Interactive Streaming Infra with JupyterHub and Apache Flink on Kubernetes

Elkhan Dadashov, Software Engineer, Apple

Agenda

- Streaming adoption challenges
  - Steep learning curve
  - Infrastructure
  - Language
- Interactive Streaming Infra
  - JupyterHub
  - Single pod development
  - At large scale development

Steep learning curve

- Challenges:
  - Bounded vs unbounded data (reading large Iceberg tables as a source)
  - Different time characteristics
  - Out-of-order data
- Solution:
  - Lots of examples with different connectors

Steep learning curve

- Challenges:
  - Connecter specific knowledge (Kafka idleness, Iceberg commits)
- Solution:
  - SDK & connectors (Kafka, Schema Registry, Iceberg)
  - Good defaults
  - Extra documentation

Steep learning curve

- Challenges:
  - Too many knobs to tune for Flink pipeline (memory, task slots, Python, Java, Flink SQL)
- Solution:
  - Flink Kubernetes Autoscaler
    - Sources: Kafka source, Iceberg Source
    - Tuning parallelism & memory

Infrastructure challenges

Running on Clouds

- Challenge:
  - Handling pod/task failures
- Solution:
  - Flink Kubernetes Operator

Running on Clouds

- Challenge:
  - Object storage limits on PUTs
- Solution:
  - Cooperation with vendors on entropy-based partitioning

Running on Clouds

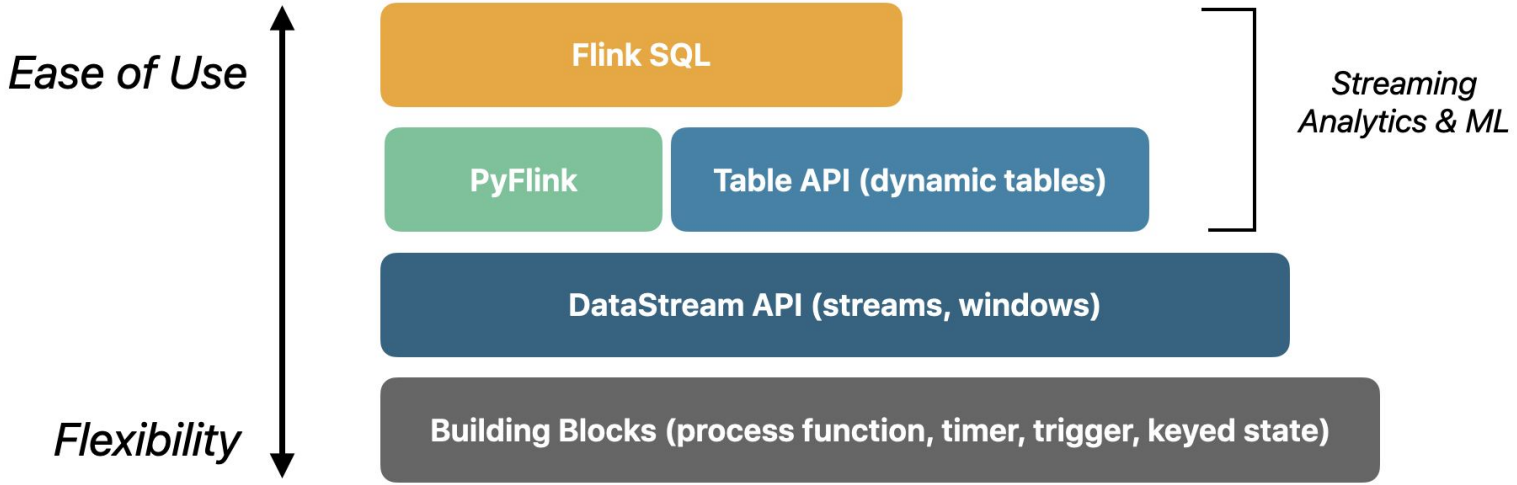
- Challenge:
  - Network costs
- Solution:
  - Co-locating Kafka and Flink in the same region/AZ

Language challenges

Language Challenges

- Challenge:
  - ML and Data science people use Python and SQL
  - Data engineers use Java/Scala
- Solution:
  - Support of different language APIs: Java, Flink SQL, PyFlink

Flink APIS for Application Development



Flink SQL

- Easier to use: what vs how
- NoCode: Well-known SQL standard: ANSI SQL + Temporal operators
- No additional cost for Infra or users
- Attract new users: ML/DS engineers/analysts

