

Release H: ksqldb

- [Introduction](#)
- [Setup](#)
 - [Kafka keystore/truststore](#)
 - [ksqldb-server](#)
 - [ksqldb-cli](#)
 - [Streams](#)
 - [Tables](#)
 - [Queries](#)
- [Links](#)

Introduction

ksqlDB is a database purpose-built to help developers create stream processing applications on top of Apache Kafka.

Setup

The following setup is designed to work with Strimzi.

Kafka keystore/truststore

Create a strimzi user (ksqldb)

Strimzi User
<pre>apiVersion: kafka.strimzi.io/v1beta1 kind: KafkaUser metadata: name: ksqldb namespace: kafka labels: strimzi.io/cluster: my-cluster spec: authentication: type: tls</pre>

Generate keystore/truststore secret

generate_stores.sh

```
#!/bin/sh

if [ -z "$1" ]
then
    echo "No argument supplied"
    exit 1
fi

kafkauser=$1

WORKDIR=$(dirname "$(realpath "$0")")

rm $WORKDIR/ca.crt $WORKDIR/user.crt $WORKDIR/user.key $WORKDIR/user-keystore.jks $WORKDIR/user.p12 $WORKDIR/
/user.password $WORKDIR/user-truststore.jks 2>/dev/null

kubectl delete secret ${kafkauser}-jks -n kafka 2>/dev/null

kubectl get secret my-cluster-cluster-ca-cert -n kafka -o jsonpath='{.data.ca\.crt}' | base64 --decode >
$WORKDIR/ca.crt

kubectl get secret ${kafkauser} -n kafka -o jsonpath='{.data.user\.key}' | base64 --decode > $WORKDIR/user.key

kubectl get secret ${kafkauser} -n kafka -o jsonpath='{.data.user\.crt}' | base64 --decode > $WORKDIR/user.crt

kubectl get secret ${kafkauser} -n kafka -o jsonpath='{.data.user\.p12}' | base64 --decode > $WORKDIR/user.p12

kubectl get secret ${kafkauser} -n kafka -o jsonpath='{.data.user\.password}' | base64 --decode > $WORKDIR/user.
password

export PASSWORD=`cat ${WORKDIR}/user.password`

keytool -import -trustcacerts -file $WORKDIR/ca.crt -keystore $WORKDIR/user-truststore.jks -storepass $PASSWORD
-noprompt

keytool -importkeystore -srckeystore $WORKDIR/user.p12 -srcstorepass ${PASSWORD} -srcstoretype pkcs12 -
destkeystore $WORKDIR/user-keystore.jks -deststorepass ${PASSWORD} -deststoretype jks

kubectl create secret generic ${kafkauser}-jks -n kafka --from-literal=keystore_password=$PASSWORD --from-
file=user-keystore.jks=${WORKDIR}/user-keystore.jks --from-literal=truststore_password=$PASSWORD --from-
file=user-truststore.jks=${WORKDIR}/user-truststore.jks --from-literal=key_password=$PASSWORD
```

Run `./generate_stores.sh ksqldb`

This will create a secret called "ksqldb-jks" which contains the keystore and truststore needed to connect to Kafka over SSL.

ksqldb-server

ksqldb-server

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ksqldb-server
  namespace: kafka
spec:
  selector:
    matchLabels:
      app: ksqldb-server
  template:
    metadata:
      labels:
        app: ksqldb-server
        version: v1
```

