Big Data Pipeline Using Confluent Kafka and MongoDB

➔ **Kafka:**

- Kafka is a big data management tool that is used to manage the data pipeline in case of streaming data.
- A data can be said as streaming data in case it is getting produced/generated continuously in real time. For example: data produced by sensors in the IOT devices.
- Kafka works on Pub(Publish)/Sub(Subscribe) architecture.
- Publishing of the data is done through Producers whereas, Subscription of published data is done through Consumers.
- Producer:
    - Producer is a programmed script. For example .py file .
    - Producer is responsible for Publishing the continuously generated streaming data from the data source(like Sensors in IOT devices) into Kafka where it is stored in form of topics.
    - Kafka can have multiple topics and each topic is having a unique name in the entire Kafka cluster. Also, each topics can have multiple partitions.
    - Topics can be seen as table (rows*columns) into which producers write data and from where consumers read data.
    - Kafka can be seen as building and topics as rooms. The way a building can have multiple rooms, Kafka can also have multiple topics. Topic basically tells us about the address that is the location where the data that is getting streamed from say sensors of IOT device, will be stored.
    - Kafka can be configured to handle multiple Producers.
    - In general since, here data is continuously getting generated the Producer script runs infinitely without any halt.
- Consumer:
    - Consumer is also a programmed script. For example .py file .
    - Consumer is responsible for Consuming the data generated by Producers from the Kafka topics.
    - Kafka can be configured to handle multiple Consumers.
    - Consumer can be configured to dump the data into databases like MongoDB, MySQL, Casandra etc.
- To summarize Kafka is a tool that acts as an mediator by providing a bridge between Producers and Consumers.

➔ **Understanding the big data pipeline for our current use case:**

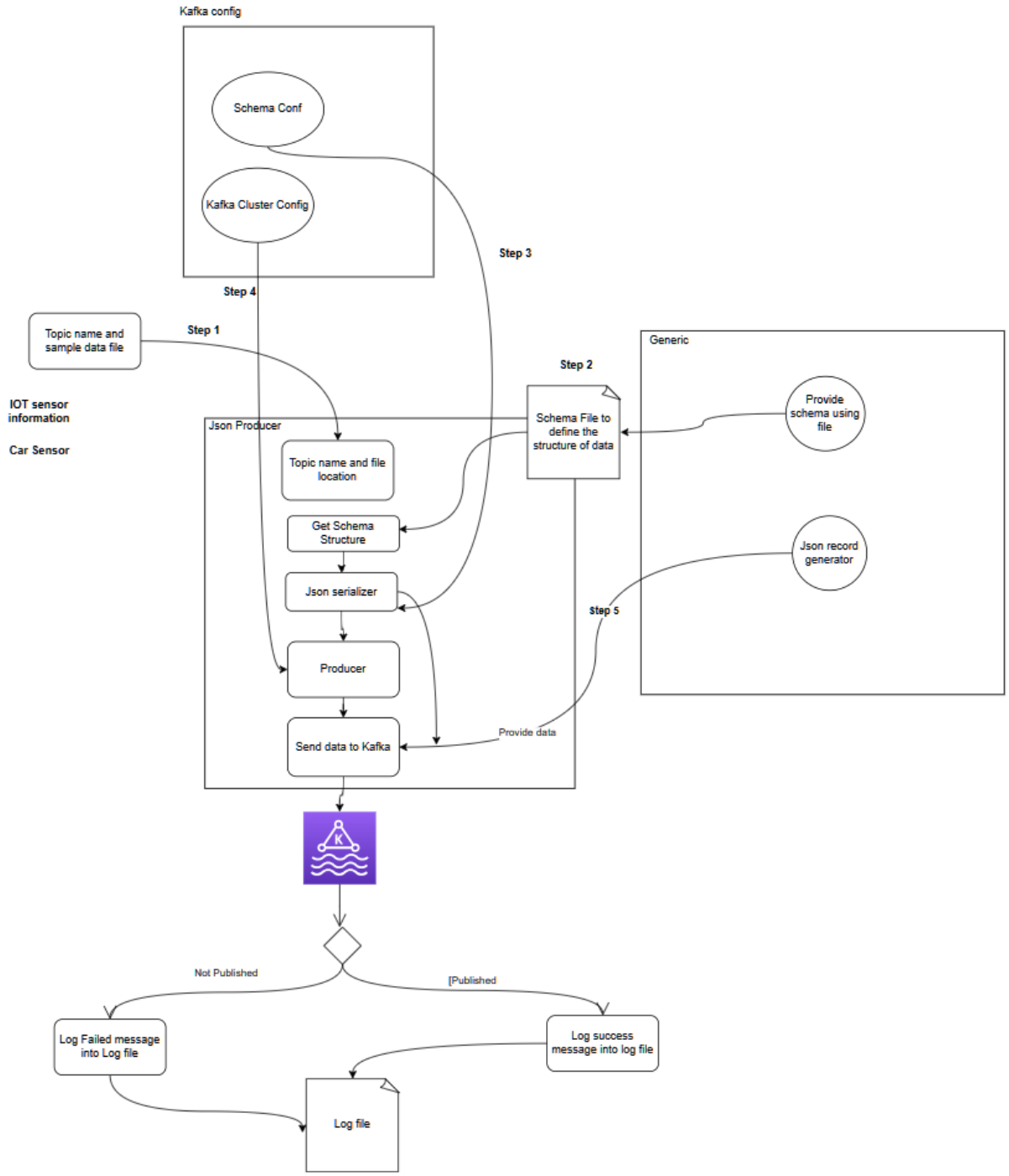
- Producer: In our use case we will be using Sensor's reading that is continuously generated by IOT devices.
- Consumer: In this use case we are going consumer data by dumping or flushing it into a MongoDB database.
- Please note that in order to demo this big data pipeline I will be using a sample_data.csv file which is having sensor readings. Reason why I am using this static pooled file is because of unavailability of respective sensors to accommodate the use case.

- *Ques*: Why are we not directly dumping data into MongoDB and are using Kafka as an mediator?
  Ans: **Reason 1**: Streaming data gets generated on one on one basis. In such a case if push data directly in MongoDB then we need to establish connectivity b/w data source and target for every streaming datapoint. This can be fine if we are working on a small dataset. But in case of streaming data which is big and can be infinitely large, creating connectivity to dump each record on one on one basis will require a lot of time.

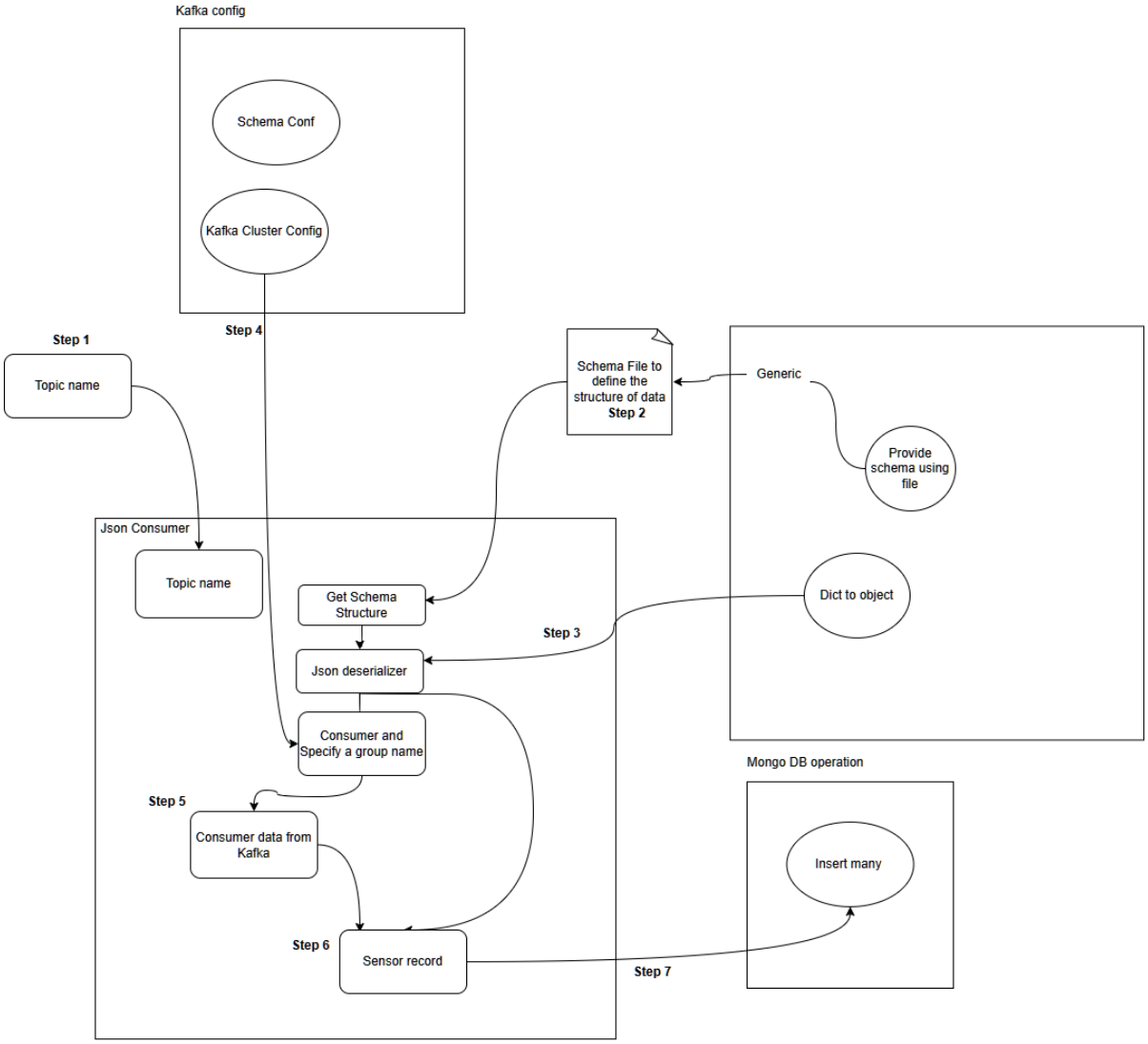
  Reason 2: We might need to perform certain preprocessing before we are going to dump our data into MongoDB. For example we need to defined the schema in which streaming data is intended to get stored in the target.

  Due to above **2 reasons** we are using Kafka that stores our streaming data on temporary basis(by default retention period is 7 days) which later based on certain threshold (eg: 5000 records per iteration) can be consumed into targets like MongoDB for persistent storage(permanent storage). This reduces the frequent connectivity required b/w data source and data target . Also, we get an option to basically register schema in Kafka which ensures that data that is getting Published into Kafka or Subscribed from Kafka follows that intended schema or structure.

- High level Producer architecture:

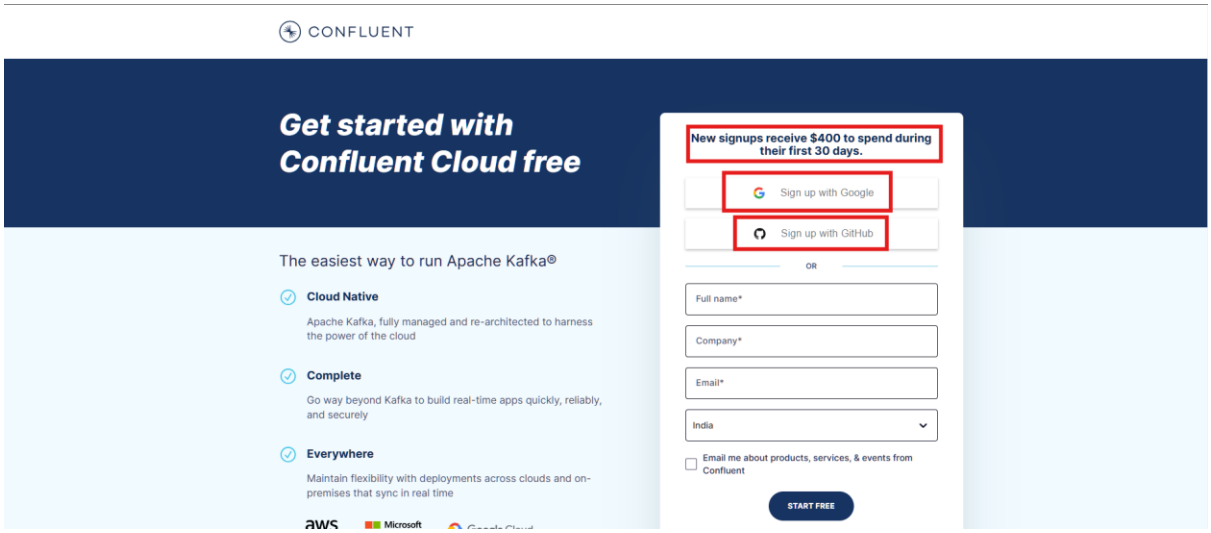


- High level Consumer architecture:



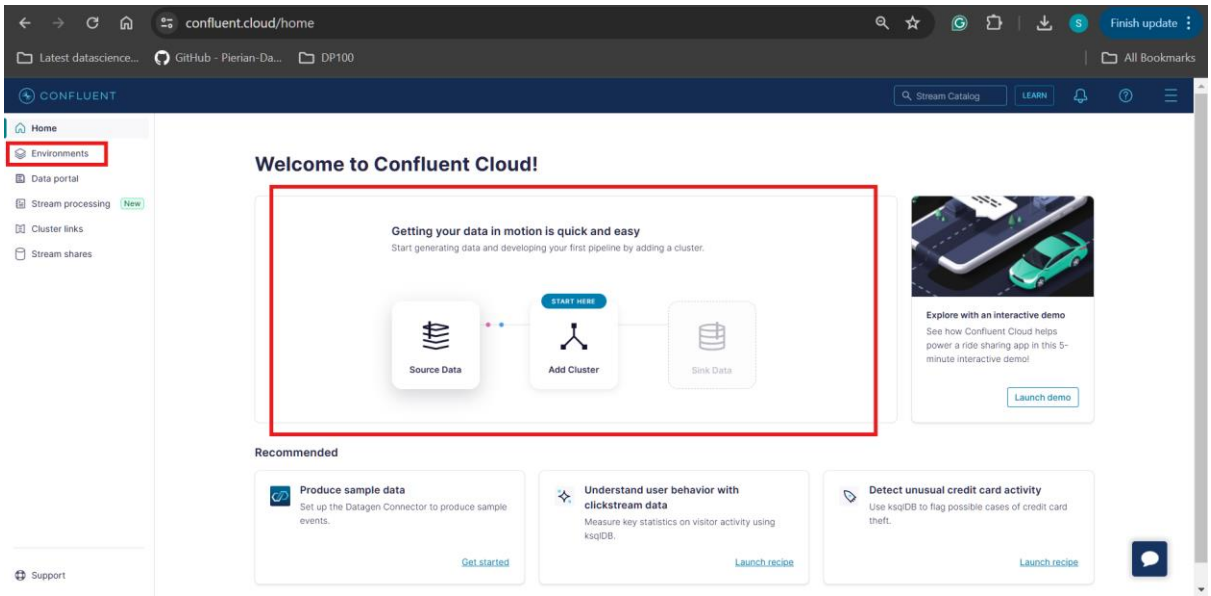
➔ **Confluent Kafka setup**:

- **Step 1 – Signup:**
  - If you are new Sign up to Confluent Kafka using url: https://confluent.cloud/signup
  - We may use our existing **Google account** or **GitHub account** to speed up the sign up process.
  - Confluent Kafka provides free **400 $** credit which we may use to explore the different features available in this **SaaS(software as a service)** version of Kafka.



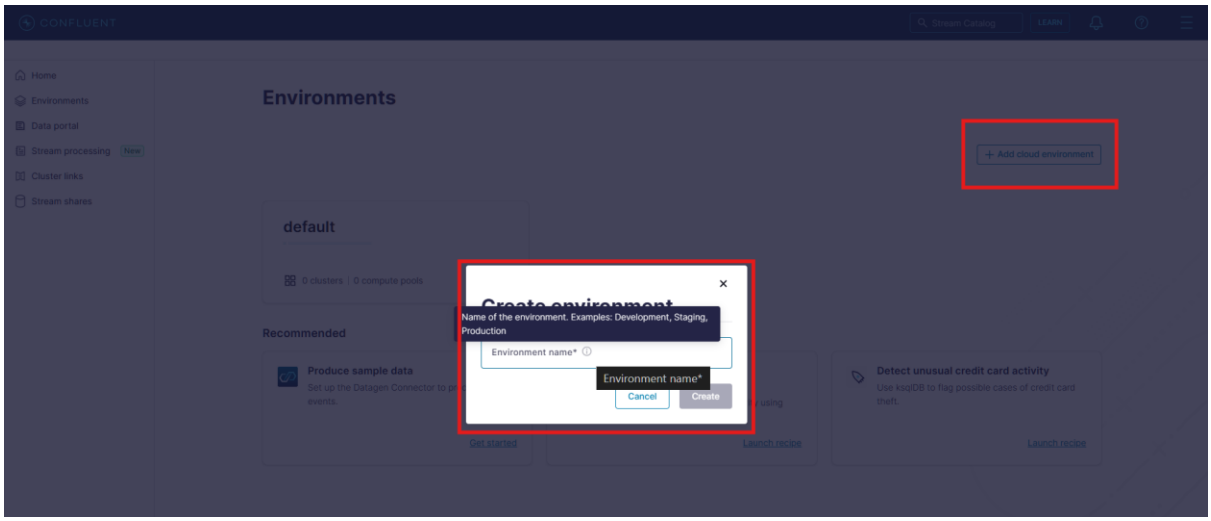
- **Step 2** – Confluent Kafka Cluster setup:
  - A Kafka cluster is a group of Kafka brokers (servers) that work together to handle the incoming(from Producers) and outgoing(to Consumer) data streams for a Kafka system. Each broker is a separate process that runs on a different machine and communicates with other brokers through a high-speed, fault-tolerant network.

  - Once you have successfully logged in, go to the home page using url: https://confluent.cloud/home

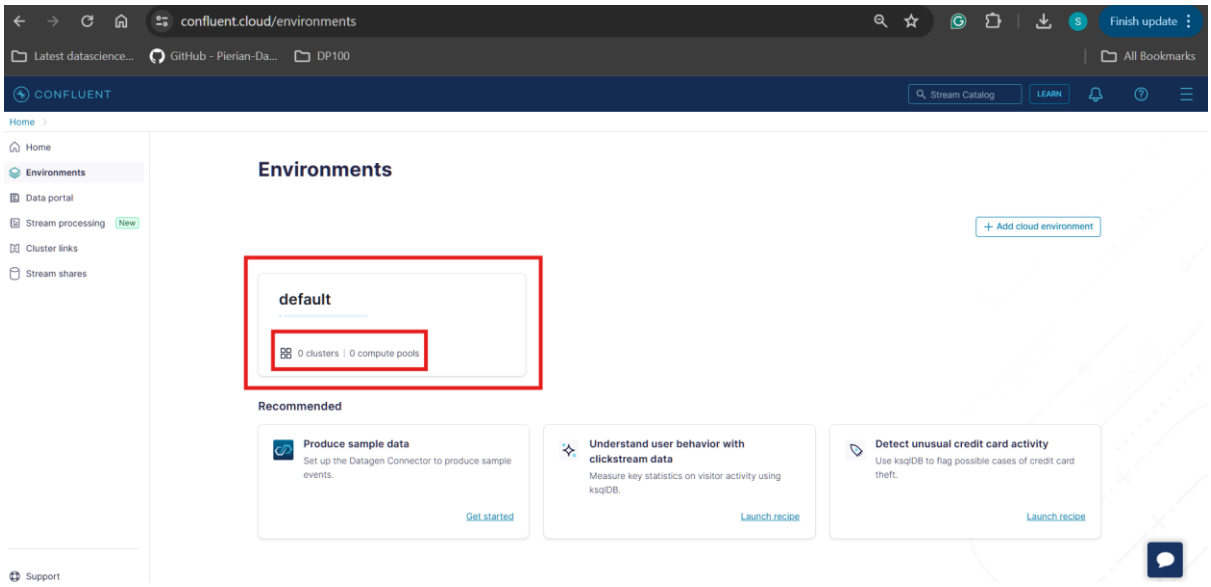


  - **Environments**: Kafka cluster that we intend to build will have certain dependencies like storage to temporarily store the incoming streaming data, compute to pre-process the data(schema matching) etc. So, all these dependencies we are configuring by creating an Environment.
  - Go to the environments section by either using url: https://confluent.cloud/environments or by navigating from the homepage.

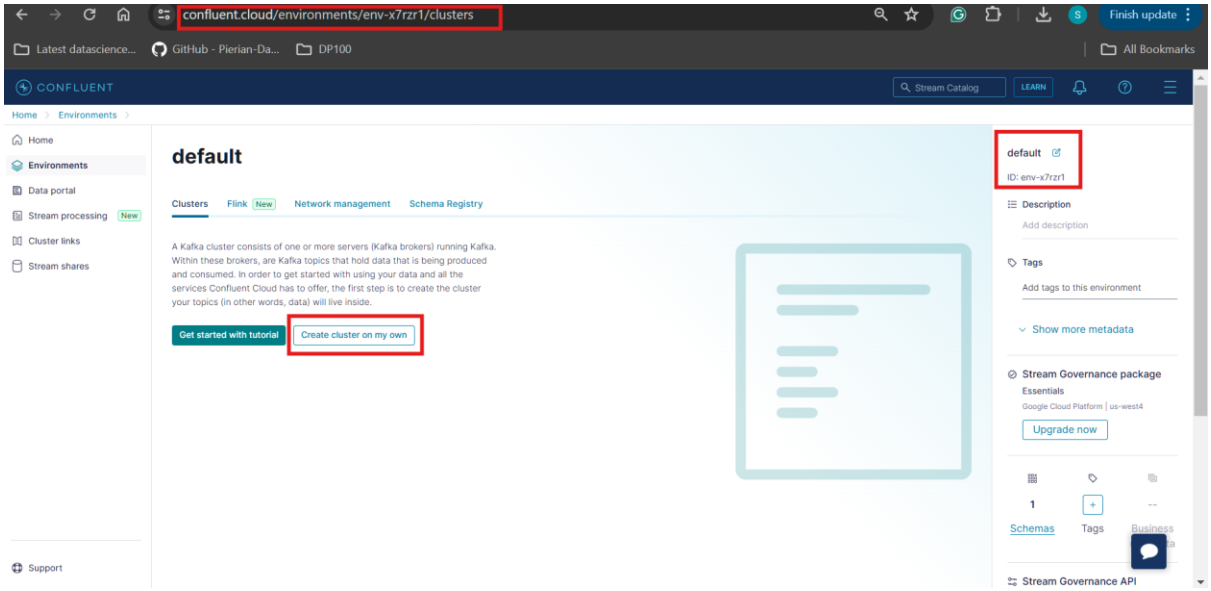
- We can either use the default environment or we can add our own environment with a custom name like Production, Development or Staging.



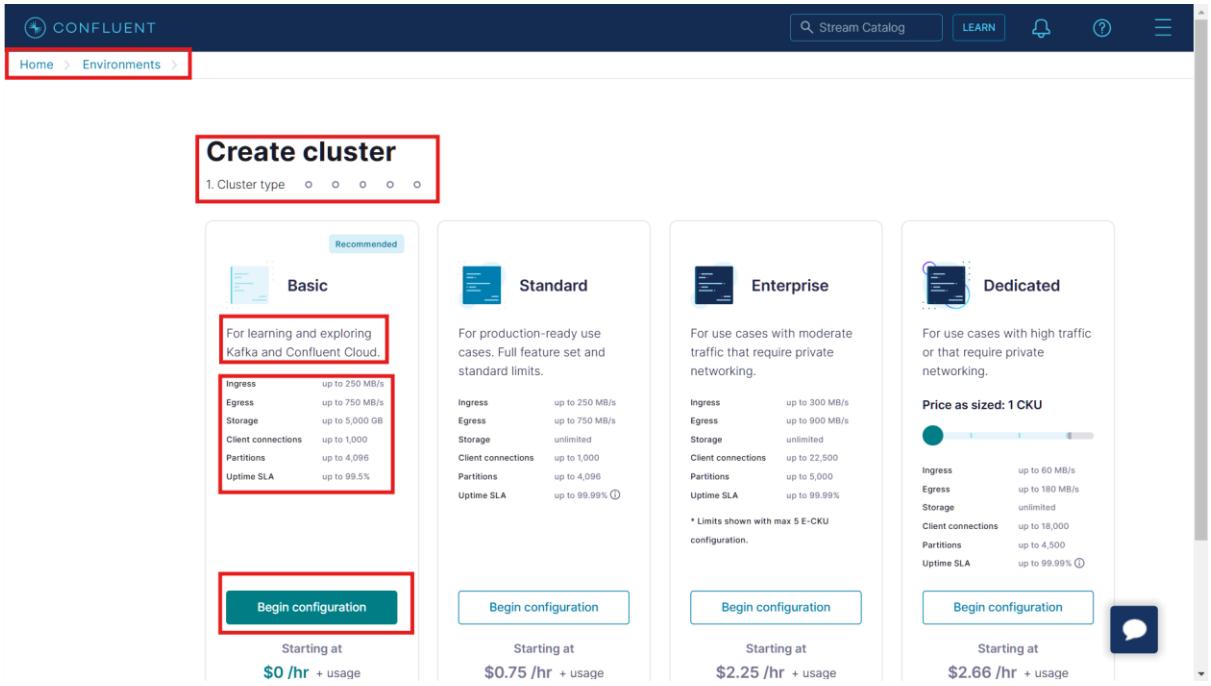
- For our use case we would be using the default environment.



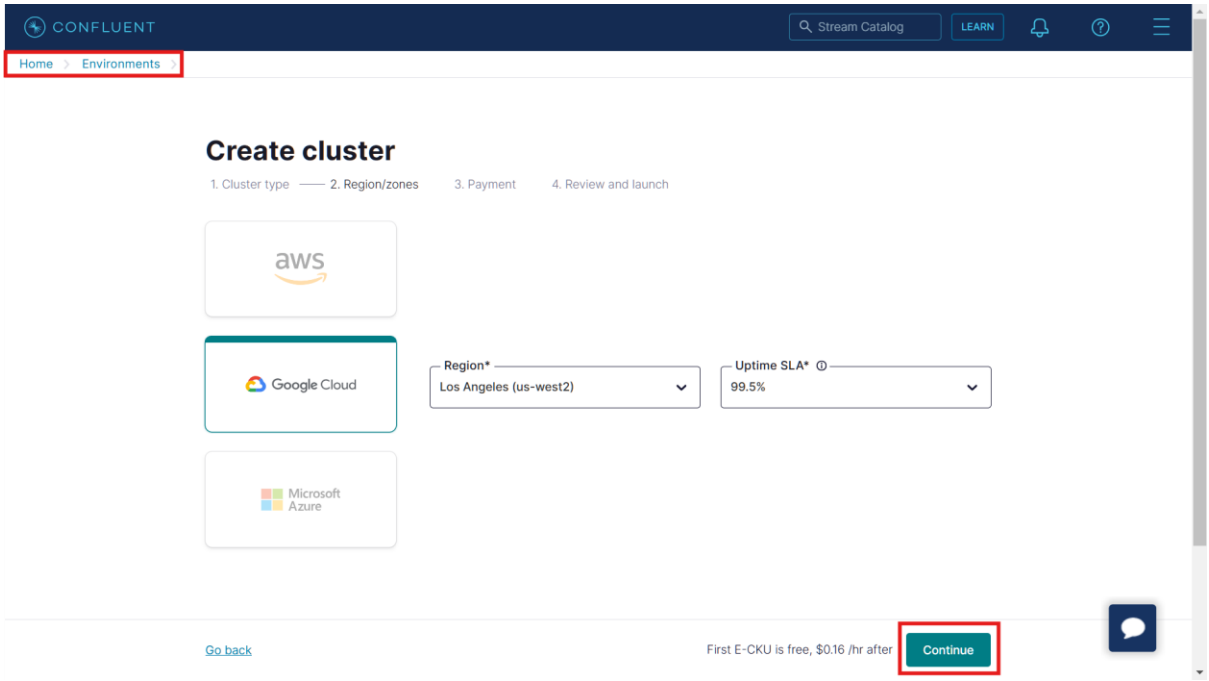
- If this is the first time we are creating a cluster in our default environment then we would be redirected to a page prompted with to **Create cluster on my own**. Also, observe in the URL followed by *environments/* there is an ID present which is representing the ID of our default environment.
  Please note that: Kafka internally assigns a unique ID for every environment which acts as means to uniquely identify the same.



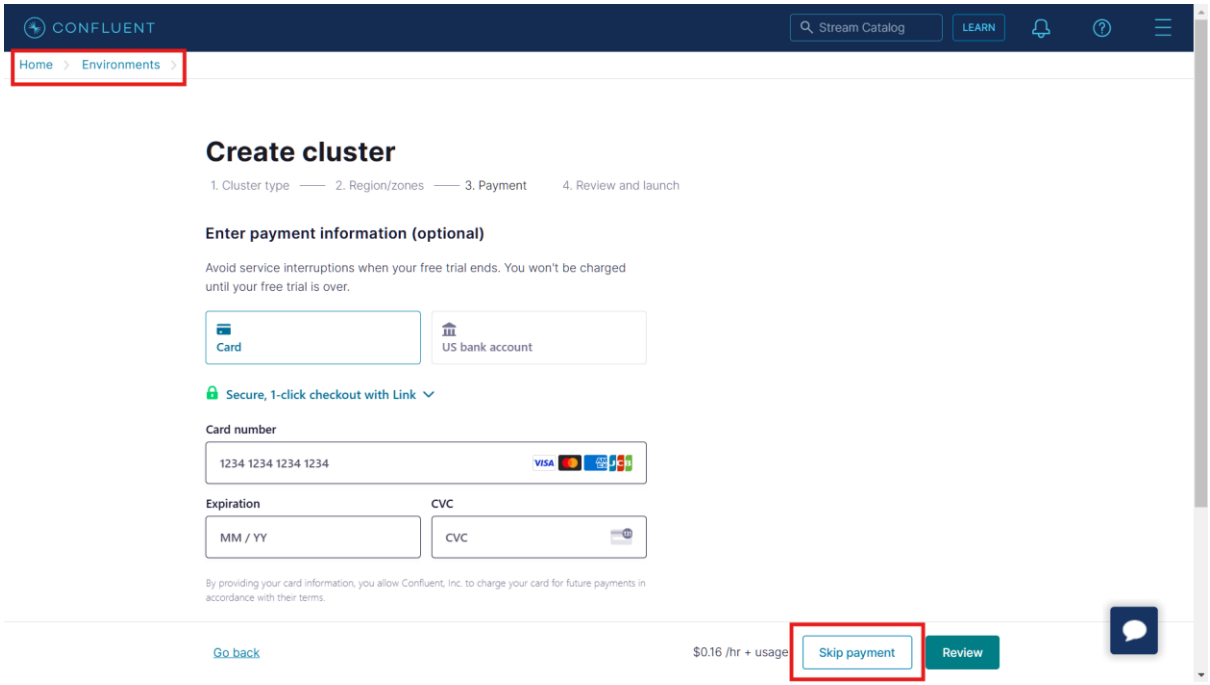
- When creating the Cluster as a first step we need to select the **Cluster type**. Based on use case and different configuration settings(representing the dependency of environment) we can choose a one according to our use case. Right now since we are creating cluster for learning and exploring the Confluent Kafka we would be using Basic version of cluster.