PyFlink

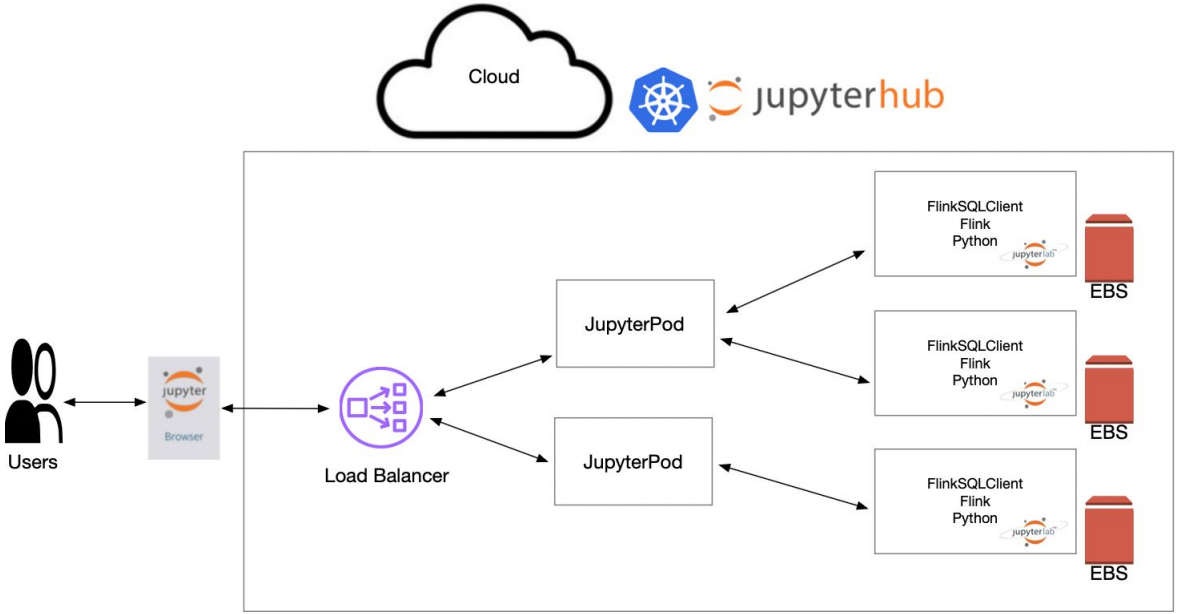
- PyFlink fully supports both the Table API and the DataStream API.
- Simplifies Flink SQL/Flink pipeline testing
- Python is one of the most popular language in data science and machine learning
- Pandas (powerful Python module) can be used in PyFlink also