```

spec:
  containers:
  - name: ksqldb-server
    image: confluentinc/ksqldb-server:0.28.2
    imagePullPolicy: IfNotPresent
    env:
    - name: KEYSTORE_PASSWORD
      valueFrom:
        secretKeyRef:
          name: ksqldb-jks
          key: keystore_password
    - name: TRUSTSTORE_PASSWORD
      valueFrom:
        secretKeyRef:
          name: ksqldb-jks
          key: truststore_password
    - name: KEY_PASSWORD
      valueFrom:
        secretKeyRef:
          name: ksqldb-jks
          key: key_password
    - name: KSQL_BOOTSTRAP_SERVERS
      value: my-cluster-kafka-bootstrap.kafka:9093
    - name: KSQL_LISTENERS
      value: http://0.0.0.0:8088
    - name: KSQL_KSQL_SERVICE_ID
      value: ksql_service_2_
    - name: KSQL_SECURITY_PROTOCOL
      value: SSL
    - name: KSQL_OPTS
      value: "-Dssl.keystore.location=/var/private/ssl/user-keystore.jks -Dssl.keystore.
password=$(KEYSTORE_PASSWORD) -Dssl.key.password=$(KEY_PASSWORD) -Dssl.truststore.location=/var/private/ssl
/user-truststore.jks -Dssl.truststore.password=$(TRUSTSTORE_PASSWORD) -Dlisteners=http://0.0.0.0:8088/"
    - name: KSQL_KSQL_EXTENSION_DIR
      value: /opt/ksqldb-udfs
    volumeMounts:
    - name: jks
      mountPath: /var/private/ssl
      readOnly: true
  volumes:
  - name: jks
    secret:
      secretName: ksqldb-jks
---
apiVersion: v1
kind: Service
metadata:
  name: ksqldb-server
  namespace: kafka
  labels:
    app: ksqldb-server
    service: ksqldb-server
spec:
  type: LoadBalancer
  selector:
    app: ksqldb-server
  ports:
  - port: 8088
    name: http-80

```

The above YAML file deploys the ksqldb server to your cluster.

It has been configured to connect to your kafka cluster over SSL using the "ksqldb-jks" secret created in the previous step.

ksqldb-cli

ksqldb-cli

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ksqldb-cli
  namespace: kafka
spec:
  selector:
    matchLabels:
      app: ksqldb-cli
  template:
    metadata:
      labels:
        app: ksqldb-cli
        version: v1
    spec:
      containers:
        - name: ksqldb-cli
          image: confluentinc/ksqldb-cli:0.28.2
          imagePullPolicy: IfNotPresent
          env:
            - name: KEYSTORE_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: ksqldb-jks
                  key: keystore_password
            - name: TRUSTSTORE_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: ksqldb-jks
                  key: truststore_password
              valueFrom:
                secretKeyRef:
                  name: ksqldb-jks
                  key: key_password
            - name: KSQL_BOOTSTRAP_SERVERS
              value: my-cluster-kafka-bootstrap.kafka:9093
            - name: KSQL_SECURITY_PROTOCOL
              value: SSL
            - name: KSQL_OPTS
              value: "-Dssl.keystore.location=/var/private/ssl/user-keystore.jks -Dssl.keystore.
password=$(KEYSTORE_PASSWORD) -Dssl.key.password=$(KEY_PASSWORD) -Dssl.truststore.location=/var/private/ssl
/user-truststore.jks -Dssl.truststore.password=$(TRUSTSTORE_PASSWORD) -Dlisteners=http://0.0.0.0:8088/"
          volumeMounts:
            - name: jks
              mountPath: /var/private/ssl
              readOnly: true
      volumes:
        - name: jks
          secret:
            secretName: ksqldb-jks
```

We can then connect to the ksqldb-server through the client.

```
kubectl exec -it <ksqldb-cli-POD-NAME> -n kafka -- ksql http://ksqldb-server:8088
```

```
=
=                                     =
=      [ _ ] [ _ ] [ _ ] [ _ ] [ _ ]   =
=      / \ / \ / \ / \ / \ / \       =
=      < - C I L L D                 =
=      | \ \ ^ \ , | \ \ / \ /         =
=                                     =
=           The Database purpose-built    =
=           for stream processing apps     =
=
```

Steams

```
CREATE STREAM IF NOT EXISTS PMS_STREAM (
    event STRUCT<
        commonEventHeader STRUCT<domain VARCHAR,
            eventName VARCHAR,
            sourceName VARCHAR,
            reportingEntityName VARCHAR,
            startEpochMicrosec BIGINT,
            lastEpochMicrosec BIGINT,
            timeZoneOffset VARCHAR>,
        perf3gppFields STRUCT<perf3gppFieldsVersion VARCHAR,
            measDataCollection STRUCT<granularityPeriod INT,
                measuredEntityUserName VARCHAR,
                measuredEntityDn VARCHAR,
                measuredEntitySoftwareVersion VARCHAR,
                measInfoList ARRAY<STRUCT<
                    measInfoId STRUCT<sMeasInfoId VARCHAR>,
                    measTypes STRUCT<
                        sMeasTypesList ARRAY<VARCHAR>>,
                    measValuesList ARRAY<STRUCT<
                        measObjInstId VARCHAR,
                        suspectFlag VARCHAR,
                        measResults ARRAY<
                            STRUCT<p INT, "+
                                sValue
>>>>>>>)
    WITH (KAFKA_TOPIC='pms',
        VALUE_FORMAT='JSON',
        PARTITIONS = 1);
```

