Big data

Apache Kafka

Using Connect and Stream

- **Import/export your data as streams of events with Kafka Connect**
- Must watch:
  - https://www.youtube.com/watch?v=J6adhl3wEj4
- There are several connectors, depending on what you want to connect to.
- As majority of us are not java programmers, we shall leverage more on ksqlDB (https://ksqldb.io/) to make our life easier.

Big data

Apache Kafka

...Using Connect and Stream

KSQLDB

- https://ksqldb.io/
- Must watch video
  - https://www.youtube.com/watch?v=7mGBxG2NhVQ
- Other video resources

Big data

Apache Kafka

KSQLDB Getting Started

- Useful for almost every aspect of kafka.
- Getting Started…
  - https://ksqldb.io/quickstart.html
  - Before running any *quickstart* statements, we will use Docker to setup all the tools that we need, except the broker(s)
  - We will also add all the requisite hostnames to our local dns (*/etc/hosts* for Linux and Mac or *c:\Windows\System32\Drivers\etc\hosts* for Windows)

Big data

Apache Kafka

…KSQLDB Getting Started

- **Setting up the hostnames**
- At this point, I am going to insist that we choose an IP alias once and for all.
  - For Linux or Mac see https://osxdaily.com/2009/08/05/how-to-create-an-ip-alias-in-mac-os-x-using-ifconfig/
    - For Mac (I run):
      - *sudo ifconfig en0 alias 192.168.1.3 255.255.255.0*
  - For Windows:
    - Open a CMD window as Administrator.
    - Check to see your network interfaces and note the Idx of the network interface you normally use [E.g. mine is 10] by running:
      - *netsh interface ipv4 show interface*
    - Enable DHCP and static coexistence for the above interface by running (it is disabled by default):
      - *netsh interface ipv4 set interface interface=10 dhcpstaticipcoexistence=enabled*
    - Add your IP alias to that interface, using the index that you noted above (mine is 10)
      - netsh interface ipv4 add address "10" 192.168.1.3 255.255.255.0
  - When all is done, you should be able to ping 192.168.1.3 and get response.

Big data

Apache Kafka

...KSQLDB Getting Started

...Setting up the hostnames

- For Mac, you will need to rerun the ip alias creation command, each time you restart your system.
- Windows will likely persist your ip alias. Please confirm on reboot.

Big data

Apache Kafka

...KSQLDB Getting Started

- **...Setting up the hostnames**
- We are going to run a number of services using Docker.
- Since we want them to reach one another, we will create hostnames for them associated with our ip alias.
- Assuming that your static ip alias is 192.168.1.3, in your *hosts* file enter:

  192.168.1.3 schema-registry ksqldb-server kafka-connect connect broker
- You can see from the above that we have setup four hostnames (*schema-registry ksqldb-server connect broker*) to point to your static ip alias (*192.168.1.3* in this illustration). We will use them in our *docker-compose.yml* file.
- As we will be using broker instead of localhost for kafka, we need to adjust *advertised.listeners* in *config\kraft\server.properties* from *advertised.listeners=PLAINTEXT://localhost:9092* to *advertised.listeners=PLAINTEXT://broker:9092*

Big data

Apache Kafka

…KSQLDB Getting Started

- Docker-compose services to setup
- **1. schema-registry**
- “Schema Registry provides a centralized repository for managing and validating schemas for topic message data, and for serialization and deserialization of the data over the network.”
- “Producers and consumers to Kafka topics can use schemas to ensure data consistency and compatibility as schemas evolve.”
- “Schema Registry is a key component for data governance, helping to ensure data quality, adherence to standards, visibility into data lineage, audit capabilities, collaboration across teams, efficient application development protocols, and system performance.”
- Ref:
  – https://docs.confluent.io/platform/current/schema-registry/index.html

Big data

Apache Kafka

…KSQLDB Getting Started

- **…1. schema-registry**
- Entry into *docker-compose.yml* file, under services:

```
schema-registry:
    image: confluentinc/cp-schema-registry:latest
    hostname: schema-registry
    container_name: schema-registry
    ports:
        - 8081:8081
    environment:
        SCHEMA_REGISTRY_HOST_NAME: schema-registry
        SCHEMA_REGISTRY_KAFKASTORE_BOOTSTRAP_SERVERS: 'broker:9092'
        SCHEMA_REGISTRY_LISTENERS: http://0.0.0.0:8081
    volumes:
        - ./data:/tmp/data
```

