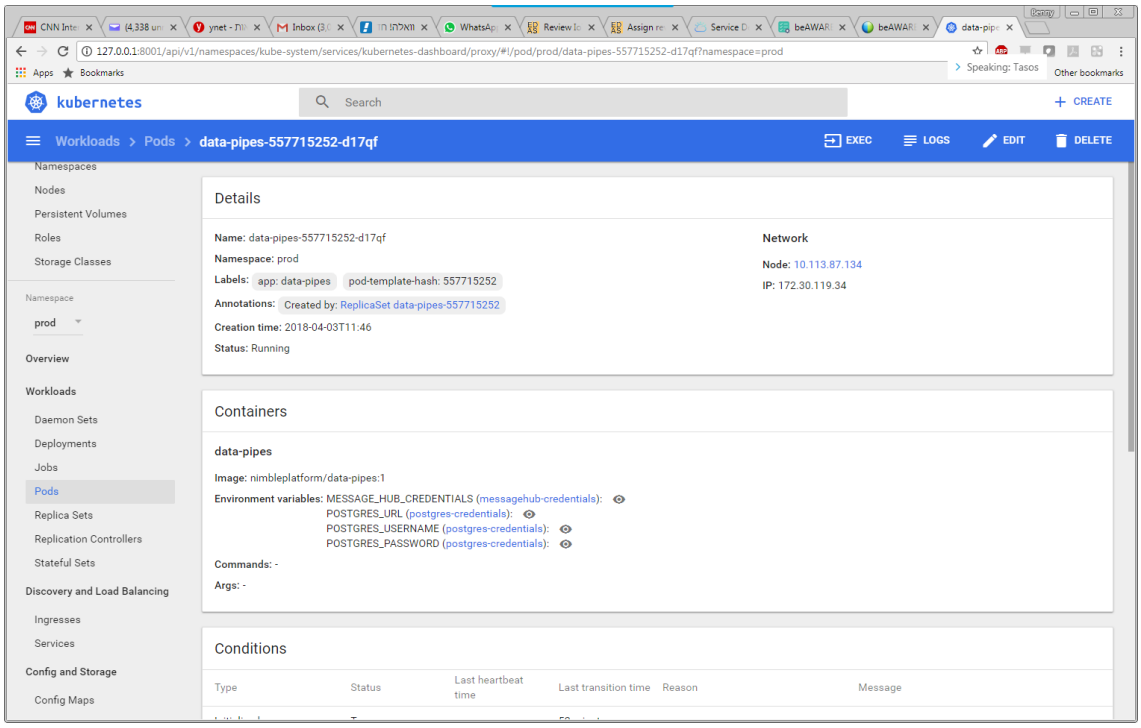


Figure 6: Data channels deployment in the NIMBLE Kubernetes cluster

6 What is being demonstrated

For the purposes of this demonstration we opted for a simple data channel definition which includes a visibility rule corresponding to a date and a specific machine as the source. The corresponding filter may be defined in the following manner:

```
Filter { Date : XXX,
         MachineId : XXX }
```

Permission is granted by the agreed upon filter to share information produced in a specific date, for all information coming from machine_id_1. There are three main components in this demonstration, namely, a data producer, streaming filter, and a data consumer.

The data producer in this demonstration sends messages through the agreed upon topic using the current date, a randomly selected machine as its source, accompanied by a random payload. An example of a sequence of messages produced by this component can be seen in Figure 7.