Interactive Streaming Development

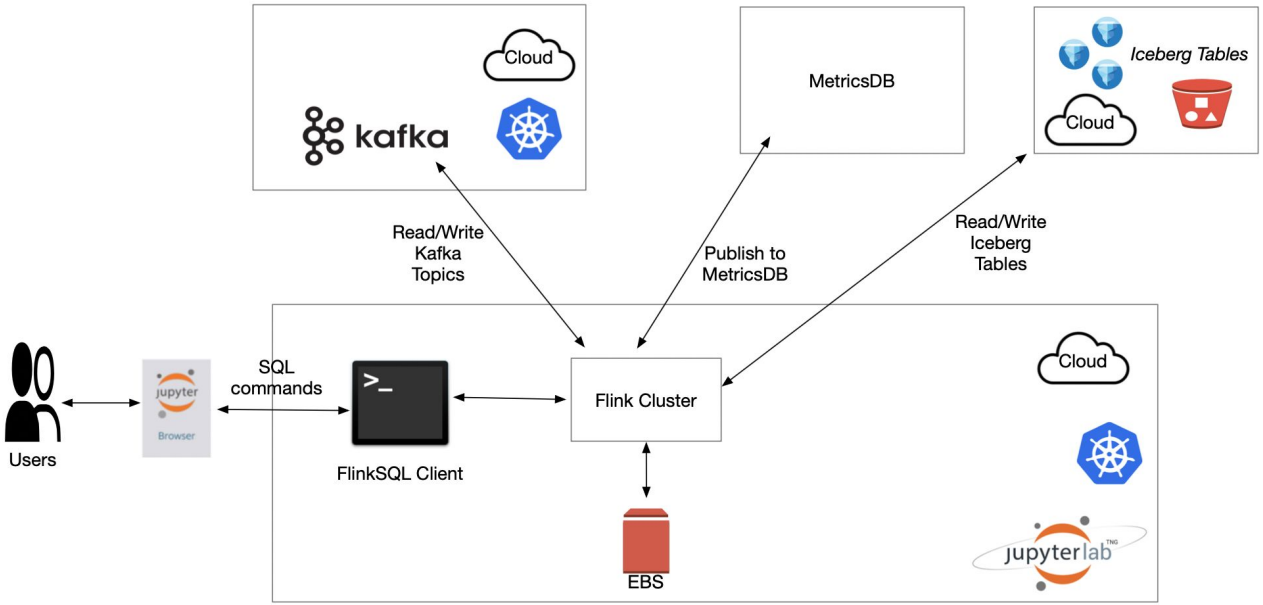
Interactive Streaming Development

- Use Flink SQL client
- Use Python notebooks

Shell on the Clouds: JupyterHub



Flink baked JupyterLab image

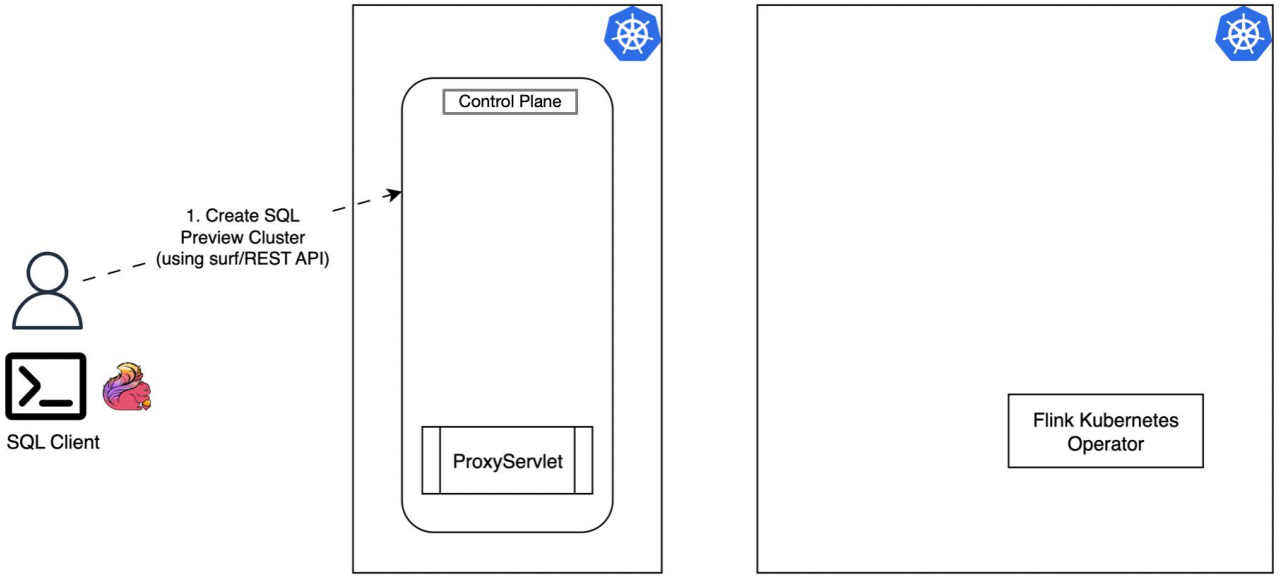


Challenges on single pod

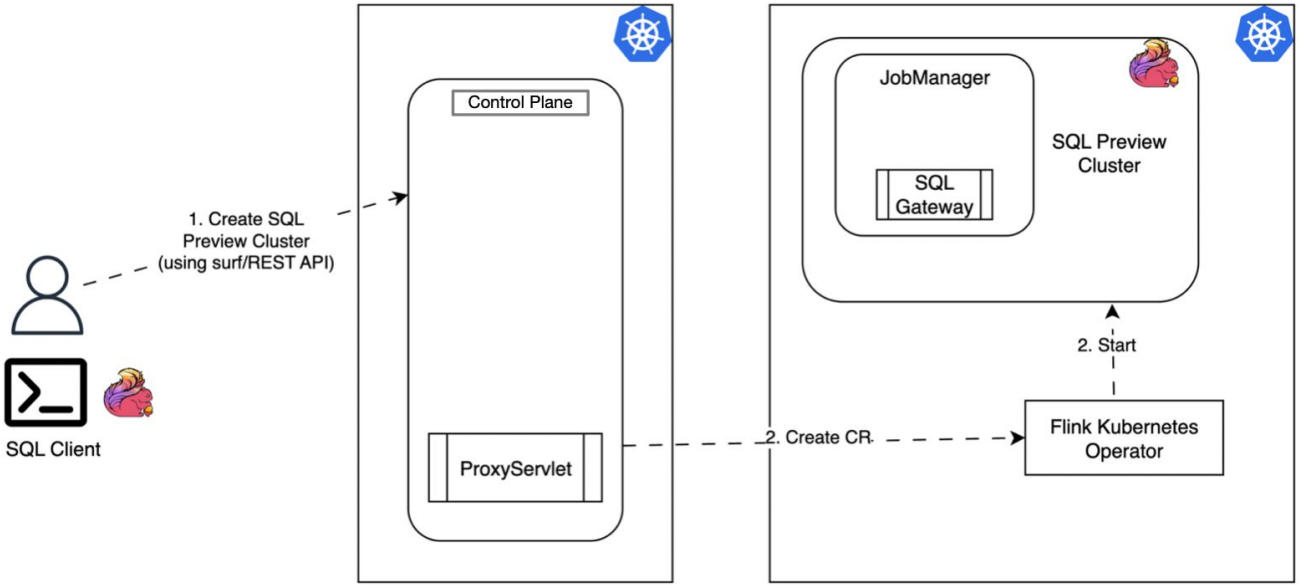
- Challenge:
  - Cannot run at large scale (reading large Iceberg table)
- Solution:
  - Remote execution of Flink pipeline for both Flink SQL & PyFlink

