

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.

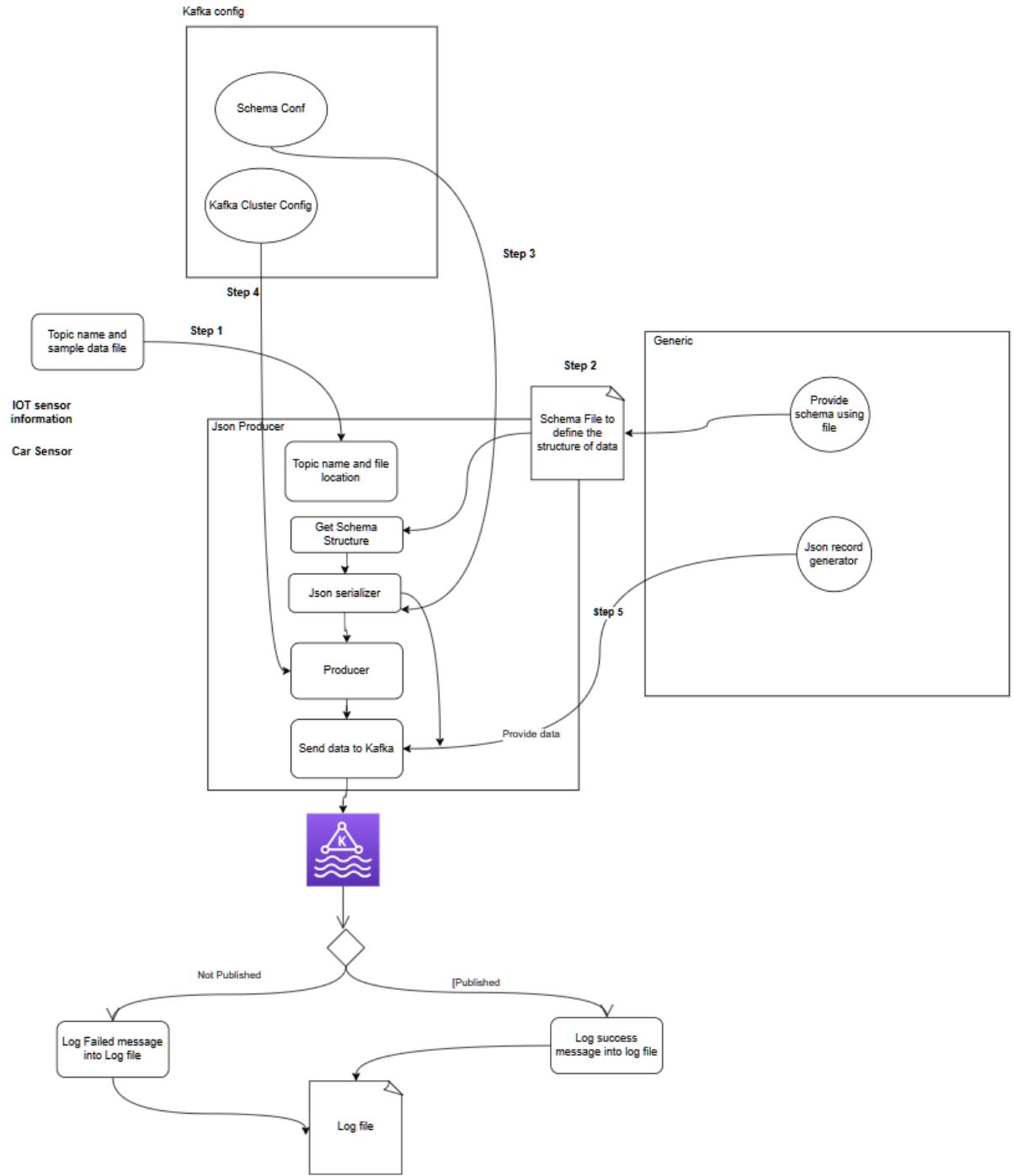
- *Ques:* Why are we not directly dumping data into MongoDB and are using Kafka as an mediator?

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.

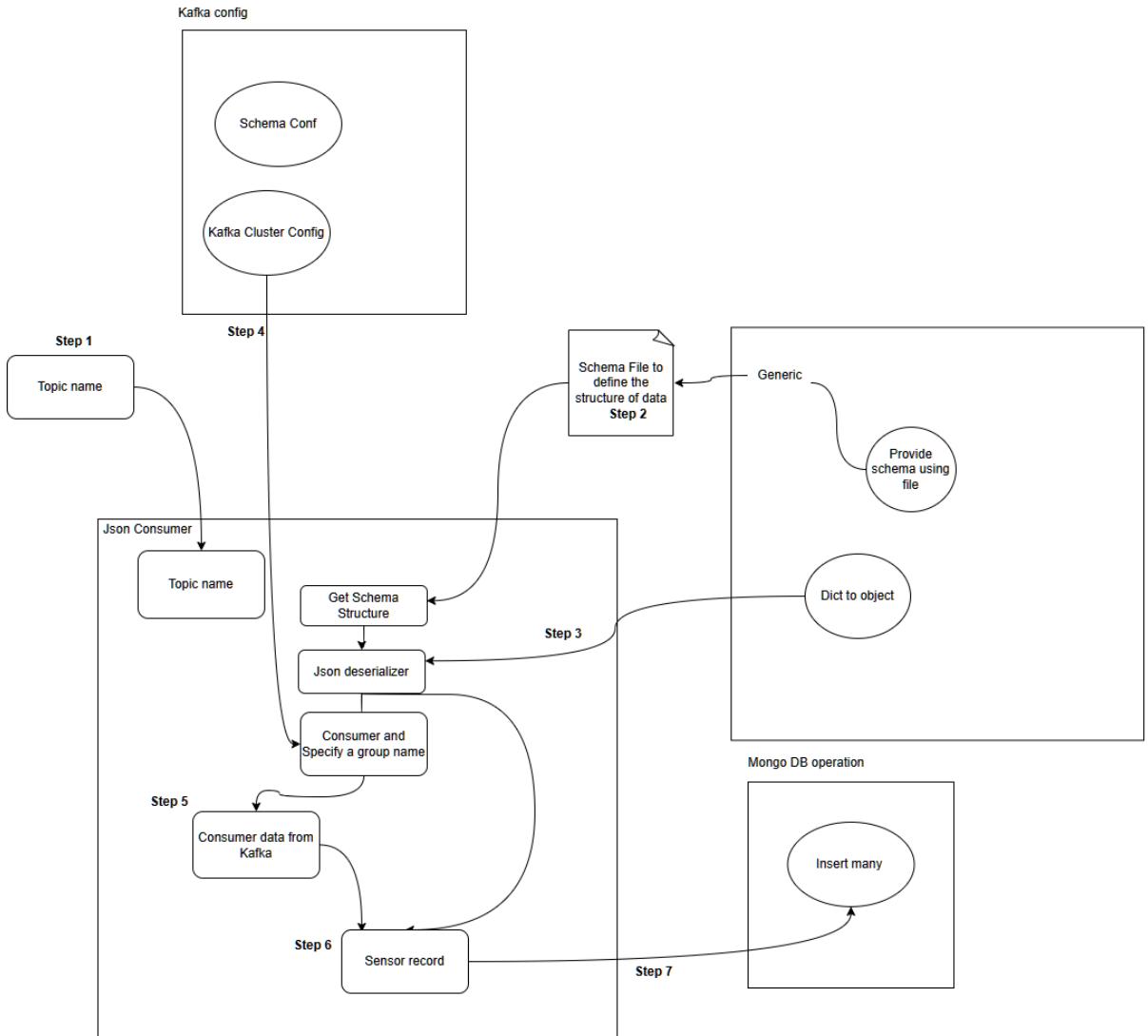
Reason 2: We might need to perform certain preprocessing before we are going to dump our data into MongoDB. For example we need to define the schema in which streaming data is intended to get stored in the target.

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:



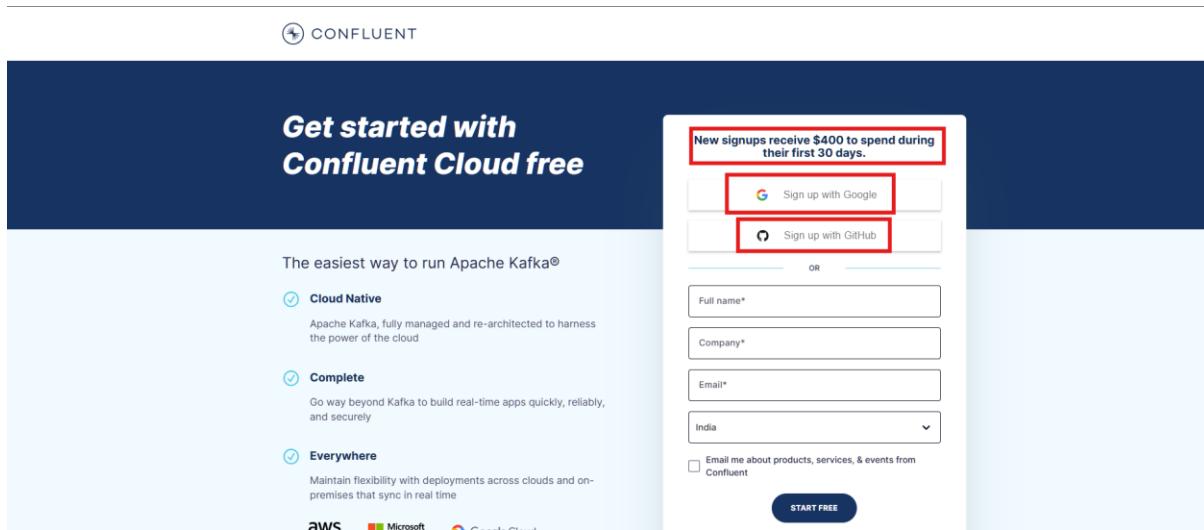
- High level Consumer architecture:



→ 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.

The screenshot shows the Confluent Cloud interface for managing environments. On the left, there's a sidebar with options like Home, Environments, Data portal, Stream processing, Cluster links, and Stream shares. The main area is titled 'default' and contains a brief description of what a Kafka cluster is. Below the description are two buttons: 'Get started with tutorial' and 'Create cluster on my own', with the latter being highlighted by a red box. To the right, there's a large icon representing a cluster and some metadata sections.

- 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.

The screenshot shows the 'Create cluster' page with four options: Recommended, Basic, Standard, Enterprise, and Dedicated. The 'Basic' option is highlighted with a red box. It has a description 'For learning and exploring Kafka and Confluent Cloud.' and a table of resource limits. Below the table is a 'Begin configuration' button, which is also highlighted with a red box. The other three options have their own descriptions and resource tables, with their 'Begin configuration' buttons visible.

| | Ingress | Egress | Storage | Client connections | Partitions | Uptime SLA |
|-------------------|----------------|----------------|----------------|--------------------|-------------|--------------|
| Basic | up to 250 MB/s | up to 750 MB/s | up to 5,000 GB | up to 1,000 | up to 4,096 | up to 99.9% |
| Standard | up to 250 MB/s | up to 900 MB/s | unlimited | up to 1,000 | up to 4,096 | up to 99.99% |
| Enterprise | up to 300 MB/s | up to 900 MB/s | unlimited | up to 22,500 | up to 5,000 | up to 99.99% |
| Dedicated | up to 60 MB/s | up to 180 MB/s | unlimited | up to 18,000 | up to 4,500 | up to 99.99% |

- 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.

The screenshot shows the 'Create cluster' interface. At the top, there are four tabs: '1. Cluster type', '2. Region/zones', '3. Payment', and '4. Review and launch'. Below these tabs, there are three options: 'aws' (selected), 'Google Cloud', and 'Microsoft Azure'. To the right of the options, there are dropdown menus for 'Region*' (set to 'Los Angeles (us-west2)') and 'Uptime SLA*' (set to '99.5%'). At the bottom left is a 'Go back' link. In the center, a message says 'First E-CKU is free, \$0.16 /hr after'. To the right, there is a 'Continue' button (which is highlighted with a red box) and a speech bubble icon.

- 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.

The screenshot shows the 'Create cluster' interface with the 'Payment' tab selected. At the top, there are four tabs: '1. Cluster type', '2. Region/zones', '3. Payment', and '4. Review and launch'. Below these tabs, there is a heading 'Enter payment information (optional)' with a note: 'Avoid service interruptions when your free trial ends. You won't be charged until your free trial is over.' There are two payment method options: 'Card' and 'US bank account'. Below these options, there is a link 'Secure, 1-click checkout with Link'. The payment form includes fields for 'Card number' (with placeholder '1234 1234 1234 1234'), 'CVC' (with placeholder '123'), and 'Expiration' (with placeholder 'MM / YY'). At the bottom, there is a note: 'By providing your card information, you allow Confluent, Inc. to charge your card for future payments in accordance with their terms.' At the bottom right, there is a 'Skip payment' button (which is highlighted with a red box), a 'Review' button, and a speech bubble icon.

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

Create cluster

Label to help identify your cluster in the console

3. Payment → 4. Review and launch

Cluster name: DEVICE-SENSOR-CLUSTER

E-CKU cost First E-CKU is free, \$0.15525 /hr after

Write \$0.0575 /GB

Read \$0.0575 /GB

Storage \$0.00012603 /GB-hour

Configuration Usage limits Uptime SLA

Cluster configuration

Settings marked with an asterisk (*) cannot be changed once you launch your cluster

Cluster type Basic Min E-CKUs 1

*Provider Google Cloud Platform Internet

*Region us-west2 *Networking Automatic

Go back Launch cluster

- 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.