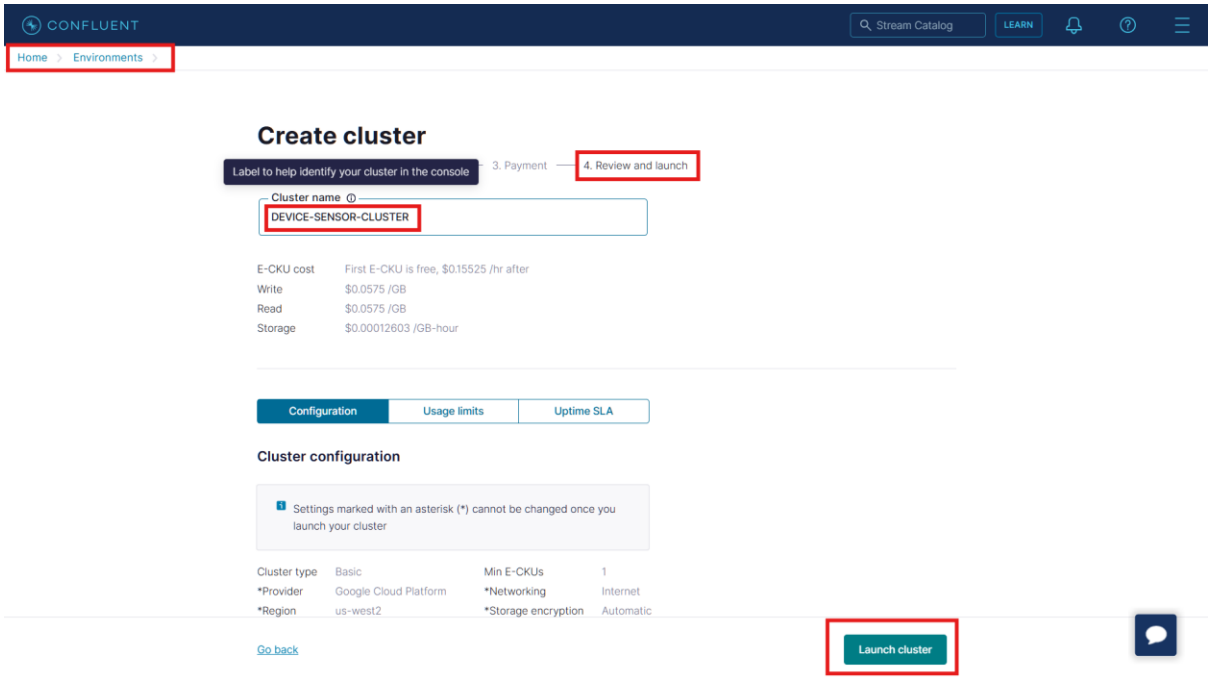
- In next step we would be selecting the Cloud provider, Region and Uptime SLA.
  Tip: Choose region closer to you for better real time inferencing and higher Uptime SLA for production read use case.



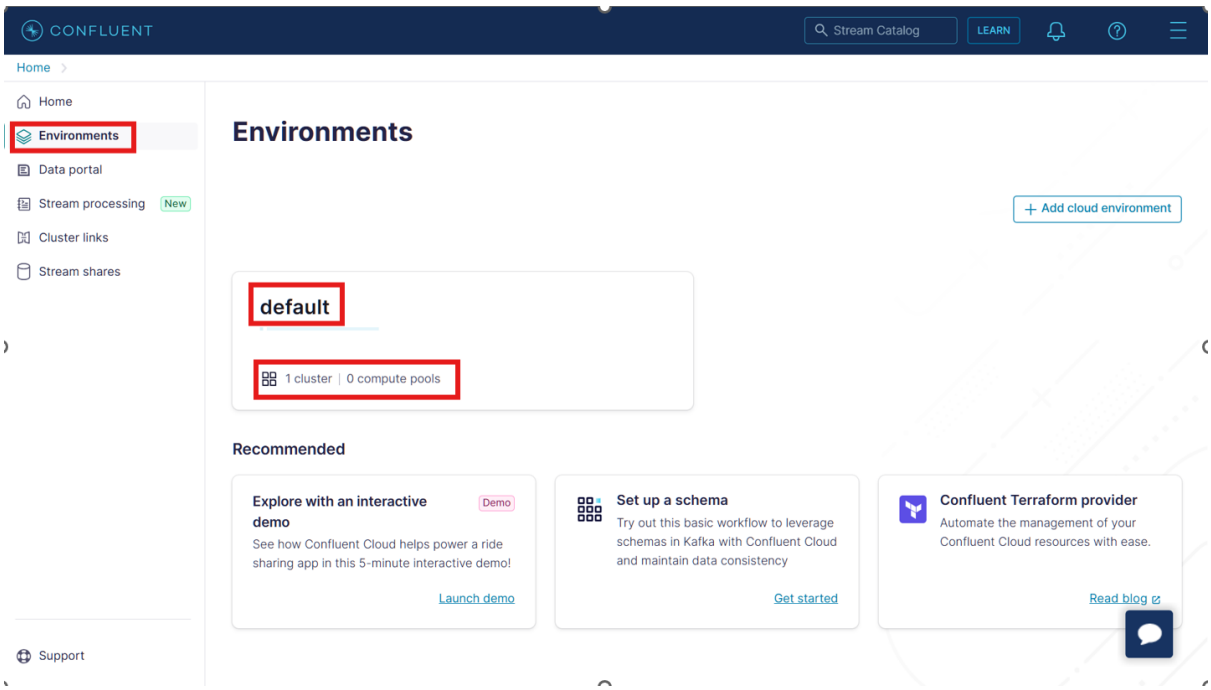
- When prompted to payments page choose **Skip Payment** option as we are currently using Kafka for learning and exploration purpose using the free 400$ credits.



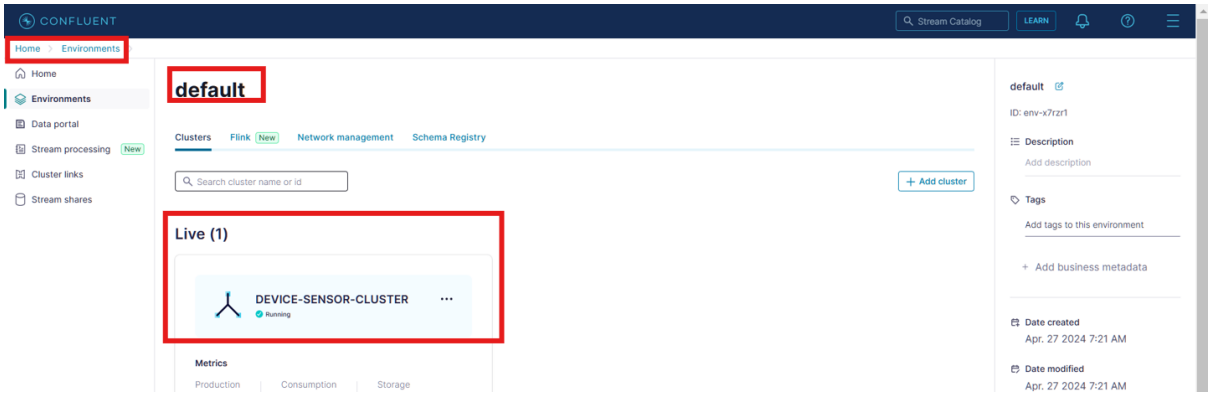
- Now, give a name to your cluster. Once you have verified all the meta information regarding your cluster, it's configuration, usage limits and uptime SLA click on **Launch Cluster** button to initiate the cluster build process in the default Confluent Kafka environment.



- Once cluster has been built successfully traverse back to the **Environments** section. Under default environment, now we can see number of cluster as 1 which was 0 earlier. This signifies that our cluster built is completed.



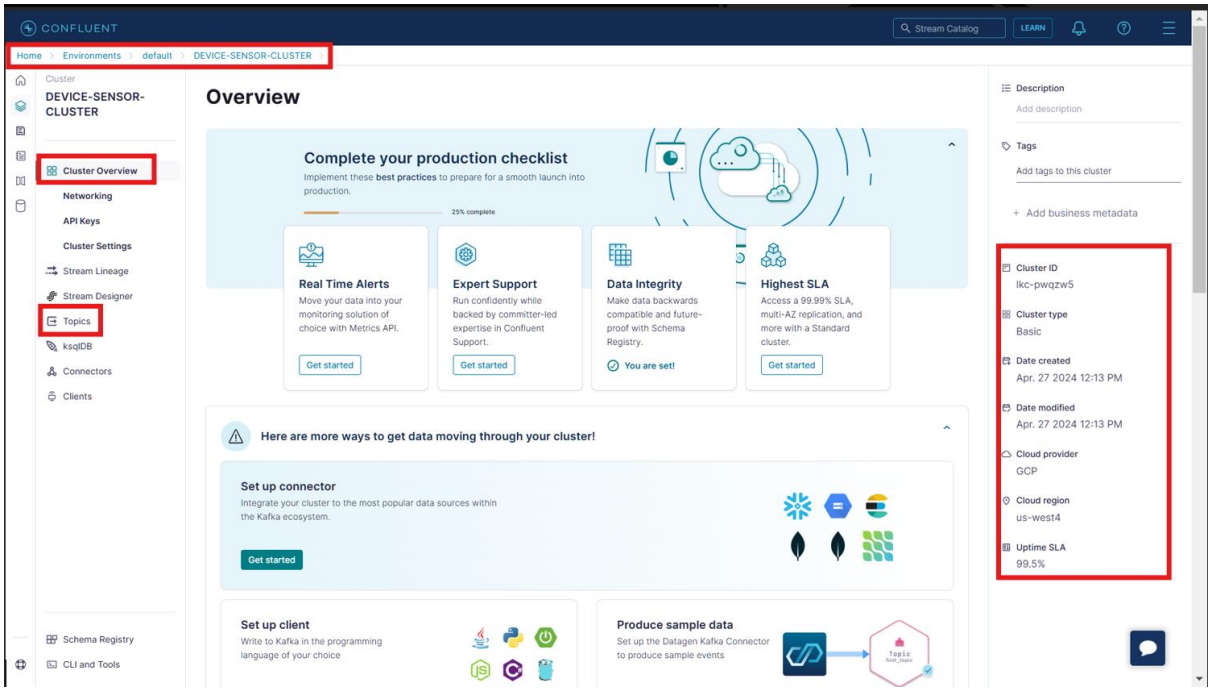
- If we go inside our default environment then we can see the newly built cluster is **Live** now.



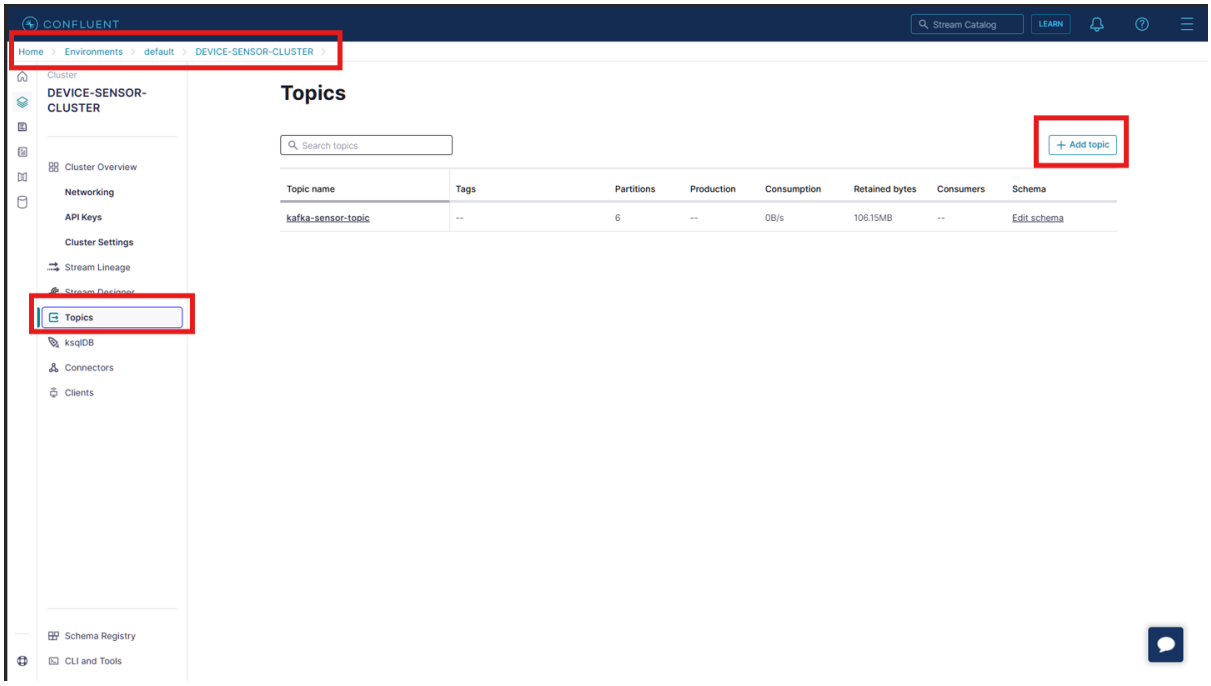
- **<u>Step 2: Kafka Topics setup</u>**

  - Next we need to create the **Topics**. Traverse inside the newly created cluster and click on the **Topics** section.

    Topics are the section that basically stores the live streaming data/messages Published to Kafka from Producer's end and then make it available for Consumption to the Consumers.



  - Click on the **Add topic**:

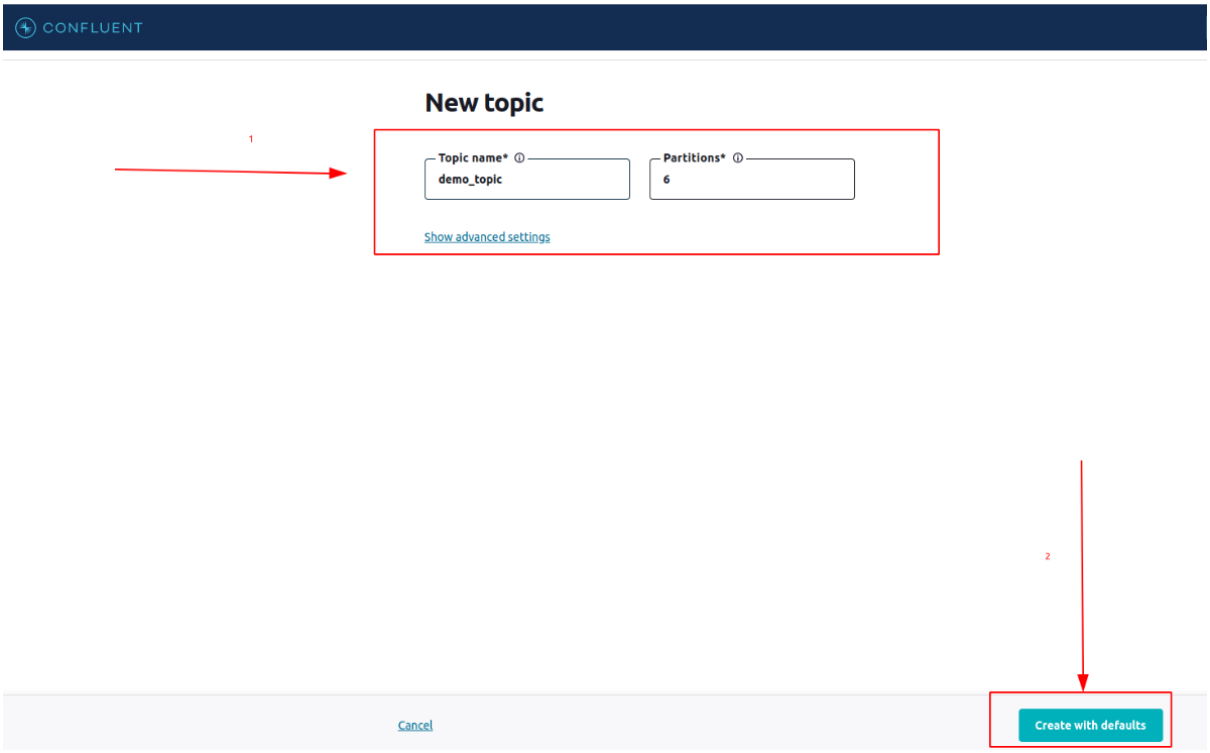


- Give a name to your Topic and the no. of Partitions followed by clicking **Create with defaults**.

  Topic_name: A Kafka cluster can have more than one topics. But we have to make sure that all these Topics are having unique name across the cluster.

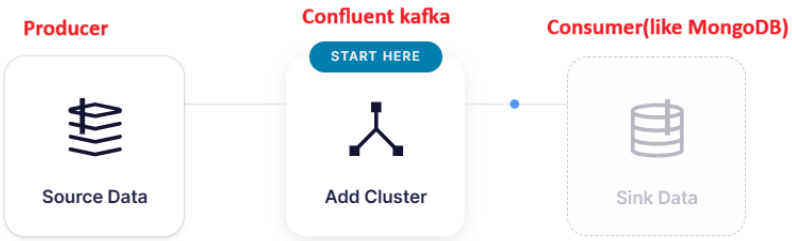
  Partitions: Let understand about Partitions with this beautiful example. Let's assume that there are 200 people waiting in queue to submit their forms. So instead of having one person to collect all 200 submissions we can split the queue in 20 groups(with 10 people in each group) and then assign a person to collect submissions from the each group. Now let's compare the same with topics. By setting the partitions, topics is divided into smaller groups so that publishing of data from Producer's end to Kafka can get fastened. Also, within each partition there is an order followed(which message/data came first, second third) but across partition order may not followed(partition1=data_from_person(1^{st},4^{th},5^{th}) and partition=data_from_person(2^{nd},3^{rd},7^{th})).

  Please note: Speed or rate of Production/Publication of data or message to Kafka from Producer's end is having direct relationship with number of partitions present in the Topics. That is higher the value of partitions, higher is the rate of Production/Publication. Therefore, number of partitions can be seen as the hyperparameter that can drastically affect the Production rate to Kafka topics.



- <u>Step 3: Generating the Secret keys</u>

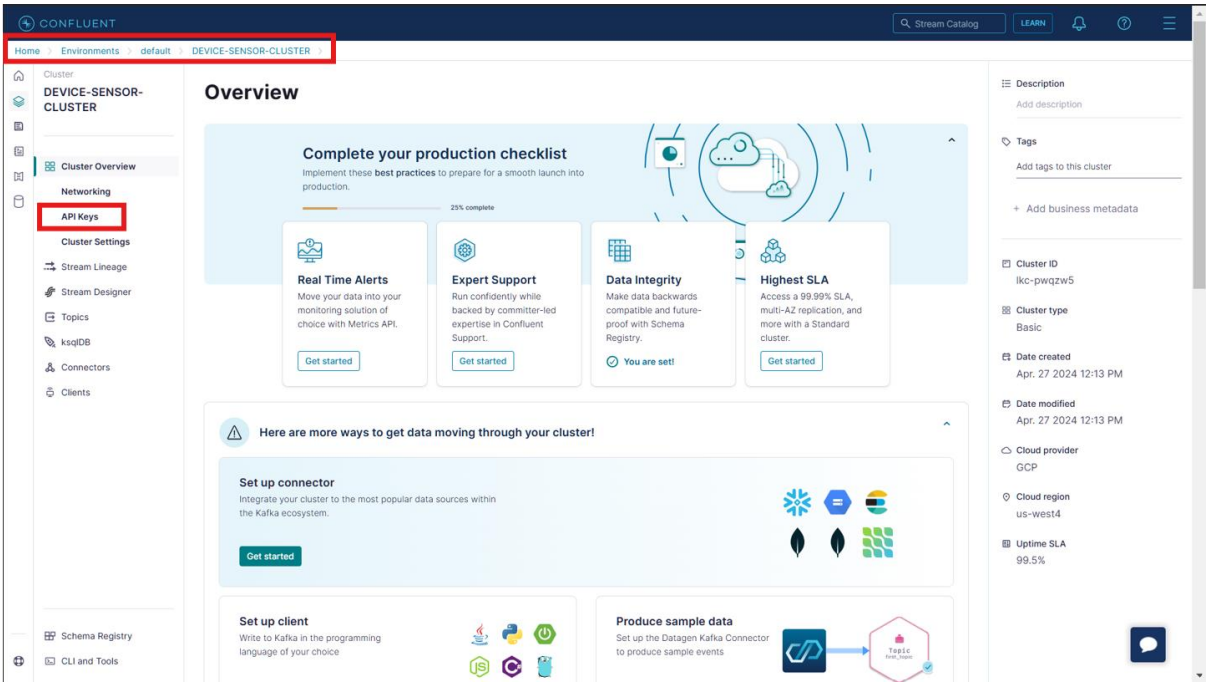
  - Once topic(s) is/are created now we would be required to obtain the secret keys from Confluent Kafka dashboard that would help us establish the connectivity with Confluent Kafka remotely so that we can use its services for creating the intended end-to-end streaming big data pipeline by integrating Producers with Consumers via Confluent Kakfa.



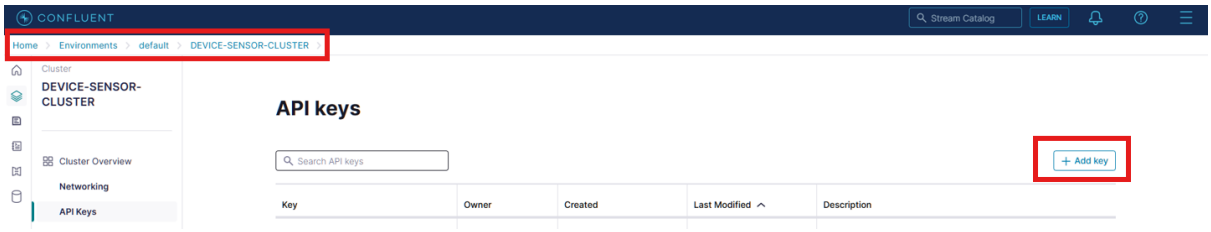
  - There are two different set of secret keys that we would be requiring – Cloud specific and Schema Specific.

  - Let's first try obtaining the **Cloud specific keys**:
    These secret keys are the API keys that would help us to establish connectivity with Confluent Kafka Cluster and access different functionalities within it.

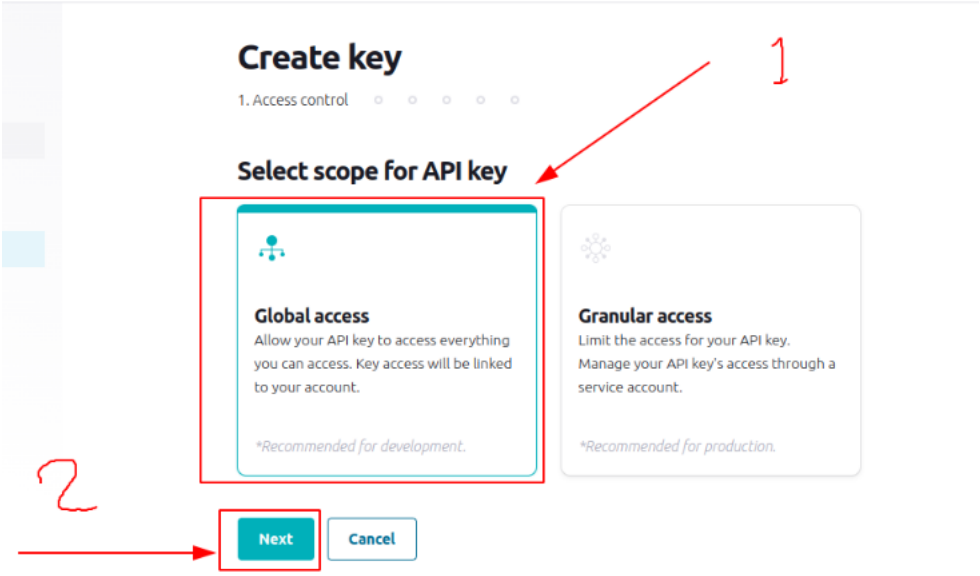
    Traverse inside the cluster that we built earlier and open the **API keys** section



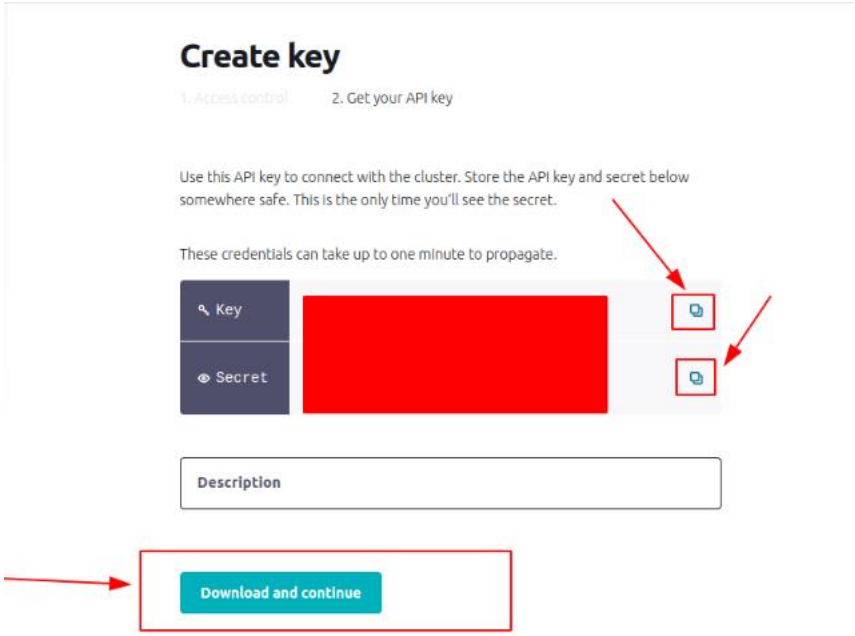
- Click on the **Add key** button:



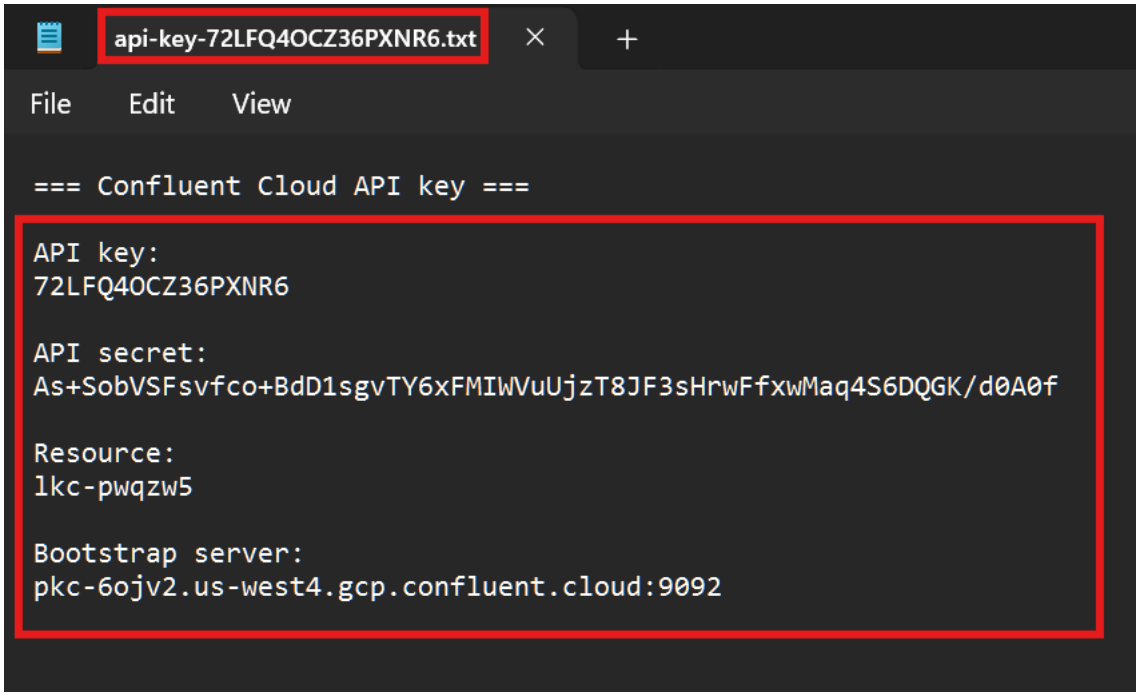
- Select **Global access** as access control and click on **Next** button.



- In the *Get your API key* page we can now either copy the generated Key and Secret and use them while establishing connectivity with the cluster. Alternatively (*recommended*), we can also click on **Download and continue** button which will download a text file with all the necessary secrets.



- This is the download .txt file with all the secrets which we will be using later to establish connectivity with Confluent Kafka cluster.