Flink SQL Interactive Development Workflow

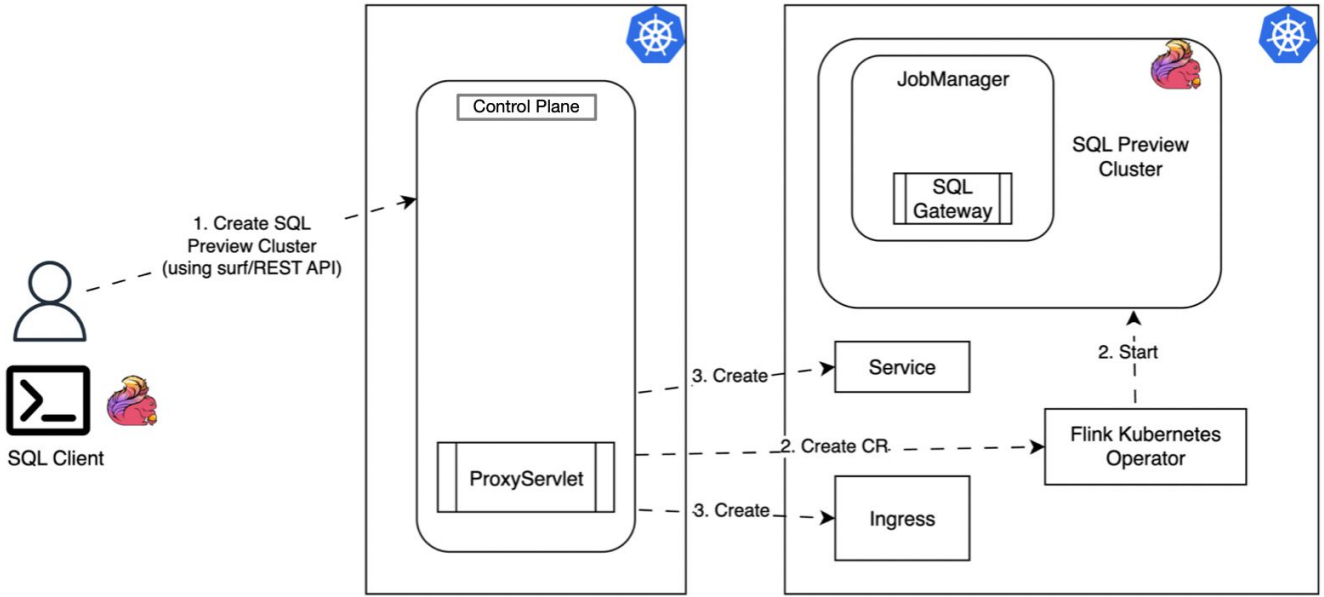
Session Cluster Initialization



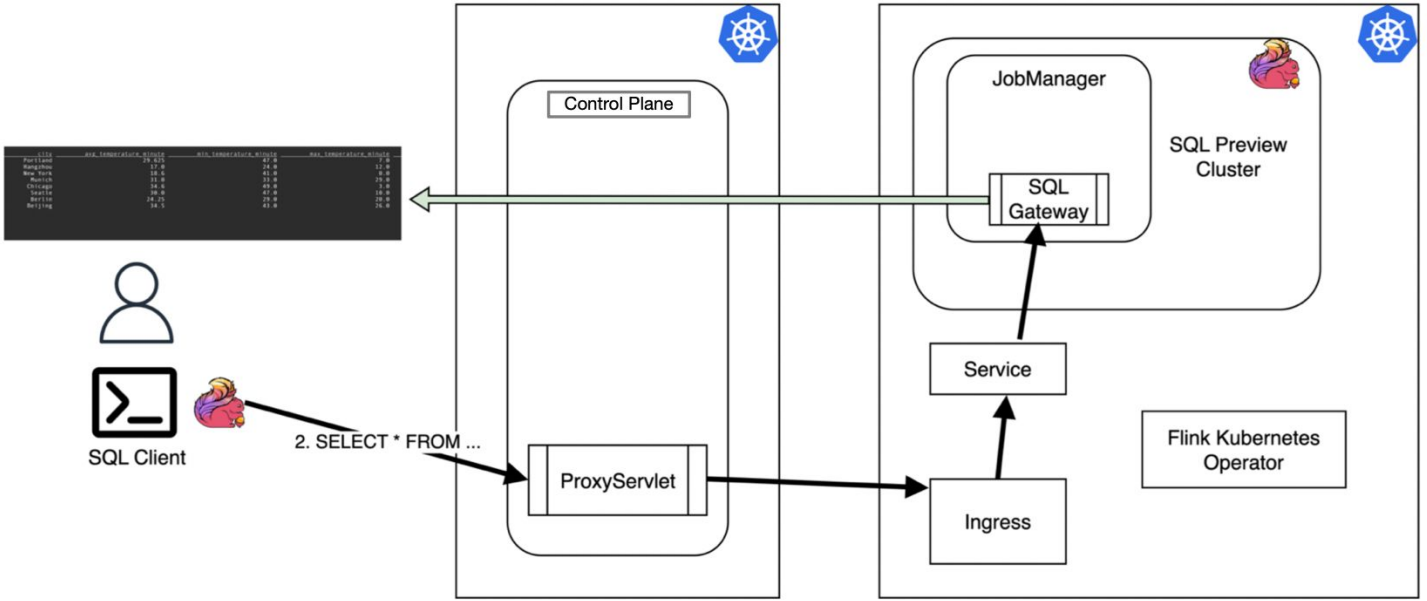