CONFLUENT

Home >

Home Environments Data portal Stream processing (New) Cluster links Stream shares + Add cloud environment

Environments

default

1 cluster | 0 compute pools

Recommended

Explore with an interactive demo [Demo](#) See how Confluent Cloud helps power a ride sharing app in this 5-minute interactive demo! [Launch demo](#)

Set up a schema Try out this basic workflow to leverage schemas in Kafka with Confluent Cloud and maintain data consistency [Get started](#)

Confluent Terraform provider Automate the management of your Confluent Cloud resources with ease. [Read blog](#)

Support

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

The screenshot shows the Confluent Platform interface. In the top left, the 'Environments' tab is selected, indicated by a red box. The main content area shows a cluster named 'DEVICE-SENSOR-CLUSTER' which is 'Running'. A red box highlights the 'Live (1)' section. On the right, there's a detailed view of the environment 'default' with fields for description, tags, and metadata like date created and modified.

- **Step 2: Kafka Topics setup**

- 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.

The screenshot shows the 'Cluster Overview' page for 'DEVICE-SENSOR-CLUSTER'. The left sidebar has a 'Topics' link highlighted with a red box. The right panel shows a production checklist with four items: 'Real Time Alerts', 'Expert Support', 'Data Integrity', and 'Highest SLA', each with a 'Get started' button. Below this is a section for setting up connectors, clients, and producing sample data. A red box highlights the 'Topics' section in the sidebar.

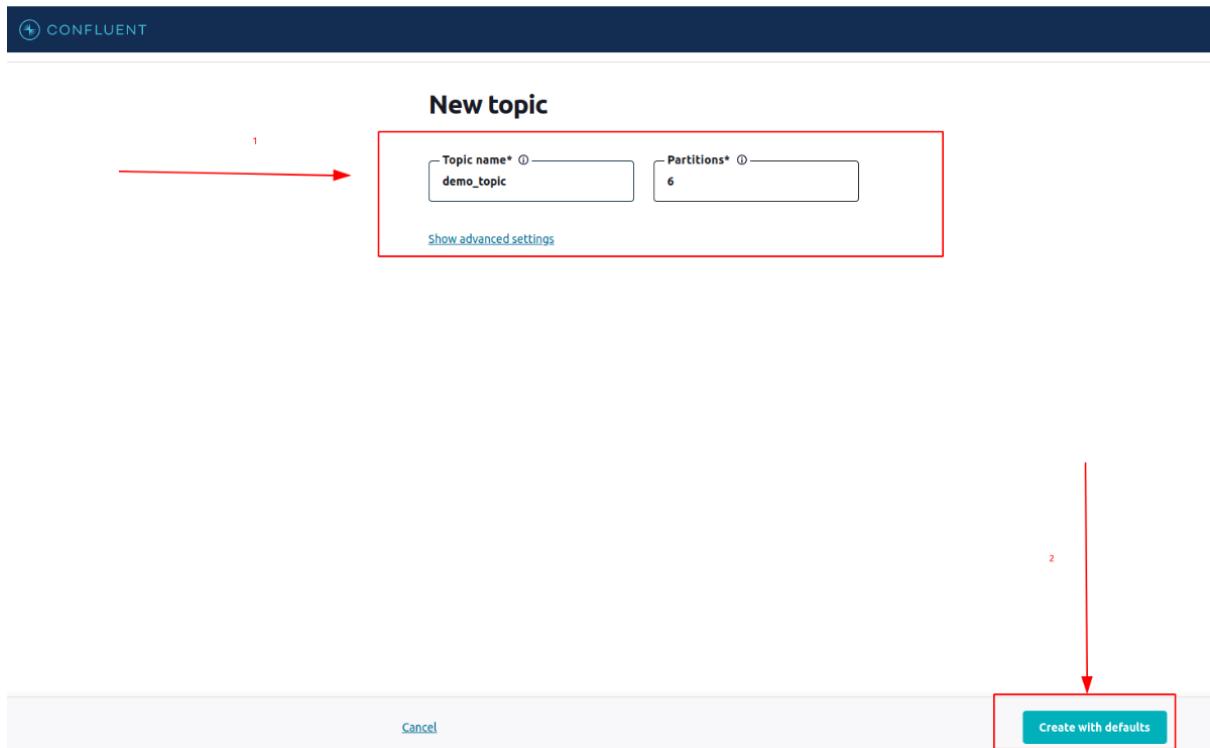
- Click on the **Add topic:**

- 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(1st,4th,5th) and partition2=data_from_person(2nd,3rd,7th)).

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.



- Step 3: Generating the Secret keys

- 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 Kafka.



- 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

DEVICE-SENSOR-CLUSTER

Overview

Complete your production checklist

- Real Time Alerts
- Expert Support
- Data Integrity
- Highest SLA

Here are more ways to get data moving through your cluster!

- Set up connector
- Set up client
- Produce sample data

API Keys

Description: Add description

Tags: Add tags to this cluster

Cluster ID: lkc-pwqzw5

Cluster type: Basic

Date created: Apr. 27 2024 12:13 PM

Date modified: Apr. 27 2024 12:13 PM

Cloud provider: GCP

Cloud region: us-west4

Uptime SLA: 99.5%

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

DEVICE-SENSOR-CLUSTER

API keys

+ Add key

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

Create key

1. Access control

Select scope for API key

Global access

Allow your API key to access everything you can access. Key access will be linked to your account.

*Recommended for development.

Granular access

Limit the access for your API key. Manage your API key's access through a service account.

*Recommended for production.

Next **Cancel**

- 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.

The screenshot shows the 'Create key' page. At the top, there are two tabs: '1. Access control' and '2. Get your API key'. Below the tabs, a note says: 'Use this API key to connect with the cluster. Store the API key and secret below somewhere safe. This is the only time you'll see the secret.' To the right of this note are two red boxes with copy icons. A red arrow points from the text above to these boxes. Below the note, another note says: 'These credentials can take up to one minute to propagate.' Underneath this is a dark blue sidebar with 'Key' and 'Secret' buttons. To the right of the sidebar is a large red rectangular area covering the 'Key' and 'Secret' fields. Below this is a 'Description' input field. At the bottom is a teal button labeled 'Download and continue'. A red arrow points from the text above to this button.

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

The screenshot shows a terminal window with a dark theme. The title bar says 'api-key-72LFQ4OCZ36PXNR6.txt'. The window has a menu bar with 'File', 'Edit', and 'View'. Below the menu is a section header '==== Confluent Cloud API key ===='. Underneath it, the text is highlighted with a red box. The text contains the following information:

```

API key:
72LFQ4OCZ36PXNR6

API secret:
As+SobVSFsvfco+BdD1sgvTY6xFMIwVuUjzT8JF3sHrwFfxwMaq4S6DQGK/d0A0f

Resource:
lkc-pwqzw5

Bootstrap server:
pkc-6ojv2.us-west4.gcp.confluent.cloud:9092

```

- 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 screenshot shows the Confluent Cloud interface for managing clusters. On the left, there's a sidebar with 'Home' and 'Environments' options. The main area is titled 'default' and shows a single cluster named 'device-sensor-cluster' (status: Running). Below the cluster card, there are sections for 'Metrics' (Production: 0B/s, Consumption: 0B/s, Storage: 0B) and 'Resources' (Topics: 1, ksqlDB: 0, Connectors: 0, Clients: 2). At the bottom, there's an 'Overview' table with columns for ID, Type, and Provider & region. On the right, a sidebar for the 'Stream Governance package' is open, showing its configuration. A red box highlights the 'Enable Stream Governance to access Schema Registry, Stream Catalog and Stream Lineage' section, specifically the 'Enable now' button, which is highlighted with a red border.

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

Confidenc's Stream Governance Suite establishes trust in the data streams moving throughout your cloud environments and delivers an easy, self-service experience for more teams to discover, understand, and put streaming data to work.



Essentials

The fundamentals for getting started.

Stream Quality
Schema Registry & validation

- 99.5% uptime SLA
- 1,000 schemas included*
- 9 cloud regions supported

Stream Catalog
Data organization & discoverability

- Auto-technical metadata ingestion
- Tags metadata
- Cloud UI & REST API

Stream Lineage
Data origin & tracking

- Real-time data streams lineage

*Starting at \$0.002/schema/hour after 1,000 schemas per environment

Begin configuration

Starting at
FREE

Upgrade to Advanced at any time



Advanced

Enterprise-ready controls for data in motion.

Stream Quality
Schema Registry & validation

- 99.95% uptime SLA
- 20,000 schemas included
- 28 cloud regions supported

Stream Catalog
Data organization & discoverability

- All existing Essentials features +
 - Business metadata
 - GraphQL API

Stream Lineage
Data origin & tracking

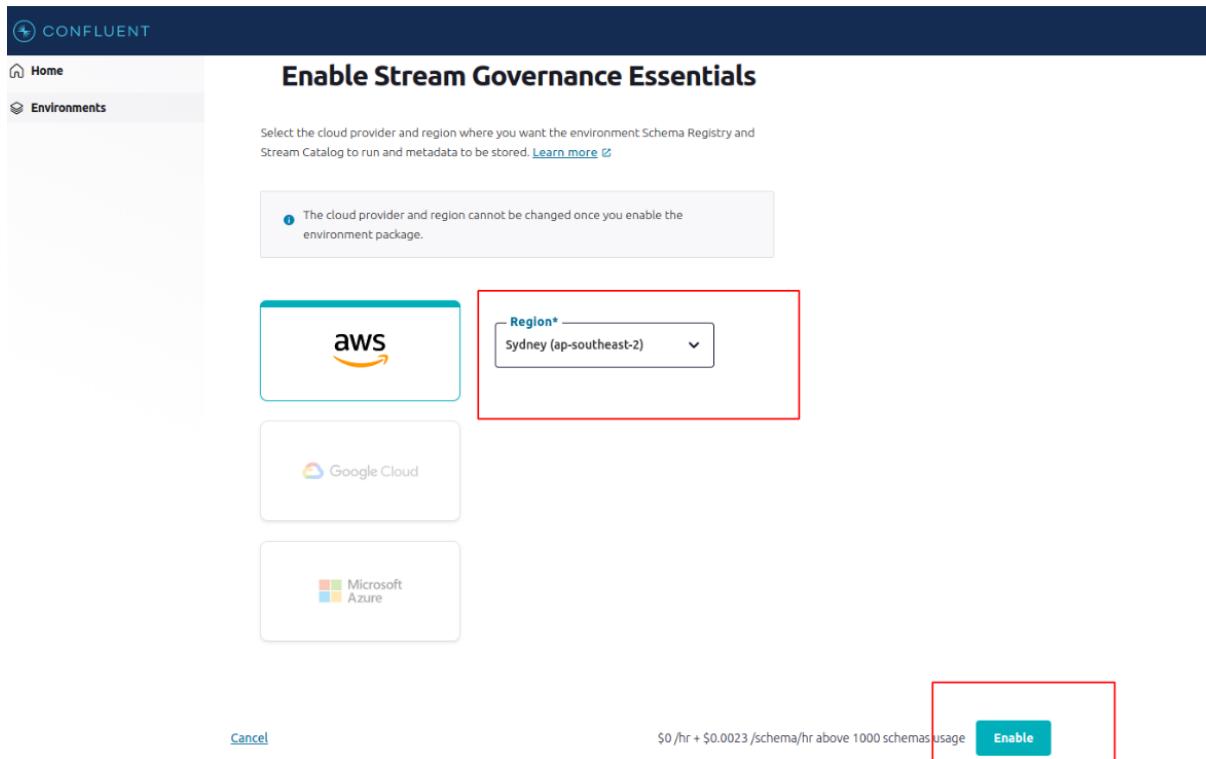
- All existing Essentials features +
 - Point-in-time lineage
 - Lineage search

Begin configuration

Starting at
\$1/hr

The full Stream Governance feature set

- 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:

The screenshot shows the Stream Governance package interface. At the top, there's a banner for 'Stream Governance package' with 'Upgrades available'. Below it, there are tabs for 'Schemas', 'Tags', and 'Business metadata'. The main area is titled 'Stream Governance API' and describes it as 'Schema Registry and Stream Catalog API'. It shows an 'Endpoint' URL: <https://parc-7q7vj.ap-southeast-2.aws.confluent.cloud>. Below the URL, there's a note about 'usage examples' and a 'Credentials' section with a red box around the 'Add key' button.

- Click on Add key:

The screenshot shows the Confluent Platform UI with the path 'Home > Environments > default' highlighted by a red box. On the left, there's a sidebar with 'Environments' selected. The main area is titled 'API credentials' and contains a table with columns 'Key', 'Owner', 'Created', 'Last Modified', and 'Description'. At the top right of the table, there are buttons for 'Edit description' and 'Add key', with a red box around the 'Add key' button.

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

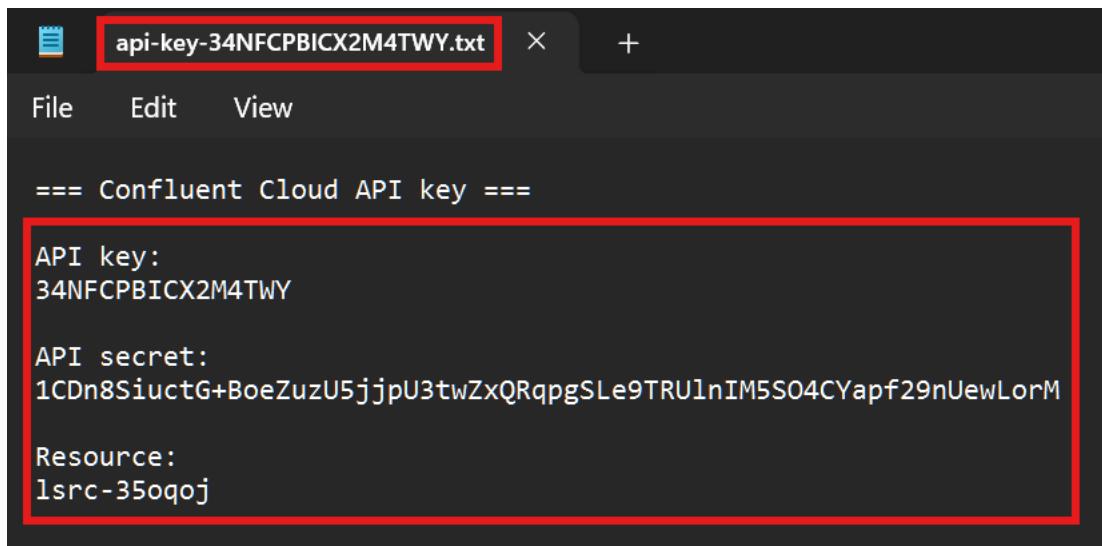
Create a new Schema Registry API key

Use this API key to connect with Schema Registry. Store the API key and secret below somewhere safe. This is the only time you'll see the secret.