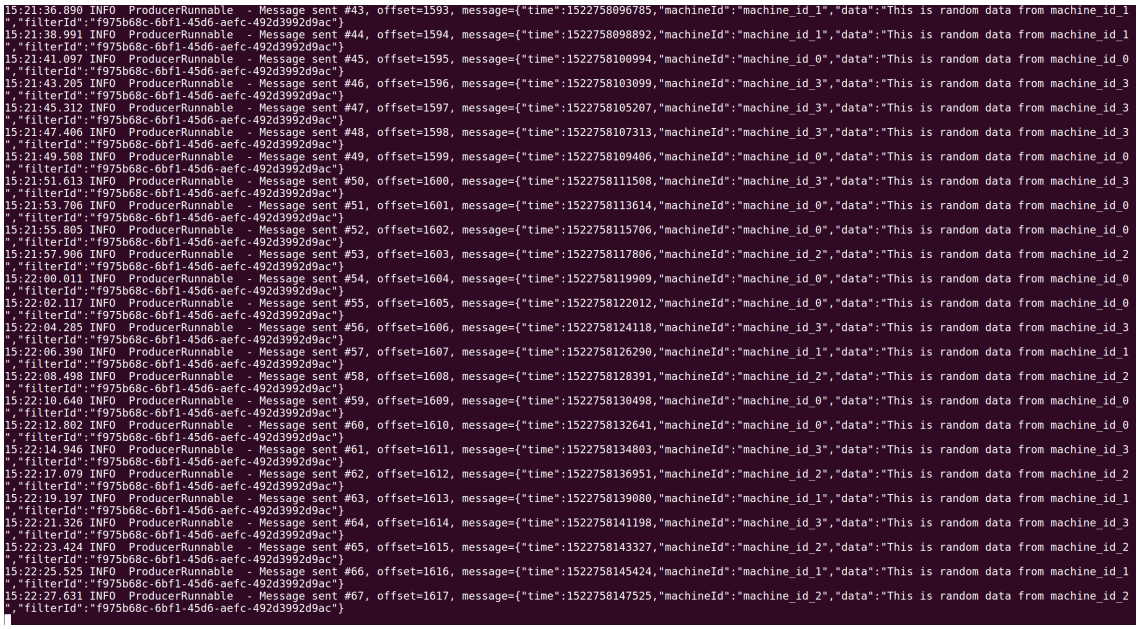


Figure 7: demo data producer

The streaming filter operation is depicted in Figure 8. In the figure one can view the data consumed by the filter, which corresponds to all the data that has been produced by the sample producer. Moreover, the figure demonstrates the application of the filter to the corresponding input in question, in which messages that abide by the corresponding filter are marked to be passed to the output topic, while messages denied by the filter are marked as such. In a production environment messages which got filtered out would be dropped altogether. For the purpose of the demonstration we created another topic to which all these messages are forwarded. At the upper part of the figure one can view the streaming process querying the filters DB to obtain the specific filter corresponding to the messages in question.

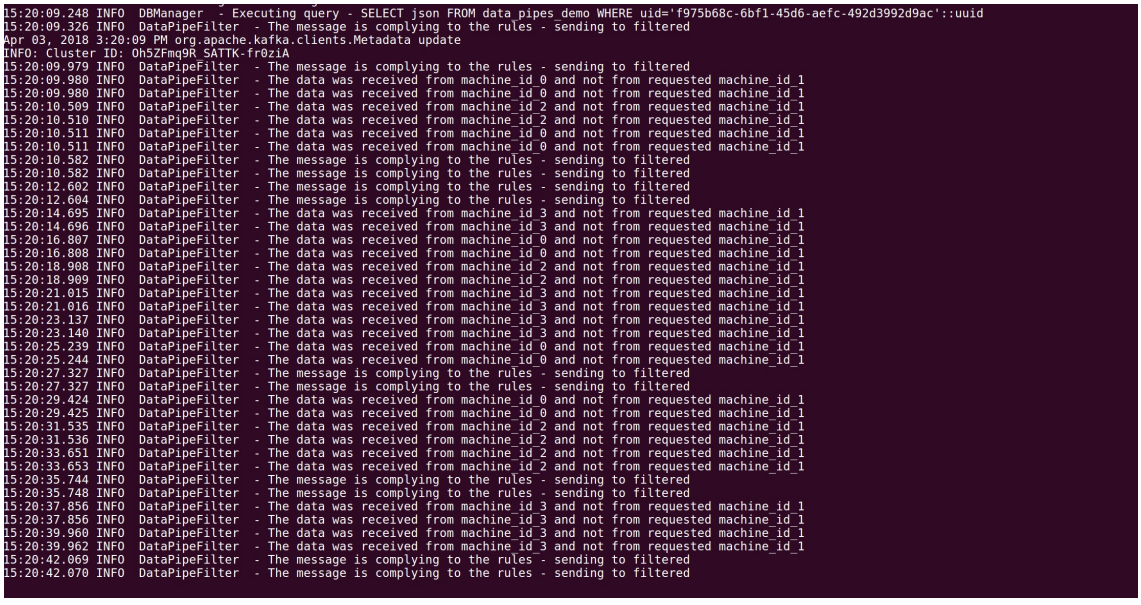


Figure 8: Demo streaming filter

Finally, two end consumers are presented. First a consumer representing the company on the receiving end of the data channel which receives only the data that has managed to pass the