Big data

Apache Kafka

…KSQLDB Getting Started

- **2. connect**
- “Kafka Connect is a free, open-source component of Apache Kafka® that serves as a centralized data hub for simple data integration between databases, key-value stores, search indexes, and file systems.”
- “You can use Kafka Connect to stream data between Apache Kafka® and other data systems and quickly create connectors that move large data sets in and out of Kafka.”
- Ref:
  – https://docs.confluent.io/platform/current/connect/index.html

Big data

Apache Kafka

…KSQLDB Getting Started

- **…1. connect**
- Entry into *docker-compose.yml* file, under services:

```
kafka-connect:
    image: confluentinc/cp-kafka-connect-base:latest
    container_name: kafka-connect
    ports:
        - 8083:8083
    environment:
        CONNECT_BOOTSTRAP_SERVERS: "broker:9092"
        CONNECT_REST_ADVERTISED_HOST_NAME: "kafka-connect"
        CONNECT_REST_PORT: 8083
        CONNECT_GROUP_ID: kafka-connect
        CONNECT_CONFIG_STORAGE_TOPIC: _kafka-connect-configs
        CONNECT_OFFSET_STORAGE_TOPIC: _kafka-connect-offsets
        CONNECT_STATUS_STORAGE_TOPIC: _kafka-connect-status
        CONNECT_KEY_CONVERTER: io.confluent.connect.avro.AvroConverter
        CONNECT_KEY_CONVERTER_SCHEMA_REGISTRY_URL: 'http://schema-registry:8081'
        CONNECT_VALUE_CONVERTER: io.confluent.connect.avro.AvroConverter
        CONNECT_VALUE_CONVERTER_SCHEMA_REGISTRY_URL: 'http://schema-registry:8081'
        CONNECT_LOG4J_ROOT_LOGLEVEL: "INFO"
        CONNECT_LOG4J_LOGGERS:
        "org.apache.kafka.connect.runtime.rest=WARN,org.reflections=ERROR"
        CONNECT_CONFIG_STORAGE_REPLICATION_FACTOR: "1"
        CONNECT_OFFSET_STORAGE_REPLICATION_FACTOR: "1"
        CONNECT_STATUS_STORAGE_REPLICATION_FACTOR: "1"
        CONNECT_PLUGIN_PATH: '/usr/share/java,/usr/share/confluent-hub-components/,/connectors/'
```

Big data

Apache Kafka

...*KSQLDB Getting Started*

```
          # If you want to use the Confluent Hub installer to d/l component, but make them available
          # when running this offline, spin up the stack once and then run :
          # docker cp kafka-connect:/usr/share/confluent-hub-components ./connectors
          # mv ./connectors/confluent-hub-components/* ./connectors
          # rm -rf ./connectors/confluent-hub-components
volumes:
          - $PWD/connectors:/connectors
          - /Users/piusonobhayedo/Documents-No-iCloud/2022-2023/SST/DataScience/DAT608/classcodes/kafka/kafka_2.13-
3.5.0/certs:/certs
          # In the command section, $ are replaced with $$ to avoid the error 'Invalid interpolation format for "command" option'
command:
          - bash
          - -c
          - |
            echo "Installing Connector”
            confluent-hub install --no-prompt confluentinc/kafka-connect-elasticsearch:14.0.7
            confluent-hub install --no-prompt confluentinc/kafka-connect-jdbc:10.7.3
            confluent-hub install --no-prompt neo4j/kafka-connect-neo4j:5.0.2
            confluent-hub install jcustenborder/kafka-connect-spooldir:latest
            #Below are for Ozone - see https://ozone.apache.org/
            confluent-hub install confluentinc/kafka-connect-hdfs3:1.1.25
            confluent-hub install confluentinc/kafka-connect-hdfs3-source:2.5.7
            confluent-hub install confluentinc/kafka-connect-s3-source:2.5.7
            confluent-hub install confluentinc/kafka-connect-s3:10.5.2#
            echo "Launching Kafka Connect worker"
            /etc/confluent/docker/run &
            #
            sleep infinity
```