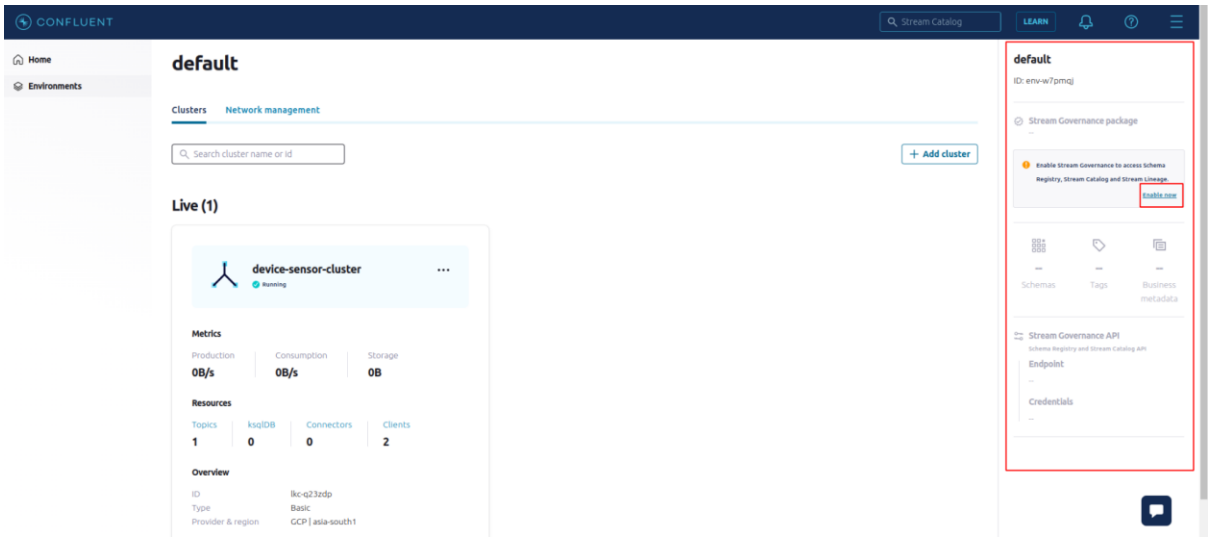


- Now, lets obtain the **Schema specific secrets**:

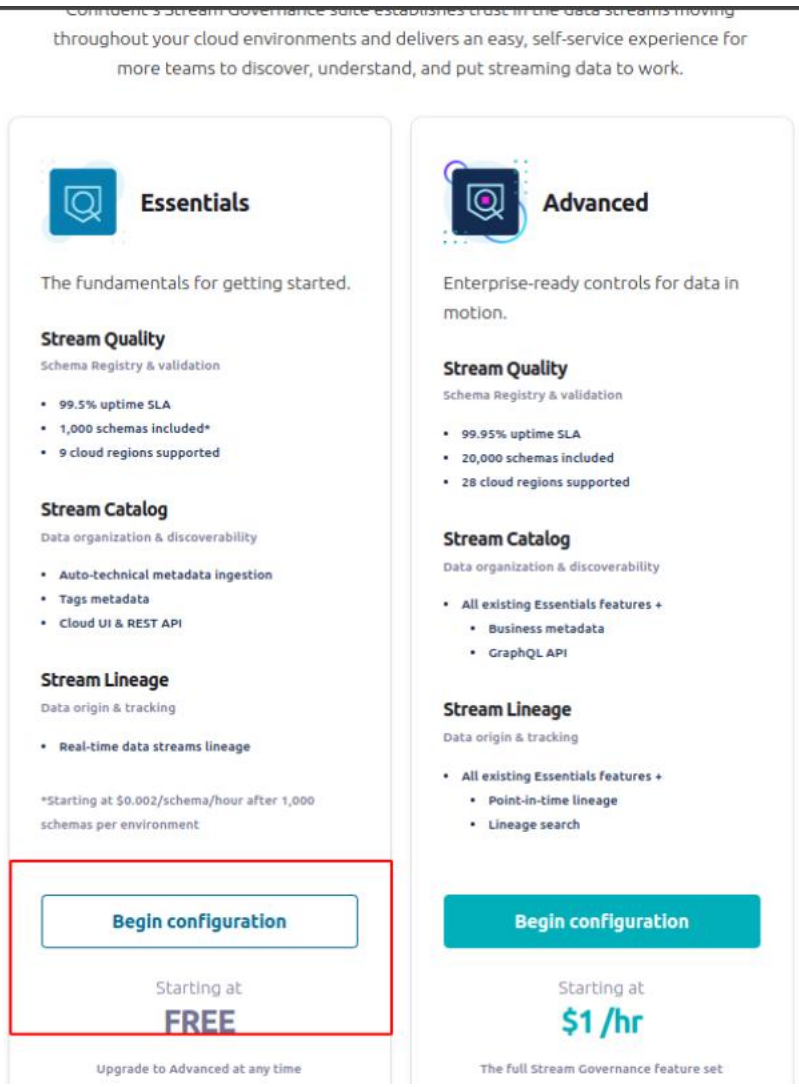
Note:

- When we are creating a big data pipeline using Kafka then we need to feed the Kafka with a Schema file.
- This Schema file is stored inside Schema Registry which is Schema storage service provided by Confluent Kafka.
- Reason why we are feeding this file is because it gives an idea to Kafka regarding the structure in which it has to Publish the data/messages coming from Producers into Topics and the intended structure of data/messages(in Kafka Topics) that needs to be sent out for Consumption to Consumers.
- We can have separate schema file for each Producer and Consumer.
- There are different formats of schema file that Confluent Kafka can access like: json format, avro format.
- We will be using json format of schema file.
- Please note that we can register schema using Confluent Kafka UI dashboard. But in our use case we will be defining the schema and registering the same during big data pipeline development phase. This is the reason why we are trying to generate **Schema Specific Secrets** which would help us to establish connectivity with **Confluent Kafka Schema Registry** for registering the intended schemas.

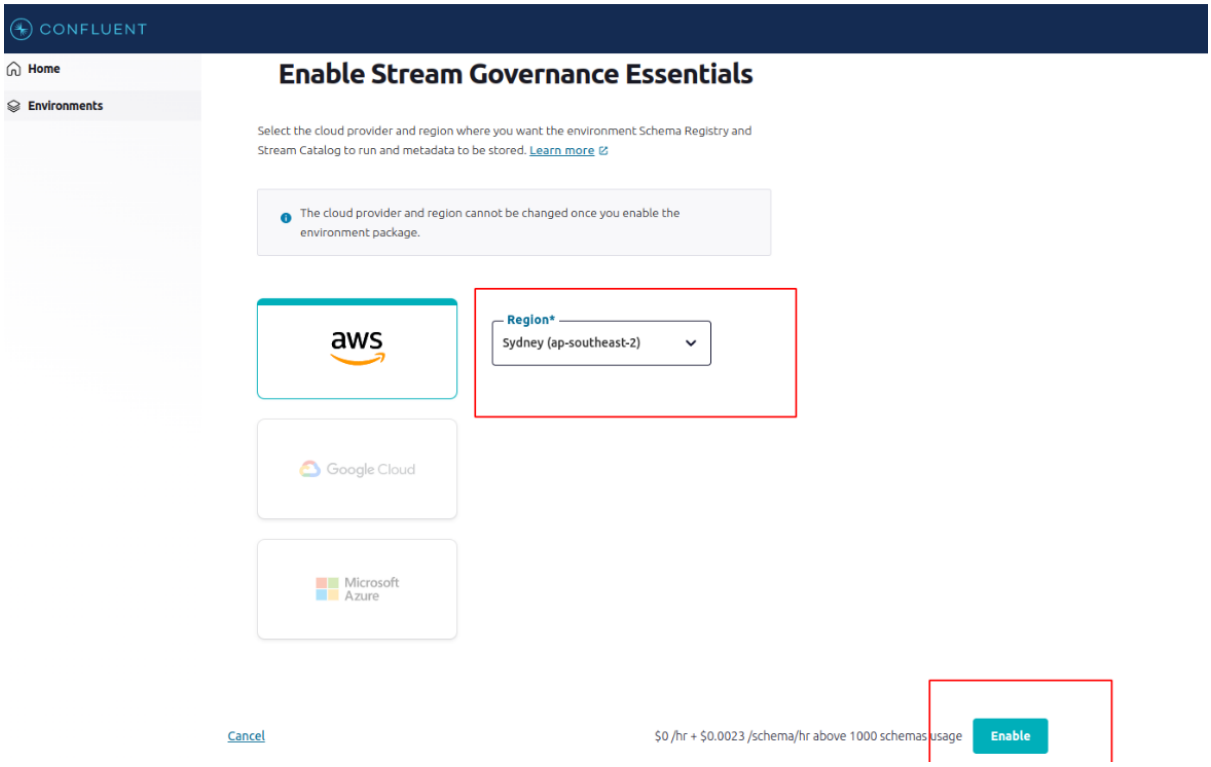
- Traverse into **Default environment** where our cluster has been built. Since we will be creating the schema registry for the first time we need to first of all enable it. Click on the **Enable now** button under **Stream Governance package**



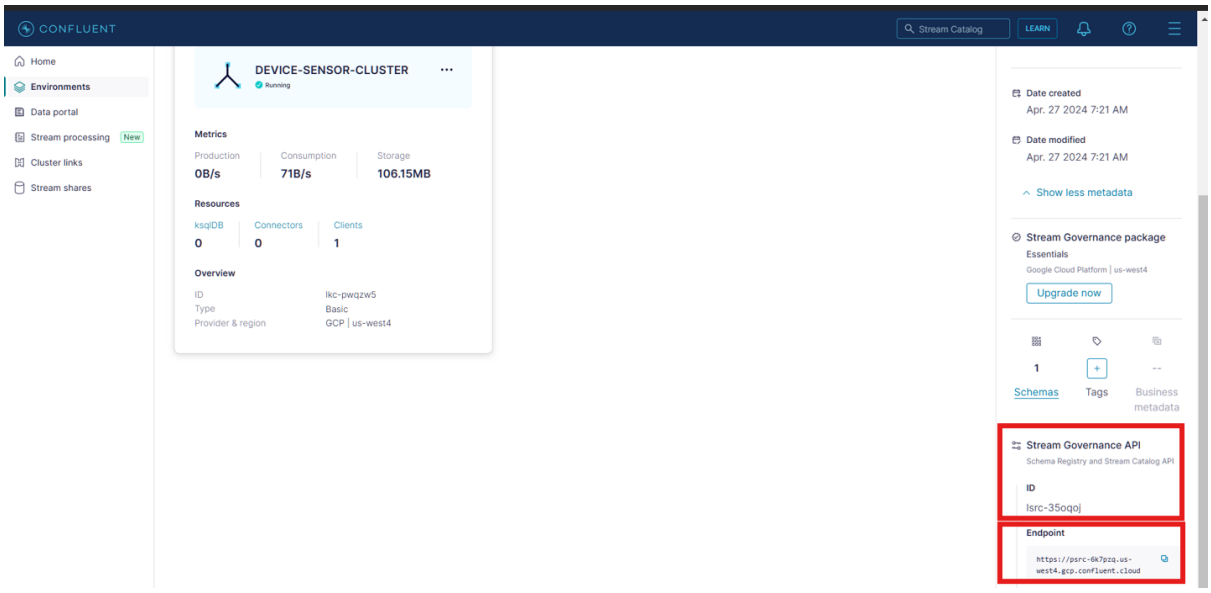
- The *free version* of **Schema Registry** can support up to 1000 schemas which is sufficient for us to explore and implement our use case.



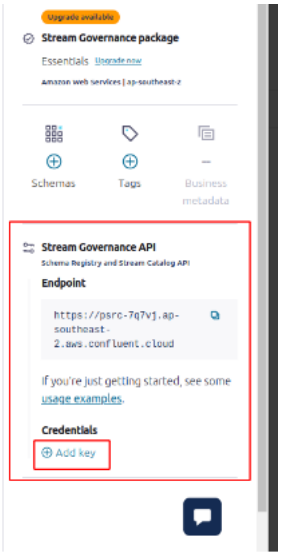
- Next select any of the cloud provider and a region closer to your current location for better real time inferencing and click the **Enable** button



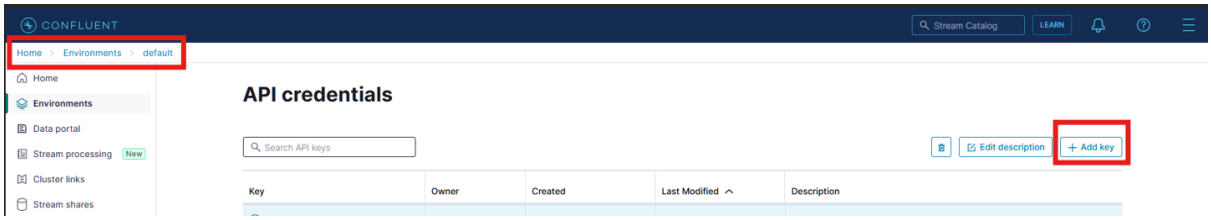
- Once we have enabled the **Stream Governance Package** then **Stream Governance API** will also get enabled. We need to copy and save the highlighted **Endpoint** URL which would be used as one of the secrets while establishing connectivity with the Schema Registry.



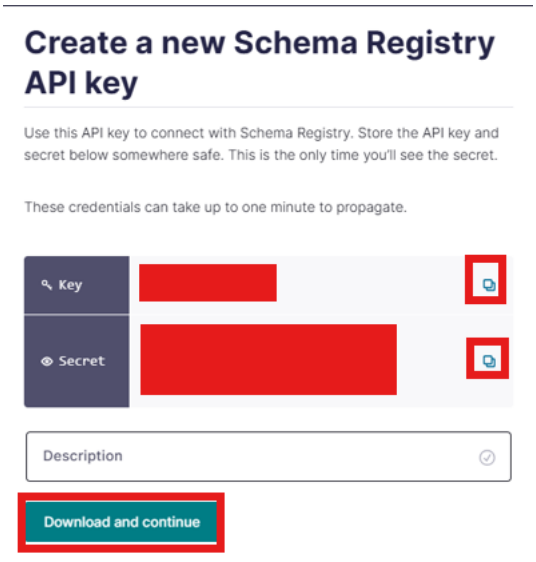
- Below Stream Governance API we can see **Credentials** section with a **Add key** button. Click on this to generate the API keys required to connect with schema registry remotely:



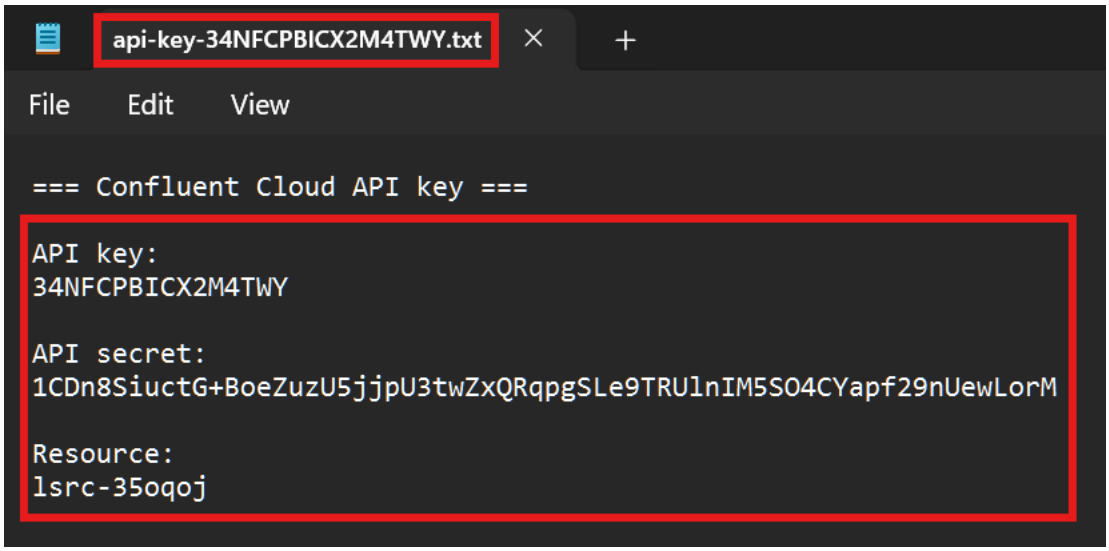
- Click on Add key:



- Now either copy the presented key of Secret or alternatively(recommended), click on **Download and continue** button to download the *.txt file* containing the secrets specific to the Schema Registry:



- This is this downloaded .txt file capturing secrets of schema registry:



Please note:

- Don't use the secrets that has been shared in this document because by the time this community documentation is published, these secrets will be no more in existence. Instead, create one for yourself self-following the steps above.
- Also please do not share these secrets with anyone as these are meant for private usage and not for general public.
- Make sure that you don't hard code these secrets in your development environment instead set these as environment variable. We will see this approach while implementing our big data pipeline.

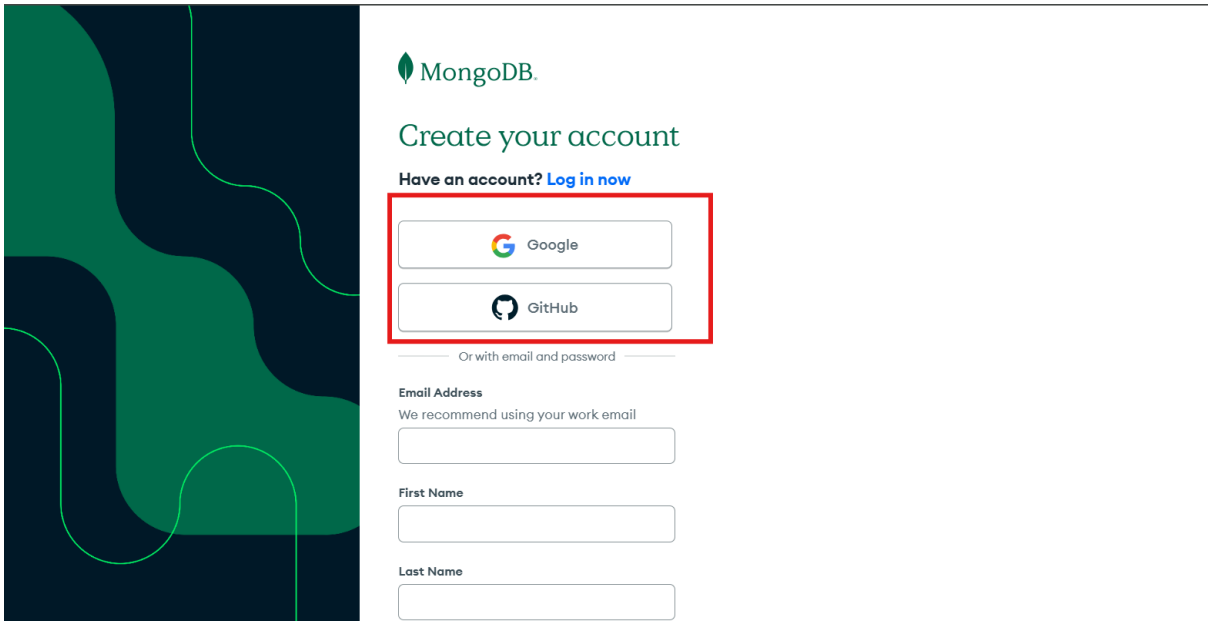
➔ MongoDB setup:

- **<u>Step 1: Signup with MongoDB:</u>**

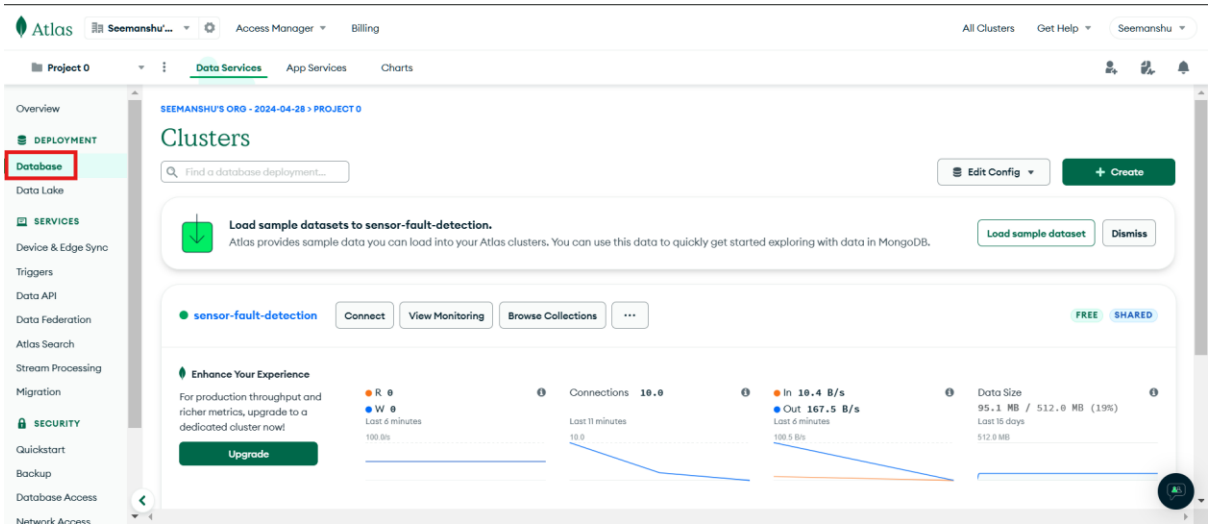
  - Kafka topics by default have the **retention policy** of **7 days**. This means by default post 7 days of Publication of data/messages to the Kafka topics it will get deleted from there.
  - This implies that Kafka does not provide the persistent storage(permanent storage). It can temporarily store the data/messages.
  - To persist our data or to permanently store our data we require another storage resource that does not have any retention rule similar to Kafka. This is the reason why we are use MongoDB where Consumers will be dumping the data.

  - Signup to MongoDB using URL: https://account.mongodb.com/account/register

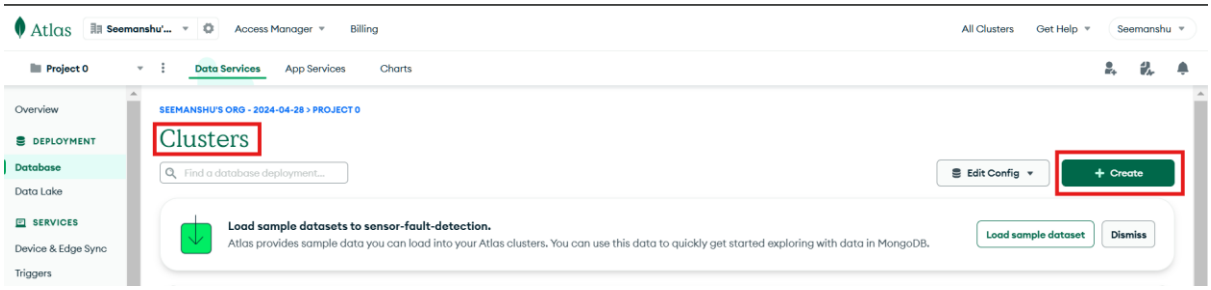
  - To make the signup process quick we can signup using our **Google** or **GitHub** account:



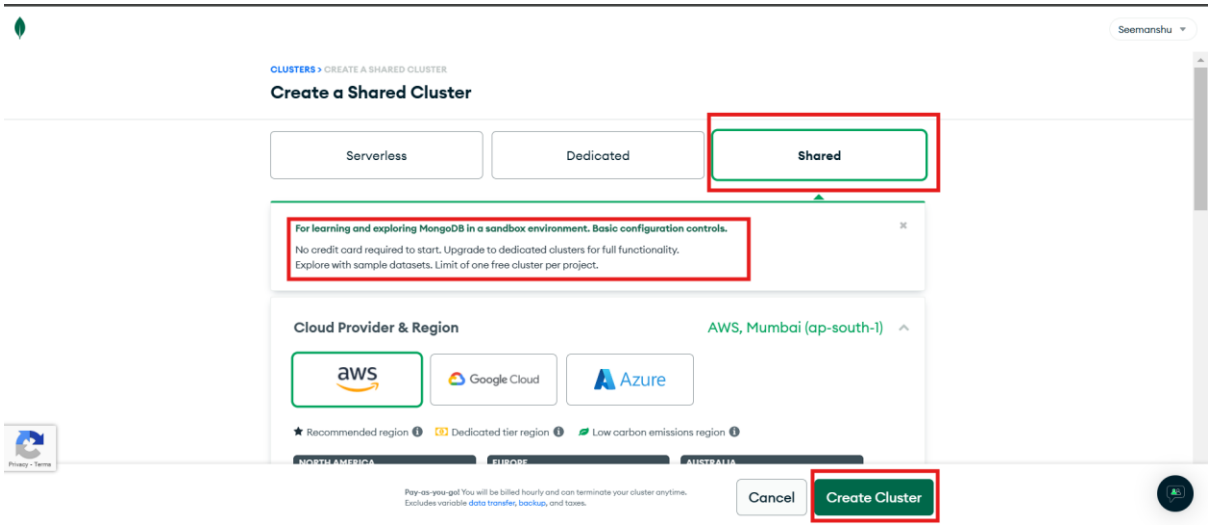
- **<u>Step 2: Create MongoDB Cluster:</u>**
  - A MongoDB is a No SQL database that is managed and stored under by the MongoDB Cluster. A MongoDB can have 1 or more than 1 databases.
  - Once you have logged in, go to the homepage of MongoDB and click on **Database** section:



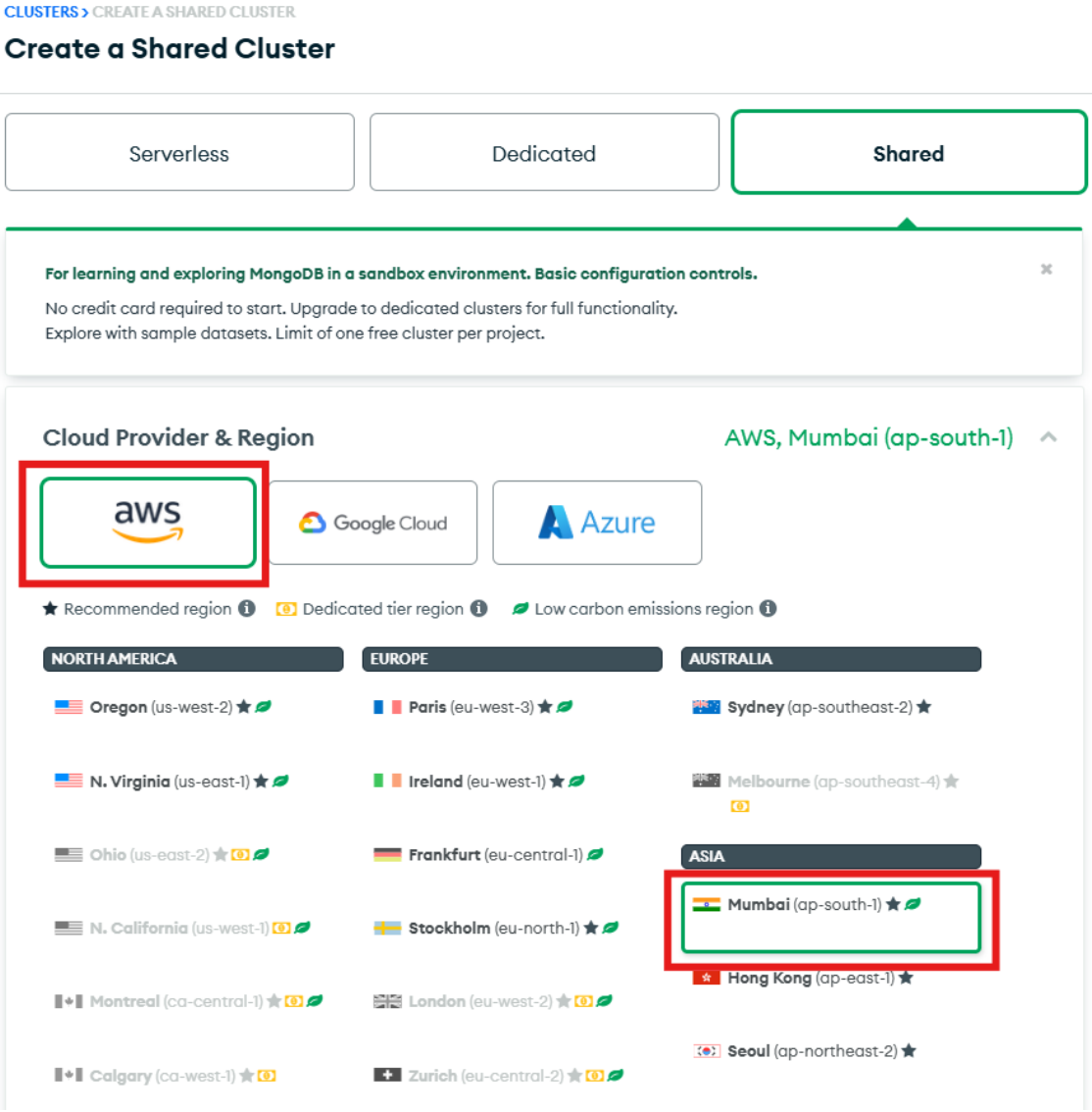
  - If this is the first then your are working with MongoDB then you would be required to create the a MongoDB cluster by clicking on **Create** button:



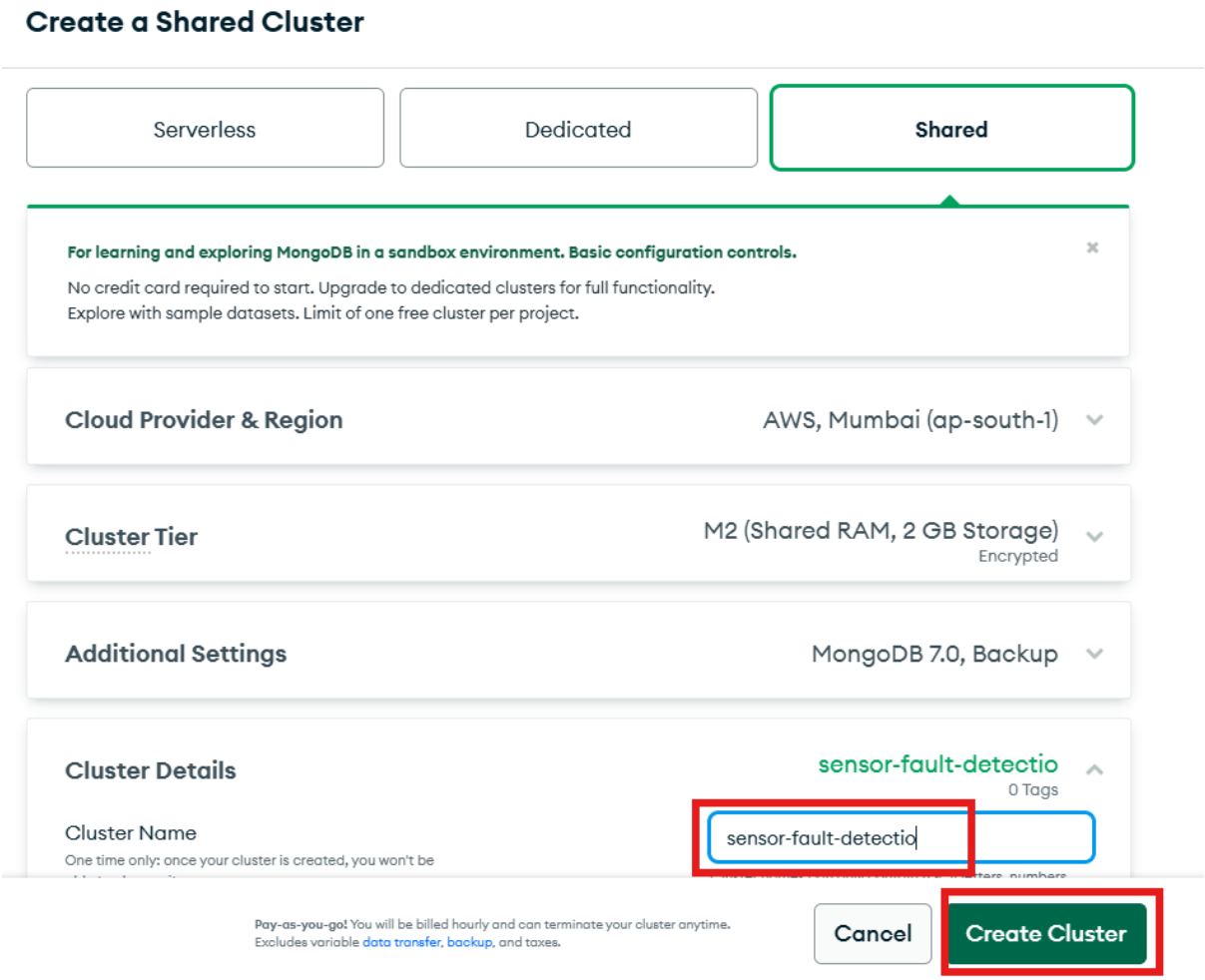
- MongoDB has 3 kinds of cluster: **Serverless, Dedicated and Shared**. Based on our use case we can create a one. In our use case we can use **Shared** cluster. Reason why we are using Shared cluster is because MongoDB provides on 1 shared cluster per project **Free of cost** which is sufficient to implement our big data pipeline use case.