These credentials can take up to one minute to propagate.

This screenshot shows the 'Create a new Schema Registry API key' page. It displays two fields: 'Key' and 'Secret', each with a redacted value and a blue copy icon. Below these fields is a 'Description' input field with a red box around it. At the bottom is a large green 'Download and continue' button with a red box around it.

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



```
api-key-34NFCPBICX2M4TWY.txt x +  
File Edit View  
==== Confluent Cloud API key ====  
  
API key:  
34NFCPBICX2M4TWY  
  
API secret:  
1CDn8SiuctG+BoeZuzU5j jpU3twZxQRqpgSLe9TRUlnIM5S04CYapf29nUewLorM  
  
Resource:  
lsrc-35oqoj
```

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:

- **Step 1: Signup with MongoDB:**

- 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:



- **Step 2: Create MongoDB Cluster:**

- 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:

A screenshot of the MongoDB Atlas Clusters page. The left sidebar has sections for Overview, Deployment (with "Database" highlighted and boxed in red), Services, Security, and Network Access. The main area shows a cluster named "SEEMANSHU'S ORG - 2024-04-28 > PROJECT 0". It includes a search bar, a "Load sample datasets to sensor-fault-detection" section with a "Load sample dataset" button, and a detailed view of the "sensor-fault-detection" cluster. This view shows metrics like R: 0, W: 0, Connections: 10.0, In: 10.4 B/s, Out: 167.5 B/s, and Data Size: 95.1 MB / 512.0 MB (19%). There are also "Edit Config" and "+ Create" buttons at the top right.

- 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:

The screenshot shows the MongoDB Atlas interface. On the left, there's a sidebar with 'Project O' selected. Under 'Data Services', 'Clusters' is highlighted. A red box highlights the 'Clusters' tab. At the top right, there are buttons for 'Edit Config' and '+ Create'. A red box highlights the '+ Create' button. Below it, a message says 'Load sample datasets to sensor-fault-detection.' with a 'Load sample dataset' 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.

The screenshot shows the 'Create a Shared Cluster' dialog. It has three tabs: 'Serverless' (disabled), 'Dedicated' (disabled), and 'Shared' (selected). A red box highlights the 'Shared' tab. Below the tabs, there's a note: 'For learning and exploring MongoDB in a sandbox environment. Basic configuration controls.' A red box highlights this note. The 'Cloud Provider & Region' section shows 'AWS, Mumbai (ap-south-1)' selected. Under 'Cloud Provider', 'aws' is selected. Other options include 'Google Cloud' and 'Azure'. Below the provider section, there are buttons for 'Cancel' and 'Create Cluster'. A red box highlights the 'Create Cluster' button.

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

Create a Shared Cluster

Serverless

Dedicated

Shared

For learning and exploring MongoDB in a sandbox environment. Basic configuration controls.

No credit card required to start. Upgrade to dedicated clusters for full functionality.
Explore with sample datasets. Limit of one free cluster per project.

Cloud Provider & Region

AWS, Mumbai (ap-south-1) ^



Google Cloud



★ Recommended region ⓘ ⚡ Dedicated tier region ⓘ 🌱 Low carbon emissions region ⓘ

NORTH AMERICA

Oregon (us-west-2) ★ 🌱

EUROPE

Paris (eu-west-3) ★ 🌱

AUSTRALIA

Sydney (ap-southeast-2) ★

N. Virginia (us-east-1) ★ 🌱

Ireland (eu-west-1) ★ 🌱

Melbourne (ap-southeast-4) ★

Ohio (us-east-2) ★ 🌱

Frankfurt (eu-central-1) 🌱

ASIA

N. California (us-west-1) 🌱

Stockholm (eu-north-1) ★ 🌱

ASIA

Mumbai (ap-south-1) ★ 🌱

Hong Kong (ap-east-1) ★

Seoul (ap-northeast-2) ★

Montreal (ca-central-1) ★ 🌱

London (eu-west-2) ★ 🌱

Calgary (ca-west-1) ★ 🌱

Zurich (eu-central-2) ★ 🌱

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

Create a Shared Cluster

Serverless Dedicated **Shared**

For learning and exploring MongoDB in a sandbox environment. Basic configuration controls. x

No credit card required to start. Upgrade to dedicated clusters for full functionality.
Explore with sample datasets. Limit of one free cluster per project.

Cloud Provider & Region AWS, Mumbai (ap-south-1) ▾

Cluster Tier M2 (Shared RAM, 2 GB Storage) ▾
Encrypted

Additional Settings MongoDB 7.0, Backup ▾

Cluster Details sensor-fault-detectio 0 Tags ▲

Cluster Name One time only: once your cluster is created, you won't be able to change it.

Pay-as-you-go! You will be billed hourly and can terminate your cluster anytime. Excludes variable [data transfer](#), [backup](#), and taxes.

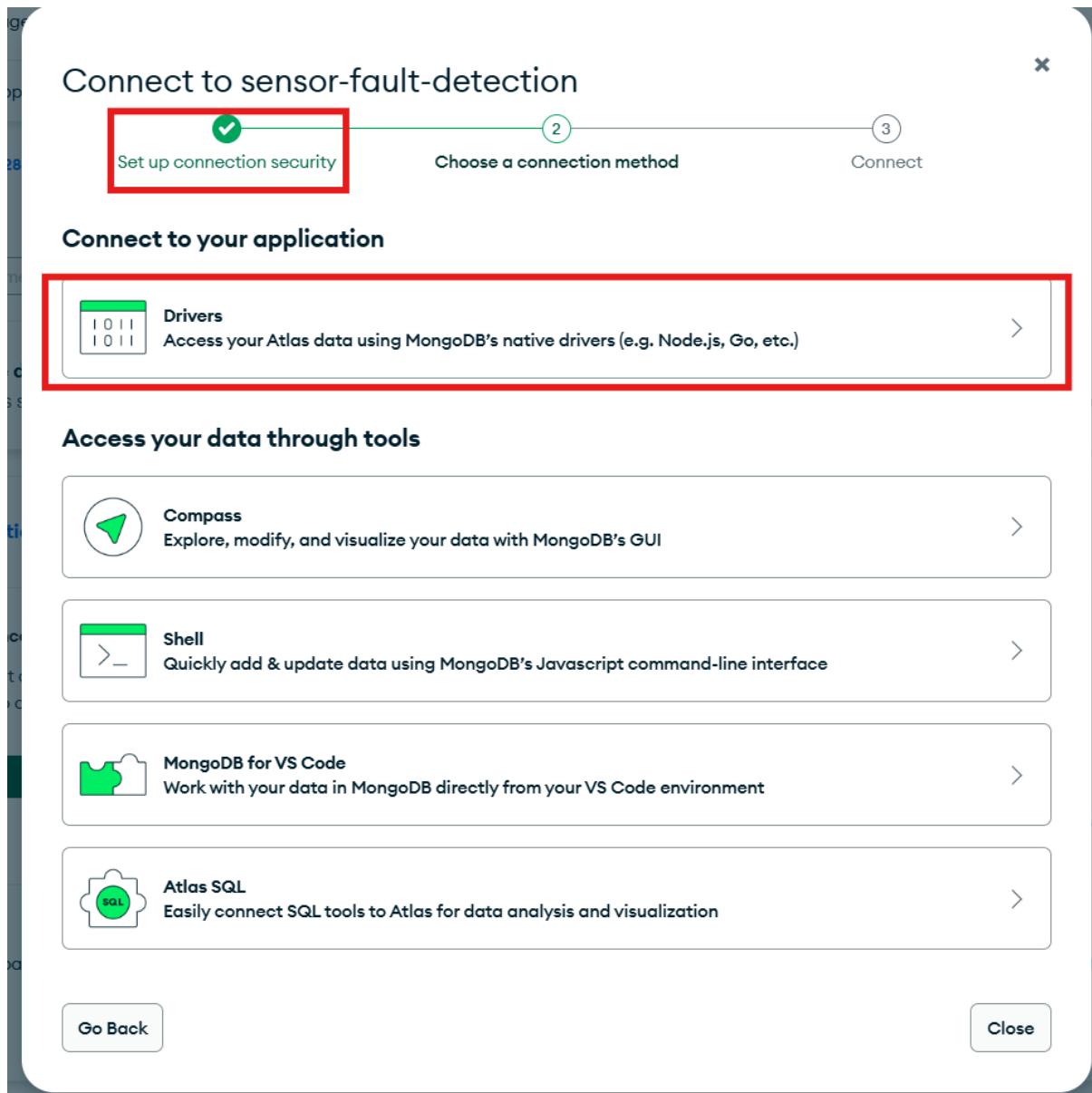
Cancel Create Cluster

- **Step 3: Generate connection string:**
 - 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:

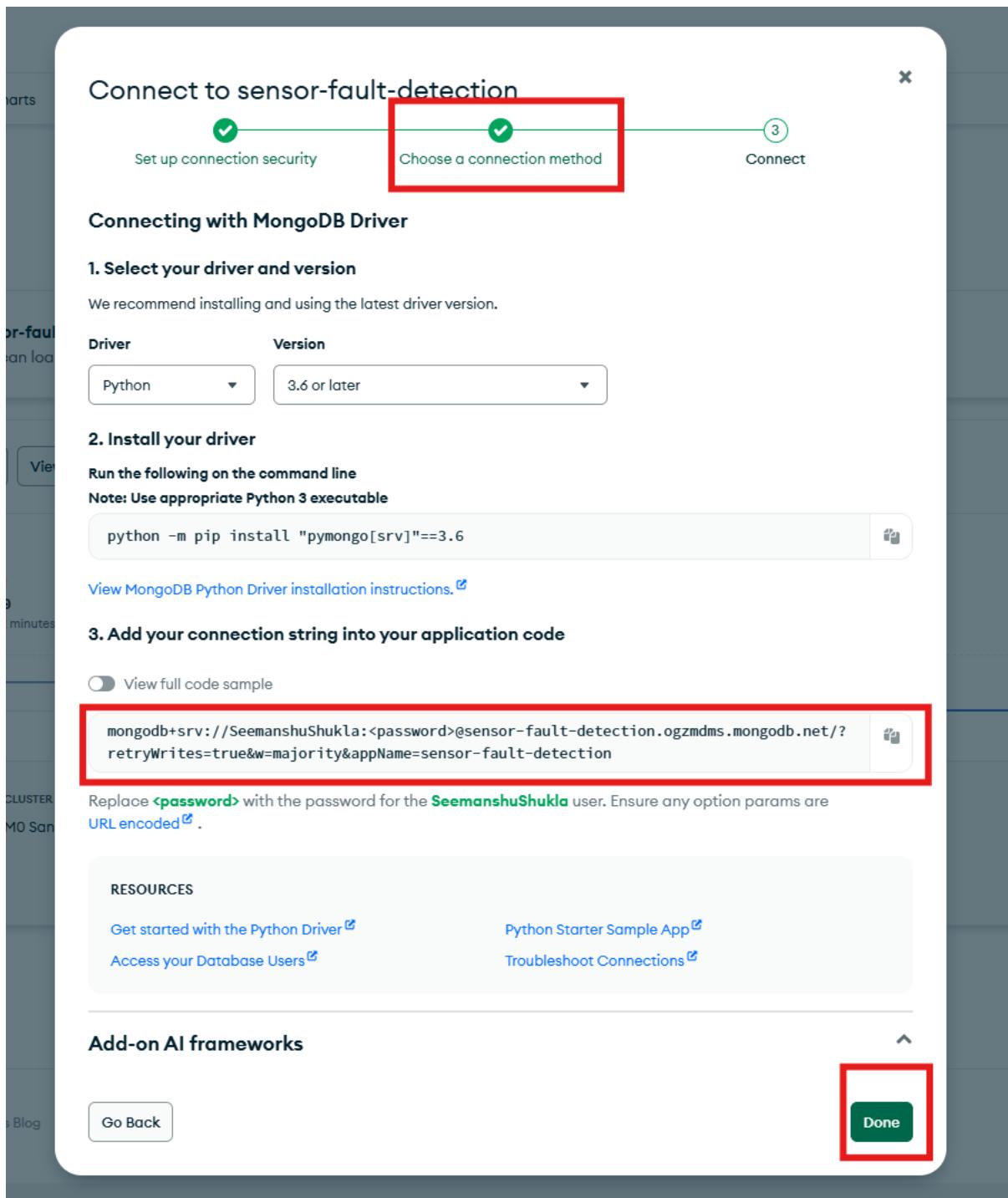
The screenshot shows the MongoDB Atlas interface. On the left, there's a sidebar with various project and service options. The main area is titled 'Clusters' and shows a single database named 'sensor-fault-detection'. This database is highlighted with a red box. Below the database name are several monitoring metrics: R: 0, W: 0, Connections: 0, In: 0.0 B/s, Out: 0.0 B/s, and Data Size: 95.1 MB / 512.0 MB (19%). The 'FREE' and 'SHARED' status is also highlighted with a red box. At the bottom, there's a table with cluster details like Version (7.0.8), Region (AWS Mumbai), Cluster Tier (M0 Sandbox (General)), Type (Replica Set - 3 nodes), and various service links.