pms transform1

```
CREATE STREAM IF NOT EXISTS pms_stream_transform1 AS
select event->commonEventHeader->domain as domain,
event->commonEventHeader->eventName as eventName,
event->commonEventHeader->sourceName as sourceName,
event->commonEventHeader->startEpochMicrosec as startEpochMicrosec,
event->commonEventHeader->lastEpochMicrosec as lastEpochMicrosec,
event->perf3gppFields->perf3gppFieldsVersion as perf3gppFieldsVersion,
event->perf3gppFields->measDataCollection->granularityPeriod as granularityPeriod,
event->perf3gppFields->measDataCollection->measuredEntityUserName as measuredEntityUserName,
event->perf3gppFields->measDataCollection->measuredEntityDn as measuredEntityDn,
EXPLODE(event->perf3gppFields->measDataCollection->measInfoList)->measTypes->sMeasTypesList as sMeasTypesList,
EXPLODE(event->perf3gppFields->measDataCollection->measInfoList)->measValuesList as measValuesList
from pms_stream EMIT CHANGES;
```

pms transform2

```
CREATE STREAM IF NOT EXISTS pms_stream_transform2 AS
select measuredEntityDn, measuredEntityUserName, sMeasTypesList,
explode(measValuesList)->measObjInstId as measObjInstId,
explode(measValuesList)->measResults as measResults
from pms_stream_transform1 EMIT CHANGES;
```

pms transform3

```
CREATE STREAM IF NOT EXISTS pms_stream_transform3 AS
select measuredEntityDn, measuredEntityUserName, explode(sMeasTypesList) as sMeasType,
measObjInstId, explode(measResults)->sValue as sValue from pms_stream_transform2;
```

pms transform4

```
CREATE STREAM IF NOT EXISTS pms_stream_transform4 AS
select measuredEntityDn + measObjInstId + sMeasType AS MY_COMPOSITE_KEY,
measuredEntityDn, measuredEntityUserName, sMeasType, measObjInstId, sValue
from pms_stream_transform3;
```

pms transform5

```
CREATE STREAM IF NOT EXISTS pms_stream_transform5 AS
select MY_COMPOSITE_KEY, measuredEntityDn, measuredEntityUserName, sMeasType, measObjInstId, sValue
from pms_stream_transform4 partition by MY_COMPOSITE_KEY;
```

Tables

pms table

```
CREATE TABLE IF NOT EXISTS PMS_TABLE (ROWKEY VARCHAR PRIMARY KEY,
    measuredEntityDn VARCHAR,
    measuredEntityUserName VARCHAR,
    sMeasType VARCHAR,
    measObjInstId VARCHAR,
    sValue VARCHAR
)
    WITH (KAFKA_TOPIC='PMS_STREAM_TRANSFORM5', VALUE_FORMAT='JSON', PARTITIONS = 1);
```

pms view

```
CREATE OR REPLACE table pms_view as select * from PMS_TABLE EMIT CHANGES;
```