- You can find connectors at the URL https://www.confluent.io/hub/

Big data

Apache Kafka

…KSQLDB Getting Started

- You can find connectors at the URL https://www.confluent.io/hub/
  - Download from there and put in your /connectors indicated in the volumes of your kafka-connect
  - Also copy your certificates to your /certs indicated in the volumes of your kafka-connect

Big data

Apache Kafka

...KSQLDB Getting Started

- **3. ksqldb-server**
- “ksqlDB is a database purpose-built for stream processing applications on top of Apache Kafka®.”
- Ref: https://docs.ksqldb.io/en/latest/

Big data

Apache Kafka

…KSQLDB Getting Started

- **…3. ksqldb-server**
- Entry into docker-compose.yml file, under services:

```
ksqldb-server:
    image: confluentinc/ksqldb-server:0.29.0
    hostname: ksqldb-server
    container_name: ksqldb-server
    ports:
        - "8088:8088"
    environment:
        KSQL_LISTENERS: http://0.0.0.0:8088
        KSQL_BOOTSTRAP_SERVERS: broker:9092
        KSQL_KSQL_LOGGING_PROCESSING_STREAM_AUTO_CREATE: "true"
        KSQL_KSQL_LOGGING_PROCESSING_TOPIC_AUTO_CREATE: "true"
        KSQL_KSQL_SCHEMA_REGISTRY_URL: http://schema-registry:8081
        KSQL_KSQL_CONNECT_URL: http://connect:8083
    volumes:
        - /Users/piusonobhayedo/Documents-No-iCloud/2022-2023/SST/DataScience/DAT608/classcodes/kafka/kafka_2.13-3.5.0/certs:/certs
```

Big data

Apache Kafka

...KSQLDB Getting Started

4. ksqldb-cli

- “The ksqlDB CLI provides a console with a command-line interface for the ksqlDB engine.”
- “Use the ksqlDB CLI to interact with ksqlDB Server instances and develop your streaming applications.”
- “The ksqlDB CLI is designed to be familiar to users of MySQL, Postgres, and similar applications.”
- The ksqlDB CLI is implemented in the io.confluent.ksql.cli package.”
- Ref:
  – https://docs.ksqldb.io/en/latest/operate-and-deploy/how-it-works/

Big data

Apache Kafka

…KSQLDB Getting Started

- **4. ksqldb-cli**
- Entry into *docker-compose.yml* file, under services:

```
ksqldb-cli:
    image: confluentinc/ksqldb-cli:0.29.0
    container_name: ksqldb-cli
    entrypoint: /bin/sh
    tty: true
```

- Start all up in order (assumes that your Kafka broker is already running):

```
docker-compose up schema-registry -d &&
docker-compose up kafka-connect -d &&
docker-compose up ksqldb-server -d
```

- Give some time for *ksqldb-server* to fully startup before starting up the client

```
docker-compose up ksqldb-cli -d
```

Big data

Apache Kafka

…KSQLDB Getting Started

- Start ksqlDB's interactive CLI

  docker exec -it ksqldb-cli ksql http://ksqldb-server:8088
- Now that our environment is set, let’s try out the commands below from https://ksqldb.io/quickstart.html. [See the url for explanations for each of the commands]
- Create a stream

  CREATE STREAM riderLocations (profileId VARCHAR, latitude DOUBLE, longitude DOUBLE)

  WITH (kafka_topic='locations', value_format='json', partitions=1);

  – You should get a message:
    - Message
    - ----------------
    - Stream created
    - ----------------

Big data

Apache Kafka

...KSQLDB Getting Started

- ...Start ksqlDB's interactive CLI
- Create materialized views (A table that only shows latest location of the rider)

```
CREATE TABLE currentLocation AS
  SELECT profileId,
         LATEST_BY_OFFSET(latitude) AS la,
         LATEST_BY_OFFSET(longitude) AS lo
  FROM riderlocations
  GROUP BY profileId
  EMIT CHANGES;
```

  – You should get a message like:
    - Message
    - ---
    - Created query with ID CTAS_CURRENTLOCATION_3
    - ---