- Choose any Cloud provider and a region closest to your current location:

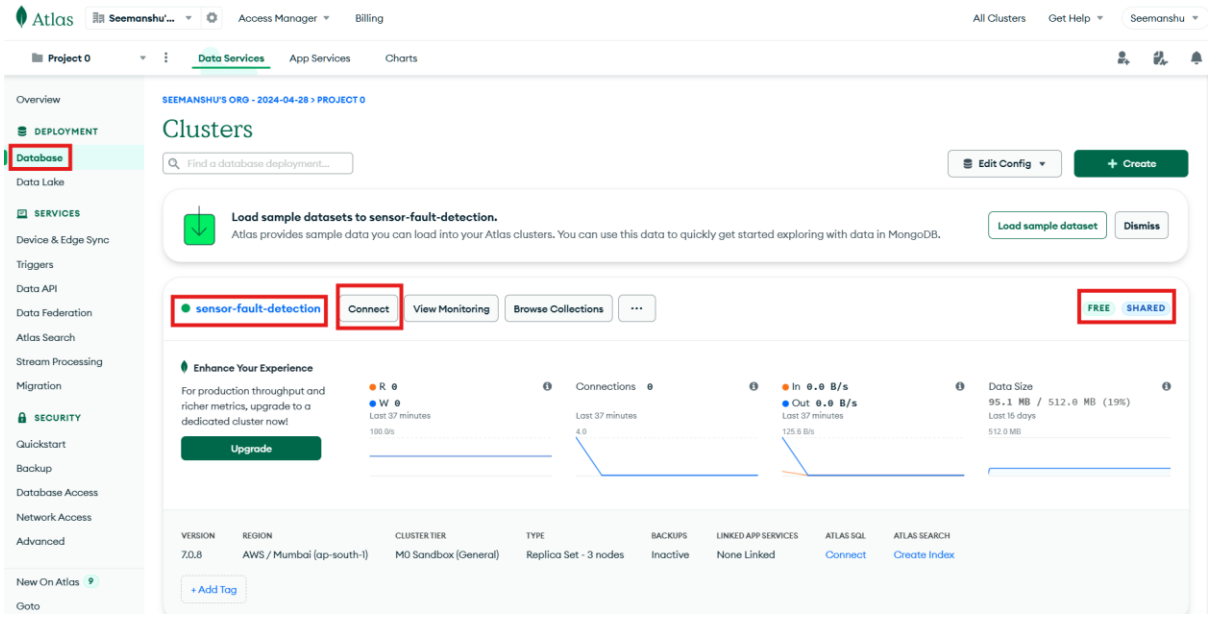


- Choose name for your cluster and click **Create Cluster** button:



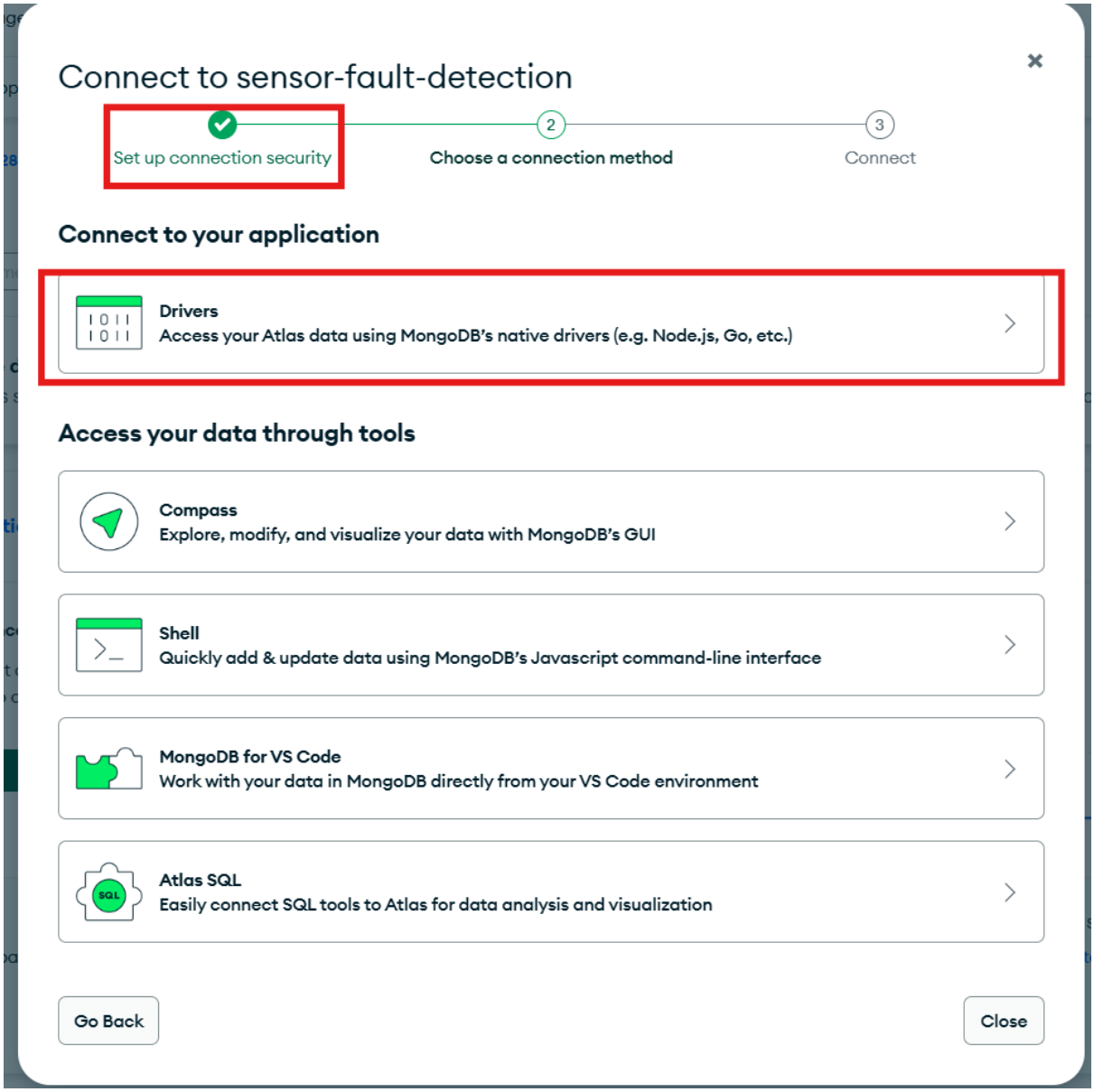
- **<u>Step 3: Generate connection string:</u>**
  - Connection string can be used to access the MongoDB cluster remotely.
  - In our use case a **Consumer** script will be dumping data from Kafka topics into MongoDB. Therefore, while building the data pipeline for the same we will utilize the generated connection string.

  - Once MongoDB Cluster is created go back to database section. Now you can see that the cluster which we have created. Click on **Connect** to generate the connection string for the respective cluster:

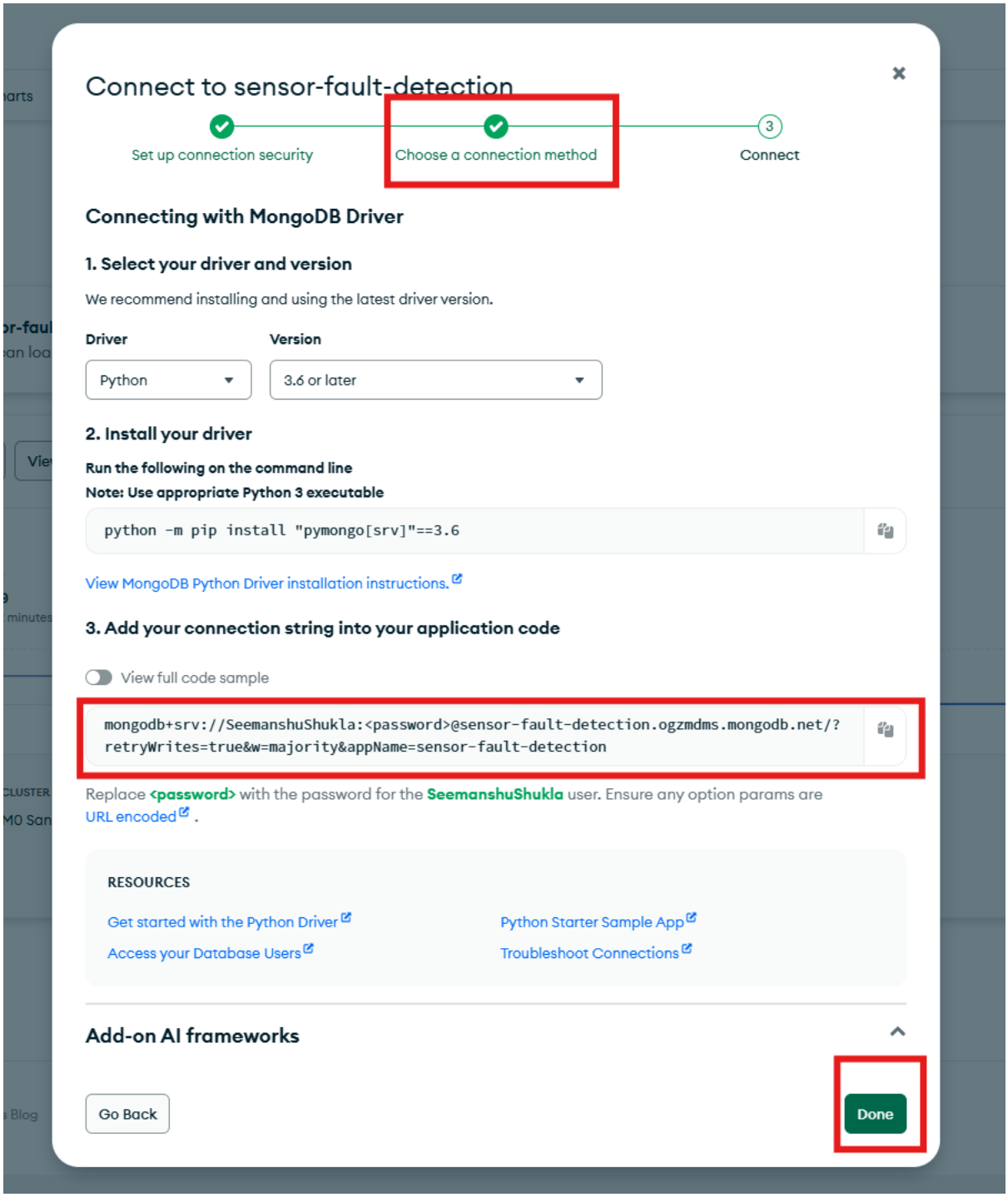


- In the ***first step,*** we are basically selecting the tool that we can use to connect with MongoDB atlas application where we wish to dump the data. There can be several tools to connect with MongoDB atlas application remotely. In our case since we would be required to build the pipeline from scratch via coding, choosing respective Drivers would be the best option. Click on **Driver:**

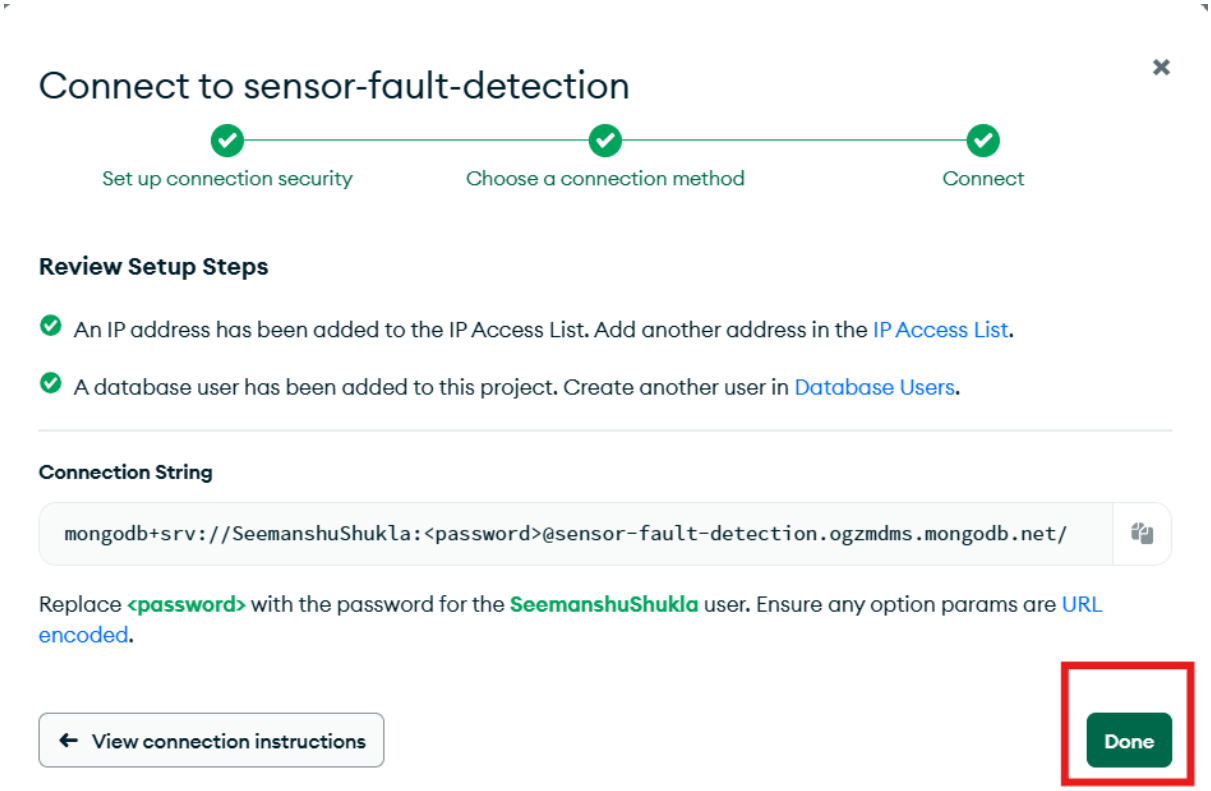
  Please Note: The driver that we are referring to here is a Python package **pymongo[srv]** which we can install in our development environment using command ***python -m pip install "pymongo[srv]"==3.6***



- Copy the highlighted connection string and Click **Done** button. We will be using this in the code while designing the data pipeline for **Consumers**.

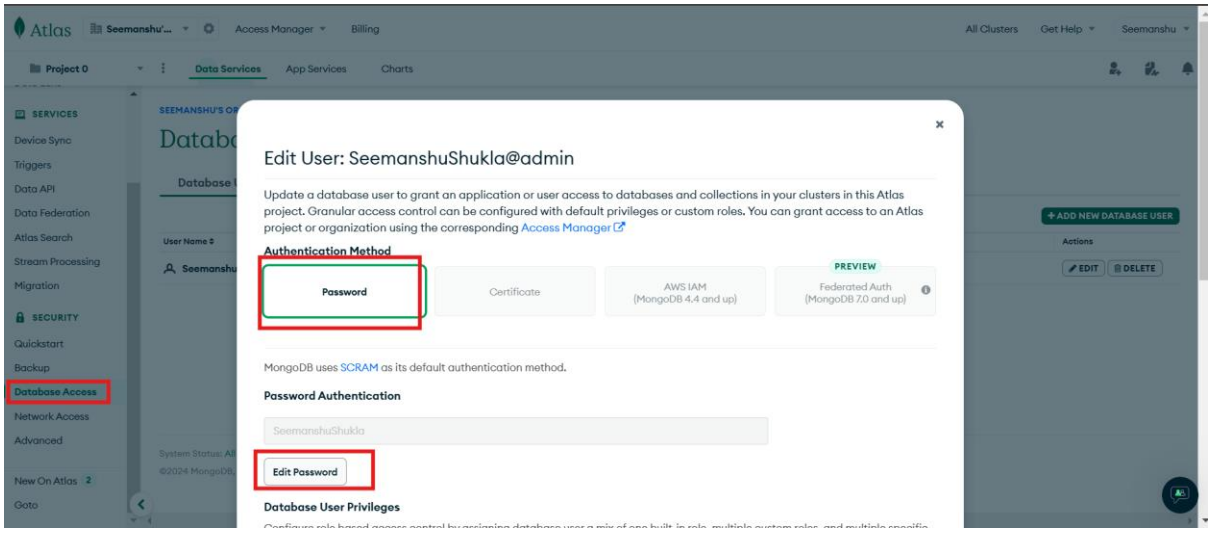


- In the third step click **Done**:



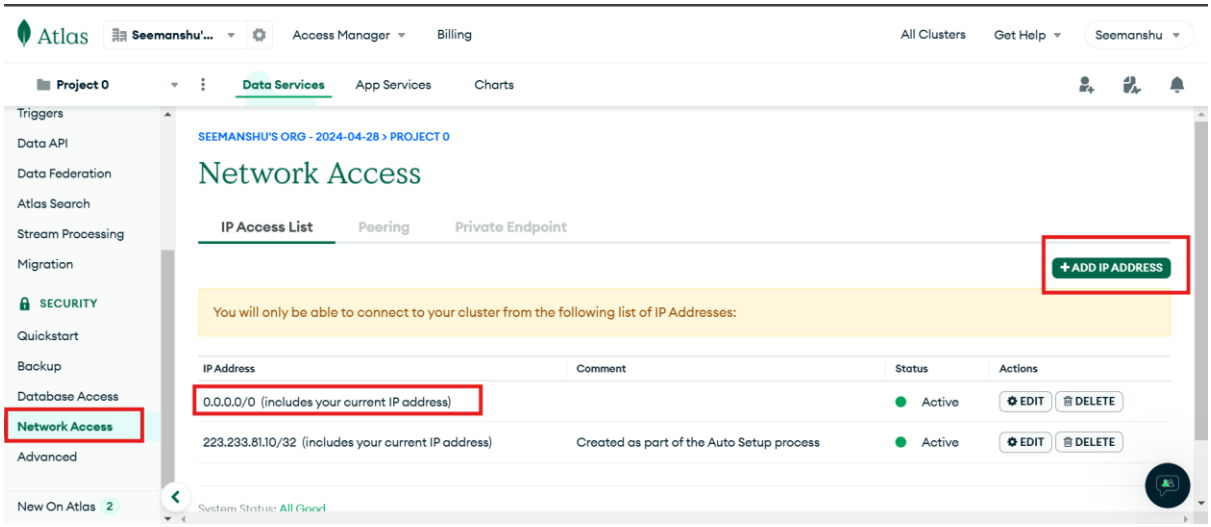
- Observe in the connection string we have ***<password>***. If this is the first time we are generating the connection string then the password will be auto populated. In case it is not auto populated then we can generate the password by following the below steps.

- Go to **Database Access** and under **Password** click **Edit Password**



- Set the new password and then replace <password> in connection string with the new password. The resultant connection string thus obtained will be used while establishing connectivity with MongoDB Atlas.

- **Step 4: Add the whitelisted IP:**
  - Next we will need to specify the list of IP address that will hit our MongoDB cluster in order to access the database and collections within it.
  - In case you are not sure about the system IP from where you wish to hit the MongoDB cluster then we can add a **whitelisted IP address: 0.0.0.0/0**
  - Go to the **Network Access** section and click the **ADD IP ADDRESS** button. In the following screen add the whitelisted IP and proceed:

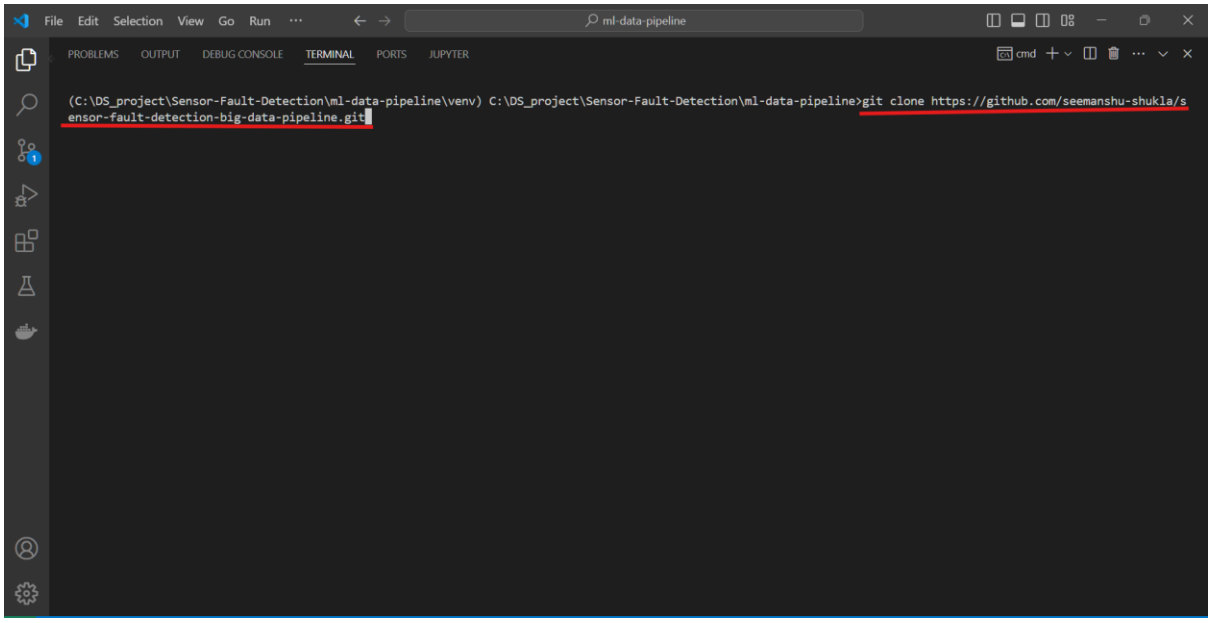


  - **Please note**: Reason why we have opted for whitelisted IP over a specific system IP is because whitelisted IP will give us more flexibility by allowing us to hit our MongoDB cluster globally from any system.

➔ **Initiating the Streaming Big data pipeline by triggering the Producer and Consumer scripts:**

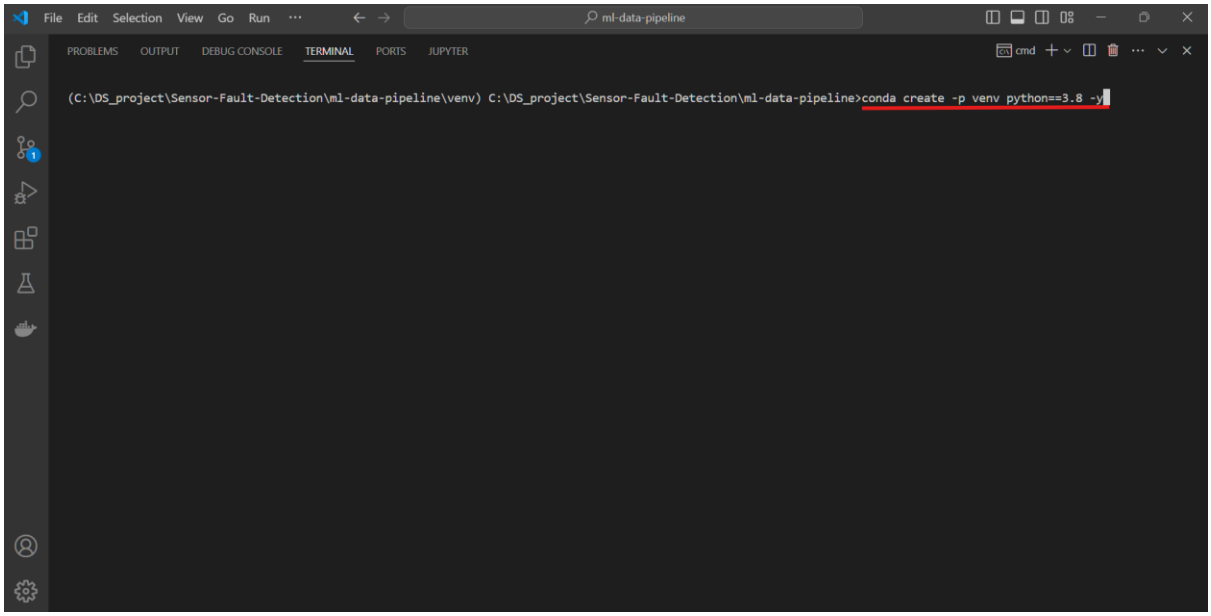
- **Clone the GitHub repository with source code:**

```
git clone https://github.com/seemanshu-shukla/sensor-fault-detection-big-data-pipeline.git
```



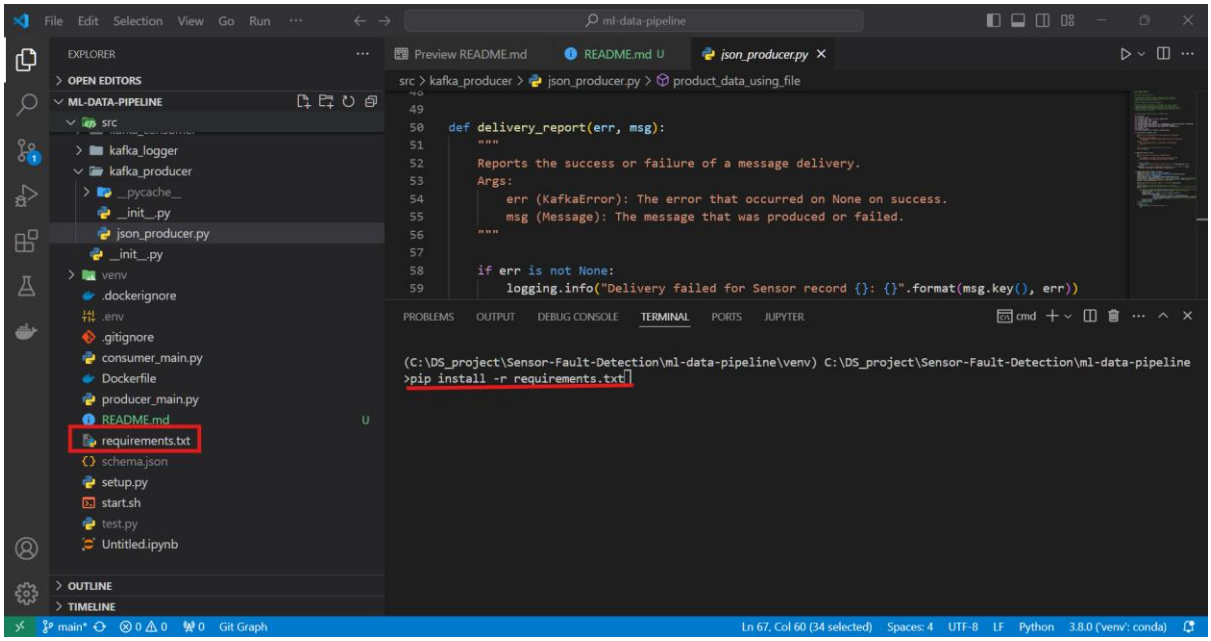
- **<u>Create the virtual environment and install the pythonic dependencies inside it:</u>**

```
conda create -p venv python==3.8 -y
```



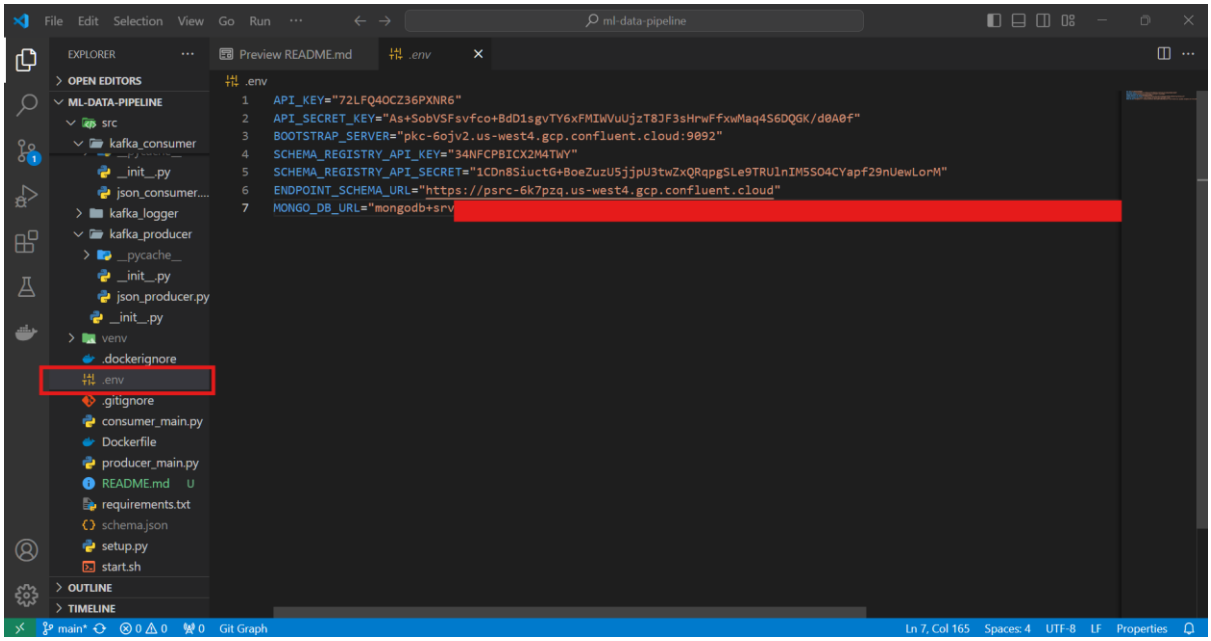
- Use the following command to install the dependencies:

```
pip install -r requirements.txt
```



- **<u>Setting the environment variables</u>:**

  Create a *.env file* in the *root directory* and define your credentials which can be obtained using the steps discussed earlier.

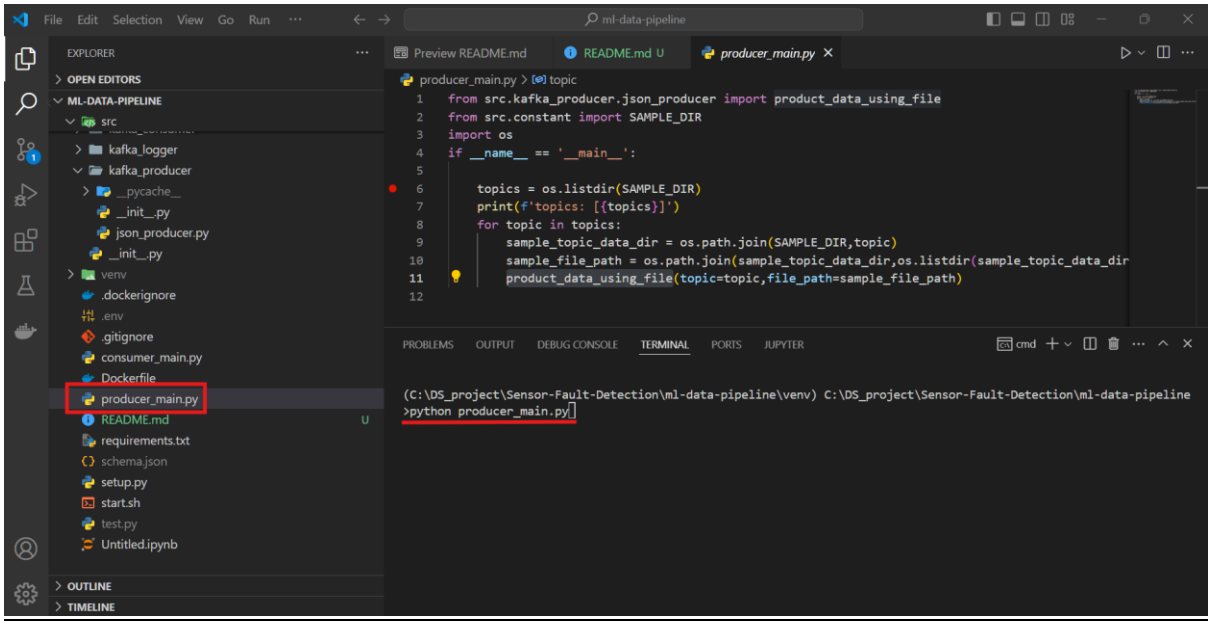


<u>Note</u>: Please don't copy my credentials as they will be no longer in use by the time this community document is live.