Queries

```
ksql> select * from pms_stream limit 1;
+-----+
| EVENT |
+-----+
| {COMMONEVENTHEADER={DOMAIN=perf3gpp, EVENTNAME=perf3gpp_gnb-Ericsson_pmMeasResult, SOURCENAME=0-DU-1122, REPORTINGENTITYNAME=, STARTEPOCHMICROSEC=-1570739712, LASTEPOCHMICROSEC=-1569839712, TIMEZONEOFFSET=+00:00}, PERF3GPPFIELDS={PERF3GPPFIELDSVERSION=1.0, MEASDATAACCOLLECTION={GRANULARITYPERIOD=900, MEASUREDENTITYUSERNAME=RNC Telecomville, MEASUREDENTITYDN=SubNetwork=CountryNN, MeContext=MEC-Gbg-1, ManagedElement=RNC-Gbg-1, MEASUREDENTITYSOFTWAREVERSION=, MEASINFOLIST=[{MEASINFOID={SMEASINFOID=, MEASTYPES={SMEASTYPESLIST=[attTCHSeizures, succTCHSeizures, attImmediateAssignProcs, succImmediateAssignProcs]}, MEASVALUESLIST=[{MEASOBJINSTID=RncFunction=RF-1,UtranCell=Gbg-997, SUSPECTFLAG=false, MEASRESULTS=[{P=1, SVALUE=813}, {P=2, SVALUE=913}, {P=3, SVALUE=1013}, {P=4, SVALUE=1113}]}], {MEASOBJINSTID=RncFunction=RF-1,UtranCell=Gbg-998, SUSPECTFLAG=false, MEASRESULTS=[{P=1, SVALUE=890}, {P=2, SVALUE=901}, {P=3, SVALUE=123}, {P=4, SVALUE=234}]}], {MEASOBJINSTID=RncFunction=RF-1,UtranCell=Gbg-999, SUSPECTFLAG=true, MEASRESULTS=[{P=1, SVALUE=456}, {P=2, SVALUE=567}, {P=3, SVALUE=678}, {P=4, SVALUE=789}]}]}, {MEASINFOID={SMEASINFOID=ENodeBFunction}, MEASTYPES={SMEASTYPESLIST=[attTCHSeizures1, succTCHSeizures2, attImmediateAssignProcs3, succImmediateAssignProcs4]}, MEASVALUESLIST=[{MEASOBJINSTID=ManagedElement=RNC-Gbg-1,ENodeBFunction=1, SUSPECTFLAG=false, MEASRESULTS=[{P=1, SVALUE=4}, {P=2, SVALUE=86,87,2,6,77,96,75,33,24}, {P=3, SVALUE=40}, {P=4, SVALUE=90}]}]}, {MEASINFOID={SMEASINFOID=, MEASTYPES={SMEASTYPESLIST=[attTCHSeizures5, succTCHSeizures6, attImmediateAssignProcs7, succImmediateAssignProcs8]}, MEASVALUESLIST=[{MEASOBJINSTID=RncFunction=RF-1,UtranCell=Gbg-997, SUSPECTFLAG=false, MEASRESULTS=[{P=1, SVALUE=238}, {P=2, SVALUE=344}, {P=3, SVALUE=563}, {P=4, SVALUE=787}]}], {MEASOBJINSTID=RncFunction=RF-1,UtranCell=Gbg-998, SUSPECTFLAG=false, MEASRESULTS=[{P=1, SVALUE=898}, {P=2, SVALUE=905}, {P=3, SVALUE=127}, {P=4, SVALUE=238}]}], {MEASOBJINSTID=RncFunction=RF-1,UtranCell=Gbg-999, SUSPECTFLAG=true, MEASRESULTS=[{P=1, SVALUE=454}, {P=2, SVALUE=569}, {P=3, SVALUE=672}, {P=4, SVALUE=785}]}]}]}
Limit Reached
Query terminated
```

```
ksql> select * from pms_view where smeasType = 'succTCHSeizures2';
+-----+-----+-----+-----+-----+
| ROWKEY | MEASUREDENTITYDN | MEASUREDENTITYUSERNA | SMEASTYPE | MEASOBJINSTID | SVALUE |
|-----+-----+-----+-----+-----+
| SubNetwork=CountryNN | SubNetwork=CountryNN | RNC Telecomville | succTCHSeizures2 | ManagedElement=RNC-Gbg-1,ENodeBFunction=1 | 86,87,2,6,77,96,75,33,24 |
| ,MeContext=MEC-Gbg-1 | ,MeContext=MEC-Gbg-1 | | | | |
| ,ManagedElement=RNC-Gbg-1 | ,ManagedElement=RNC-Gbg-1 | | | | |
| Gbg-1ManagedElement=Gbg-1 | | | | | |
| RNC-Gbg-1,ENodeBFunction=1 | | | | | |
| tion=1succTCHSeizures2 | | | | | |
Query terminated
ksql>
```

Links

[ksqldb](#)

[How to create a user-defined function](#)

[Scalar functions](#)

[Table Functions](#)

Cleaning messy sensor data in Kafka with ksqlDB