Big data

Apache Kafka

…KSQLDB Getting Started

- Materialize the derived table *currentLocation* above

```
CREATE TABLE ridersNearMountainView AS
  SELECT ROUND(GEO_DISTANCE(la, lo, 37.4133, -122.1162), -1) AS distanceInMiles,
         COLLECT_LIST(profileId) AS riders,
         COUNT(*) AS count
  FROM currentLocation
  GROUP BY ROUND(GEO_DISTANCE(la, lo, 37.4133, -122.1162), -1);
```

- Run a push query over the stream (This is a continuously running consumer)

```
SELECT * FROM riderLocations
  WHERE GEO_DISTANCE(latitude, longitude, 37.4133, -122.1162) <= 5 EMIT CHANGES;
```

  – This should give us a consumer type console that waits until terminated.

- Let’s push data to the stream via another console and observe the outcome in the consumer console above. See next page.

Big data

Apache Kafka

…KSQLDB Getting Started

- Start another ksqlDB's interactive CLI for producer activity using ksql client.

  docker exec -it ksqldb-cli ksql http://ksqldb-server:8088

- Populate the stream with events

  INSERT INTO riderLocations (profileId, latitude, longitude) VALUES ('c2309eec', 37.7877, -122.4205);

  INSERT INTO riderLocations (profileId, latitude, longitude) VALUES ('18f4ea86', 37.3903, -122.0643);

  INSERT INTO riderLocations (profileId, latitude, longitude) VALUES ('4ab5cbad', 37.3952, -122.0813);

  INSERT INTO riderLocations (profileId, latitude, longitude) VALUES ('8b6eae59', 37.3944, -122.0813);

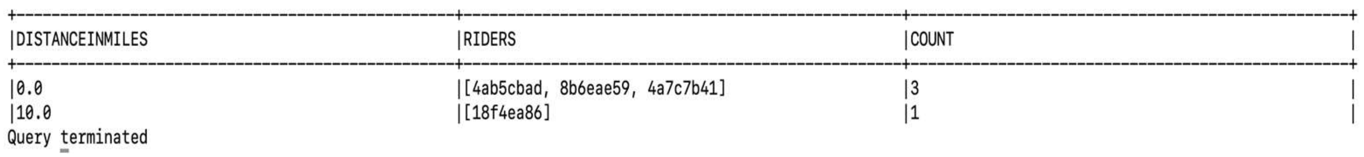
  INSERT INTO riderLocations (profileId, latitude, longitude) VALUES ('4a7c7b41', 37.4049, -122.0822);

  INSERT INTO riderLocations (profileId, latitude, longitude) VALUES ('4ddad000', 37.7857, -122.4011);

- Run a Pull query against the materialized view

  *SELECT * from ridersNearMountainView WHERE distanceInMiles <= 10;*

  – This should give output like:



```
+--------------------------------------------+--------------------------------------------+--------------------------------------------+
|DISTANCEINMILES                             |RIDERS                                      |COUNT                                       |
+--------------------------------------------+--------------------------------------------+--------------------------------------------+
|0.0                                         |[4ab5cbad, 8b6eae59, 4a7c7b41]              |3                                           |
|10.0                                        |[18f4ea86]                                  |1                                           |
Query terminated
```

- All is well? Good to go with your environment

Big data

Apache Kafka

…KSQLDB Getting Started

- Get to know more on your own…
  - https://www.youtube.com/watch?v=7mGBxG2NhVQ
  - https://developer.confluent.io/learn-kafka/ksqldb/intro/

Big data

Apache Kafka

KSQLDB Illustrations

- **For illustrations, see** https://ksqldb.io/examples.html#show-the-contents-of-a-kafka-topic
- Interacting with broker via ksqldb client. Example commands:
  - List all topics
    - SHOW TOPICS;
  - Show the contents of a Kafka topic (as consumer)
    - PRINT 'topic_name' FROM BEGINNING;
      - E.g.,
        - » *PRINT 'quickstart-events' FROM BEGINNING;*

Big data

Apache Kafka

…KSQLDB Illustrations

- KSQLDB is particularly useful for creating Kafka streams and tables.
- Let's conceptually distinguish between **events**, **streams** and **tables**
  - Ref: https://www.confluent.io/blog/kafka-streams-tables-part-1-event-streaming/
- An **event** records the fact that "something happened" in the world.
  - For instance:
    - Event key: "Alice"
    - Event value: "Has arrived in Rome"
    - Event timestamp: "Dec. 3, 2019 at 9:06 a.m."