Session Cluster Initialization



Session Cluster Initialization

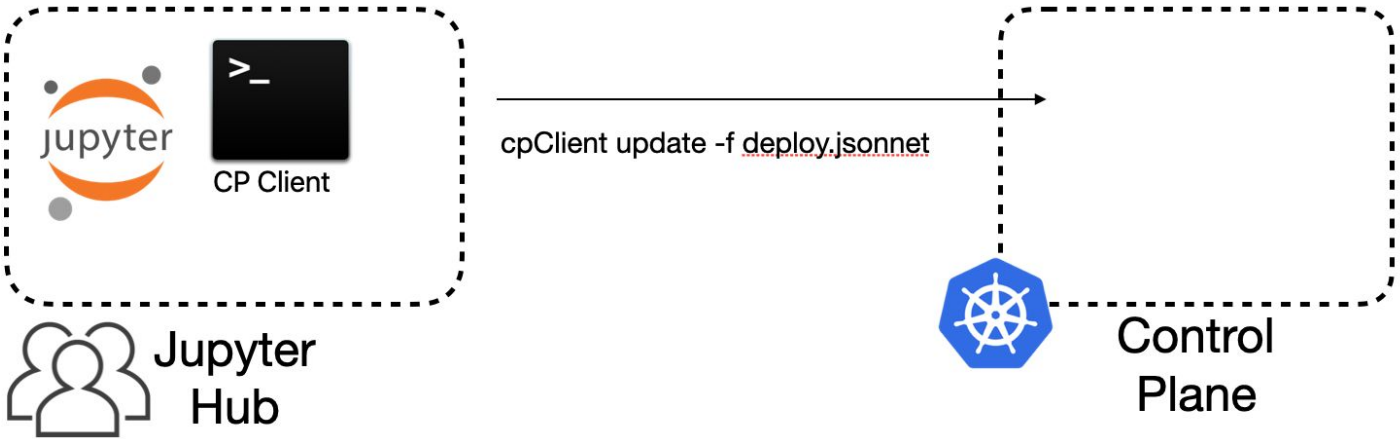


Interactive Flink SQL

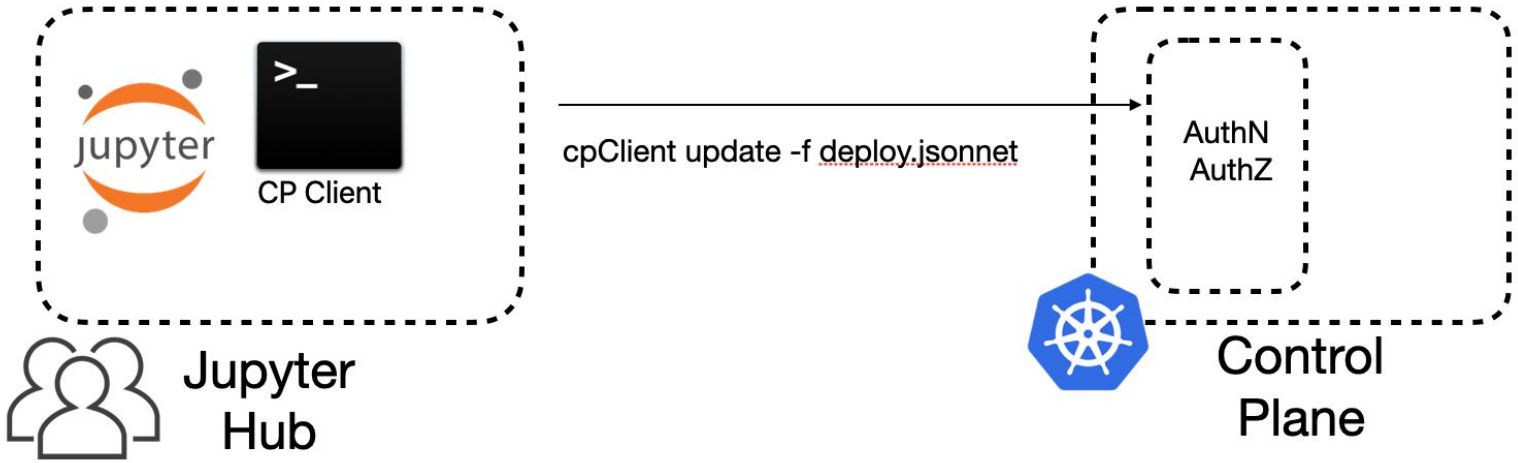