- 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**:

Connect to sensor-fault-detection



Review Setup Steps

- ✓ An IP address has been added to the IP Access List. Add another address in the [IP Access List](#).
- ✓ A database user has been added to this project. Create another user in [Database Users](#).

Connection String

```
mongodb+srv://SeemanshuShukla:<password>@sensor-fault-detection.ogzmdms.mongodb.net/
```



Replace **<password>** with the password for the **SeemanshuShukla** user. Ensure any option params are [URL encoded](#).

[← View connection instructions](#)

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**

The screenshot shows the MongoDB Atlas interface. On the left, there's a sidebar with 'Project 0' selected. Under 'Data Services', 'Database Access' is highlighted with a red box. In the main area, a modal window titled 'Edit User: SeemanshuShukla@admin' is open. Inside the modal, the 'Authentication Method' section is visible, with the 'Password' input field highlighted with a red box. Below it, the 'Edit Password' button is also highlighted with a red box. The background shows other tabs like 'Data Services', 'App Services', and 'Charts', along with some system status information.

- 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:

The screenshot shows the MongoDB Atlas interface for a project named 'SEEMANSHU'S ORG - 2024-04-28 > PROJECT 0'. On the left sidebar, 'Network Access' is selected and highlighted with a red box. The main page title is 'Network Access' and the sub-section is 'IP Access List'. A prominent green button at the top right of the list area is also highlighted with a red box and labeled '+ADD IP ADDRESS'. Below the button, a yellow banner states: 'You will only be able to connect to your cluster from the following list of IP Addresses:'. The table lists two entries:

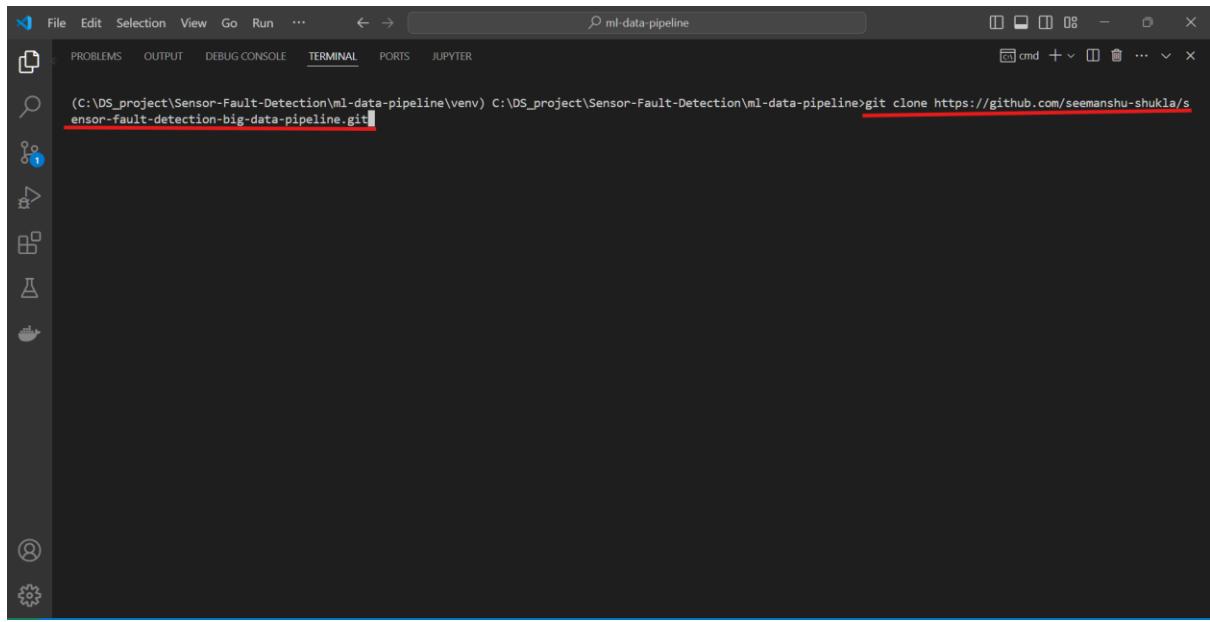
| IP Address | Comment | Status | Actions |
|---|---|--------|---|
| 0.0.0.0/0 (includes your current IP address) | | Active | <button>EDIT</button> <button>DELETE</button> |
| 223.233.81.10/32 (includes your current IP address) | Created as part of the Auto Setup process | Active | <button>EDIT</button> <button>DELETE</button> |

- **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:**

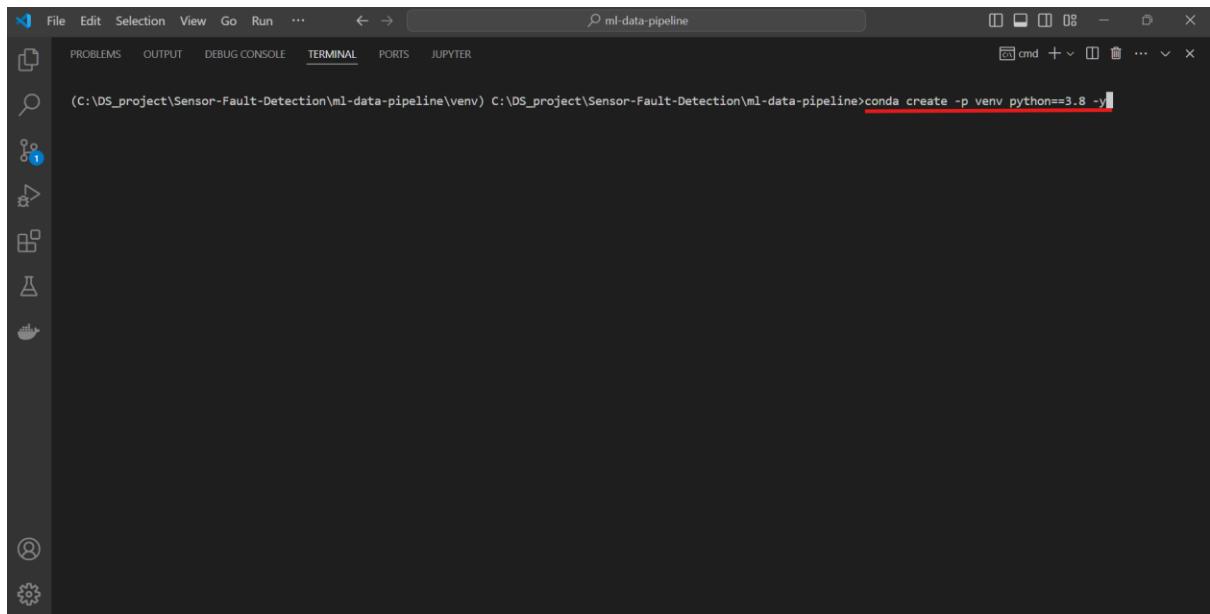
```
git clone https://github.com/seemanshu-shukla/sensor-fault-detection-big-data-pipeline.git
```



```
(C:\DS_project\Sensor-Fault-Detection\ml-data-pipeline\venv) C:\DS_project\Sensor-Fault-Detection\ml-data-pipeline>git clone https://github.com/seemanshu-shukla/sensor-fault-detection-big-data-pipeline.git
```

- **Create the virtual environment and install the pythonic dependencies inside it:**

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



```
(C:\DS_project\Sensor-Fault-Detection\ml-data-pipeline\venv) C:\DS_project\Sensor-Fault-Detection\ml-data-pipeline>conda create -p venv python==3.8 -y
```

- Use the following command to install the dependencies:

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

```

src > kafka_producer > json_producer.py > product_data_using_file
src/kafka_producer/_pycache_/_init_.py
src/kafka_producer/json_producer.py
src/venv/.dockerignore
src/.env
src/.gitignore
src/consumer_main.py
src/Dockerfile
src/producer_main.py
src/README.md
src/requirements.txt
src/schema.json
src/setup.py
src/start.sh
src/test.py
src/Untitled.ipynb

```

The screenshot shows the VS Code interface with the 'ml-data-pipeline' project open. The Explorer sidebar shows files like 'README.md', 'requirements.txt', and various Python scripts. The 'Terminal' tab shows the command 'pip install -r requirements.txt' being run. The 'requirements.txt' file is highlighted with a red box.

- **Setting the environment variables:**

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

```

API_KEY="72LFQ40CZ36PXNR6"
API_SECRET_KEY="A+sObvSfsvfco+BdD1sgvTY6xFMIMVuUjzT8JF3sHrwFfxwMaq4S6DQGK/d0A0F"
BOOTSTRAP_SERVER="pkc-6ojv2.us-west4.gcp.confluent.cloud:9092"
SCHEMA_REGISTRY_API_KEY="34NFCPBICX2M4TWY"
SCHEMA_REGISTRY_API_SECRET="icDn8iuctg+BoeZuzU5jjpU3twZxQRqpgSLe9TRUlInIM5S04CYapf29nUewLorM"
ENDPOINT_SCHEMA_URL="https://psrc-6k7pq.us-west4.gcp.confluent.cloud"
MONGO_DB_URL="mongodb+srv://"

```

The screenshot shows the VS Code interface with the 'ml-data-pipeline' project open. The Explorer sidebar shows files like 'README.md', 'requirements.txt', and various Python scripts. The '.env' file is highlighted with a red box. The content of the '.env' file is shown in the terminal, containing sensitive API keys and secrets.

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

- **Trigger the Producer script:**

- Vscode:

Trigger producer script

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under "ML-DATA-PIPELINE". The "src" folder contains "kafka_logger", "kafka_producer" (which includes __init__.py, json_producer.py, and __init__.py), "venv" (which includes .env, .gitignore, consumer_main.py, Dockerfile, and producer_main.py), requirements.txt, schema.json, setup.py, start.sh, test.py, and Untitled.ipynb.
- Preview README.md:** A preview of the README.md file is shown in the top right.
- Editor:** The "producer_main.py" file is open in the editor. The code imports from src.kafka_producer.json_producer and src.constant, and defines a main function that lists topics in SAMPLE_DIR, prints them, and then iterates over each topic to produce data using product_data_using_file.
- Terminal:** The terminal tab shows the command "python producer_main.py" entered.

Real time terminal logs while Publishing data to Kafka topics

File Edit Selection View Go ... ↶ ↷ ml-data-pipeline

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PUBLISH JUPYTER

EXPLORER

ML-DATA-Pipeline

- ipynb_checkpoints
- vscode
- assignment
- flowchart
- kafka-egg.info
- logs

sample_data/kafka-sensor-topic

- aps_failure_training.set.csv

src

- __pycache__
- constant
- database

 - __pycache__
 - mongodb.py

entity

- __pycache__
- __init__.py
- generic.py

kafka_config

kafka_consumer

- __pycache__
- __init__.py
- json_consumer.py

kafka_logger

kafka_producer

- __init__.py

venv

- dockeringuide
- .env

OUTLINE

TIMELINE

Flushing records...

```
{'class': 'neg', 'ab_000': '40222', 'ab_000': 'na', 'ac_000': '698', 'ad_000': '628', 'ae_000': '0', 'af_000': '0', 'ag_000': '0', 'ah_000': '0', 'ai_000': '0', 'aj_000': '1226', 'ak_000': '46284', 'al_000': '1981140', 'am_000': '855376', 'an_000': '6144', 'ap_000': '6318', 'ar_000': '2504090', 'as_000': '2109168', 'at_000': '573660', 'au_000': '334864', 'av_000': '0', 'aw_000': '0', 'ax_000': '0', 'ay_000': '0', 'az_000': '0', 'ba_000': '0', 'bb_000': '84', 'bd_000': '152', 'be_000': '98', 'bf_000': '2', 'bg_000': '123580', 'bh_000': '37110', 'bi_000': '310028', 'bj_000': '260378', 'bk_000': '191300', 'bl_000': '266289', 'bm_000': 'na', 'bn_000': 'na', 'bo_000': 'na', 'bp_000': 'na', 'br_000': 'na', 'bs_000': '197280', 'bt_000': '202116', 'bu_000': '3076406', 'bv_000': '3075406', 'bx_000': '3075902', 'by_000': '28954', 'bz_000': '0', 'ca_000': '61016', 'cb_000': '649540', 'cc_000': '2872088', 'cd_000': '1209600', 'ce_000': '85760', 'cf_000': '0', 'cp_000': '120', 'ch_000': '0', 'ci_000': '2480113.92', 'cj_000': '0.8', 'ck_000': '38936.08', 'cl_000': '0', 'cm_000': '0', 'cn_000': '0', 'cn_001': '0', 'cn_002': '0', 'cn_003': '32', 'cn_004': '497628', 'cn_005': '1708490', 'cn_006': '626690', 'cn_007': '30774', 'cn_008': '6542', 'cn_09': '532', 'co_000': '0', 'cp_000': '14', 'cq_000': '3076406', 'cr_000': 'na', 'cs_000': '4526', 'cs_001': '562', 'cs_002': '81282', 'cs_003': '195104', 'cs_004': '284388', 'cs_005': '2064878', 'cs_006': '233308', 'cs_007': '8882', 'cs_008': '12', 'cs_009': '0', 'ct_000': '936', 'cu_000': '728', 'cv_000': '245378', 'cx_000': '9240', 'cy_000': '18', 'cz_000': '3104', 'da_000': '0', 'db_000': '0', 'dc_000': '2506464', 'dd_000': '2116', 'de_000': '214', 'df_000': '0', 'dg_000': '0', 'dh_000': '0', 'di_000': '0', 'dj_000': '0', 'dk_000': '0', 'dl_000': '0', 'dm_000': '0', 'dn_000': '21254', 'do_000': '25634', 'dp_000': '4644', 'dq_000': '0', 'dr_000': '0', 'ds_000': '79872', 'dt_000': '14390', 'du_000': '152740', 'dv_000': '17080', 'dx_000': '0', 'dy_000': '0', 'dz_000': '0', 'ea_000': '0', 'eb_000': '39221060', 'ec_000': '183200', 'ee_000': '1035.72', 'ed_000': '1256', 'ee_000': '409788', 'ef_000': '686416', 'eg_000': '440666', 'ee_003': '183200', 'ee_004': '1035456', 'ee_005': '25488', 'ee_006': '225148', 'ee_007': '158304', 'ee_008': '170384', 'ee_009': '158', 'ef_000': '0', 'eg_000': '0'}
```

Flushing records...