Big data

Apache Kafka

…KSQLDB Illustrations

- …conceptual distinction cont'd
- An event **stream** records the history of what has happened in the world as a sequence of events.
- A **stream** provides immutable data. (Immutable for as long as you set the topic events to last)
  - For instance:
    - Sales ledger (with sales recorded in chronological order)
    - Sequence of moves in a chess game
    - Record the history of your business for hundreds of years.
      - Read the New York Times case: https://www.confluent.io/blog/publishing-apache-kafka-new-york-times/

Big data

Apache Kafka

…KSQLDB Illustrations

- …conceptual distinction cont’d
  - Compared to an event stream, a **table** represents the state of the world at a particular point in time, typically “now.”
  - A **table** is a view of an event stream.
  - A **table** provides mutable data.
  - For instance:
    - Total sales
    - The current state of the chess board

Big data

Apache Kafka

…KSQLDB Illustrations

- **stream-table duality**
  - We can turn a **stream** into a **table** by aggregating the stream with operations such as **COUNT()** or **SUM()**
  - We can turn a table into a stream by capturing the changes made to the table—inserts, updates, and deletes—into a “change stream.” This process is often called *change data capture* or *CDC* for short.

Big data

Apache Kafka

…KSQLDB Illustrations

- Advantageous to use ksqldb to create streams because:
  - It can automatically create the underlying topic in *kafka* for us, if it does not exist.
  - It can automatically create appropriate entry in *schema-registry* for us, for the key and value data types for the topic, where required; e.g. the case of database.

Big data

Apache Kafka

…KSQLDB Illustrations

- Let us run some Examples:
- Ref: https://ksqldb.io/examples.html
  - Create a stream over a non-existing Kafka topic:

    CREATE STREAM s1 (c1 VARCHAR, c2 INTEGER)

    WITH (kafka_topic='s1', value_format='json', partitions=2);
  - Create a stream over an existing Kafka topic:

    CREATE STREAM s1 (c1 VARCHAR, c2 INTEGER)

    WITH (kafka_topic='s1', value_format='json');
  - Notice that the difference between the above two statements is that *partitions* option needs to be set when the topic does not yet exist.

Big data

Apache Kafka

…KSQLDB Illustrations

- …Let us run some Examples:
  - Create a table (existing vs non-existing scenario also applies here)

CREATE TABLE t1 (c1 VARCHAR PRIMARY KEY, c2 INTEGER)

WITH (kafka_topic='t1', KEY_FORMAT='KAFKA', PARTITIONS=2, value_format='json');

Big data

Apache Kafka

…KSQLDB Illustrations

- …Let us run some Examples:
  - Populate streams
    - *INSERT INTO s1 (c1, c2) VALUES ('0', 1);*
    - *INSERT INTO t1 (c1, c2) VALUES ('1', 1);*
- Check things out:
  - List topics:
    - *SHOW TOPICS;*
  - List streams:
    - *SHOW STREAMS;*
  - List tables:
    - *SHOW TABLES;*
  - Describe a stream or table
    - *DESCRIBE s1;*
    - *DESCRIBE t1;*
  - For runtime statistics and query details run:
    - DESCRIBE <Stream,Table> EXTENDED;
    - E.g.,
      - *DESCRIBE s1 EXTENDED;*
      - *DESCRIBE t1 EXTENDED;*

Big data

Apache Kafka

…KSQLDB Illustrations

- …Let us run some Examples:
  - Show all events in a topic using as usual:
    - PRINT 'topic_name' FROM BEGINNING;
    - E.g.,

      PRINT s1 FROM BEGINNING;

      Returns, key: <null>, value: {"C1":"0","C2":1}

      PRINT t1 FROM BEGINNING;

      Returns, key: 1, value: {"C2":1}
    - From the above, you can see that in t1, the value you supplied for C1 is used as the key, as you indicated earlier when you ran,

      CREATE TABLE t1 (c1 VARCHAR PRIMARY KEY, c2 INTEGER)

      WITH (kafka_topic='t1', KEY_FORMAT='KAFKA', PARTITIONS=2, value_format='json');