PyFlink Interactive Development Workflow

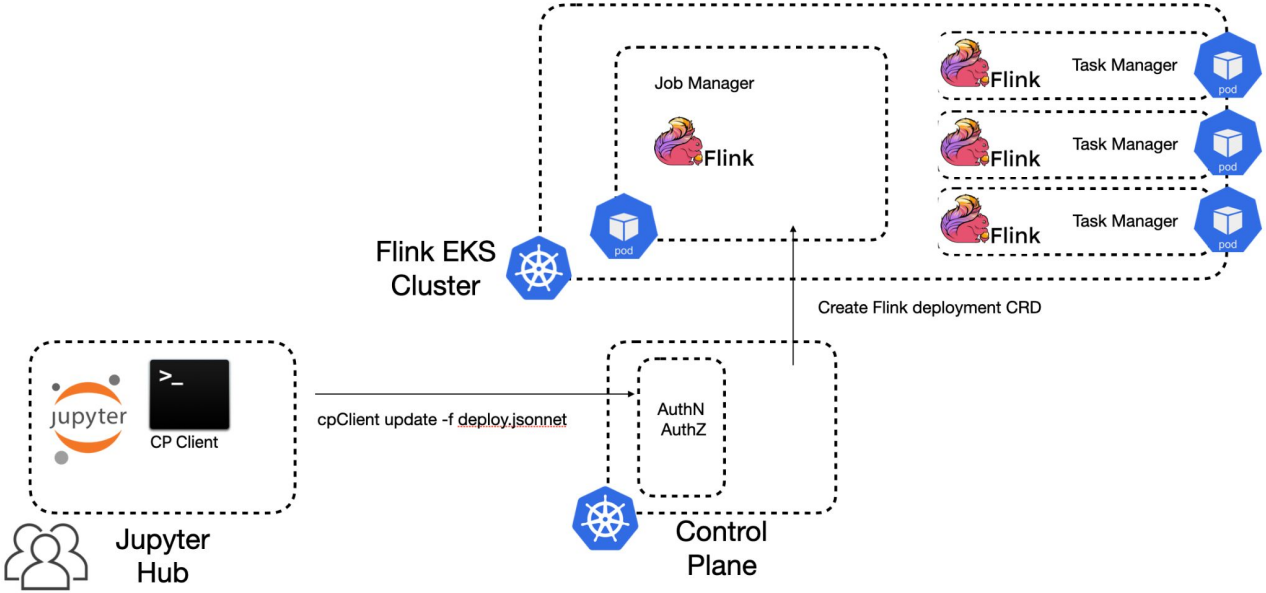
Step 1: Create Standalone cluster on remote Flink EKS cluster