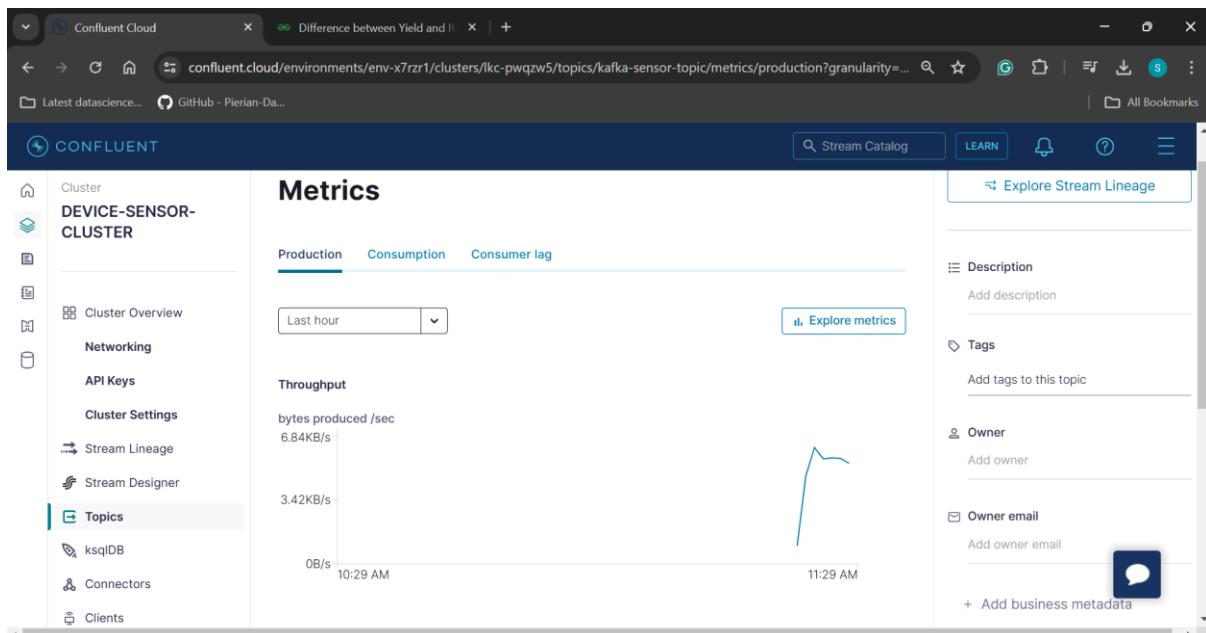
```
(c:\05_project\Sensor-Fault-Detection\ml-data-pipeline\venv) C:\05_project\Sensor-Fault-Detection\ml-data-pipeline[]
```

- 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**

The screenshot shows the Confluent Cloud interface for a Kafka topic named 'kafka-sensor-topic'. The 'Production' section displays a value of 974 bytes per second. The 'Consumption' section displays a value of 0 bytes per second. The 'Topics' menu item in the left sidebar is highlighted with a red box.

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:

The screenshot shows the Confluent Platform UI for a Kafka cluster named 'DEVICE-SENSOR-CLUSTER'. The top navigation bar includes links for Home, Environments, default, DEVICE-SENSOR-CLUSTER, and Topics. The left sidebar lists various cluster management options like Cluster Overview, Networking, API Keys, Cluster Settings, Stream Lineage, Stream Designer, and Topics. The 'Topics' section is currently selected.

The main view displays a Kafka topic named 'kafka-sensor-topic'. It features a summary card with metrics: Production in last hour (6,445 messages), Consumption in last hour (1,032 messages), Total messages (22,249), and Retention time (1 week). Below this, a search bar and dropdown filters for partitions, offset, and max results are shown. A table lists 50 messages with columns for Timestamp, Offset, Partition, Key, and Value. Each message row contains a blue circular icon with a white dot.

The screenshot shows the Confluent Platform interface. On the left sidebar, under the 'Topics' section, the 'Message details' card is highlighted with a red box. The main content area displays a table of 50 messages with columns: Timestamp, Offset, Partition, Key, and Value. The 'Value' column for the first message is also highlighted with a red box. The 'Value' for the first message is a JSON object:

```
1 {  
2   "class": "neg",  
3   "aa_000": "60702",  
4   "ab_000": "na",  
5   "ac_000": "696",  
6   "ad_000": "646",  
7   "ae_000": "0",  
8   "af_000": "0",  
9   "ag_000": "0",  
}
```

At the bottom right, there are download buttons for CSV, JSON, and a link to 'How to consume'.

The screenshot shows the Confluent Cloud UI for a Kafka cluster. On the left, there's a sidebar with various navigation options like Cluster Overview, Networking, API Keys, Cluster Settings, Stream Lineage, Stream Designer, Topics, Schema Registry, and CLI & Tools. The Topics section is currently selected. In the main area, there are two summary cards: 'Production in last hour' (6,679 messages) and 'Consumption in last hour' (1,704 messages). Below these are search and filter controls. A table lists 50 messages shown, with columns for Timestamp, Offset, Partition, Key, and Value. The 'Headers' tab in the message details panel is highlighted with a red box. The table shows several rows of message data.

- **Trigger the Consumer script:**

- Vscode:

Triggering the consumer script

The screenshot shows a VS Code interface with the title bar 'ml-data-pipeline'. The Explorer sidebar on the left shows a project structure with files like README.md, consumer_main.py, kafka_consumer.json_consumer, kafka_logger, kafka_producer, .pycache_, .env, Dockerfile, requirements.txt, schema.json, setup.py, start.sh, test.py, and Untitled.ipynb. The consumer_main.py file is selected and highlighted with a red box. The code editor shows the following Python script:

```

1  from src.kafka_consumer.json_consumer import consumer_using_sample_file
2
3  from src.constant import SAMPLE_DIR
4  import os
5  if __name__=='__main__':
6
7      topics = os.listdir(SAMPLE_DIR)
8      print(f'topics: {[topics]}')
9      for topic in topics:
10          sample_topic_data_dir = os.path.join(SAMPLE_DIR,topic)
11          sample_file_path = os.path.join(sample_topic_data_dir,os.listdir(sample_topic_data_dir))
12          consumer_using_sample_file(topic="kafka-sensor-topic",file_path = sample_file_path)

```

The terminal at the bottom shows the command being run: '(C:\DS_project\Sensor-Fault-Detection\ml-data-pipeline\venv) C:\DS_project\Sensor-Fault-Detection\ml-data-pipeline>python consumer_main.py'.

Real time terminal logs while Consuming data from Kafka topics

```

File Edit Selection View Go ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER
+ - ×
cmd powershell cmd
ML-DATA.PIPELINE
> ipynb_checkpoints
> .vscode
> assignment
> flowchart
> kafka.egg.info
> logs
> sample_data
> src
> venv
> .dockerignore
> .env
> .gitignore
consumer.main.py
Dockerfile
producer.main.py
README.md
requirements.txt
schema.json
setup.py
start.sh
test.py
Untitled.ipynb

```

```

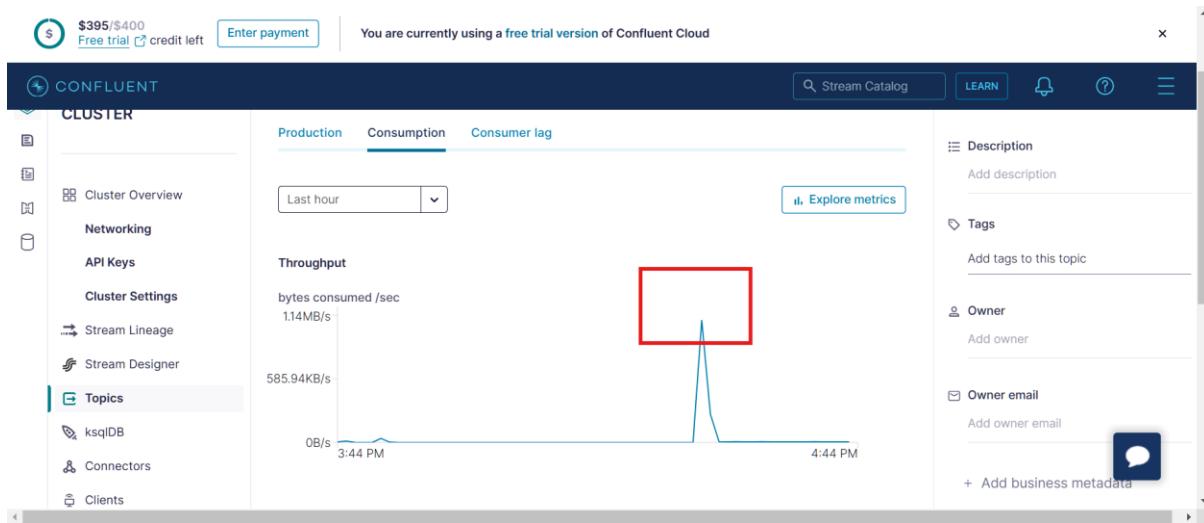
{'class': 'msg', 'aa_000': '2286', 'ab_000': 'na', 'ac_000': '230706538', 'ad_000': '224', 'ae_000': '0', 'af_000': '0', 'ag_000': '0', 'ag_001': '0', 'ag_002': '0', 'ag_003': '0', 'ag_004': '104', 'ag_005': '99186', 'ag_006': '36564', 'ag_007': '128', 'ag_008': '0', 'ag_009': '0', 'ah_000': '6982', 'ai_000': '0', 'aj_000': '0', 'ak_000': '0', 'al_000': '0', 'am_000': '0', 'an_000': '123604', 'ao_000': '108768', 'ap_000': '265640', 'aq_000': '54496', 'ar_000': '478800', 'as_000': '293652', 'at_000': '0', 'az_000': '0', 'd2_000': '0', 'ea_000': '0', 'eb_000': '88125300', 'ec_000': '2912.04', 'ed_000': '2858', 'ee_000': '1608888', 'e0_001': '1472666', 'ee_002': '998500', 'ee_003': '566884', 'ee_004': '1298398', 'ee_005': '1218244', 'ee_006': '1019768', 'ee_007': '717762', 'ee_008': '898642', 'ee_009': '28588', 'ef_000': '0', 'eg_000': '0'} <> confluent_kafka.serialization.SerializationContext object at 0x000002404f7c2b8>
{'class': 'msg', 'aa_000': '2286', 'ab_000': 'na', 'ac_000': '230706538', 'ad_000': '224', 'ae_000': '0', 'af_000': '0', 'ag_000': '0', 'ag_001': '0', 'ag_002': '0', 'ag_003': '0', 'ag_004': '104', 'ag_005': '99186', 'ag_006': '36564', 'ag_007': '128', 'ag_008': '0', 'ag_009': '0', 'ah_000': '6982', 'ai_000': '0', 'aj_000': '0', 'ak_000': '0', 'al_000': '0', 'am_000': '0', 'an_000': '123604', 'ao_000': '108768', 'ap_000': '265640', 'aq_000': '54496', 'ar_000': '478800', 'as_000': '293652', 'at_000': '0', 'az_000': '0', 'd2_000': '0', 'ea_000': '0', 'eb_000': '88125300', 'ec_000': '2912.04', 'ed_000': '2858', 'ee_000': '1608888', 'e0_001': '1472666', 'ee_002': '998500', 'ee_003': '566884', 'ee_004': '1298398', 'ee_005': '1218244', 'ee_006': '1019768', 'ee_007': '717762', 'ee_008': '898642', 'ee_009': '28588', 'ef_000': '0', 'eg_000': '0'} <> confluent_kafka.serialization.SerializationContext object at 0x000002404f7c2b8>
{'class': 'msg', 'aa_000': '2286', 'ab_000': '28', 'az_001': '34', 'az_002': '276', 'az_003': '134204', 'az_005': '174', 'az_007': '0', 'az_008': '0', 'az_009': '0', 'ba_000': '4428', 'ba_001': '69952', 'ba_002': '17376', 'ba_003': '8610', 'ba_004': '2990', 'ba_005': '1048', 'ba_006': '592', 'ba_007': '186', 'ba_008': '0', 'ba_009': '0', 'bb_000': '147638', 'bc_000': '2', 'bd_000': '14', 'be_000': '716', 'bf_000': '0', 'bg_000': '1000', 'bh_000': '1000', 'bi_000': '14404', 'bj_000': '9572', 'bk_000': 'na', 'bl_000': 'na', 'bm_000': 'na', 'bn_000': 'na', 'bo_000': 'na', 'bp_000': 'na', 'br_000': 'na', 'bs_000': '23720', 'bt_000': '2285.92', 'bu_000': '147638', 'bx_000': '147638', 'by_000': '158094', 'bz_000': '525', 'bz_000': '294', 'ca_000': '16558', 'cb_000': '147260', 'cc_000': '135982', 'cd_000': '1209500', 'ce_000': '3126', 'cf_000': '2', 'cg_000': '4', 'ch_000': '0', 'ci_000': '123344.64', 'cj_000': '0.0', 'ck_000': '11387.52', 'cl_000': '0', 'cm_000': '28', 'cn_000': '0', 'cn_001': '0', 'cn_002': '0', 'cn_003': '61572', 'cn_004': '63554', 'cn_005': '10126', 'cn_006': '602', 'cn_007': '128', 'cn_008': '0', 'cn_009': '0', 'co_000': '0', 'cp_000': '6', 'cq_000': '147638', 'cr_000': 'na', 'cs_000': '1340', 'cs_001': '32', 'cs_002': '360', 'cs_003': '9604', 'cs_004': '2112', 'cs_005': '40190', 'cs_006': '75776', 'cs_007': '3066', 'cs_008': '196', 'cs_009': '0', 'ct_000': '330', 'cu_000': '214', 'cv_000': '123730', 'cx_000': '66658', 'cy_000': '0', 'cz_000': '18', 'da_000': '0', 'db_000': '10', 'dc_000': '124152', 'dd_000': '114', 'de_000': '56', 'df_000': '0', 'dg_000': '0', 'dh_000': '0', 'di_000': '0', 'dj_000': '0', 'dm_000': '1450', 'dt_000': '510', 'du_000': '1066', 'do_000': '880', 'dp_000': '600', 'dr_000': '0', 'dr_000': '0', 'ds_000': '1450', 'dt_000': 'd', 'v_000': '82346', 'ox_000': '16440', 'oy_000': '29', 'oz_000': '0', 'eo_000': '0', 'eo_001': '128598', 'ec_000': '88.4', 'ed_000': '82', 'ee_000': '13934', 'eo_002': '15824', 'eo_003': '10578', 'eo_004': '6760', 'eo_005': '21126', 'eo_006': '68424', 'eo_007': '136', 'ee_008': '0', 'ee_009': '0', 'ef_000': '0', 'eg_000': '0'} <> confluent_kafka.serialization.SerializationContext object at 0x000002404f7c2b8>
{'class': 'msg', 'aa_000': '2286', 'ab_000': '117', 'ac_000': '0', 'ad_000': '18', 'ae_000': '0', 'af_000': '0', 'ag_000': '0', 'ag_001': '0', 'ag_002': '0', 'ag_003': '0', 'ag_004': '28', 'ag_005': '11592', 'ag_006': '11530', 'ag_007': '0', 'ag_008': '0', 'ag_009': '0', 'ah_000': '0', 'al_000': '0', 'ak_000': '0', 'al_000': '0', 'am_000': '0', 'an_000': '29688', 'ao_000': '23720', 'ap_000': '13970', 'ar_000': '0', 'as_000': '0', 'at_000': '0', 'au_000': '0', 'av_000': '36', 'aw_000': '20', 'ay_000': '0', 'ay_001': '0', 'ay_002': '0', 'ay_003': '0', 'ay_004': '0', 'ay_005': '0', 'ay_006': '0', 'ay_007': '784', 'ay_008': '22454', 'ay_009': '0', 'az_000': '782', 'az_001': '524', 'az_002': '0'} <> confluent_kafka.serialization.SerializationContext object at 0x000002404f7c2b8>