filter (see Figure 9). Second, as mentioned above, for demo purposes we created another consumer which receives all the messages dropped by the filter (see Figure 10).

```
INFO: [Consumer clientId=kafka-streams-consumer, groupId=kafka-streams-consumer-group] Setting newly assigned partitions [streams-filtered-0]
15:20:12.730 INFO  ConsumerRunnable  - Message consumed: {"time":1522758012484,"machineId":"machine_id_1","data":"This is random data from machine_id_1","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:12.803 INFO  ConsumerRunnable  - Message consumed: {"time":1522758012484,"machineId":"machine_id_1","data":"This is random data from machine_id_1","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:15.805 INFO  ConsumerRunnable  - No messages consumed
15:20:18.806 INFO  ConsumerRunnable  - No messages consumed
15:20:21.807 INFO  ConsumerRunnable  - No messages consumed
15:20:24.807 INFO  ConsumerRunnable  - No messages consumed
15:20:27.461 INFO  ConsumerRunnable  - Message consumed: {"time":1522758027233,"machineId":"machine_id_1","data":"This is random data from machine_id_1","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:27.544 INFO  ConsumerRunnable  - Message consumed: {"time":1522758027233,"machineId":"machine_id_1","data":"This is random data from machine_id_1","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:30.548 INFO  ConsumerRunnable  - No messages consumed
15:20:33.549 INFO  ConsumerRunnable  - No messages consumed
15:20:35.862 INFO  ConsumerRunnable  - Message consumed: {"time":1522758035639,"machineId":"machine_id_1","data":"This is random data from machine_id_1","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:35.955 INFO  ConsumerRunnable  - Message consumed: {"time":1522758035639,"machineId":"machine_id_1","data":"This is random data from machine_id_1","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:38.956 INFO  ConsumerRunnable  - No messages consumed
15:20:41.958 INFO  ConsumerRunnable  - No messages consumed
15:20:42.269 INFO  ConsumerRunnable  - Message consumed: {"time":1522758041962,"machineId":"machine_id_1","data":"This is random data from machine_id_1","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:42.334 INFO  ConsumerRunnable  - Message consumed: {"time":1522758041962,"machineId":"machine_id_1","data":"This is random data from machine_id_1","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:45.336 INFO  ConsumerRunnable  - No messages consumed
15:20:48.337 INFO  ConsumerRunnable  - No messages consumed
15:20:51.337 INFO  ConsumerRunnable  - No messages consumed
15:20:54.337 INFO  ConsumerRunnable  - No messages consumed
15:20:56.949 INFO  ConsumerRunnable  - Message consumed: {"time":1522758056689,"machineId":"machine_id_1","data":"This is random data from machine_id_1","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:57.040 INFO  ConsumerRunnable  - Message consumed: {"time":1522758056689,"machineId":"machine_id_1","data":"This is random data from machine_id_1","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:00.041 INFO  ConsumerRunnable  - No messages consumed
15:21:03.044 INFO  ConsumerRunnable  - No messages consumed
15:21:06.044 INFO  ConsumerRunnable  - No messages consumed
15:21:09.046 INFO  ConsumerRunnable  - No messages consumed
15:21:09.724 INFO  ConsumerRunnable  - Message consumed: {"time":1522758069469,"machineId":"machine_id_1","data":"This is random data from machine_id_1","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:09.790 INFO  ConsumerRunnable  - Message consumed: {"time":1522758069469,"machineId":"machine_id_1","data":"This is random data from machine_id_1","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:11.803 INFO  ConsumerRunnable  - Message consumed: {"time":1522758071578,"machineId":"machine_id_1","data":"This is random data from machine_id_1","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:11.870 INFO  ConsumerRunnable  - Message consumed: {"time":1522758071578,"machineId":"machine_id_1","data":"This is random data from machine_id_1","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:14.878 INFO  ConsumerRunnable  - No messages consumed
15:21:17.878 INFO  ConsumerRunnable  - No messages consumed
15:21:18.126 INFO  ConsumerRunnable  - Message consumed: {"time":1522758077891,"machineId":"machine_id_1","data":"This is random data from machine_id_1","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:21.128 INFO  ConsumerRunnable  - No messages consumed
```