Step 1: Create Standalone cluster on remote Flink EKS cluster



Step 1: Create Standalone cluster on remote Flink EKS cluster



Step 2: PyFlink job submission

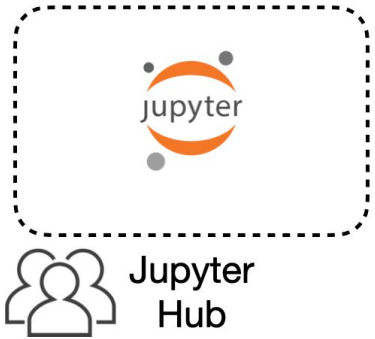


Table API PyFlink Examples for Apache Flink

```
[2]:
import argparse
import json
import logging
import random
import sys

import numpy as np
import pandas as pd
from pyflink.common import Row, Types
from pyflink.common.time import Instant
from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import (
    DataTypes,
    EnvironmentSettings,
    FormatDescriptor,
    Schema,
    StreamTableEnvironment,
    TableDescriptor,
    TableEnvironment,
)
from pyflink.table.expressions import col, lit
from pyflink.table.udf import udaf, udf, udtf
from pyflink.table.window import Tumble
```

```
[3]:
env = StreamExecutionEnvironment.get_execution_environment()
env.set_parallelism(1)
t_env = StreamTableEnvironment.create(stream_execution_environment=env)

# start a checkpoint every 1000 ms
env.enable_checkpointing(1000)

# advanced options:

# # make sure 500 ms of progress happen between checkpoints
env.get_checkpoint_config().set_min_pause_between_checkpoints(500)

# # checkpoints have to complete within one minute, or are discarded
env.get_checkpoint_config().set_checkpoint_timeout(60000)

# # allow only one checkpoint to be in progress at the same time
env.get_checkpoint_config().set_max_concurrent_checkpoints(1)

# # enables the unaligned checkpoints
env.get_checkpoint_config().enable_unaligned_checkpoints()

t_env.get_config().set("parallelism.default", "1")
```

```
[3]: <pyflink.table.table_config.TableConfig at 0x7fe842d6c310>
```

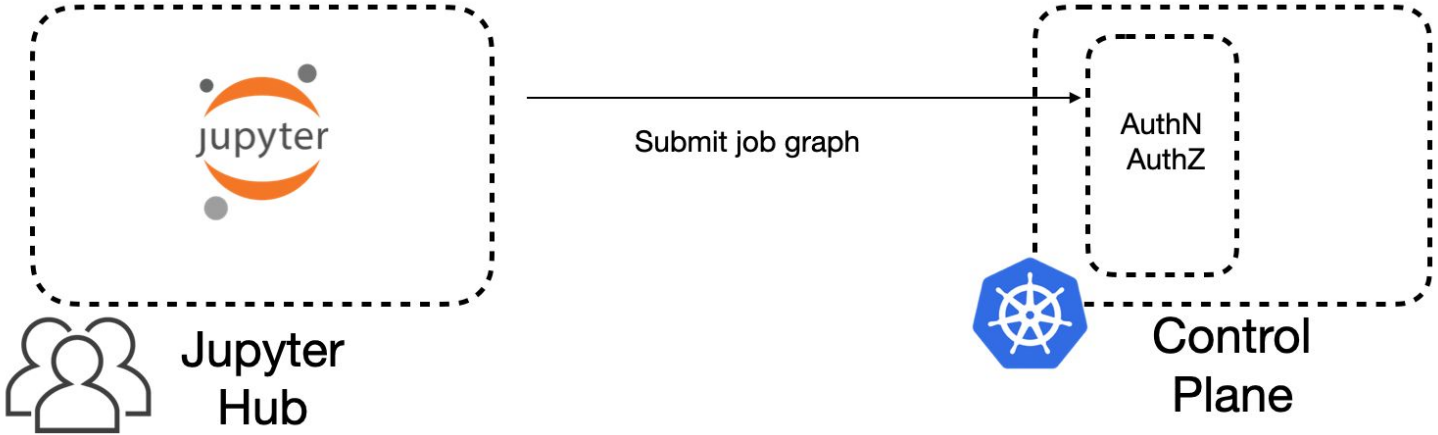
Lab: Ingest data into Iceberg from Kafka

- Create local table with `datagen` connector (controlled random data)
- Create Kafka topic and ingest data into Kafka topic from local table
- Create Iceberg table and ignest data into Iceberg table from Kafka topic