```

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.

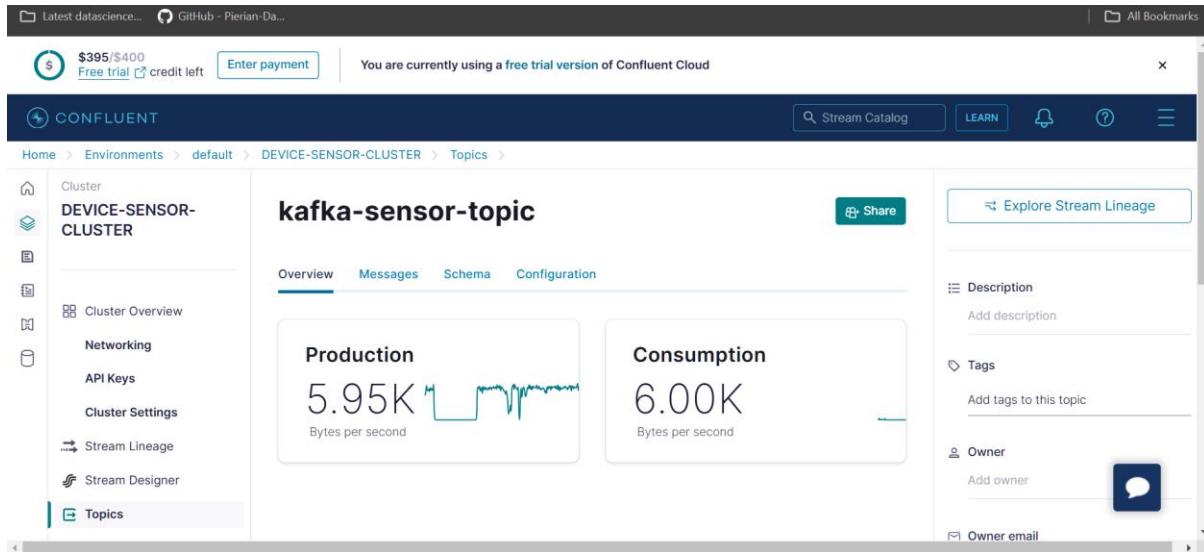
- Visualizing the Confluent Kafka dashboard during consumption of data from the Kafka topics by the consumer script:

*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.

kafka-sensor-topic

Overview Messages Schema Configuration

Production in last hour: 7,164 messages

Consumption in last hour: 30,816 messages

Total messages: 29,853 | Retention time: 1 week

Filter by timestamp, offset, key or value | All partitions | Latest | Max 50 results

50 messages shown | Auto-refresh on | CSV | JSON

Timestamp | Offset | Partition | Key | Value

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.

Metrics

Production | Consumption | Consumer lag

Last hour | Explore metrics

Throughput

bytes produced /sec
6.84KB/s
3.42KB/s

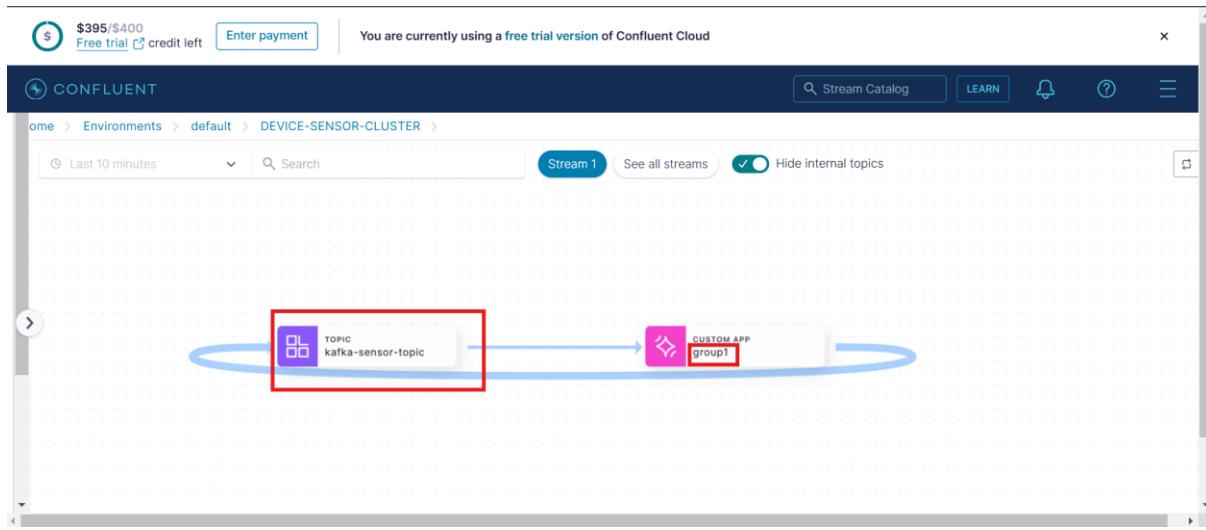
Explore Stream Lineage

Description
Add description

Tags
Add tags to this topic

Owner
Add owner

Owner email



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.

The screenshot shows the AWS Atlas MongoDB dashboard. The top navigation bar includes 'Atlas', 'Seemanshu...', 'Access Manager', 'Billing', 'All Clusters', 'Get Help', and 'Seemanshu'. The main area shows a 'Project O' cluster with a 'sensor-fault-detection' database. Under 'Collections', there is a 'ineuron.car' collection. A specific document is selected, showing its fields: '_id: ObjectId("662e2b4ca50d3db267a4e662")', 'class: "car"', 'model: "Tata Altroz"', 'ab_400: "n/a"', 'ac_400: "n/a"', 'ad_400: "n/a"', 'af_400: "n/a"', 'ag_400: "n/a"', 'ag_401: "n/a"', 'ag_402: "n/a"', 'ag_403: "n/a"', 'ag_404: "n/a"', 'ag_405: "n/a"', 'ag_406: "n/a"', 'ag_407: "n/a"', 'ag_408: "n/a"', 'ag_409: "n/a"', 'ab_400: "n/a"', 'ab_401: "n/a"', 'ab_402: "n/a"', 'ab_403: "n/a"', 'ab_404: "n/a"', 'ab_405: "n/a"', 'ab_406: "n/a"', 'ab_407: "n/a"', 'ab_408: "n/a"', 'ab_409: "n/a"', 'af_400: "n/a"', 'af_401: "n/a"', 'af_402: "n/a"', 'af_403: "n/a"', 'af_404: "n/a"', 'af_405: "n/a"', 'af_406: "n/a"', 'af_407: "n/a"', 'af_408: "n/a"', 'af_409: "n/a"', 'ag_400: "n/a"', 'ag_401: "n/a"', 'ag_402: "n/a"', 'ag_403: "n/a"', 'ag_404: "n/a"', 'ag_405: "n/a"', 'ag_406: "n/a"', 'ag_407: "n/a"', 'ag_408: "n/a"', 'ag_409: "n/a"', 'ab_400: "n/a"', 'ab_401: "n/a"', 'ab_402: "n/a"', 'ab_403: "n/a"', 'ab_404: "n/a"', 'ab_405: "n/a"', 'ab_406: "n/a"', 'ab_407: "n/a"', 'ab_408: "n/a"', 'ab_409: "n/a"', 'af_400: "n/a"', 'af_401: "n/a"', 'af_402: "n/a"', 'af_403: "n/a"', 'af_404: "n/a"', 'af_405: "n/a"', 'af_406: "n/a"', 'af_407: "n/a"', 'af_408: "n/a"', 'af_409: "n/a"', 'ag_400: "n/a"', 'ag_401: "n/a"', 'ag_402: "n/a"', 'ag_403: "n/a"', 'ag_404: "n/a"', 'ag_405: "n/a"', 'ag_406: "n/a"', 'ag_407: "n/a"', 'ag_408: "n/a"', 'ag_409: "n/a"', 'ab_400: "n/a"', 'ab_401: "n/a"', 'ab_402: "n/a"', 'ab_403: "n/a"', 'ab_404: "n/a"', 'ab_405: "n/a"', 'ab_406: "n/a"', 'ab_407: "n/a"', 'ab_408: "n/a"', 'ab_409: "n/a"', 'af_400: "n/a"', 'af_401: "n/a"', 'af_402: "n/a"', 'af_403: "n/a"', 'af_404: "n/a"', 'af_405: "n/a"', 'af_406: "n/a"', 'af_407: "n/a"', 'af_408: "n/a"', 'af_409: "n/a"', 'ag_400: "n/a"', 'ag_401: "n/a"', 'ag_402: "n/a"', 'ag_403: "n/a"', 'ag_404: "n/a"', 'ag_405: "n/a"', 'ag_406: "n/a"', 'ag_407: "n/a"', 'ag_408: "n/a"', 'ag_409: "n/a"'}. A note on the right side of the document details: 'Each document/message is uniquely identified through _id which is internally maintained by MongoDB. Also, these documents or messages are stored in key-value format where key represents the feature's name whereas, value holds the data of the respective key'. Arrows point from various labels to specific parts of the interface: 'MongoDB Cluster name' to the cluster name, 'Data base name' to the database name, 'Collection name' to the collection name, and 'Data base name' to the collection name again.

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:

The screenshot shows the Confluent Cloud Home page. At the top left, there is a red box around the account status bar which displays '\$377/\$400' and 'Free trial 23 days left'. Below this, the 'CONFLUENT' logo and a navigation menu with 'Home' selected are visible. The main content area features a 'Welcome back to Confluent Cloud!' message and a 'Your Confluent overview' section. This section includes a table titled 'Summary of your existing resources' with the following data:

| Environments | Clusters | Topics | Partitions | ksqlDB | Connectors | Cluster.links |
|---------------|----------------|--------|------------|--------|------------|---------------|
| 1 | 1 | 1 | 6 | 0 | 0 | 0 |
| Compute pools | SQL workspaces | | | | | |
| 0 | 0 | | | | | |

A red box highlights this entire table. Below this, a note says 'There's no payment information for your account' and shows two progress bars: 'Free trial' (23 days left) and 'Remaining free credit' (\$377/\$400). A red box highlights these bars. On the right, there is a sidebar with a 'Stream Catalog' search bar, 'LEARN' button, notification bell, help icon, and a three-dot menu. A callout box says 'Enter payment to avoid disruptions' with a 'Recommended' link, followed by three bullet points: '✓ You won't be charged until your free trial is over', '✓ Avoid service interruptions when your free trial ends', and '✓ Cancel or change services anytime'. A blue 'Update payment info' button is at the bottom of this sidebar.

- **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:

The screenshot shows the Confluent Cloud interface. In the top navigation bar, the 'Environments' tab is highlighted with a red box. Below the navigation bar, there's a sidebar with links like 'Home', 'Environments' (which is selected), 'Data portal', 'Stream processing', 'Cluster links', and 'Stream shares'. On the right side, there's a search bar for 'Stream Catalog', a 'LEARN' button, and a help icon. The main content area is titled 'Environments' and shows a card for the 'default' environment, which contains '1 cluster | 0 compute pools'. Below this, there's a 'Recommended' section with three cards: 'Explore with an interactive demo' (with a 'Launch demo' button), 'Set up a schema' (with a 'Get started' button), and 'Confluent Terraform provider' (with a 'Read blog' button). A red box highlights the 'default' environment card.