Figure 9: Demo filter consumer

```
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:48.590 INFO  ConsumerRunnable  - Message consumed: {"time":1522758048275,"machineId":"machine_id_0","data":"This is random data from machine_id_0","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:50.605 INFO  ConsumerRunnable  - Message consumed: {"time":1522758050372,"machineId":"machine_id_0","data":"This is random data from machine_id_0","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:50.676 INFO  ConsumerRunnable  - Message consumed: {"time":1522758050372,"machineId":"machine_id_0","data":"This is random data from machine_id_0","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:52.712 INFO  ConsumerRunnable  - Message consumed: {"time":1522758052463,"machineId":"machine_id_3","data":"This is random data from machine_id_3","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:52.778 INFO  ConsumerRunnable  - Message consumed: {"time":1522758052463,"machineId":"machine_id_3","data":"This is random data from machine_id_3","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:54.822 INFO  ConsumerRunnable  - Message consumed: {"time":1522758054577,"machineId":"machine_id_2","data":"This is random data from machine_id_2","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:54.891 INFO  ConsumerRunnable  - Message consumed: {"time":1522758054577,"machineId":"machine_id_2","data":"This is random data from machine_id_2","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:57.895 INFO  ConsumerRunnable  - No messages consumed
15:20:59.056 INFO  ConsumerRunnable  - Message consumed: {"time":1522758058824,"machineId":"machine_id_3","data":"This is random data from machine_id_3","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:20:59.122 INFO  ConsumerRunnable  - Message consumed: {"time":1522758058824,"machineId":"machine_id_3","data":"This is random data from machine_id_3","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:01.174 INFO  ConsumerRunnable  - Message consumed: {"time":1522758060929,"machineId":"machine_id_0","data":"This is random data from machine_id_0","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:01.249 INFO  ConsumerRunnable  - Message consumed: {"time":1522758060929,"machineId":"machine_id_0","data":"This is random data from machine_id_0","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:03.313 INFO  ConsumerRunnable  - Message consumed: {"time":1522758063048,"machineId":"machine_id_0","data":"This is random data from machine_id_0","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:03.386 INFO  ConsumerRunnable  - Message consumed: {"time":1522758063048,"machineId":"machine_id_0","data":"This is random data from machine_id_0","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:05.442 INFO  ConsumerRunnable  - Message consumed: {"time":1522758065178,"machineId":"machine_id_0","data":"This is random data from machine_id_0","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:05.529 INFO  ConsumerRunnable  - Message consumed: {"time":1522758065178,"machineId":"machine_id_0","data":"This is random data from machine_id_0","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:07.612 INFO  ConsumerRunnable  - Message consumed: {"time":1522758067316,"machineId":"machine_id_2","data":"This is random data from machine_id_2","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:07.679 INFO  ConsumerRunnable  - Message consumed: {"time":1522758067316,"machineId":"machine_id_2","data":"This is random data from machine_id_2","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:10.680 INFO  ConsumerRunnable  - No messages consumed
15:21:13.683 INFO  ConsumerRunnable  - No messages consumed
15:21:13.977 INFO  ConsumerRunnable  - Message consumed: {"time":1522758073673,"machineId":"machine_id_0","data":"This is random data from machine_id_0","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:14.044 INFO  ConsumerRunnable  - Message consumed: {"time":1522758073673,"machineId":"machine_id_0","data":"This is random data from machine_id_0","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:16.028 INFO  ConsumerRunnable  - Message consumed: {"time":1522758075785,"machineId":"machine_id_2","data":"This is random data from machine_id_2","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:16.098 INFO  ConsumerRunnable  - Message consumed: {"time":1522758075785,"machineId":"machine_id_2","data":"This is random data from machine_id_2","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:19.099 INFO  ConsumerRunnable  - No messages consumed
15:21:20.231 INFO  ConsumerRunnable  - Message consumed: {"time":1522758079989,"machineId":"machine_id_3","data":"This is random data from machine_id_3","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
15:21:22.334 INFO  ConsumerRunnable  - Message consumed: {"time":1522758082089,"machineId":"machine_id_0","data":"This is random data from machine_id_0","filterId":"f975b68
c-6bf1-45d6-aefc-492d3992d9ac"}
```

Figure 10: Demo non-filter consumer