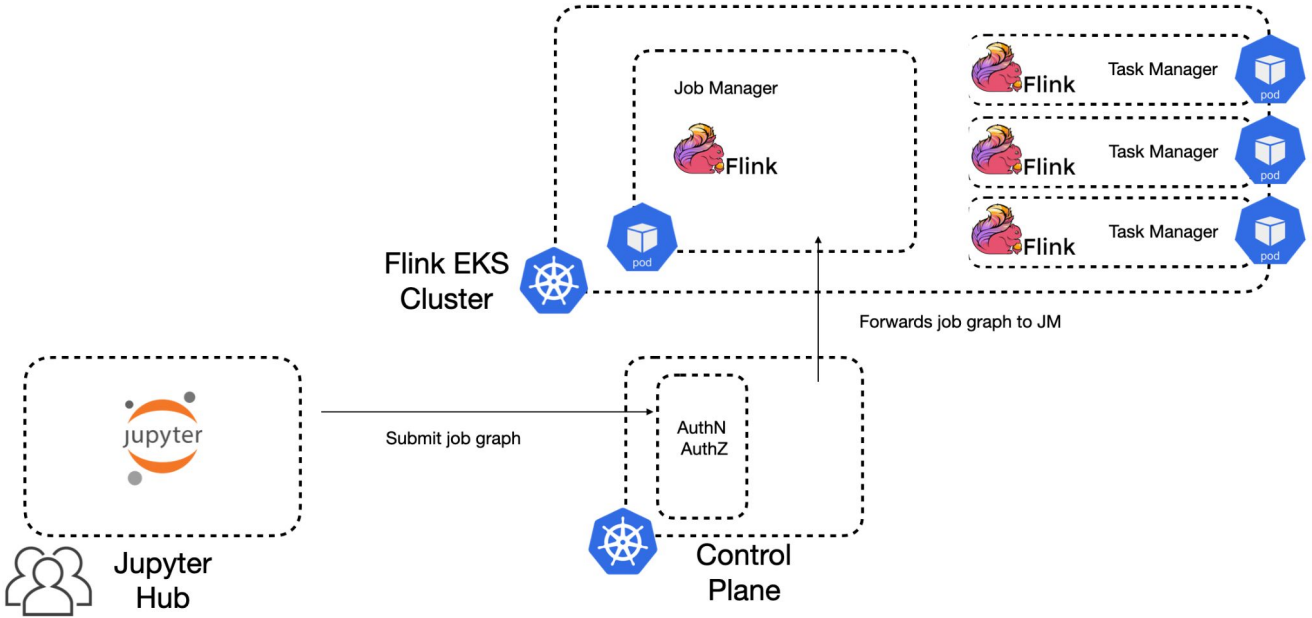
Create `engagement_datagen_table` table with datagen connector for random data generation

```
[4]:
```

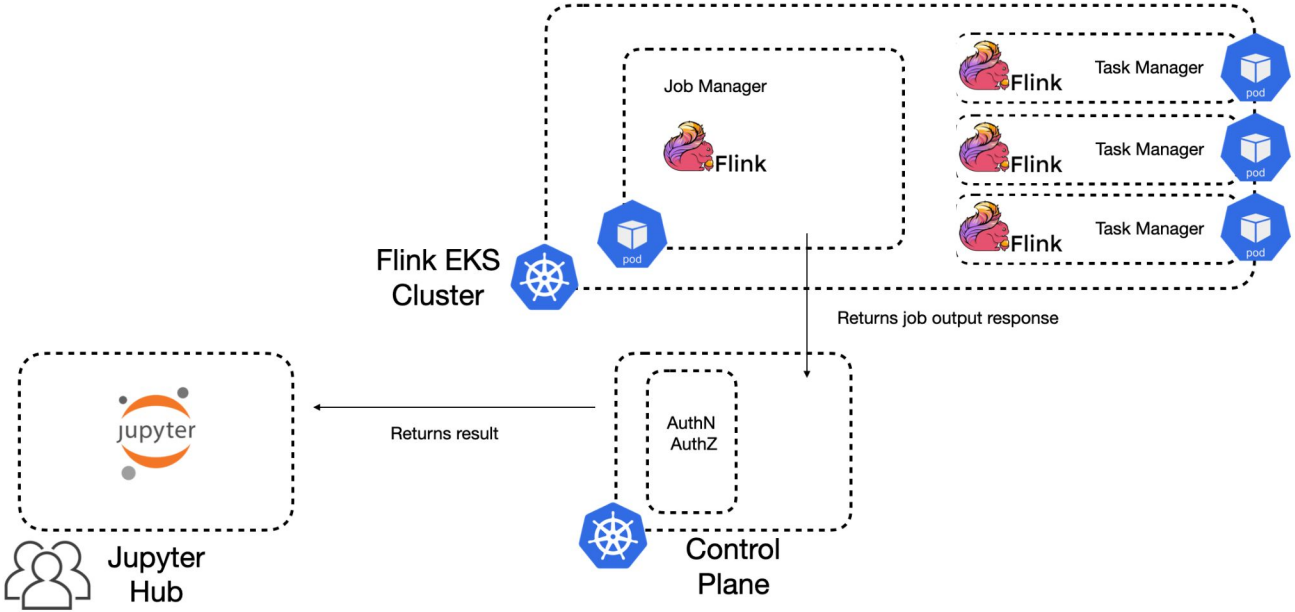
Step 2: PyFlink job submission



Step 2: PyFlink job submission



Step 2: PyFlink job submission



Thank you.