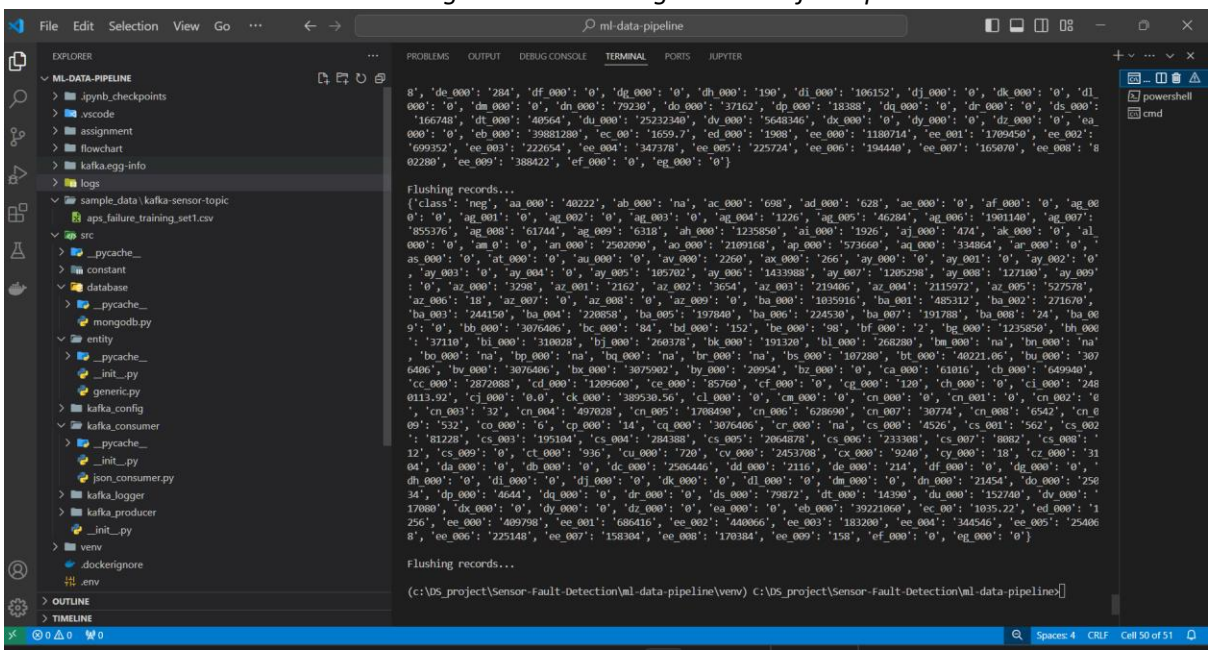
- **<u>Trigger the Producer script:</u>**

  - Vscode:

Trigger producer script

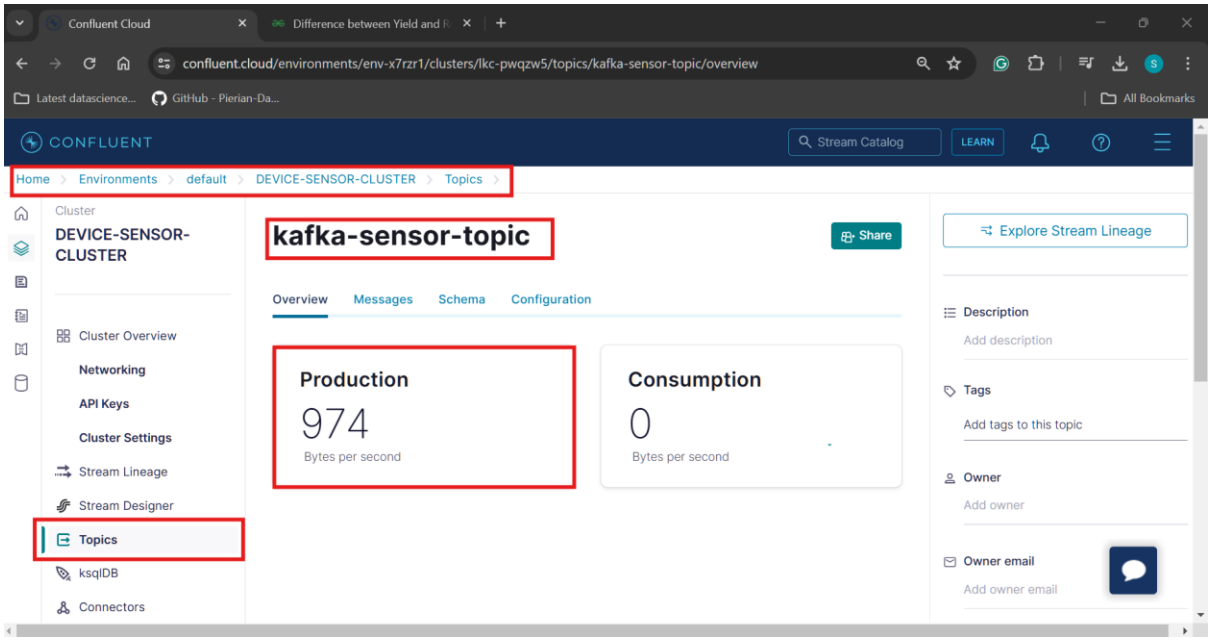


Real time terminal logs while Publishing data to Kafka topics

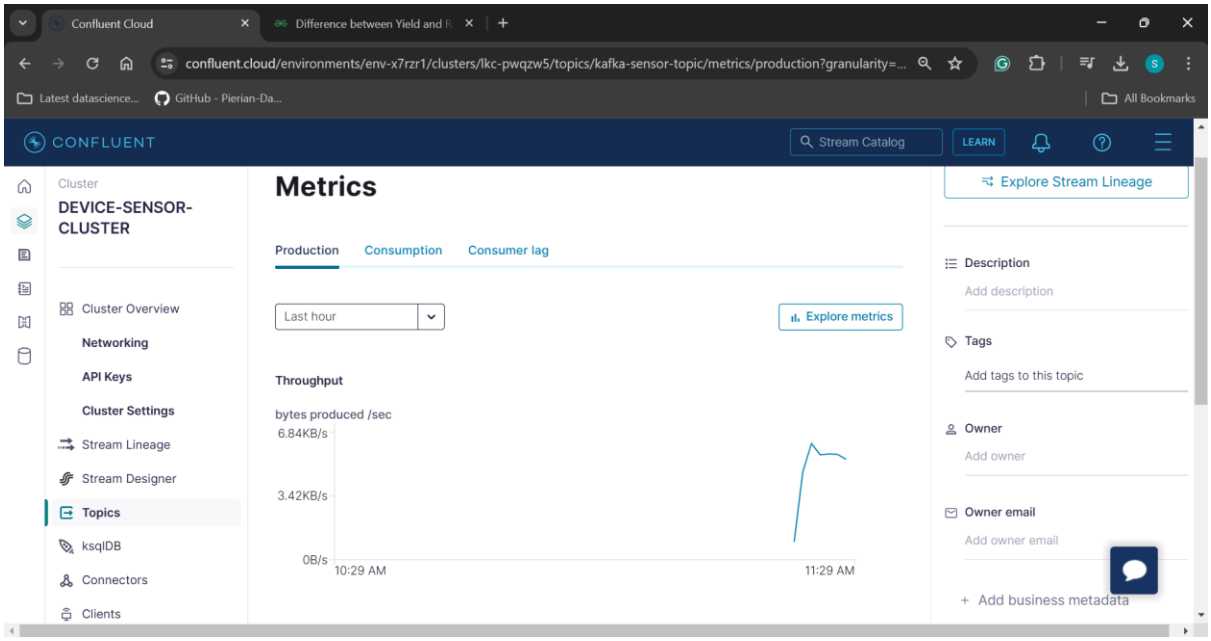


- Visualizing the Confluent Kafka dashboard during publication of data to the Kafka topics by the consumer script:

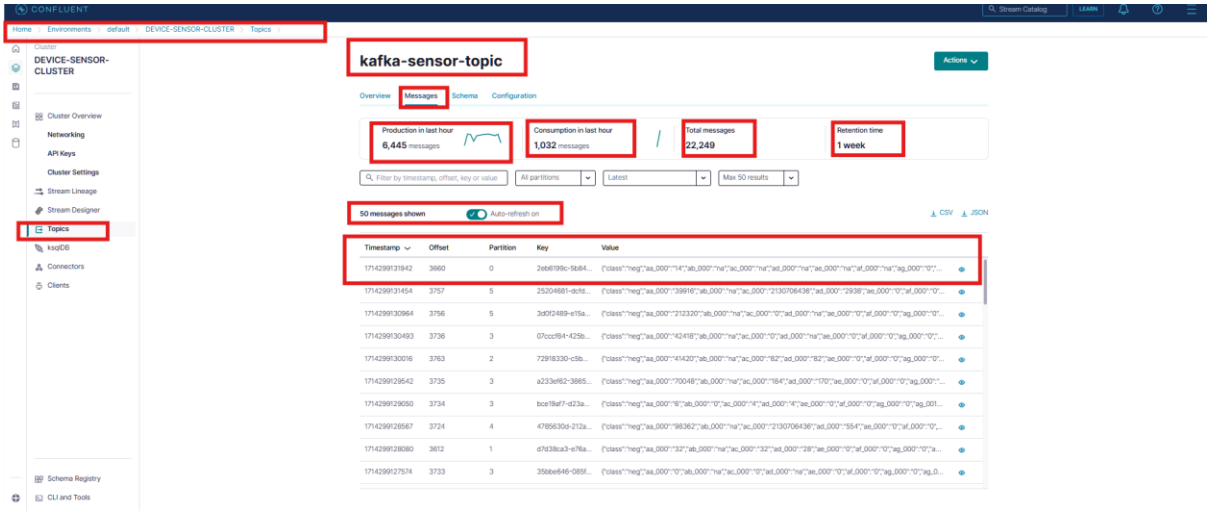
Current rate/speed at which data is getting published to kafka topics = ***964 bytes/sec***

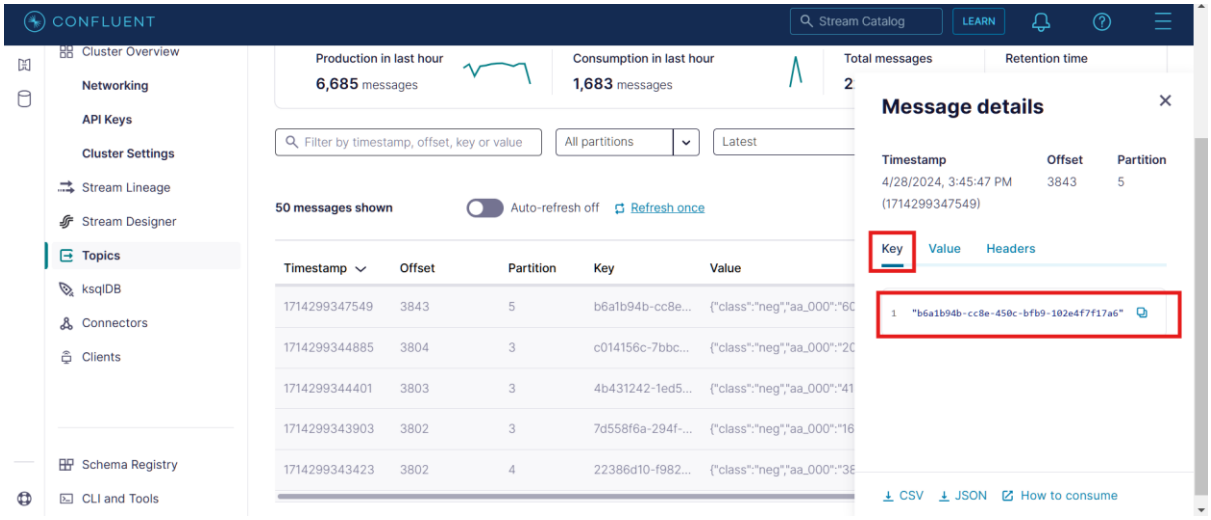


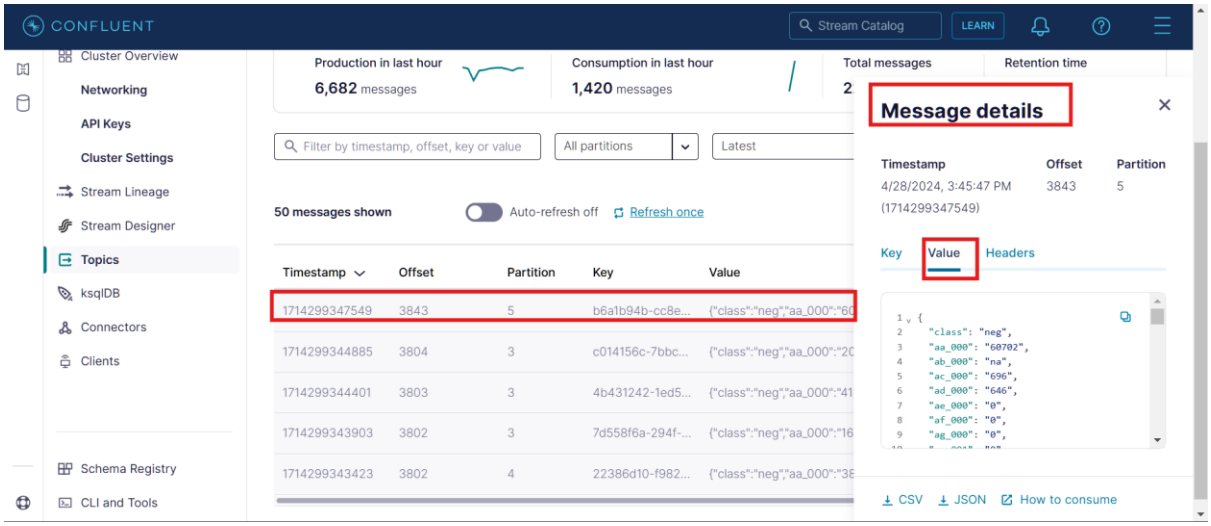
Graphical representation of rate with which data is getting published to Kafka topics across different instance of time

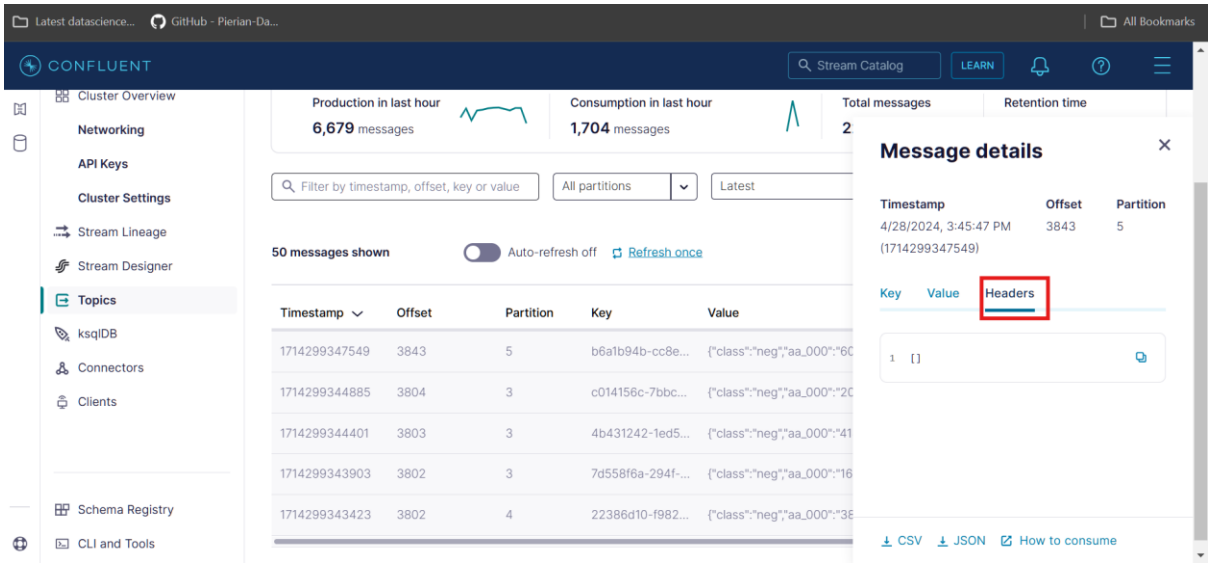


Checking the messages/data that is getting published:


kafka-sensor-topic
DEVICE-SENSOR-CLUSTER
Production in last hour
6,445 messages
Consumption in last hour
1,032 messages
Total messages
22,249
Retention time
1 week
50 messages shown
Auto-refresh on

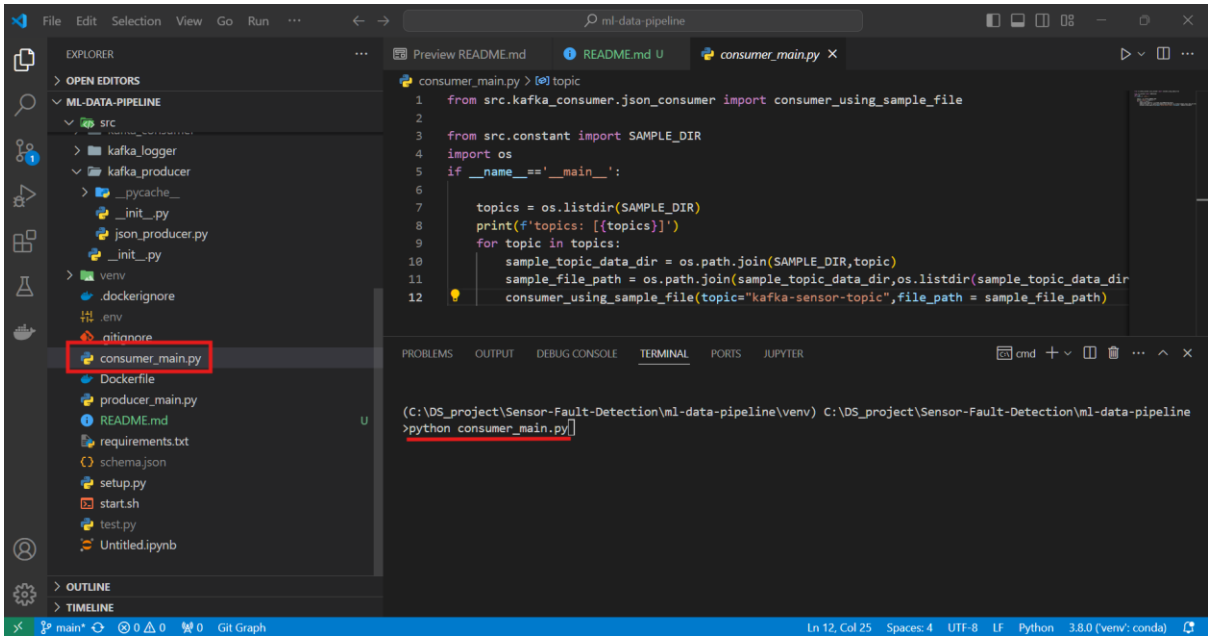

CONFLUENT
Stream Catalog
LEARN
Production in last hour
6,685 messages
Consumption in last hour
1,683 messages
Total messages
Retention time
Message details
Timestamp
4/28/2024, 3:45:47 PM
(1714299347549)
Offset
3843
Partition
5
Key
Value
Headers
"b6a1b94b-cc8e-450c-bfb9-102e4f7f17a6"
50 messages shown
Auto-refresh off
Refresh once
Filter by timestamp, offset, key or value
All partitions
Latest
Timestamp
Offset
Partition
Key
Value
1714299347549
3843
5
1714299344885
3804
3
1714299344401
3803
3
1714299343903
3802
3
1714299343423
3802
4
Cluster Overview
Networking
API Keys
Cluster Settings
Stream Lineage
Stream Designer
Topics
ksqlDB
Connectors
Clients
Schema Registry
CLI and Tools
CSV
JSON
How to consume


CONFLUENT
Stream Catalog
LEARN
Production in last hour
6,682 messages
Consumption in last hour
1,420 messages
Total messages
Retention time
Message details
Timestamp
4/28/2024, 3:45:47 PM
(1714299347549)
Offset
3843
Partition
5
Key
Value
Headers
{
"class": "neg",
"aa_000": "60702",
"ab_000": "na",
"ac_000": "696",
"ad_000": "646",
"ae_000": "0",
"af_000": "0",
"ag_000": "0",
50 messages shown
Auto-refresh off
Refresh once
Filter by timestamp, offset, key or value
All partitions
Latest
Timestamp
Offset
Partition
Key
Value
1714299347549
3843
5
1714299344885
3804
3
1714299344401
3803
3
1714299343903
3802
3
1714299343423
3802
4
Cluster Overview
Networking
API Keys
Cluster Settings
Stream Lineage
Stream Designer
Topics
ksqlDB
Connectors
Clients
Schema Registry
CLI and Tools
CSV
JSON
How to consume

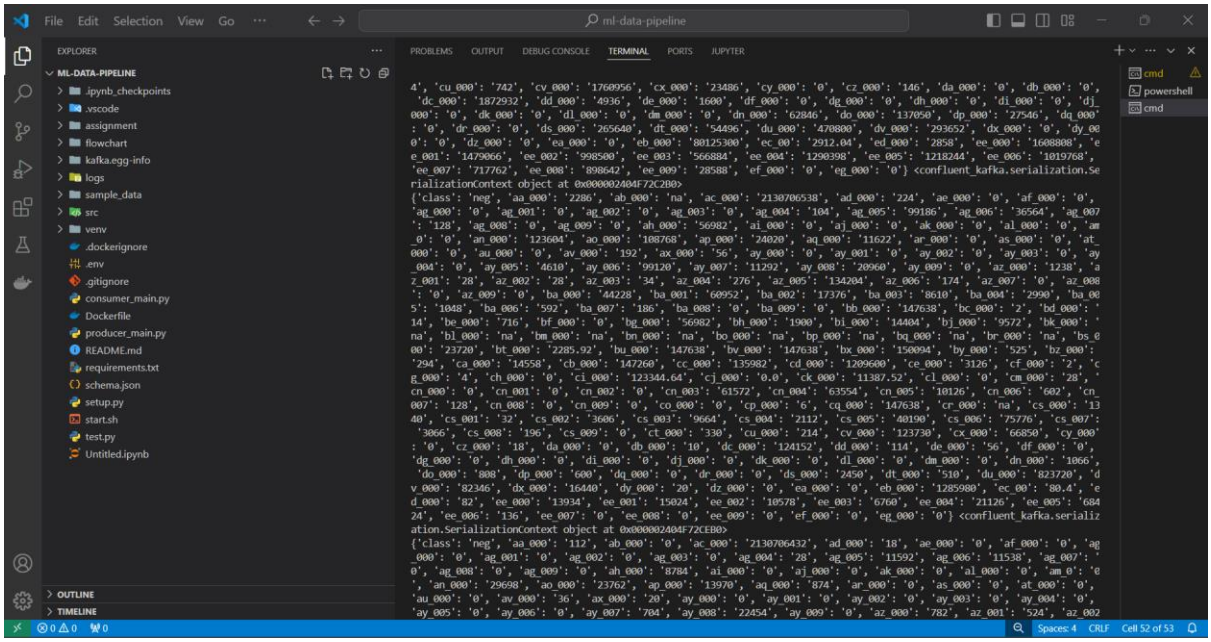


- **<u>Trigger the Consumer script:</u>**

  - Vscode:
    Triggering the consumer script



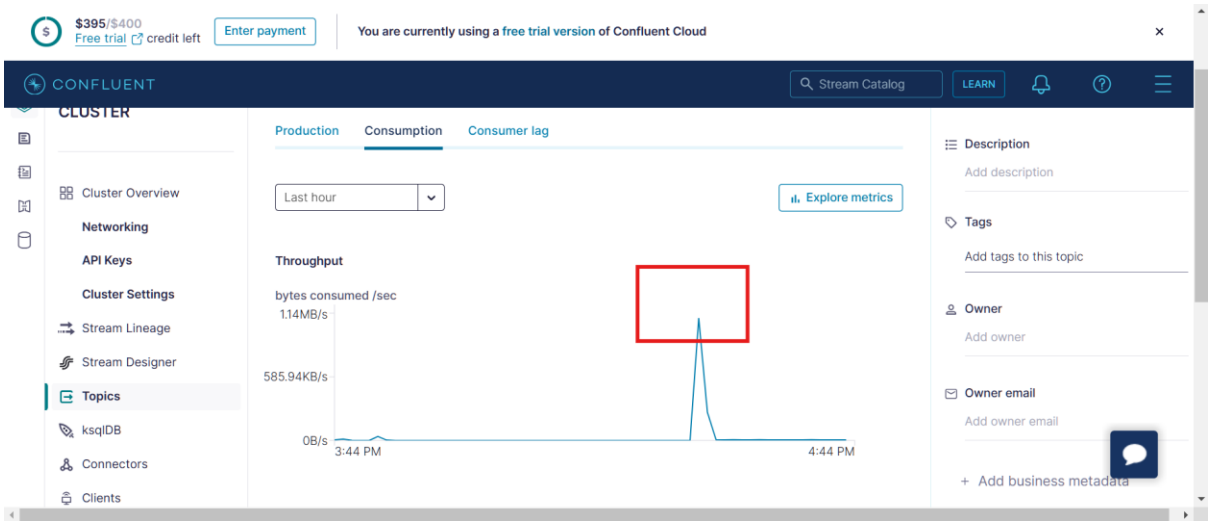
Real time terminal logs while Consuming data from Kafka topics



Note: Ideally Producer and Consumer scripts are part of different projects. Producer script is meant to be deployed over edge devices from where it can continuously stream the data to the Kafka topics. Whereas, consumer script is scheduled to run on a different environment. But since here we are not having any edge device, where we can separately place our Producer script. Therefore, here both the scripts are running under same project but on different terminal. We can see in above snip we have 2 cmd terminals. The first cmd terminal is running the producer script whereas the second cmd terminal is running the consumer script where it is consuming data from Kafka topics and via our system sending the consumed data to the MongoDB database for the persistence purpose.

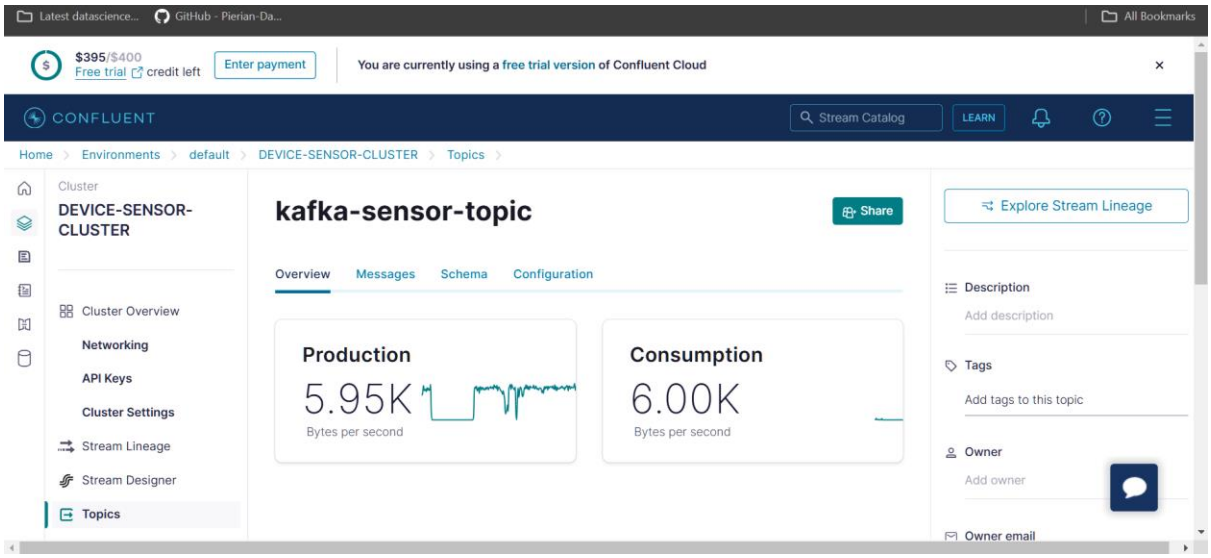
- Visualizing the Confluent Kafka dashboard during consumption of data from the Kafka topics by the consumer script:

  We can see in the below Throughput chart that as soon as we triggered the consumer script, the consumption rate peaked suddenly and went up to the rate of ***1.14 MB/sec*** *which is way higher than the rate of production.*

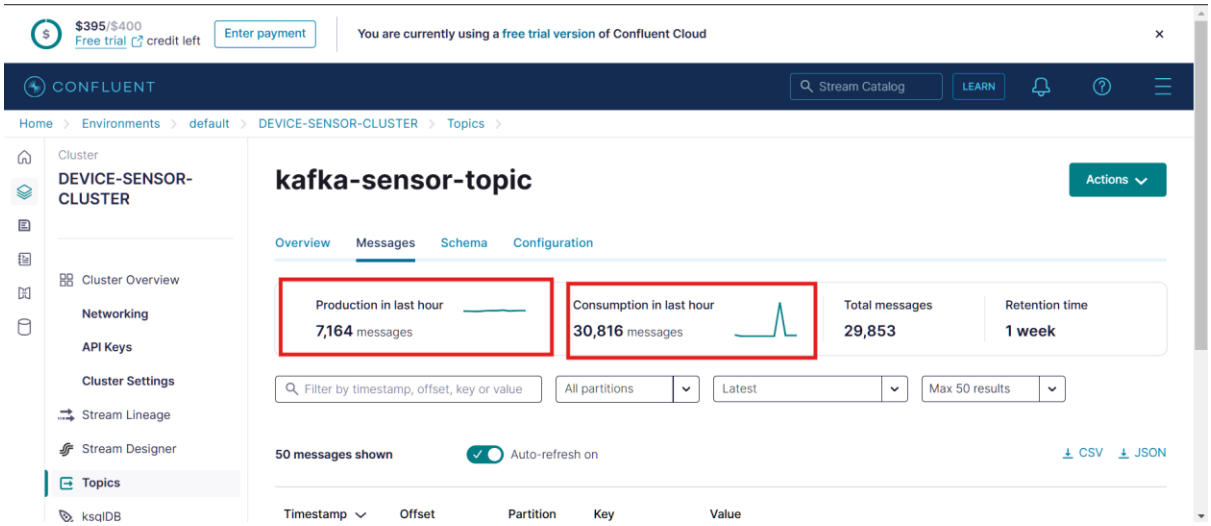


Note: The rate of consumption is more than the rate of production. This is the reason why in the above snip we can see that as soon as consumer script is triggered the consumption rate ran at the high speed 1.14 MB/sec since data was already available for consumption.

After some time Production and Consumption rate were in sync. Reason behind this is that inbound speed(rate of production) is not enough to accommodate the outbound speed(rate of consumption) therefore, consumer script is not able to consumer messages at the top speed because of we can see that it is having a sync with Production rate.