- **Home > Environments >:**

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

The screenshot shows the 'Clusters' tab selected in the navigation bar. The main content area displays a cluster named 'DEVICE-SENSOR-CLUSTER' under the 'Live (1)' section. This section includes a summary card with metrics: Production (0B/s), Consumption (71B/s), and Storage (106.15MB). Below this, there are sections for 'Resources' (ksqldb: 0, Connectors: 0, Clients: 1) and 'Overview' (ID: lkc-pwqzw5, Type: Basic, Provider & region: GCP | us-west4). To the right of the cluster card, there's a detailed sidebar for the 'default' environment, which lists metadata: ID: env-x7r1, Description (Add description), Tags (Add tags to this environment), Date created (Apr. 27 2024 7:21 AM), and Date modified (Apr. 27 2024 7:21 AM). It also shows Stream Governance package information (Essentials, Google Cloud Platform | us-west4, Upgrade now), and links for 'Schemas', 'Tags', and 'Business metadata'. A red box highlights the cluster card and the detailed sidebar.

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:

- [Home > Environments > default >](#)

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:

The screenshot shows the Confluent Schema Registry interface. The left sidebar has links for Home, Environments, Data portal, Stream processing (New), Cluster links, and Stream shares. The breadcrumb path 'Home > Environments > default >' is highlighted with a red box. The top navigation bar includes 'Stream Catalog', 'LEARN', a bell icon, and a help icon. The main area is titled 'default'. It shows 'Compatibility mode: BACKWARD' and a search bar 'Search by schema subject name'. A table lists schemas with columns 'Subject', 'Schema type', and 'Current version'. The row for 'kafka-sensor-topic-value' is highlighted with a red box. In the top right, there's a '+ Add schema' button.

- [Home > Environments > default > Schemas >](#):

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

The screenshot shows the schema details for 'kafka-sensor-topic-value'. The top navigation bar and breadcrumb path are identical to the previous screenshot. The schema details include 'Type: JSON Schema', 'Compatibility mode: Backward', 'Used by topic: kafka-sensor-topic', and 'Schema ID: 100001'. The 'Schema tree' section shows the schema structure: object with properties like type: object, additionalProperties: false, \$id, \$schema, title: SampleRecord, and description: Sample schema to help you get started. To the right, there are sections for 'Description', 'Tags', 'Schema doc', and 'Date created' (Apr. 28 2024 11:22 AM). A blue arrow points from the 'Schema tree' section to the schema structure.

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

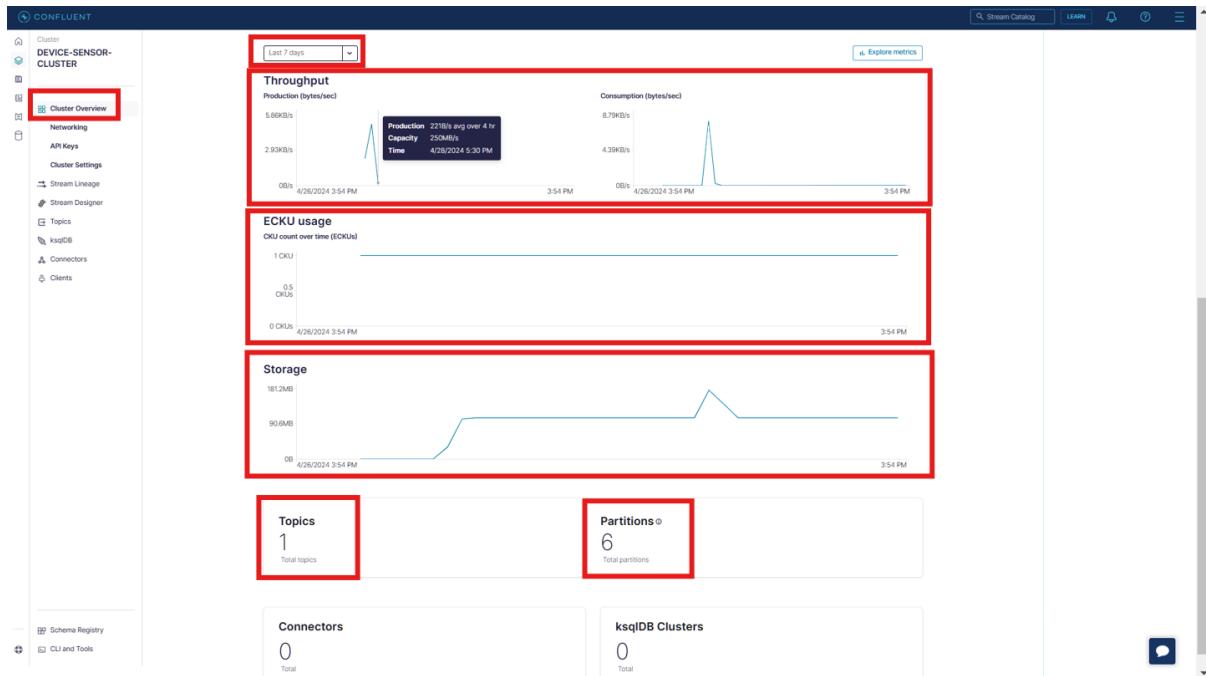
The screenshot shows the Confluent Schema Registry interface. The top navigation bar includes 'Stream Catalog', 'LEARN', 'Stream processing', and other options. The main page title is 'kafka-sensor-topic-value'. It displays a JSON schema with a red box highlighting the code area. The schema is a complex object with nested properties and descriptions. To the right of the schema, there are sections for 'Description' (with 'Add description'), 'Tags' (with 'Add tags to this version'), 'Schema doc' (with 'Sample schema to help you get started'), and 'Date created' (Apr. 28 2024 11:22 AM). A blue button labeled 'Evolve schema' is also visible.

- **Home > Environments > default > DEVICE-SENSOR-CLUSTER > :**

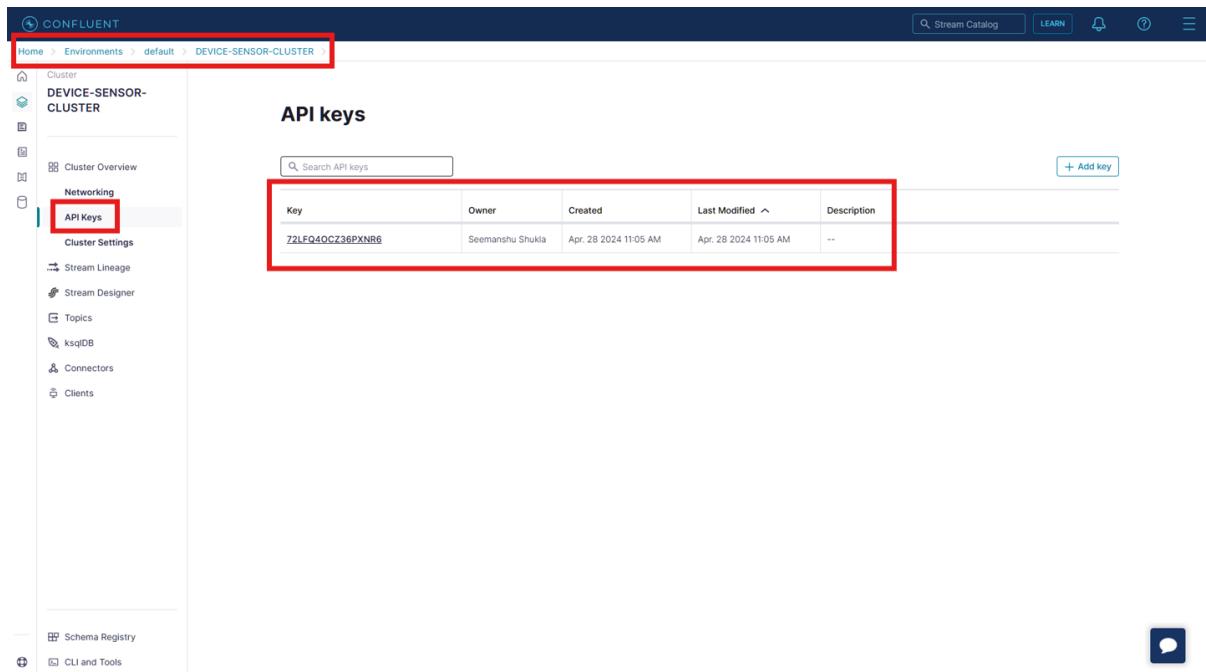
- Below snip gives us the complete overview our cluster named as **DEVICE-SENSOR-CLUSTER**.

The screenshot shows the Confluent Cluster Overview page for 'DEVICE-SENSOR-CLUSTER'. The left sidebar lists various cluster management options like 'Networking', 'API Keys', 'Cluster Settings', etc. The main 'Overview' section features a 'Complete your production checklist' with four items: 'Real Time Alerts', 'Expert Support', 'Data Integrity', and 'Highest SLA'. Below this, there's a section for 'Set up connector' and 'Produce sample data'. On the right, a large red box highlights the 'Cluster Overview' details, which include fields like 'Cluster ID' (lkc-pwqzw5), 'Cluster type' (Basic), 'Date created' (Apr. 27 2024 12:13 PM), 'Date modified' (Apr. 27 2024 12:13 PM), 'Cloud provider' (GCP), 'Cloud region' (US-West4), and 'Uptime SLA' (99.5%). Red arrows point to each of these highlighted fields.

- 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:

Cluster settings

General Capacity

Identification

Name: DEVICE-SENSOR-CLUSTER
Cluster ID: lkc-pwqzw5

Cluster details

Basic

Provider: GCP
Region: us-west4
Uptime SLA: 99.5%

Enhance your experience
Upgrade to higher SLA
Upgrade to a Standard cluster for production-ready features.
See details

Endpoints

Bootstrap server: pkc-6ojv2.us-west4.gcp.confluent.cloud:9092
REST endpoint: https://pkc-6ojv2.us-west4.gcp.confluent.cloud:443

⚠ Use the Kafka REST API to interact with your cluster and produce records.

Delete cluster

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

Cluster settings

General Capacity

Fixed limits

| | | |
|------------|-------------|----------------|
| Ingress | Last 7 days | up to 250 MB/s |
| Egress | Last 7 days | up to 750 MB/s |
| Storage | 6 | up to 5 TB |
| Partitions | 6 | up to 4,096 |

- 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:

The screenshot shows the Confluent Platform interface. The left sidebar has a tree view with 'DEVICE-SENSOR-CLUSTER' selected. Under 'Topics', there is a table with one row: 'Topic name: kafka-sensor-topic, Tags: --, Partitions: 6, Production: --, Consumption: 0B/s, Retained bytes: 106.15MB, Consumers: --, Schema: Edit schema'. A red box highlights this table.

- [Home > Environments > default > DEVICE-SENSOR-CLUSTER > Topics >](#):

- This section is divided into 4 parts: **Overview, Messages, Schema, Configuration**.
- **Part 1st Overview:**

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.

The screenshot shows the 'kafka-sensor-topic' details page. The left sidebar has a tree view with 'DEVICE-SENSOR-CLUSTER' selected. Under 'Topics', the 'Overview' tab is active. The 'Production' section shows '0 Bytes per second'. The 'Consumption' section shows '0 Bytes per second'. On the right, there is a sidebar with fields: 'Description' (Add description), 'Tags' (Add tags to this topic), 'Owner' (Add owner), 'Owner email' (Add owner email), '+ Add business metadata', 'Date created' (Apr. 27 2024 12:28 PM), 'Date modified' (Apr. 27 2024 12:28 PM), 'Retention time' (1 week), 'Retention size' (Infinite), and 'Number of partitions' (6). Red arrows point to the 'Date created', 'Retention time', and 'Number of partitions' fields.

- **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.

The screenshot shows the Confluent Platform interface for the 'DEVICE-SENSOR-CLUSTER'. The left sidebar is collapsed. The main area displays the 'kafka-sensor-topic' overview. The 'Messages' tab is active. Key statistics are highlighted: Production in last hour (63 messages), Consumption in last hour (63 messages), Total messages (36,187), and Retention time (1 week). Below the stats are search and filter controls. A table lists 50 messages with columns for Timestamp, Offset, Partition, Key, and Value. The Value column contains JSON data representing sensor measurements.

- **Part 3rd Schema:**

The screenshot shows the Confluent Platform interface for the 'DEVICE-SENSOR-CLUSTER'. The left sidebar is collapsed. The main area displays the 'kafka-sensor-topic' schema. The 'Schema' tab is active. It shows a JSON schema for 'Value' with fields like type: object, additionalProperties: false, and properties. The 'Key' tab is also visible. To the right, there are sections for Description, Tags, Owner, and metadata like Date created and Retention time.

- **Part 4th Configuration:**

The screenshot shows the Confluent Platform interface for managing a Kafka topic. The left sidebar shows the environment 'DEVICE-SENSOR-CLUSTER' and the 'Topics' section, which is highlighted with a red box. The main area displays the configuration for the 'kafka-sensor-topic'. A large red box surrounds the 'General settings' table, which lists various Kafka configuration properties like 'name', 'partitions', 'cleanup.policy', etc. To the right of the configuration table is a panel for topic metadata, also enclosed in a red box. This panel includes fields for 'Description', 'Tags', 'Owner', 'Owner email', and various timestamp and retention settings.