Big data

Apache Kafka

…KSQLDB Illustrations

- …Let us run some Examples:
  - Let us create a **derived stream** from another stream
  - Let’s use the REST API option from python code.
  - You can create a new python file named for e.g., *kafka-ksql-REST.py* and add the following code:

```
import requests
url = 'http://ksqldb-server:8088/ksql'
headers = {
  'Accept' : 'application/vnd.ksql.v1+json',
  'Content-Type' : 'application/vnd.ksql.v1+json'
  }
body = {
 "ksql": "CREATE STREAM s2 AS SELECT * FROM s1 EMIT CHANGES;",
 "streamsProperties": {
  "ksql.streams.auto.offset.reset": "earliest"
 }
}
response = requests.post(
  url,
  headers=headers,
  json=body
)

# Print the response
post_response_json = response.json()
print(post_response_json)
```

Big data

Apache Kafka

…KSQLDB Illustrations

- …Let us run some Examples:
  - Running *python kafka-ksql-REST.py* from a *.venv* activated terminal should create our stream, as specified in the body of our http post above. i.e.,

    ```
    body = {
      "ksql": "CREATE STREAM S1_Expanded WITH (KAFKA_TOPIC = 'S1_Expanded', VALUE_FORMAT = 'JSON', PARTITIONS = 4) AS SELECT * FROM S1;",
      "streamsProperties": {
        "ksql.streams.auto.offset.reset": "earliest"
      }
    }
    ```
  - For info on *streamsProperties* see https://docs.ksqldb.io/en/latest/reference/server-configuration/
  - The above creates another stream named s2 that listens for changes in s1 and picks them up.
  - This should take up at least 1 partition from s1. So if s1 has only one partition and is occupied, there will be none available for s2.
  - You can also send http post commands via curl

Big data

Apache Kafka

…KSQLDB Illustrations

- …Let us run some Examples:
  - In the derived stream we can also achieve data **transformation** using appropriate SQL select statement.
  - Before we do this with s1, let us be sure that we have enough partitions in s1. Originally, we had only 2 which should be used up by now.
    - Unfortunately, we can't change existing topics number of partitions on the fly; so, plan well initially.
    - A way out is to create a new topic derived from s1 that has enough partitions and that contains all the events in s1. E.g.,
      - CREATE STREAM S1_Expanded WITH (KAFKA_TOPIC = 's1', VALUE_FORMAT = 'JSON', PARTITIONS = 4, REPLICAS = 2) AS SELECT * FROM S1;
    - The *REPLICAS* option above is right now useless because we do not yet have multiple brokers clustered. The idea is to explicitly instruct kafka to ensure that all the events of the topic is available in at least two brokers, in this case. In this way, if a broker goes down, data is not lost.
    - So, http post the above without the *REPLICAS*.

Big data

Apache Kafka

…KSQLDB Illustrations

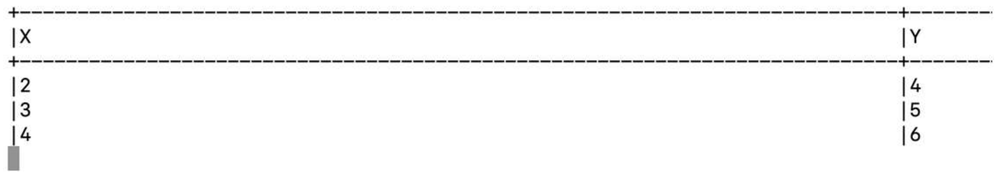
- …Let us run some Examples:
  - E.g., let's create another stream that is a result of transformation of *s1* columns:
    - *CREATE STREAM s1_transformed AS*
    - *SELECT c1 AS x, c2 + 1 AS y FROM s1 EMIT CHANGES;*