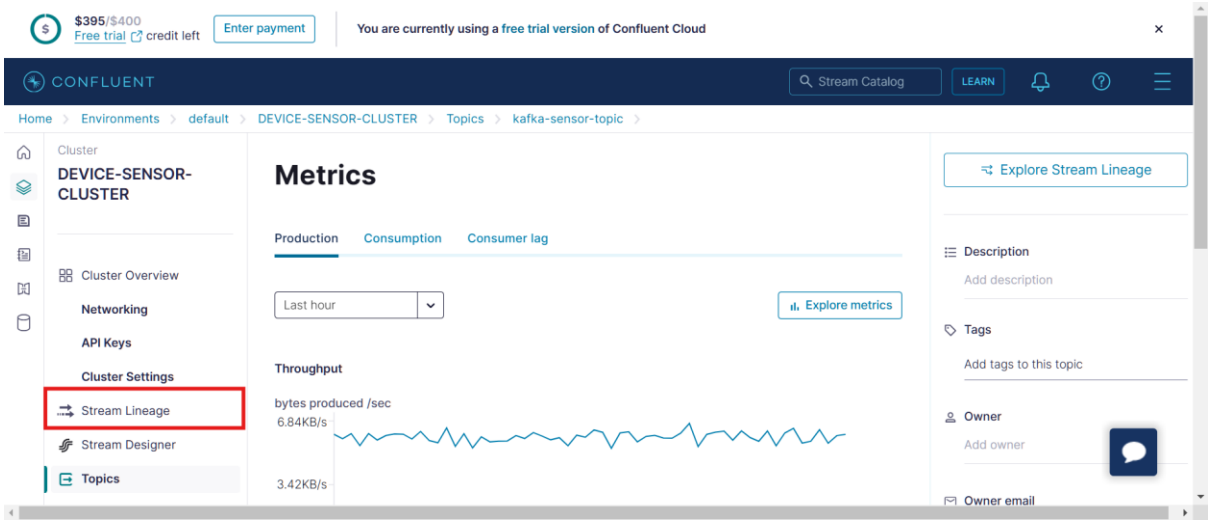


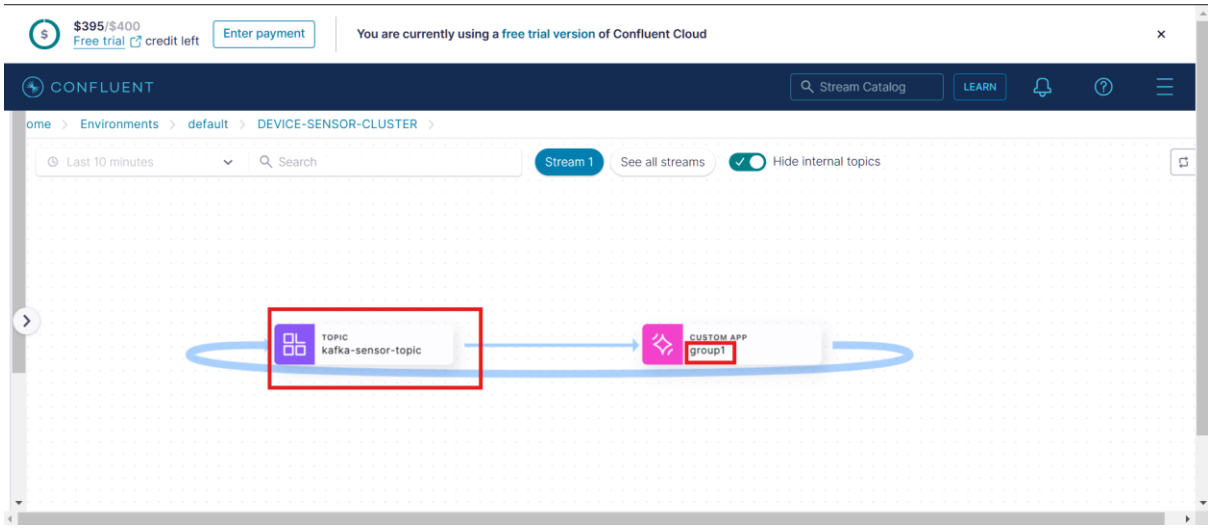
Based on messages processed by both Production and Consumption components in the last hour interval, we can conclude that consumption rate is much higher than the production rate.



Note: We have triggered Producer script first and after some time we have triggered the Consumer script allowing consumer script to work at top speed by consuming already produced data. Later, both consumption and production rate get in sync since producer script was not able to produce the messages with the rate with which consumer script was consuming data.

*Stream Lineage allows us to visualize the different data streams. In below snip we can see that data is moving from Kafka topic to the consumer named as "***group1"** which we have programmatically defined while designing the streaming big data pipeline.

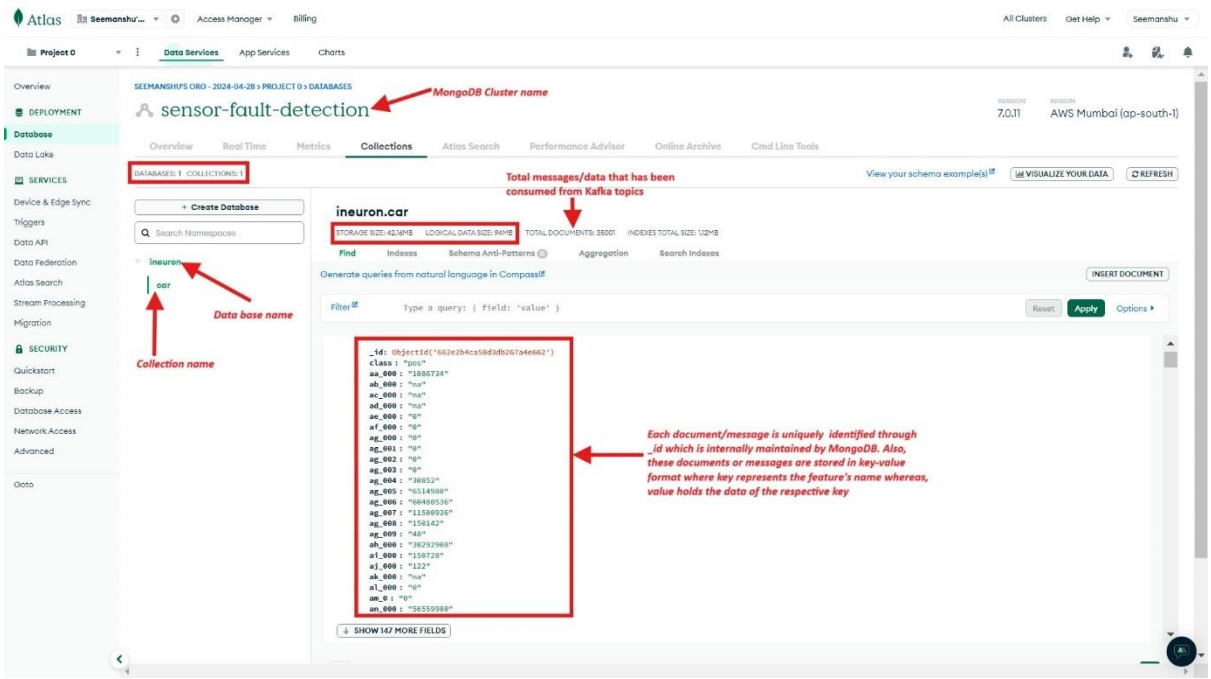




Note: In the above Production Throughput chart we can observe that the production rate is almost constant across time.

- Checking the MongoDB dashboard:

  Consumer script has been designed in a way that the consumed data/messages from Kafka topics via our machine(where consumer script is running) is pushed to MongoDB for persistence(permanent storage) purpose.



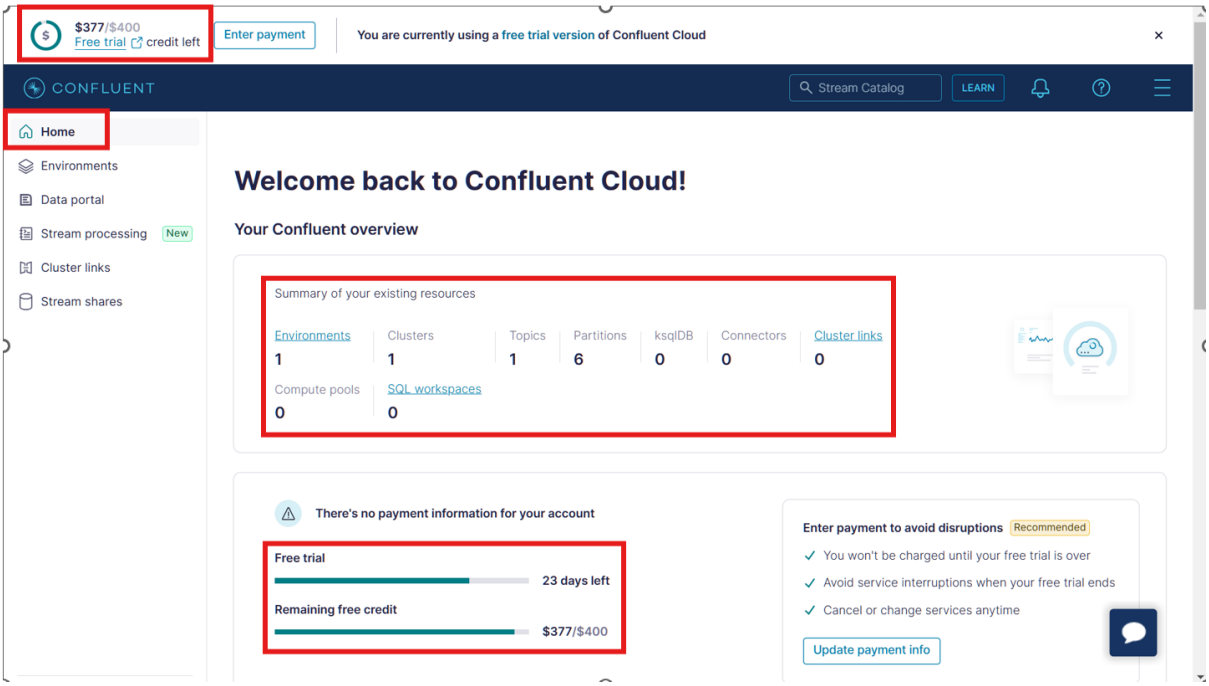
Note: We have designed the consumer script in such a way that at a time **5000 messages** are getting pooled at our local(where consumer script is running) and from there we are using insert_many MongoDB operation to push all these 5000 messages to MongoDB. Please note that pooling 5000 messages at a time from the Kafka topics is just a hyperparameter and can be

decided based on experimentation. An appropriate value drastically improve the consumption rate followed by pushing consumed messages to MongoDB.

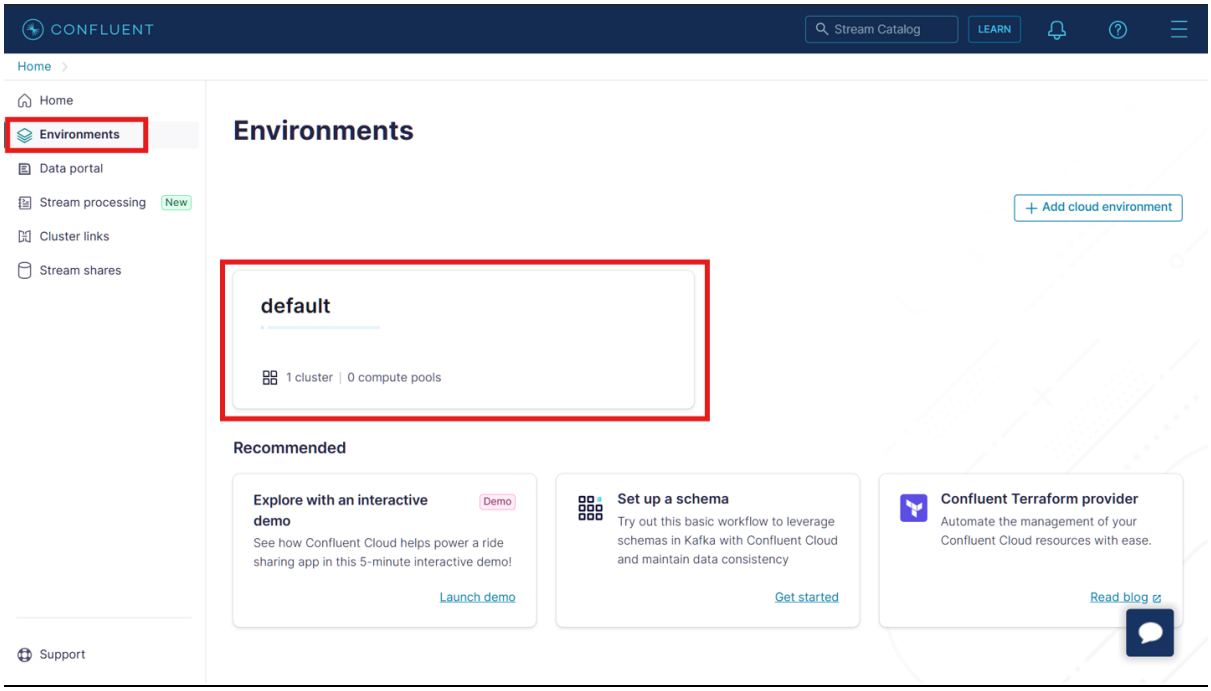
➔ **Exploring different sections of Confluent Kafka before deleting the associated resources:**

- Under this section we will be exploring the Confluent Kafka one last time before deleting the associated resources post data pipeline completion in order to save the costs.

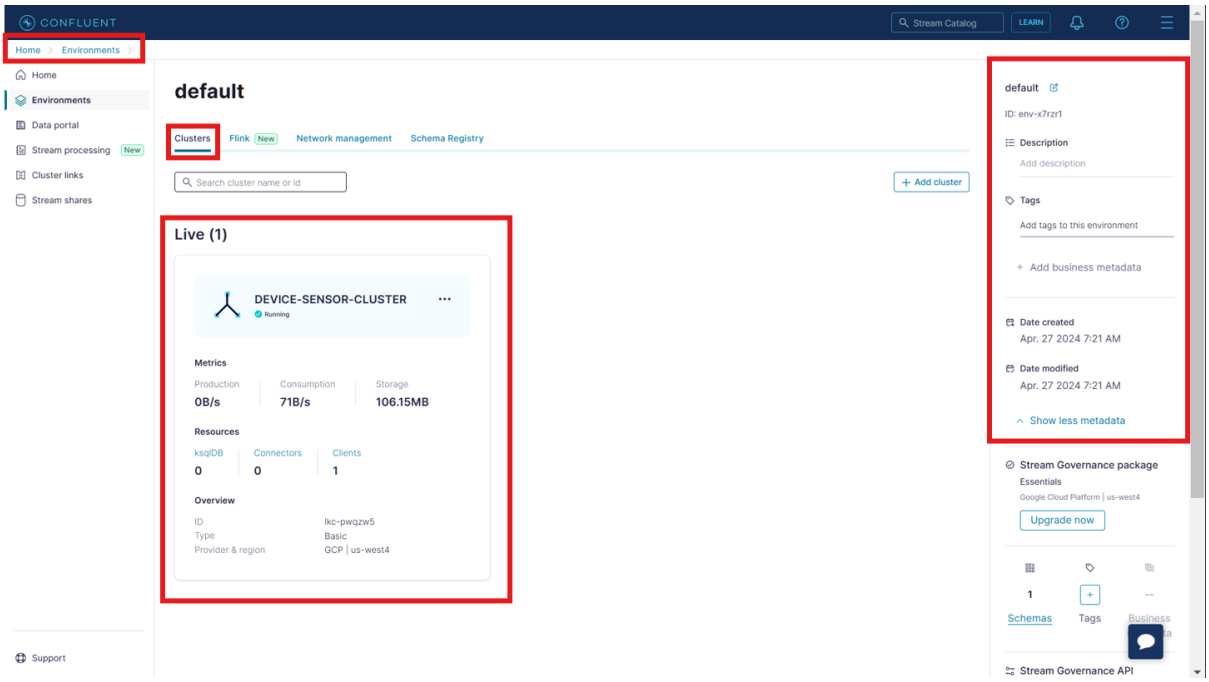
- **Home:**
  - In the homepage we can see the summary of our existing resources along with remaining free credits:



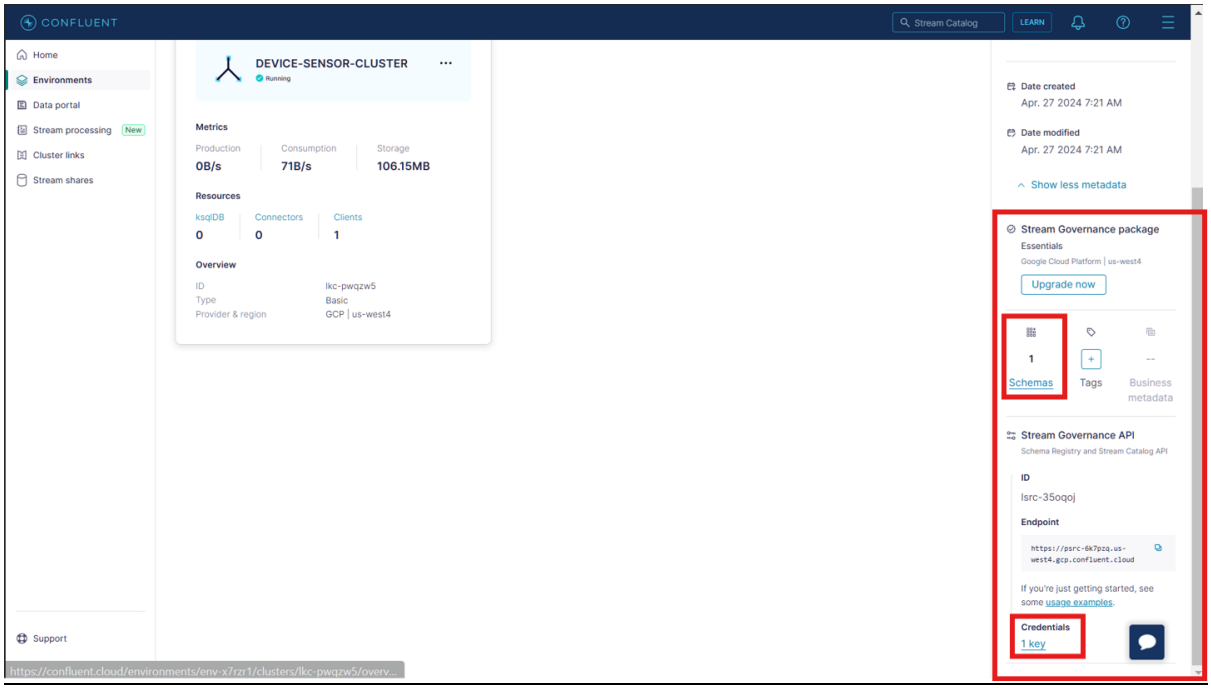
- **Environments:**
  - In the Environment section we can see the overview of the environment defined earlier while setting up the Kafka cluster. In our case we were using the **default** environment within which we built 1 cluster:



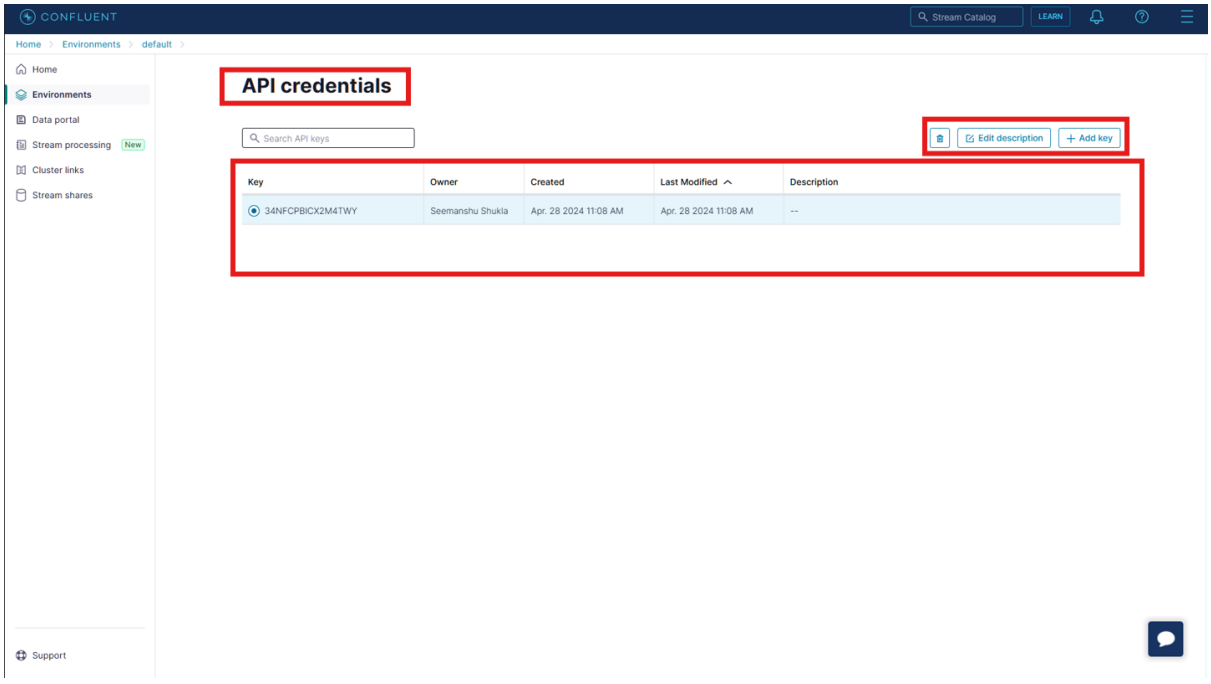
- **<u>Home > Environments >:</u>**
  Let's go inside default environment and check out more detailed overview of our cluster named as **DEVICE-SENSOR-CLUSTER:**



The key highlighted in the below snip represents the schema registry key that we created earlier



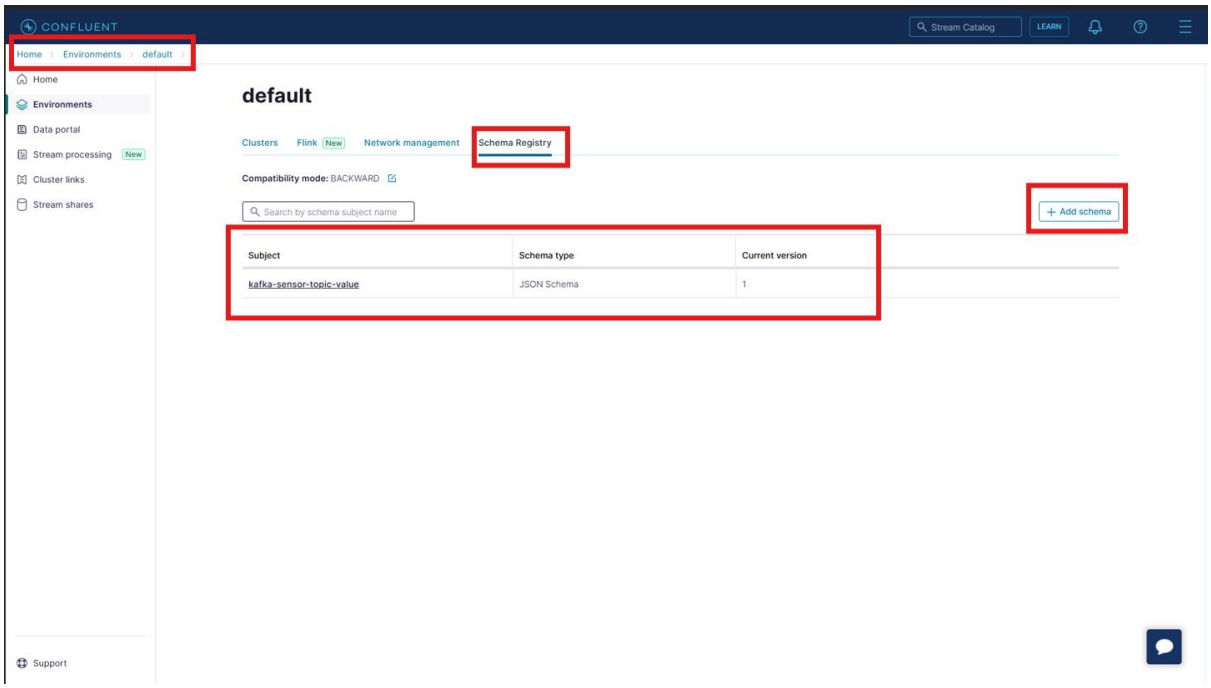
If we click on **1Key** highlighted in the above snip then it will redirect us to the page where we can manage the API credentials related to Schema Registry:



- **<u>Home > Environments > default >:</u>**

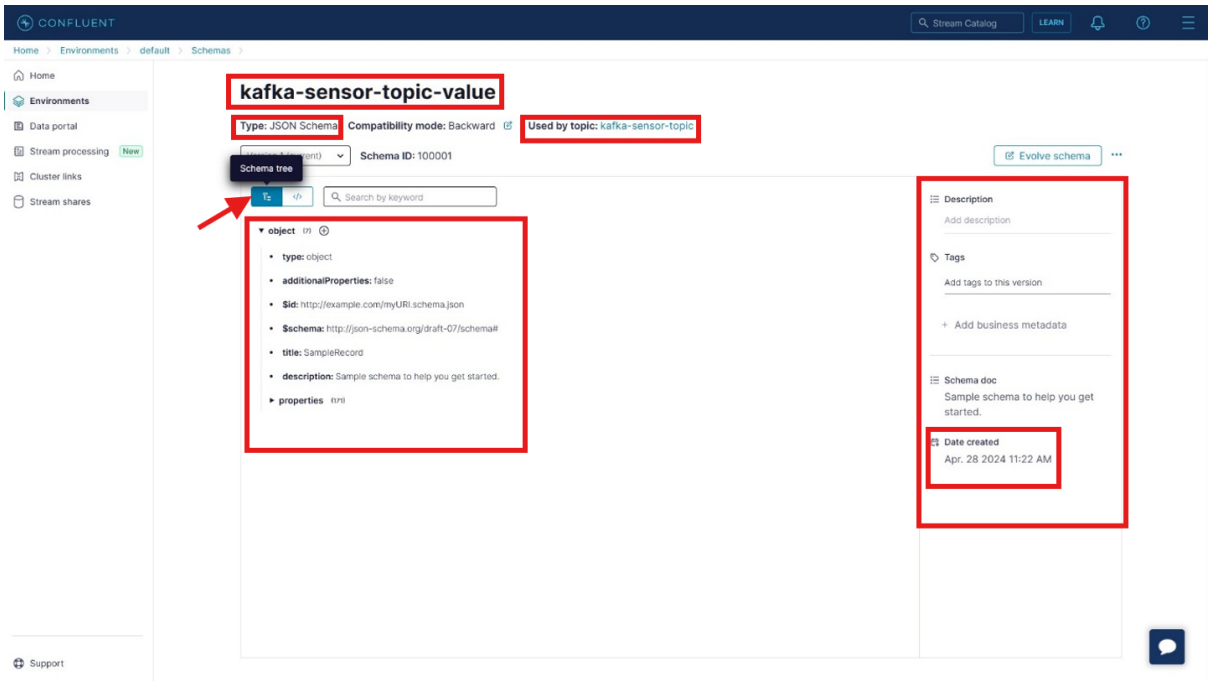
  Under **Schema Registry** section we can manage the schema's which will be utilized during Publication of the data to topics or Consumption of the data from the topics. In below snip

kafka-sensor-topic-value is the name of schema that we have registered earlier by running our data pipeline in the development environment:

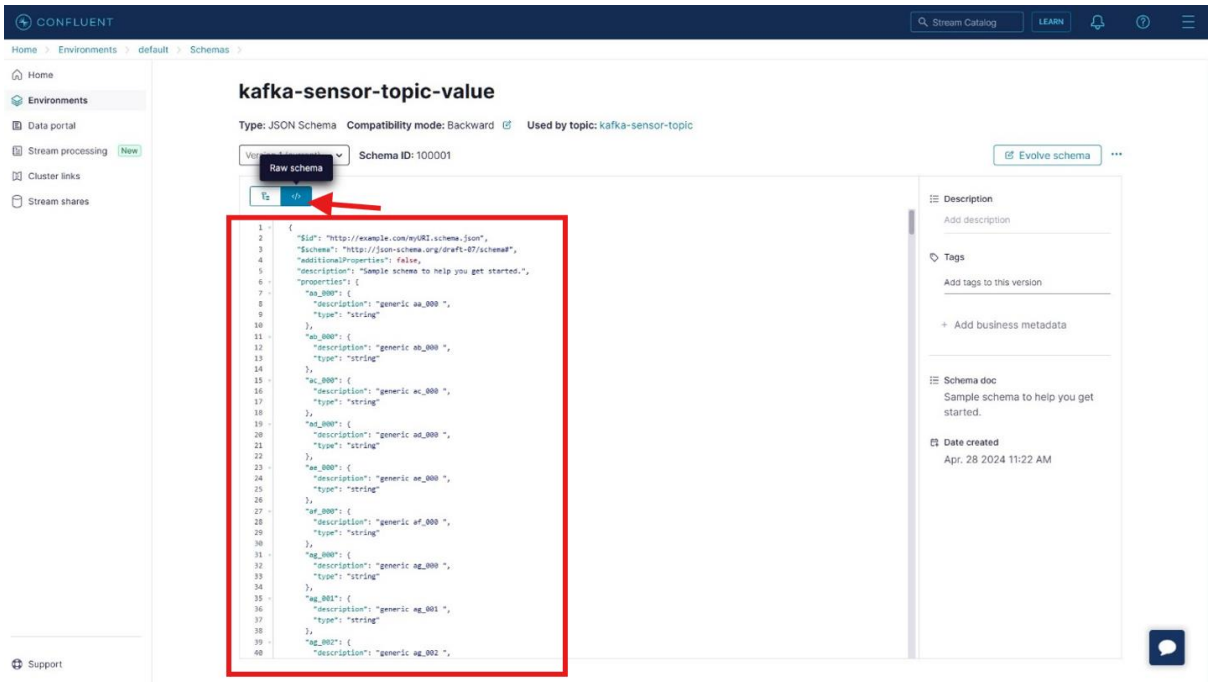


- **<u>Home > Environments > default > Schemas >:</u>**

    - Below snip depicts the **Schema tree** view which is one way to visualize our schema named as **kafka-sensor-topic-value**

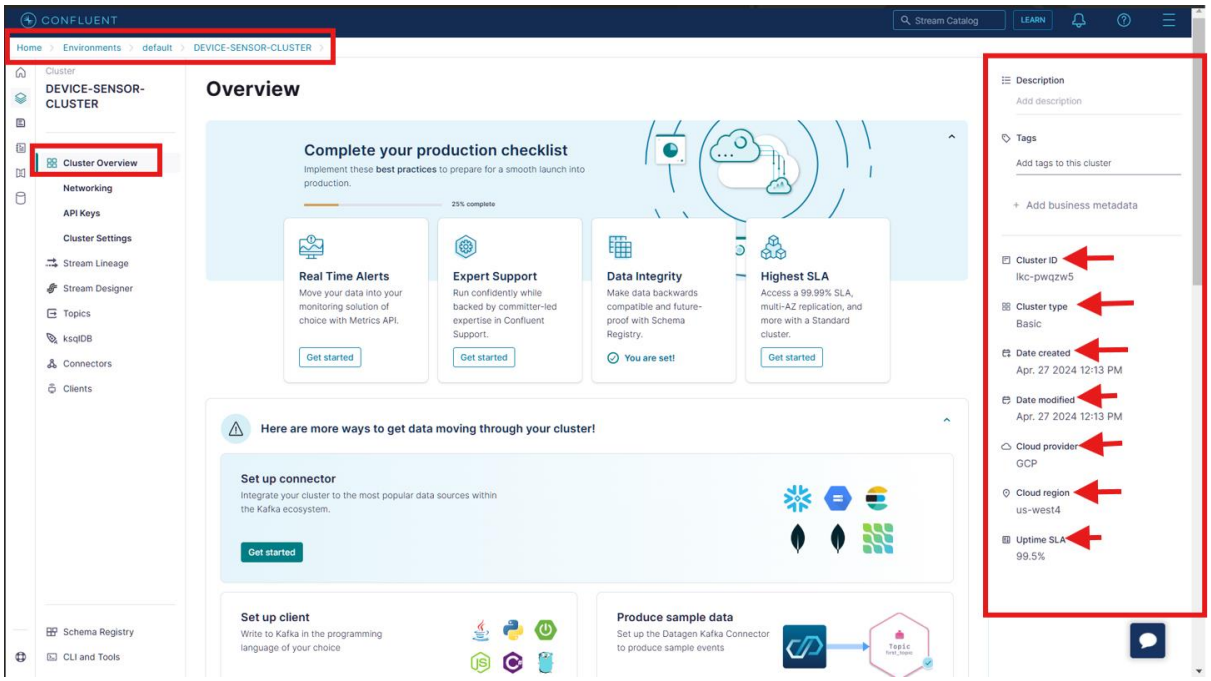


- Below snip depicts the **Raw schema** view which is another way to visualize our registered schema