- **Home > Environments > default > DEVICE-SENSOR-CLUSTER > Topics > kafka-sensor-topic >:**

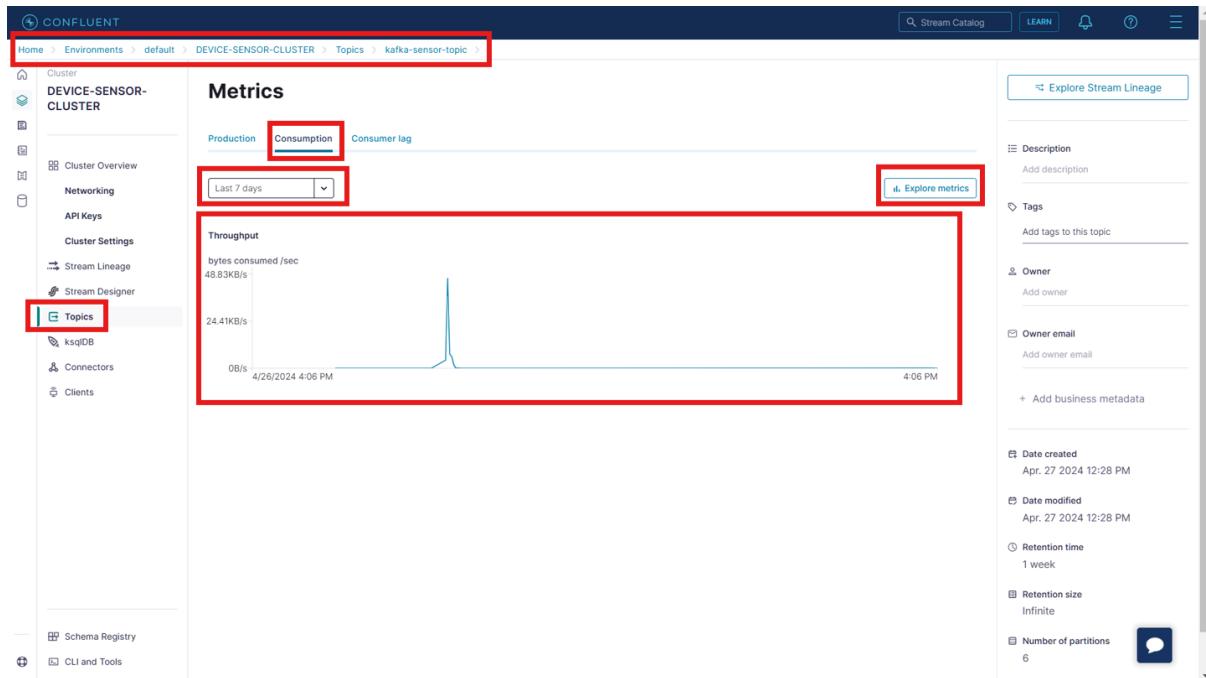
- **Part 1st Production:**

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

The screenshot shows the 'Metrics' tab for the 'kafka-sensor-topic' topic. The left sidebar shows the environment 'DEVICE-SENSOR-CLUSTER' and the 'Topics' section, which is highlighted with a red box. The main area displays production metrics. A red box highlights the 'Production' tab. Below it, a dropdown menu for selecting time intervals (e.g., 'Last 7 days') is also highlighted with a red box. A callout box points to a specific data point on a throughput chart, indicating a value of '5.83KB/s' at the time '4/28/2024 12:00 PM'. To the right of the metrics chart is a panel for topic metadata, also enclosed in a red box. This panel includes fields for 'Description', 'Tags', 'Owner', 'Owner email', and various timestamp and retention settings.

- **Part 2nd Consumption:**

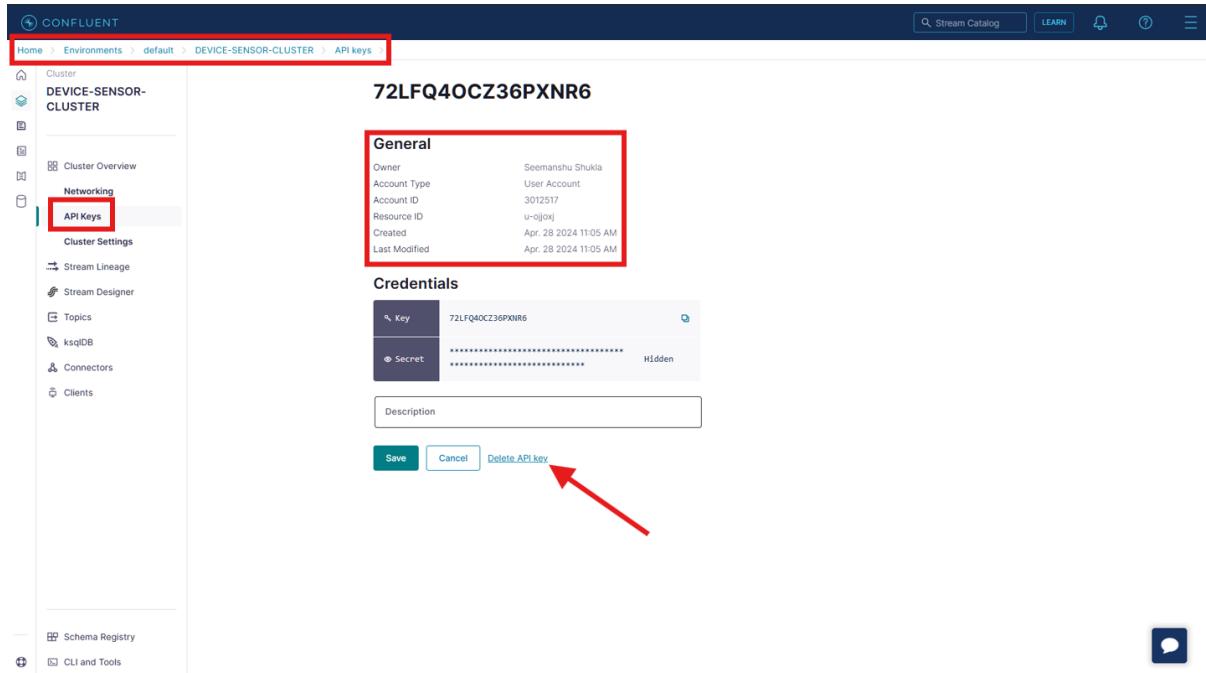
Here we can explore the different metrics related to the Consumption component.



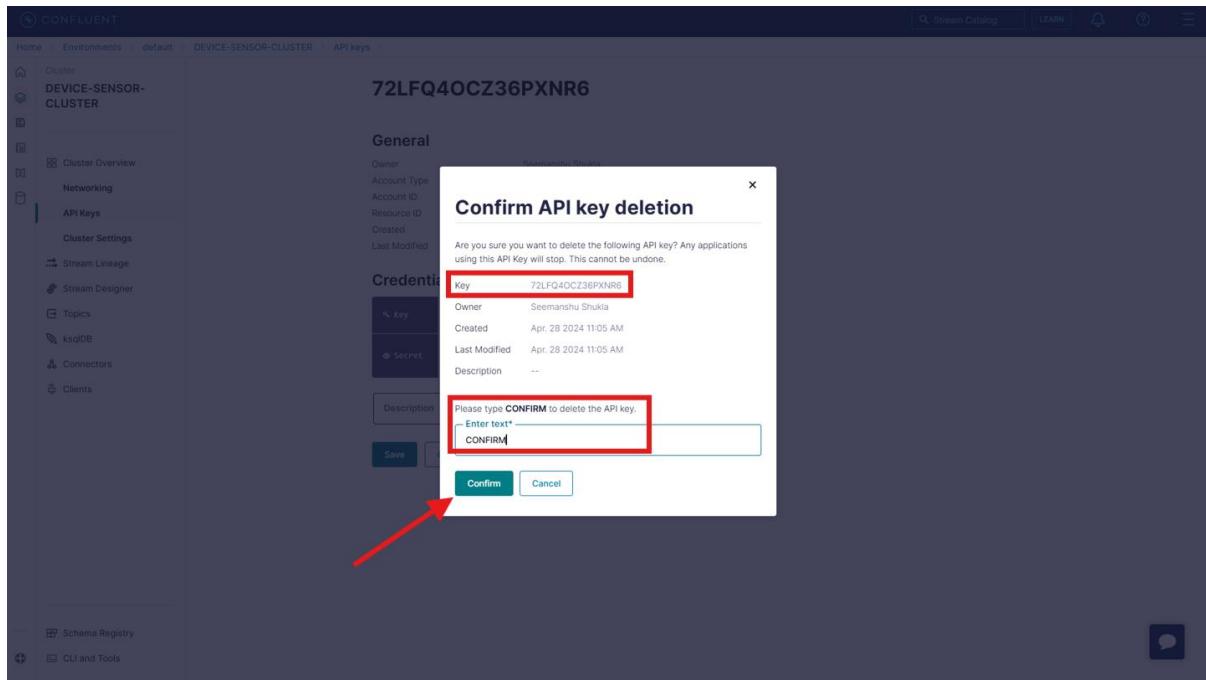
→ **Deleting the Kafka resources to save the cost:**

- **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:



- **Schema Registry API key deletion:**

- Traverse inside **default environment** and click **key** button under **Credentials:**

| | |
|---|-------|
| <input type="radio"/> Schema Registry API key | 1 key |
|---|-------|

<https://confluent.cloud/environments/env-x7zr1/schema-registry/schemas>

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

The screenshot shows the Confluent platform interface. In the top navigation bar, 'Environments' is highlighted with a red box. On the left sidebar, 'Environments' is also highlighted with a red box. The main content area is titled 'API credentials'. A table lists an API key: '34NFCPBICX2M4TWY' (Owner: Seemanshu Shukla, Created: Apr. 28 2024 11:08 AM, Last Modified: Apr. 28 2024 11:08 AM). The entire row for this key is highlighted with a red box. At the top right of the table, there are three buttons: a trash icon, 'Edit description', and '+ Add key'. A red arrow points from the bottom right towards the trash icon.

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

The screenshot shows a 'Confirm API key deletion' dialog box. It contains a message: 'Are you sure you want to delete the following API key? Any applications using this API Key will stop. This cannot be undone.' Below this is a table with the same API key details as the previous screenshot. At the bottom of the dialog, there is a text input field with placeholder text 'Please type CONFIRM to delete the API key.' and a red box around it. The word 'CONFIRM' is typed into this field. There are two buttons at the bottom: 'Confirm' (highlighted with a red box) and 'Cancel'.

- **Deleting the Kafka topic:**

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

The screenshot shows the Confluent Platform interface for managing Kafka topics. The left sidebar is expanded, showing sections like Cluster Overview, Networking, API Keys, Cluster Settings, Stream Lineage, Stream Designer, and Topics. The 'Topics' section is highlighted with a red box. The main content area is titled 'kafka-sensor-topic' and shows the 'Configuration' tab selected. Under 'General settings', there are fields for name (kafka-sensor-topic), partitions (6), cleanup.policy (delete), max.message.bytes (2097164), retention.bytes (-1), and retention.ms (604800000). Below these are 'Edit settings' and 'Show full config' buttons. To the right of the configuration panel is a sidebar with various topic metadata: Description, Tags, Owner, Owner email, Date created, Date modified, Retention time, Retention size, and Number of partitions (6). A red arrow points from the 'Delete topic' button in the configuration panel towards the sidebar.

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

This screenshot shows the 'Confirm marked for deletion' dialog box that appears after clicking the 'Delete topic' button in the previous screen. The dialog asks if you're sure you want to mark the topic for deletion. It contains a text input field with 'kafka-sensor-topic' and two buttons at the bottom: 'Continue' (highlighted with a red box) and 'Cancel'. The background shows the same topic configuration page as before, with the 'Delete topic' button now highlighted by a red arrow.

- **Delete schema registry:**

- Traverse to Schemas section inside default environment. Click on the 3 dot icon and click the **Delete** button:

The screenshot shows the Confluent Schema Registry interface. The left sidebar has a 'Environments' menu item highlighted with a red box. The main content area displays a schema named 'kafka-sensor-topic-value'. On the right, there is a context menu with options: 'Download', 'Duplicate', 'Delete', and 'Compare versions'. A red arrow points to the 'Delete' option.

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

The screenshot shows a 'Deletion' confirmation dialog box. It asks what to delete, mentioning a soft delete operation. It provides two options: 'Delete schema (version 1)' and 'Delete the entire subject including all versions'. The second option is selected and highlighted with a red box. At the bottom, there are 'Cancel' and 'Delete' buttons, with the 'Delete' button also highlighted with a red box.

- **Delete the Kafka Cluster:**

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

The screenshot shows the Confluent Cloud interface. In the top left, there's a navigation bar with 'CONFLUENT' and links for 'Home', 'Environments', 'Data portal', 'Stream processing', 'Cluster links', and 'Stream shares'. Below this is a sidebar with 'Environments' highlighted. The main area is titled 'default' and shows a 'Live (1)' section with a card for 'DEVICE-SENSOR-CLUSTER'. The card displays metrics like Production (0B/s), Consumption (43B/s), and Storage (106.15MB). It also shows resources (ksqldb, Connectors, Clients) and an overview with ID, Type, and Provider & region details. On the right, there's a sidebar with 'default' environment details, Stream Governance package (Essentials, Google Cloud Platform | us-west4), and Stream Governance API (Endpoint: https://psrc-6k7pq.us-west4.gcp.confluent.cloud). A red box highlights the 'Clusters' tab in the top navigation bar, and another red box highlights the 'Environments' link in the sidebar. A red arrow points from the 'Delete cluster' button on the DEVICE-SENSOR-CLUSTER card to the 'Delete cluster' button in the confirmation dialog.

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

The screenshot shows the Confluent Cloud interface with the 'Clusters' tab selected. A red box highlights the 'Clusters' tab in the top navigation bar. A red box highlights the 'DEVICE-SENSOR-CLUSTER' card in the 'Live (1)' section. A red box highlights the input field in the 'Confirm deletion' dialog where 'DEVICE-SENSOR-CLUSTER' is typed. A red arrow points from the 'Continue' button in the dialog to the 'Continue' button in the confirmation dialog.

- **Deleting the environment:**

- 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! 😊