Big data

Apache Kafka

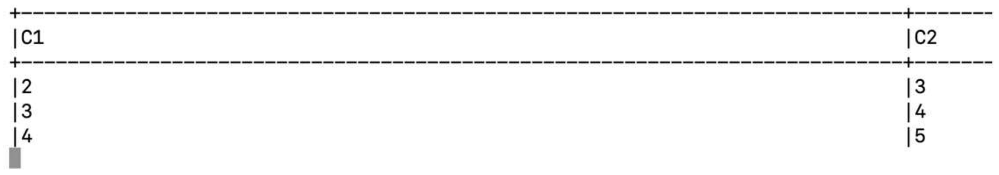
...KSQLDB Illustrations

- ...Let us run some Examples:
  - Let's consume our *s2* and *s1_transformed* streams, both derived from *s1*.
    - Use one ksql-client window to emit changes in (i.e. consume) *s2*
      - *select * from s2 emit changes;*
    - Use another ksql-client window to emit changes in (i.e. consume) *s1_transformed*
      - *select * from s1_transformed emit changes;*
  - Use another ksql-client window to add records to s1
    - *INSERT INTO s1 (c1, c2) VALUES ('2', 3);*
    - *INSERT INTO s1 (c1, c2) VALUES ('3', 4);*
    - *INSERT INTO s1 (c1, c2) VALUES ('4', 5);*
  - If you check:
    - *s1_transformed* consume window, you will see columns x and y and the y column containing the transformed value.



```
+------------------------------------------------------------------------------+--------
|X                                                                             |Y
+------------------------------------------------------------------------------+--------
|2                                                                             |4
|3                                                                             |5
|4                                                                             |6
```

    - s2 consume window will reflect same columns and emitted values



```
+------------------------------------------------------------------------------+--------
|C1                                                                            |C2
+------------------------------------------------------------------------------+--------
|2                                                                             |3
|3                                                                             |4
|4                                                                             |5
```

Big data

Apache Kafka

…KSQLDB Illustrations

- …Let us run some Examples:
  - Exercise: Produce content for the s1 topic from your python code. E.g.,
    - Create a file kafka_producer_exercise.py and add the following content

```
import logging
from kafka import KafkaProducer
from kafka.errors import KafkaError
import json

try:
    producer = KafkaProducer(bootstrap_servers=['broker:9092'],value_serializer=lambda m: json.dumps(m).encode('utf-8'))
    for n in range(3):
    producer.send('s1', {'c1': str(n + 7), 'c2' : n + 8})
except KafkaError as e:
    logging.error ("Error sending to kafka broker: {}", e)
    exit(1)

producer.flush()
producer.close()
```

    - Run as usual and observe the s2 and s1_transformed consumer windows you opened earlier. There will be update.

- …Let us run some Examples:
  - There are many other transformations you can do for derived streams, following SQL select possibilities. E.g.:
    - Select a subset of columns
    - Filter rows by values
    - Apply a function to columns
      - See https://docs.ksqldb.io/en/latest/developer-guide/ksqldb-reference/functions/ for functions references
    - Select NULL or non-NULL columns
    - Timestamp comparison
    - Date/time operations
    - Etc.

Big data

Apache Kafka

…KSQLDB Illustrations

- …Let us run some Examples:
- Go through and do as much exercises as possible from:
  - https://docs.ksqldb.io/en/latest/developer-guide/ksqldb-reference/create-stream/
  - https://docs.ksqldb.io/en/latest/developer-guide/ksqldb-reference/create-table/
  - https://docs.ksqldb.io/en/latest/developer-guide/ksqldb-reference/create-stream-as-select/
  - https://docs.ksqldb.io/en/latest/developer-guide/ksqldb-reference/create-table-as-select/
  - https://docs.ksqldb.io/en/latest/developer-guide/ksqldb-reference/select-push-query/
  - https://docs.ksqldb.io/en/latest/developer-guide/ksqldb-reference/select-pull-query/
- Reference index is at https://docs.ksqldb.io/en/latest/developer-guide/ksqldb-reference/

Big data

Apache Kafka

…KSQLDB Illustrations

- …Let us run some Examples:
  - In production, headless mode is recommended for ksqldb.
  - In this mode, ksql-cli console or REST connection is no longer allowed.
  - Mount a volume for the tested and trusted queries, in *volumes* section of *ksqldb-server* service in *docker-compose.yml*.

    volumes:

    - *./kafka/ksqldb-server/src:/opt/app/src*
  - Add command line option via *KSQL_OPTS* environment variable.
    - *KSQL_OPTS: "-Dksql.queries.file=/opt/app/src/queries.sql"*
  - Comment the line out, if you are in a non-production ksqldb-server and testing your queries and doing initial setup with partitions via command line or REST.