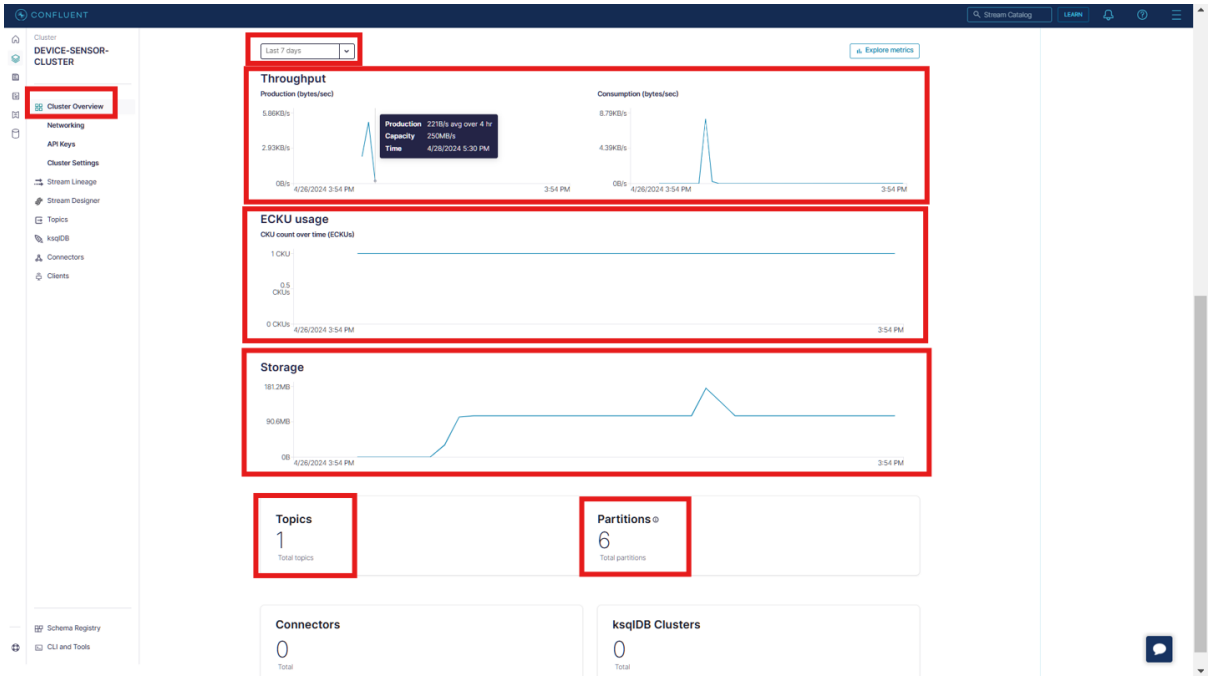


- **<u>Home > Environments > default > DEVICE-SENSOR-CLUSTER > :</u>**

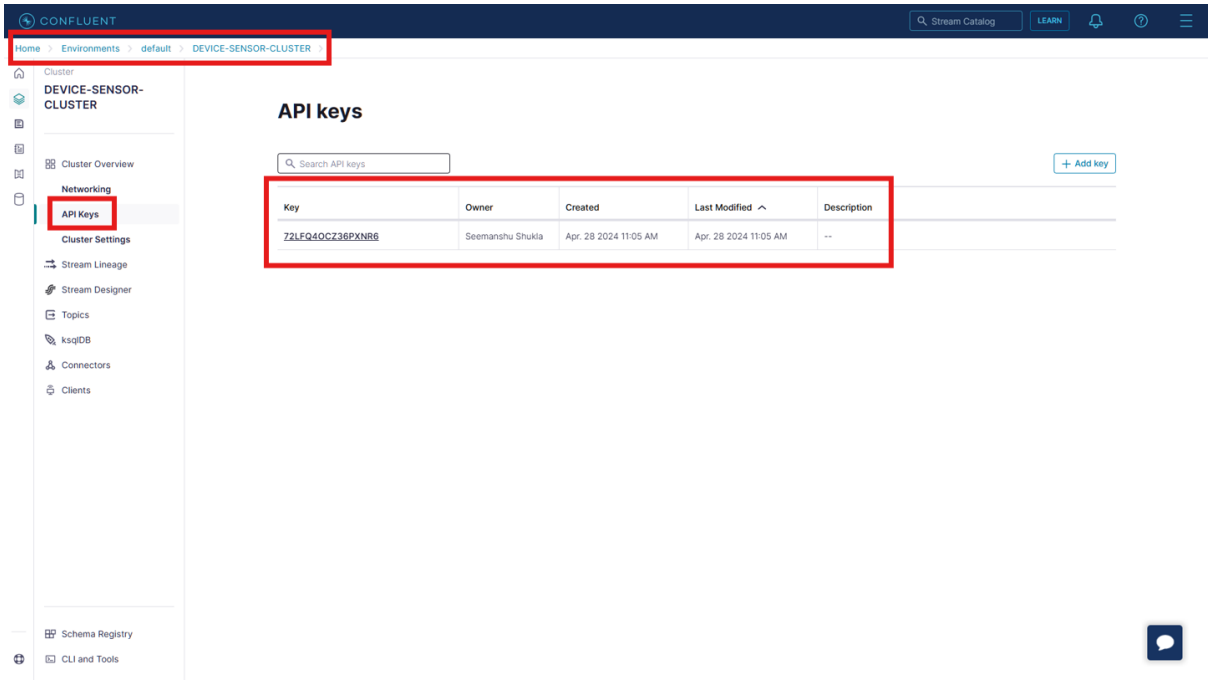
- Below snip gives us the complete overview our cluster named as **DEVICE-SENSOR-CLUSTER.**



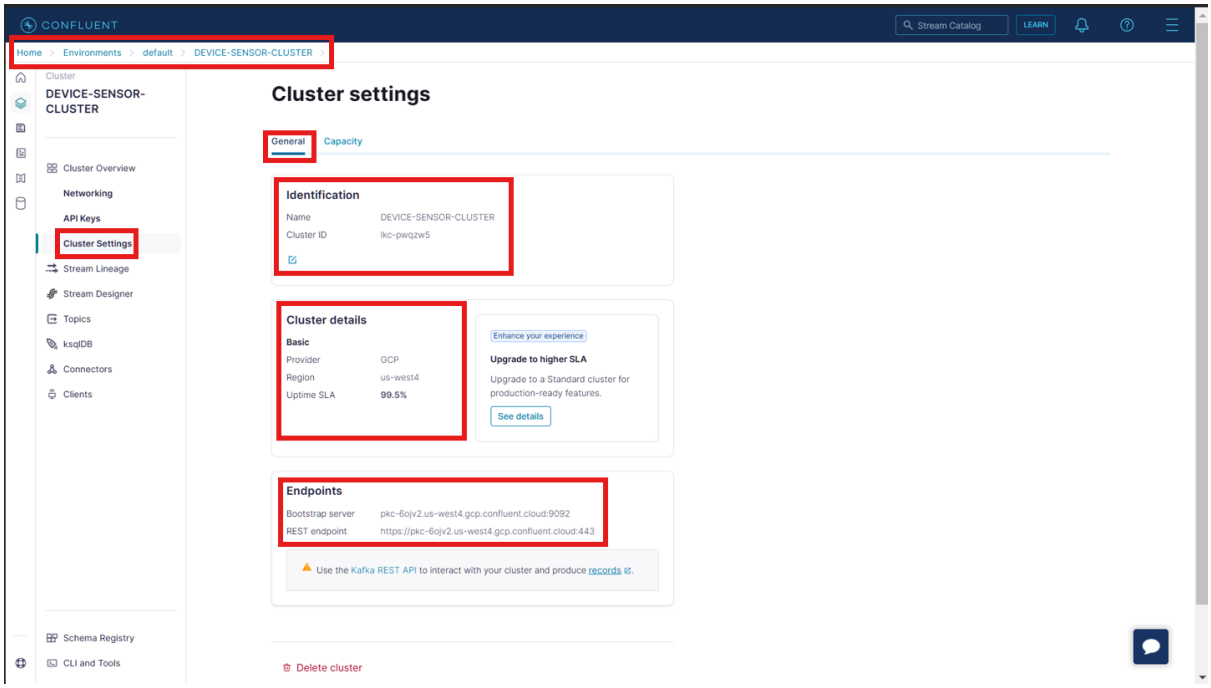
- Below snip depicts different metrices related to the cluster.



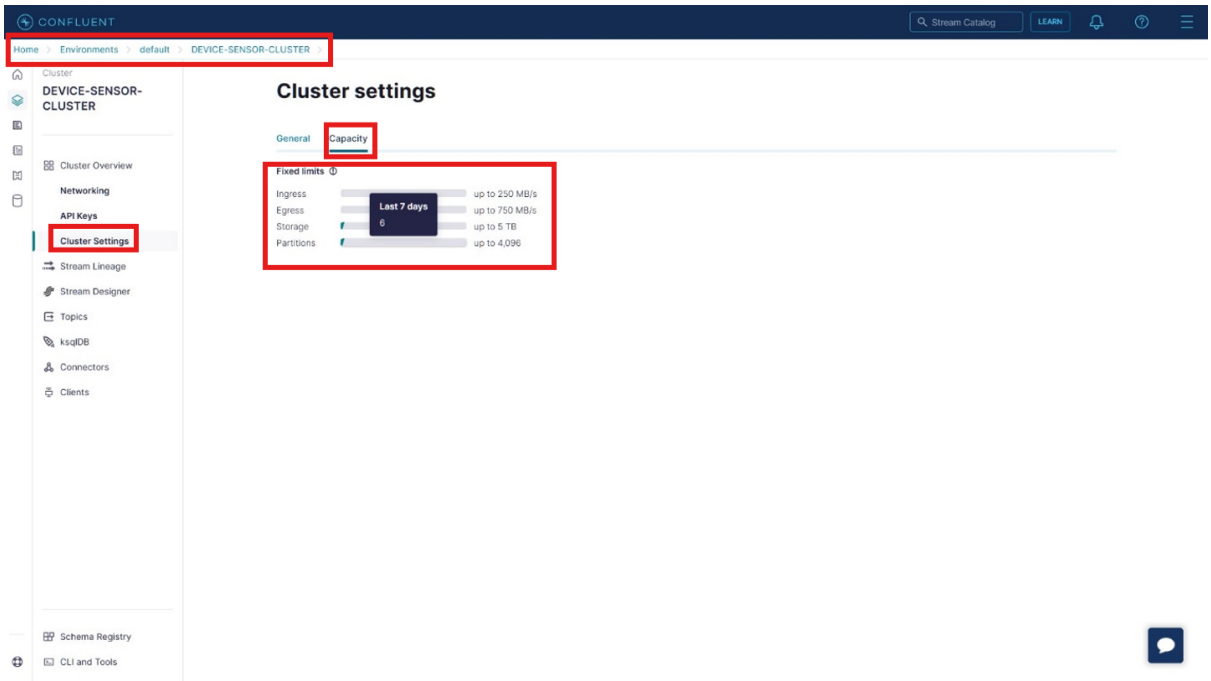
- Below snip depicts the **API Keys** section from where we can manage the API Keys needed for establishing the connectivity with the Kafka Cluster:



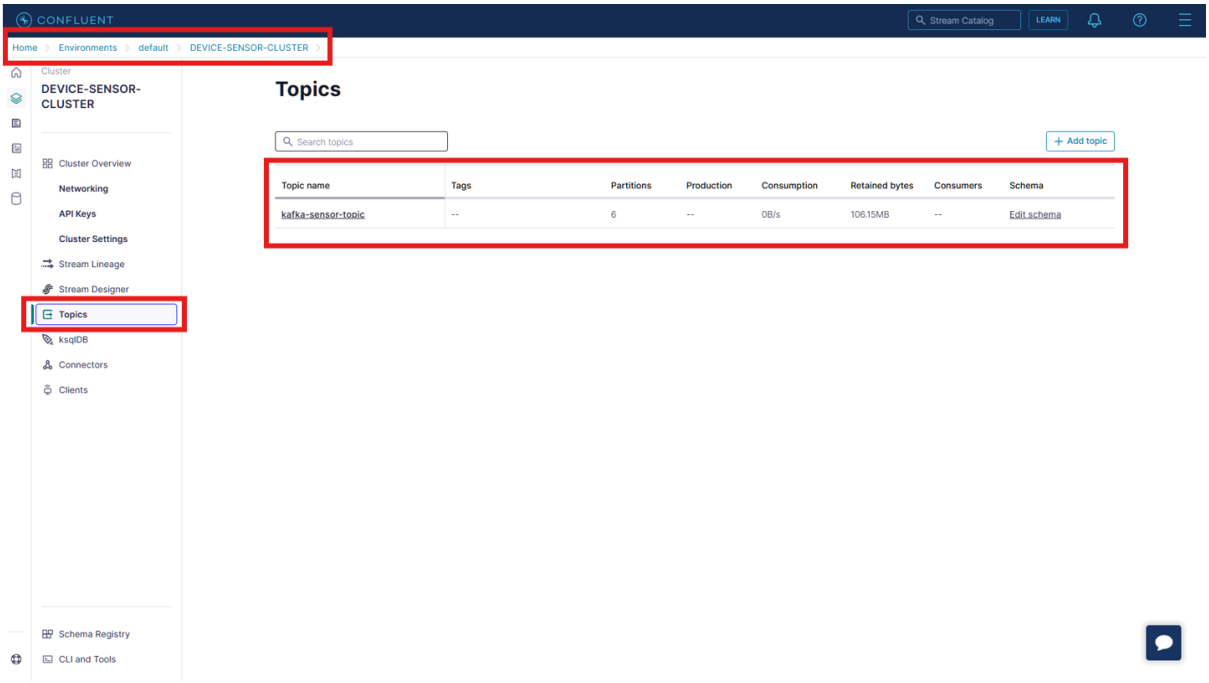
- Below snip depicts the **General** configurations of our Kafka Cluster:



- Below snip depicts **Capacity** related configuration of our Kafka Cluster:

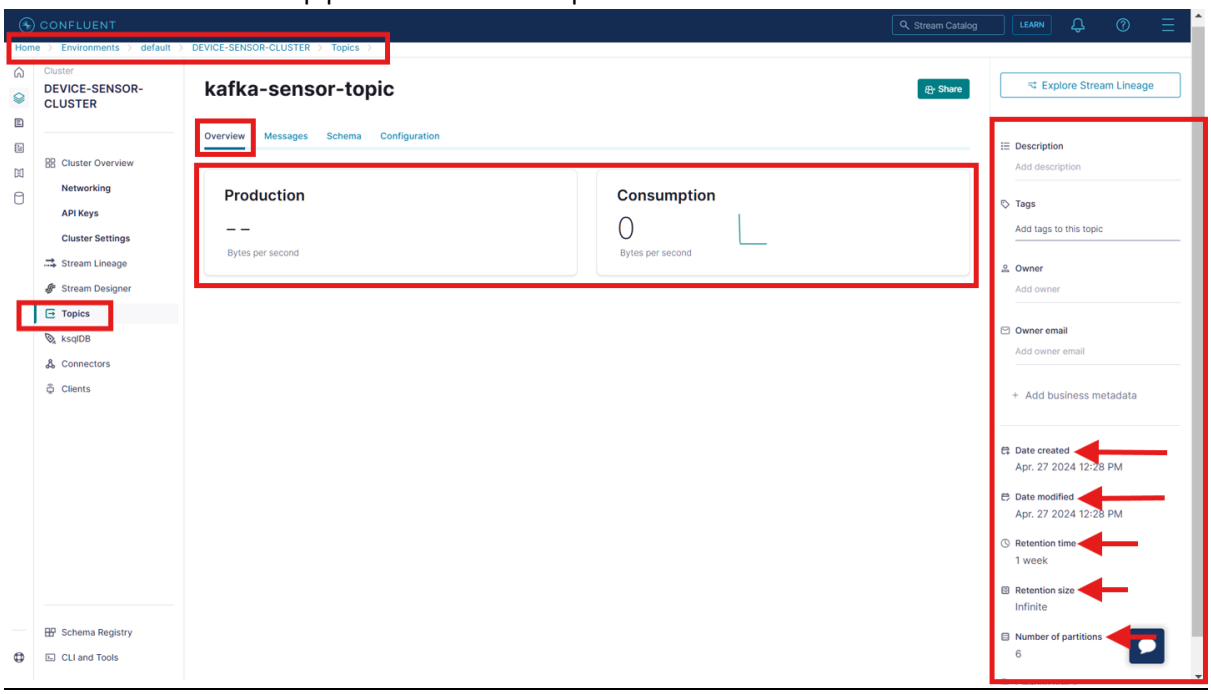


- Under **Topics** section we can manage the Kafka topics. In the below snip **kafka-sensor-topic** is the name of Topic that we defined while designing the data pipeline:



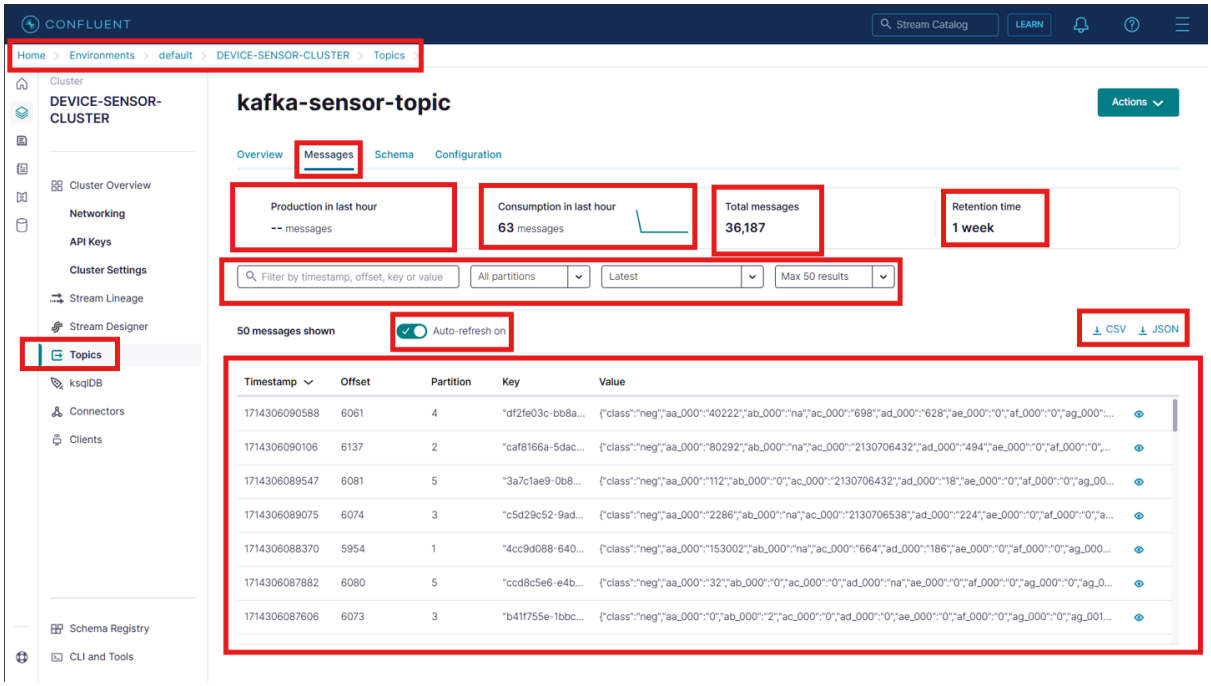
- **<u>Home > Environments > default > DEVICE-SENSOR-CLUSTER > Topics > :</u>**

  - This section is divided into 4 parts: **Overview, Messages, Schema, Configuration**.
  - **<u>Part 1st Overview:</u>**
    The speed with which data/messages is sent from Producer to Kafka topics(**Production**) and the speed with which data/messages is sent from Kafka topics to Consumers(**Consumption**) is measured in *Bytes per second*. In below snip this speed is 0 because at this time there is no data to Produce and Consume since our data pipeline has been completed.

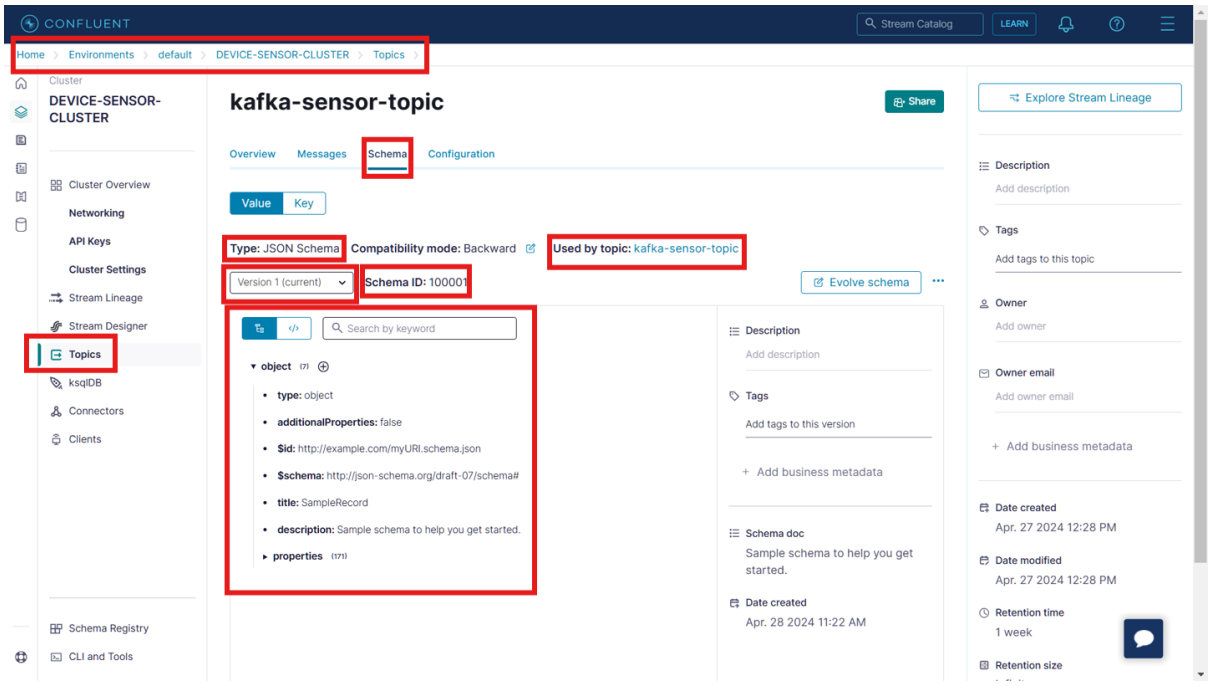


- **Part 2nd Messages:**

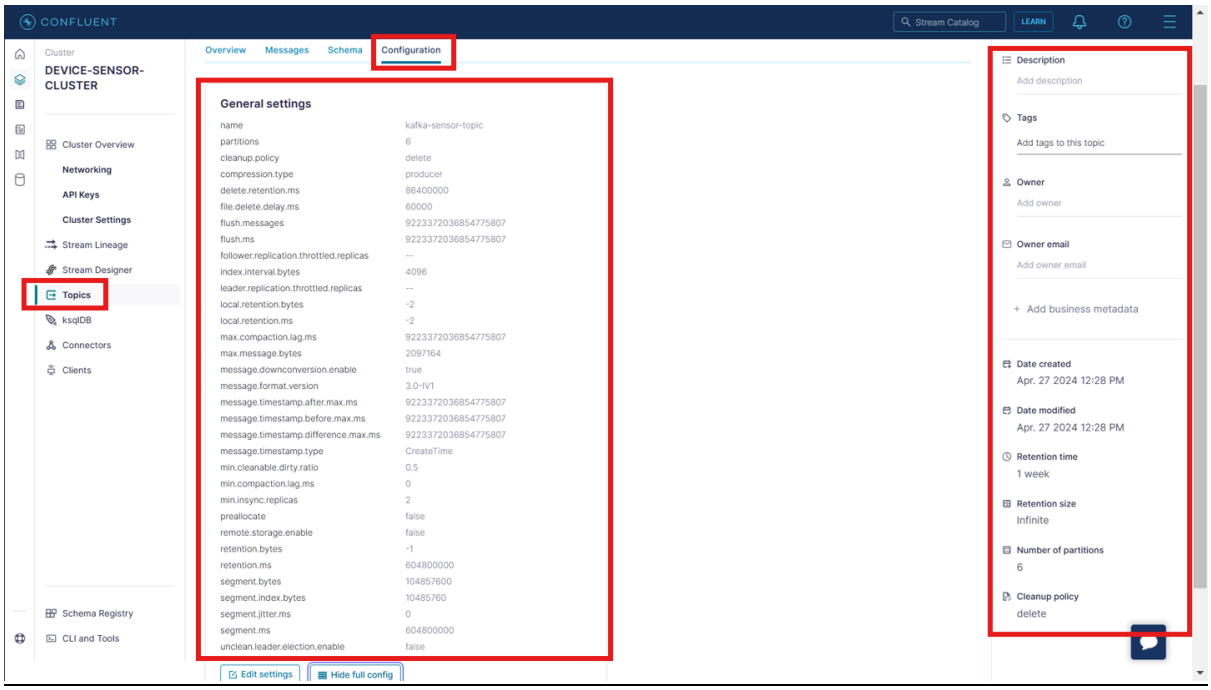
Under this section we can see total messages, how many messages are processed by Production and Consumption component in the last hour, time up to which messages will be retained in the Kafka topics(Retention time), different filters using which we can embed the searching mechanism followed by exporting the searched result in csv or json format.



- **Part 3rd Schema:**

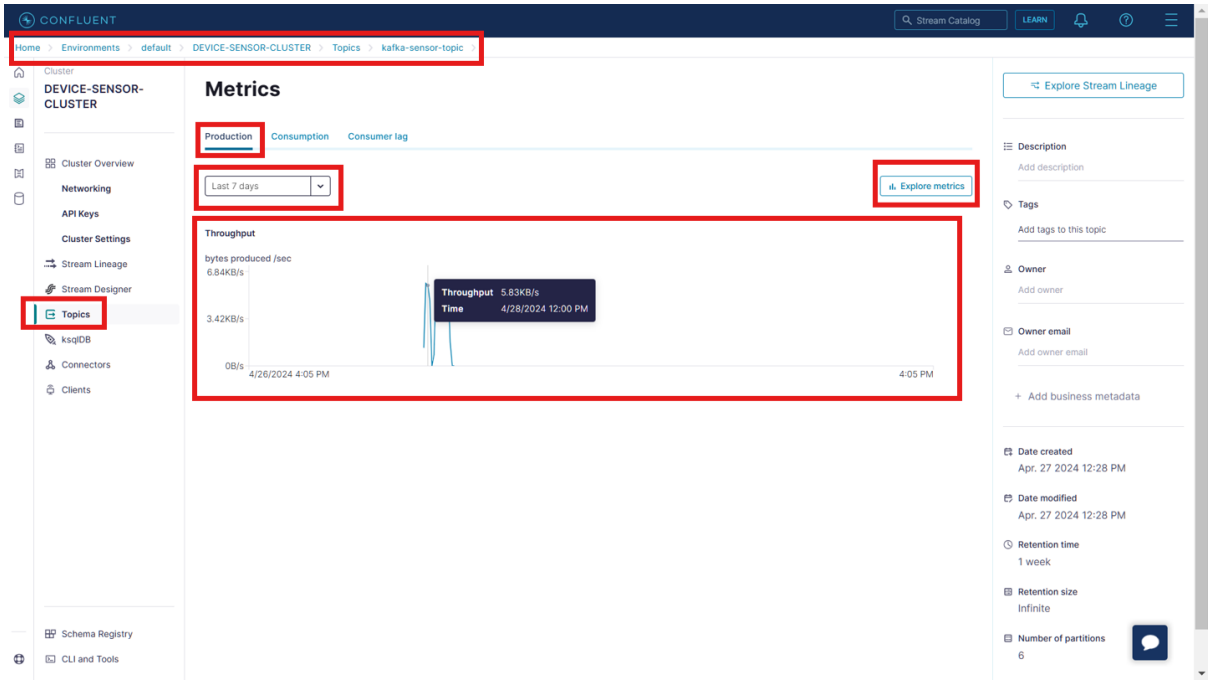


- **Part 4th Configuration:**



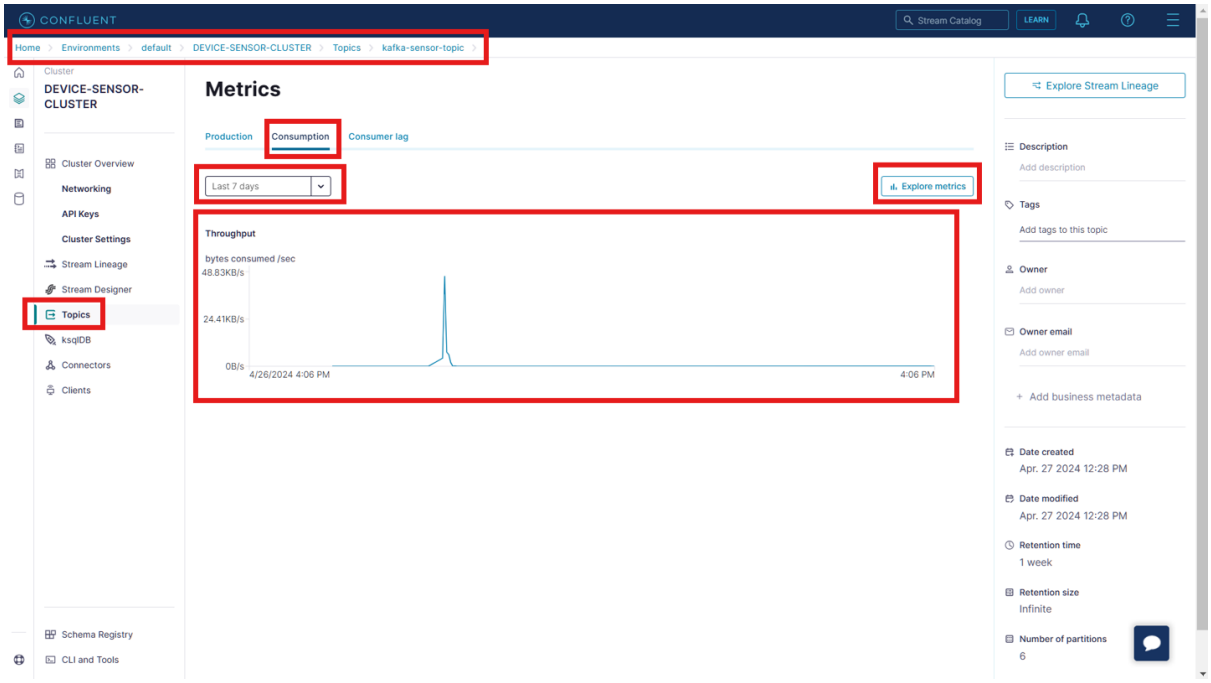
- **Home > Environments > default > DEVICE-SENSOR-CLUSTER > Topics > kafka-sensor-topic >:**
  - **Part 1st Production:**

    Here we can explore the different metrices related to the Production component.



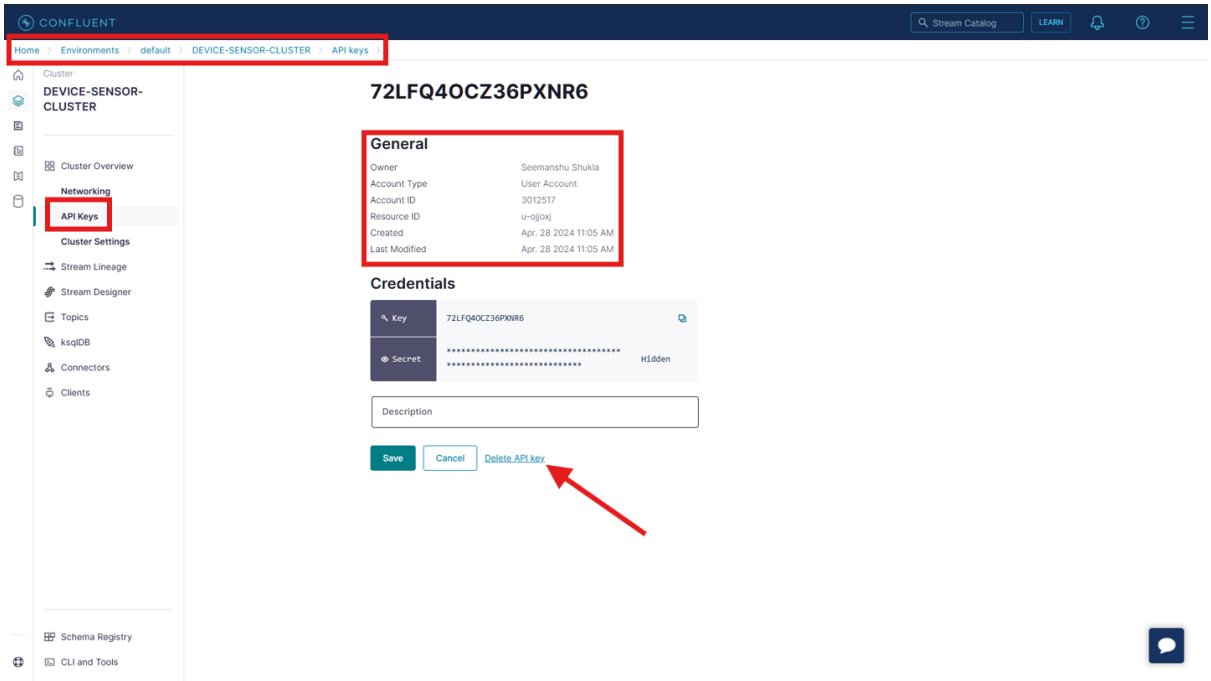
  - **Part 2nd Consumption:**

    Here we can explore the different metrices related to the Consumption component.

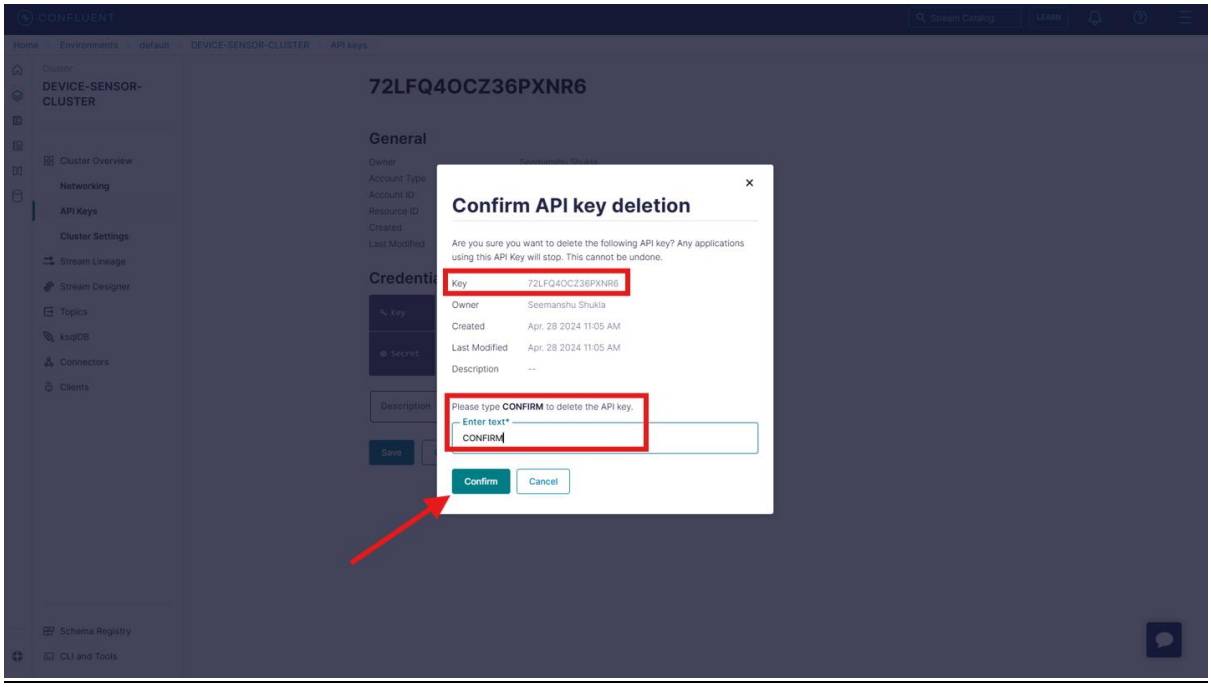


➔ **Deleting the Kafka resources to save the cost:**

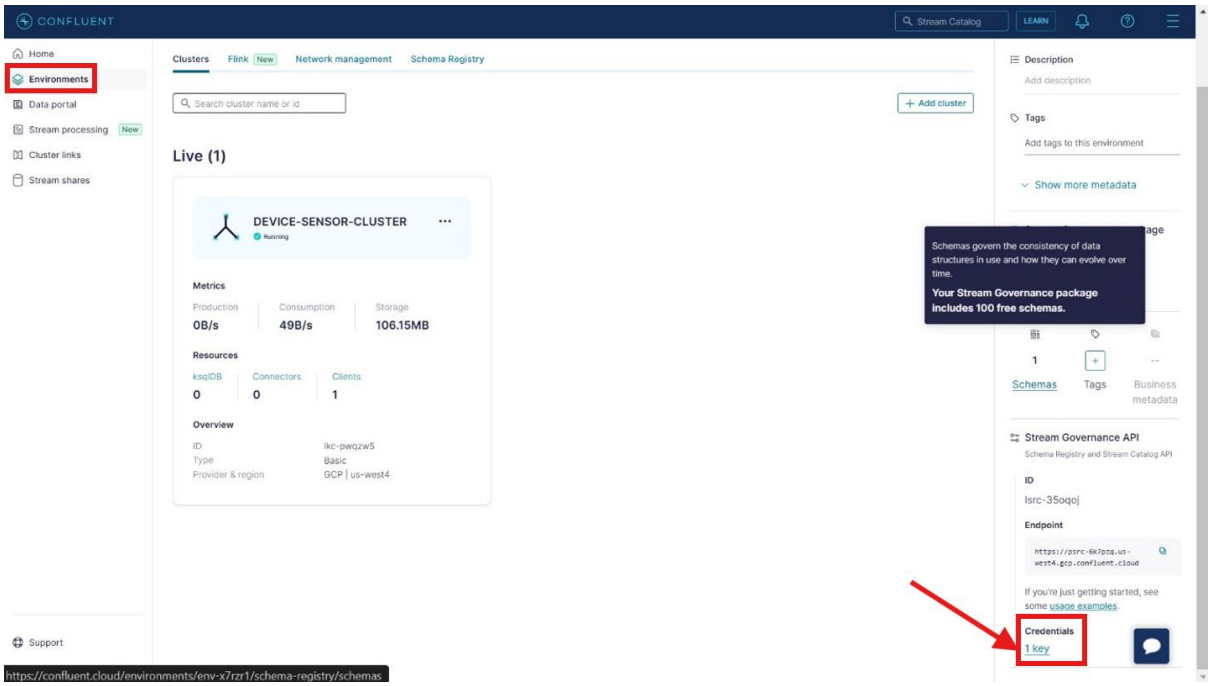
- **Kafka Cluster API key deletion:**
  - Traverse to **API Keys** inside the Kafka cluster (in our case it is **DEVICE-SENSOR-CLUSTER**) and click the **Delete API key** button:



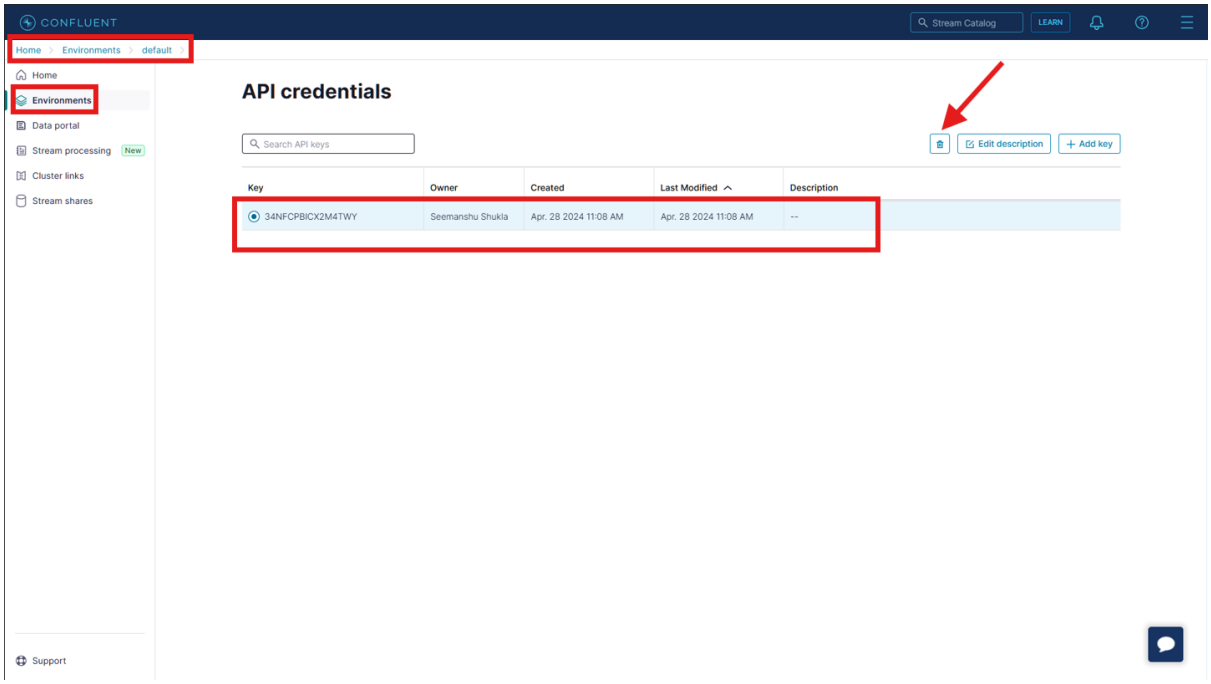
- In the **Confirm API key deletion** verify the API key and type **CONFIRM** in the **Enter text** field followed by clicking the **Confirm** button:



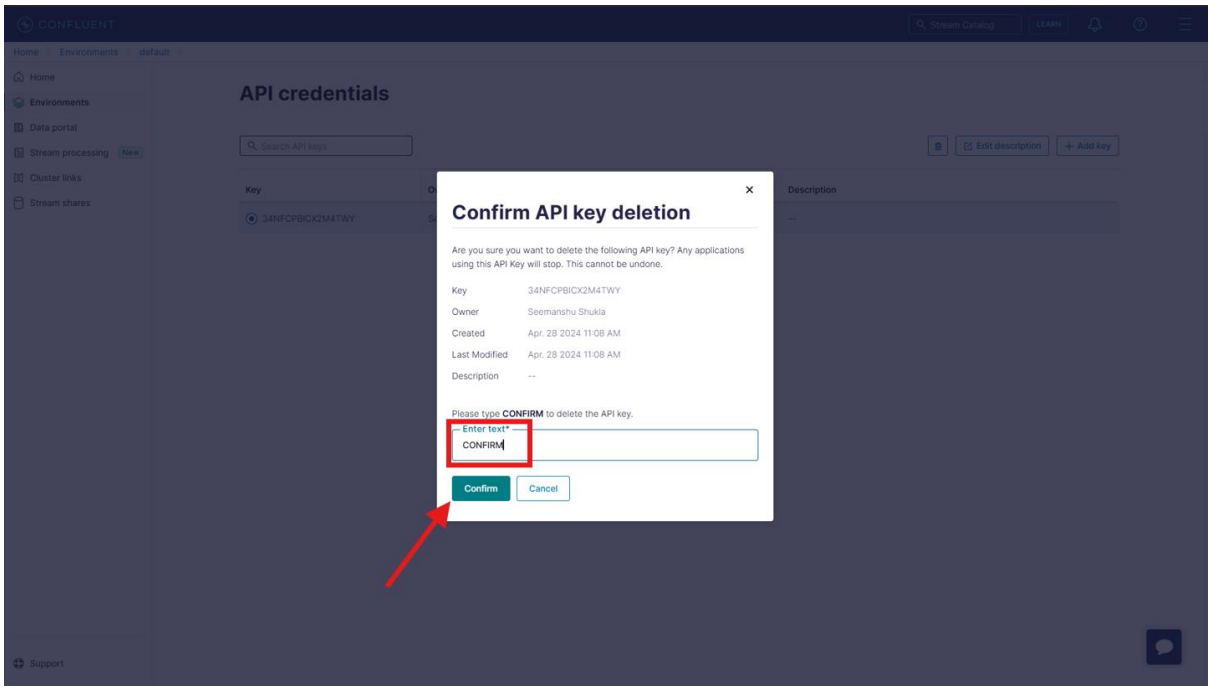
- **<u>Schema Registry API key deletion:</u>**
  - Traverse inside **default environment** and click **key** button under **Credentials**:



  - Select the radio button against the Schema Registry API key that you wish to delete followed by clicking the delete icon button:

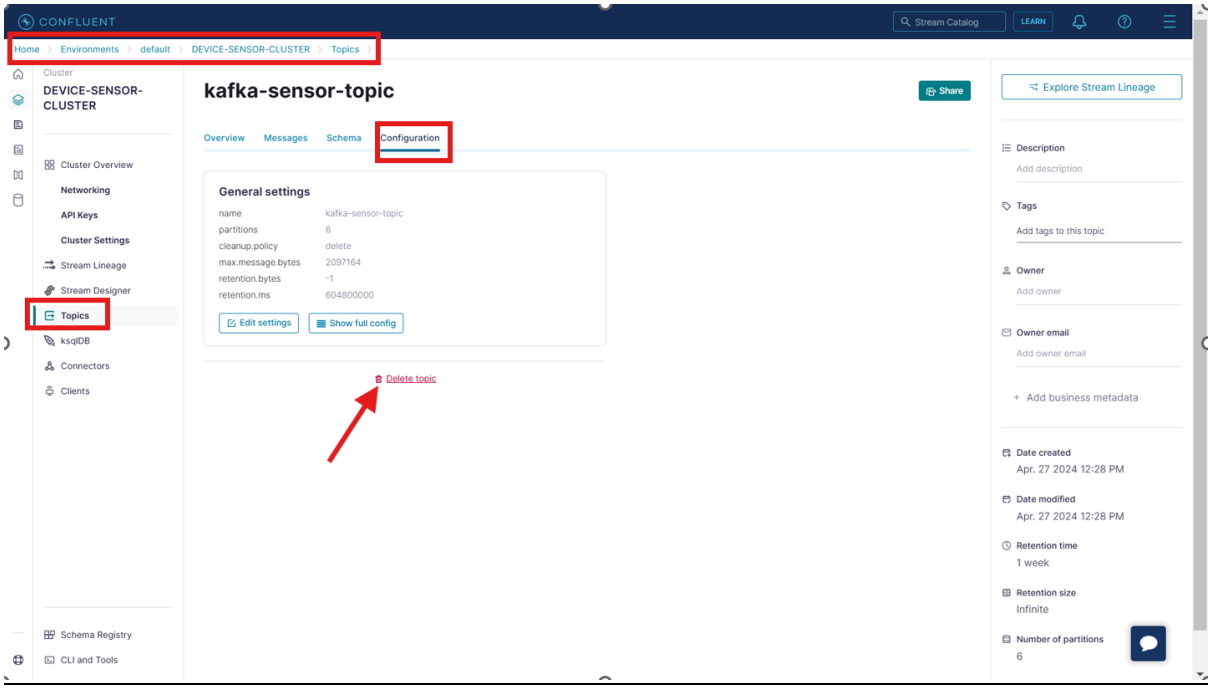


- In the Confirm API key deletion page type CONFIRM and click the **Confirm** button:

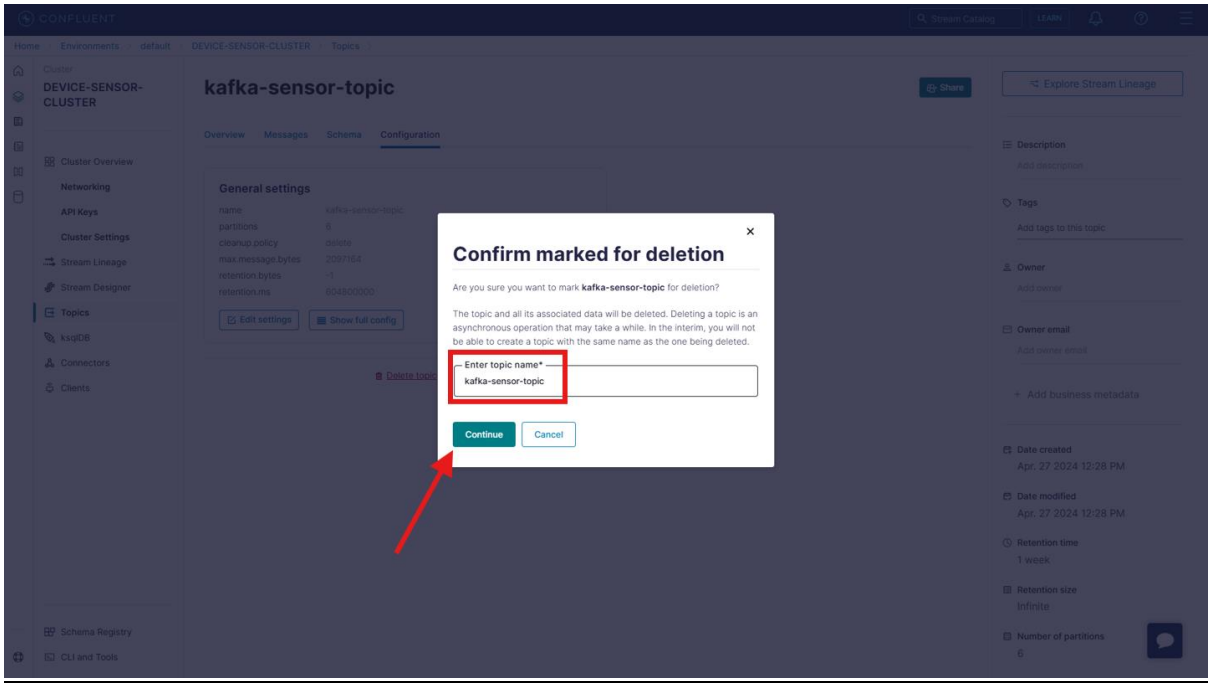


- **<u>Deleting the Kafka topic:</u>**

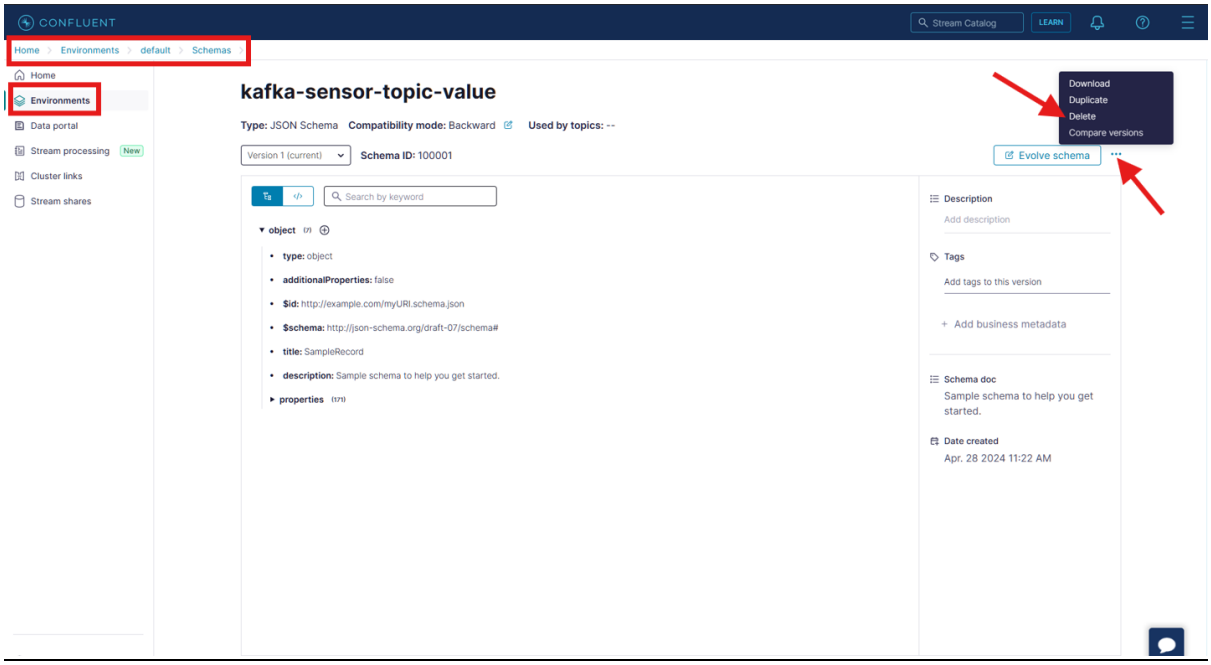
  - Traverse to the **Configuration** section inside Kafka topics and click the **Delete topic** button:



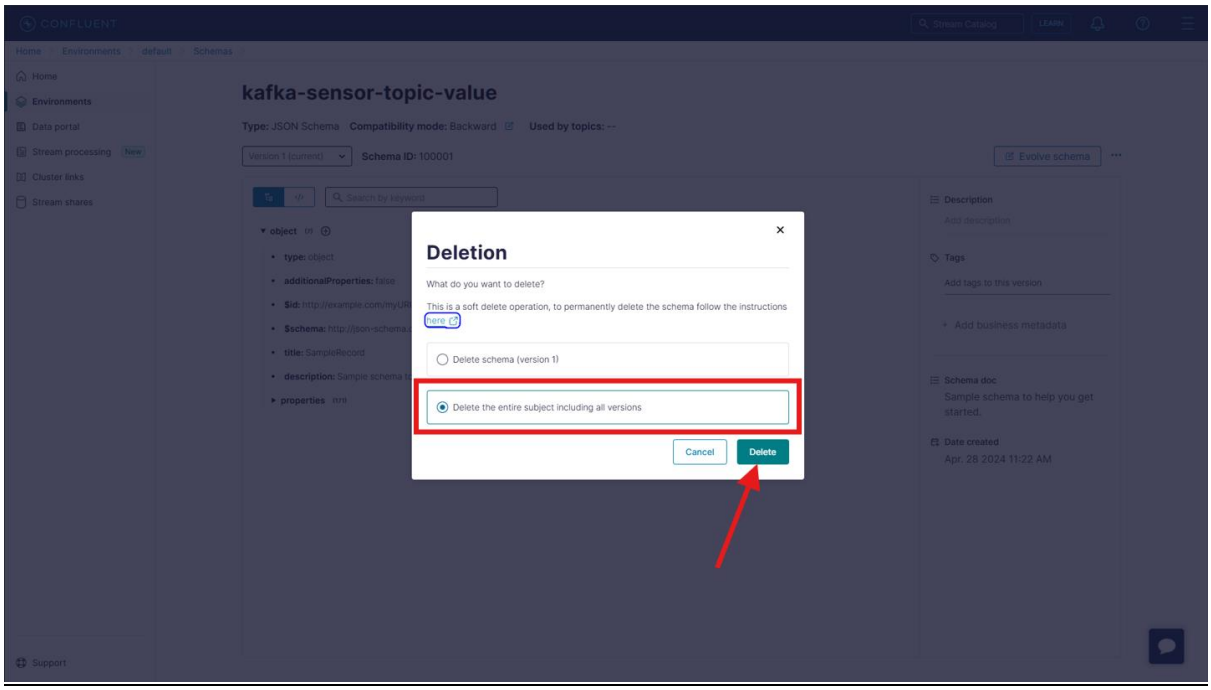
- In the **Confirm marked for deletion** page enter the name of the Kafka topic and click **Continue** button:



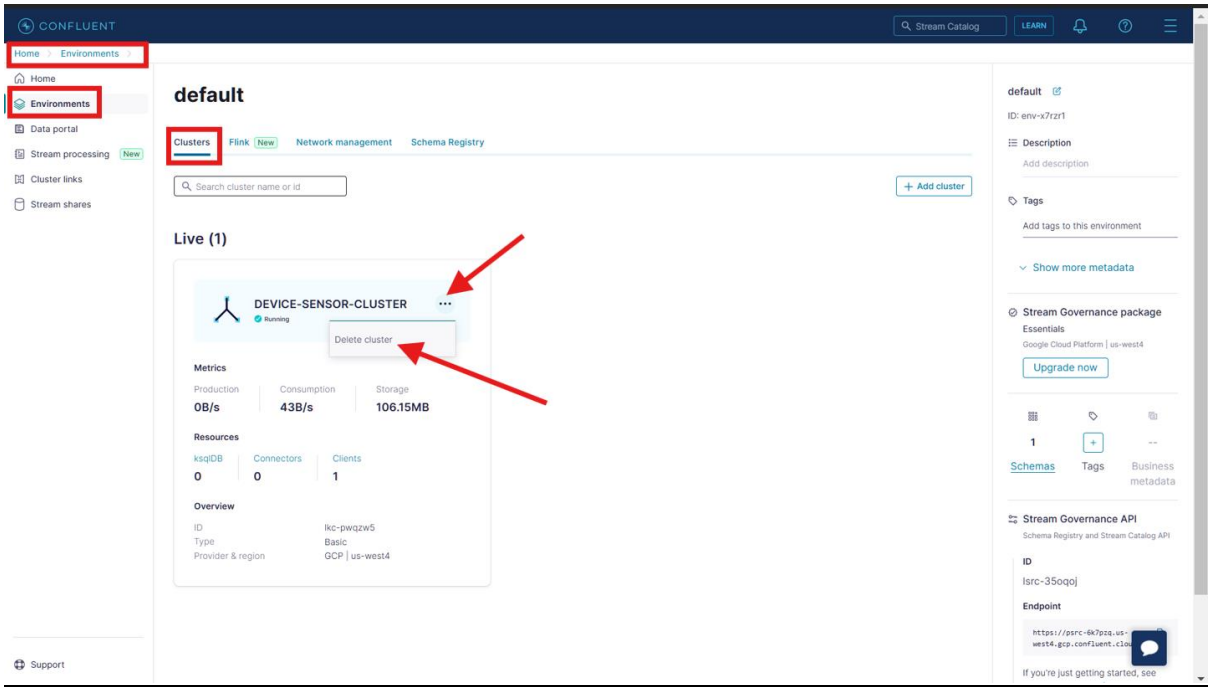
- **<u>Delete schema registry:</u>**
  - Traverse to Schemas section inside default environment. Click on the *3 dot icon* and click the **Delete** button:



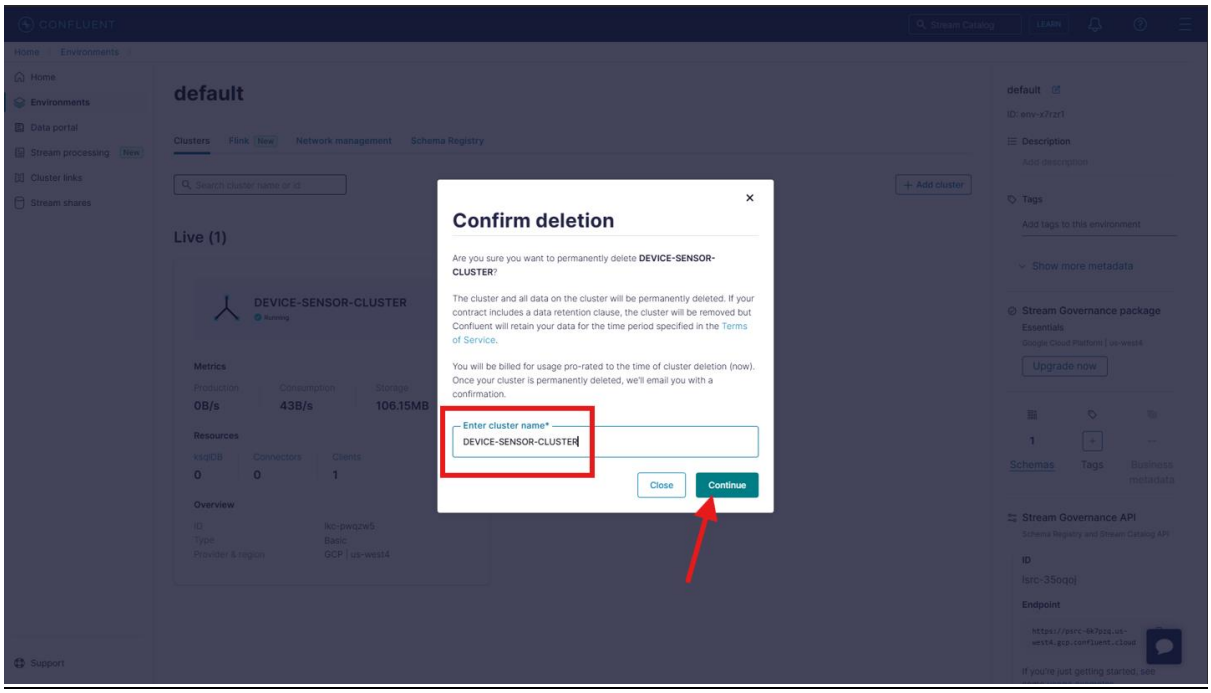
- Select the highlighted radio option and click the **Delete** button:



- **<u>Delete the Kafka Cluster:</u>**
  - Go to the Cluster Overview page inside the default environment. Then click the 3 dot icon and click **Delete cluster** button:



- Enter the name of the Kafka cluster to confirm the deletion and click the **Continue** button:



- **<u>Deleting the environment:</u>**
  - In our case we choose the **default** environment for building the Kafka cluster, then we cannot delete this default environment.
  - But in case we created the separate environment then in that case we will be required to delete that environment as well.

HAPPY LEARNING! 😊