

Problem Statement:

In professional basketball, particularly within the NBA, real-time data analytics are pivotal for enhancing game analysis, optimizing player performance, and boosting fan engagement. Despite the potential of live data, its effective utilization is often hampered by the complexity of processing and visualizing this data in real-time. Our project accesses live NBA data using the `nba_api`, which provides a rich set of game statistics through endpoints like `https://nba-prod-us-east-1-mediaops-stats.s3.amazonaws.com/NBA/liveData/scoreboard/todaysScoreboard_00.json`. The API outputs data in JSON format, encapsulating game details such as scores, player stats, and team performance metrics across various game periods.

However, challenges persist in swiftly ingesting, processing, and displaying these data points. The structure of the data includes nested elements under keys like `games`, `homeTeam`, and `awayTeam`, detailing real-time updates like game scores, individual player performances, and game status. The goal of our proposed data pipeline—integrating Apache Kafka, Apache Spark, InfluxDB, and Grafana—is to streamline these processes. By setting up Kafka for efficient data ingestion, using Spark for real-time data analytics, storing processed data in InfluxDB, and visualizing insights through Grafana, we aim to facilitate quick, informed decision-making for coaches and teams, and enhance the interactive experience for fans. This pipeline promises to revolutionize how stakeholders interact with live game data, turning raw stats into actionable insights instantaneously.

Task 1: Data Ingestion

Apache Kafka is the platform used for handling real-time data feeds. It is a distributed event streaming platform capable of handling trillions of events a day.

Apache Kafka Requires Java to run, as Kafka itself is a Java-based system.

Install Java:

When setting up a data pipeline involving Apache Kafka, Java is a necessary dependency. Go to Oracle's official Java download page. Choose the version of Java Development Kit (JDK) you need. For most users working with Kafka and Spark, JDK 22 or the latest version will be suitable. Run the installer and check if the java has been installed or not from the terminal.

oracle.com/java/technologies/downloads/#jdk22-mac

ORACLE Products Industries Resources Customers Partners Developers Company

Java downloads Tools and resources Java archive

JDK 22 JDK 21 JDK 17 GraalVM for JDK 22 GraalVM for JDK 21 GraalVM for JDK 17

JDK Development Kit 22.0.1 downloads

JDK 22 binaries are free to use in production and free to redistribute, at no cost, under the [Oracle No-Fee Terms and Conditions \(NFTC\)](#).
JDK 22 will receive updates under these terms, until September 2024, when it will be superseded by JDK 23.

Linux **macOS** Windows

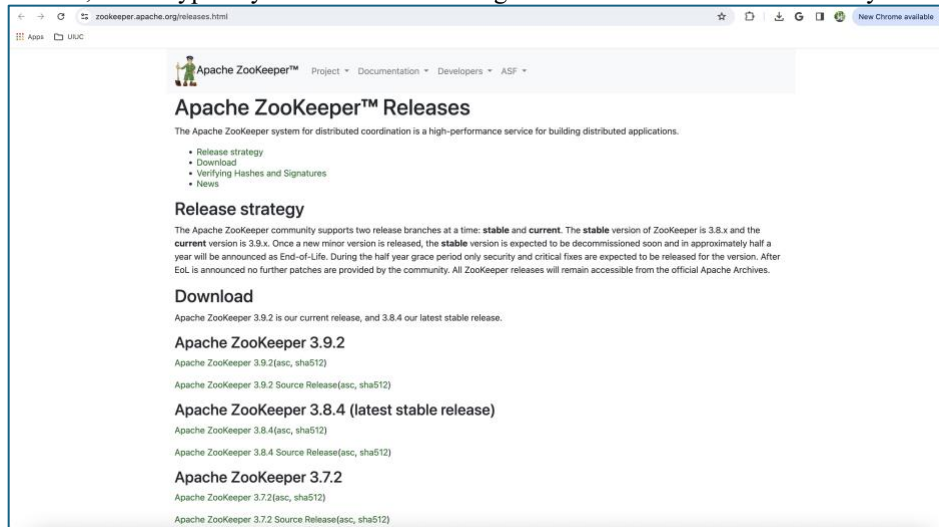
Product/file description	File size	Download
ARM64 Compressed Archive	180.03 MB	https://download.oracle.com/java/22/latest/jdk-22_macos-aarch64_bin.tar.gz (sha256)
ARM64 DMG Installer	179.47 MB	https://download.oracle.com/java/22/latest/jdk-22_macos-aarch64_bin.dmg (sha256)
x64 Compressed Archive	182.21 MB	https://download.oracle.com/java/22/latest/jdk-22_macos-x64_bin.tar.gz (sha256)
x64 DMG Installer	181.65 MB	https://download.oracle.com/java/22/latest/jdk-22_macos-x64_bin.dmg (sha256)

```
echo $JAVA_HOME
java -version
```

```
(base) nik@Nikhils-MacBook-Air-3 ~ % echo $JAVA_HOME
/Library/Java/JavaVirtualMachines/jdk-22.jdk/Contents/Home
(base) nik@Nikhils-MacBook-Air-3 ~ % java -version
java version "22.0.1" 2024-04-16
Java(TM) SE Runtime Environment (build 22.0.1+8-16)
Java HotSpot(TM) 64-Bit Server VM (build 22.0.1+8-16, mixed mode, sharing)
(base) nik@Nikhils-MacBook-Air-3 ~ %
```

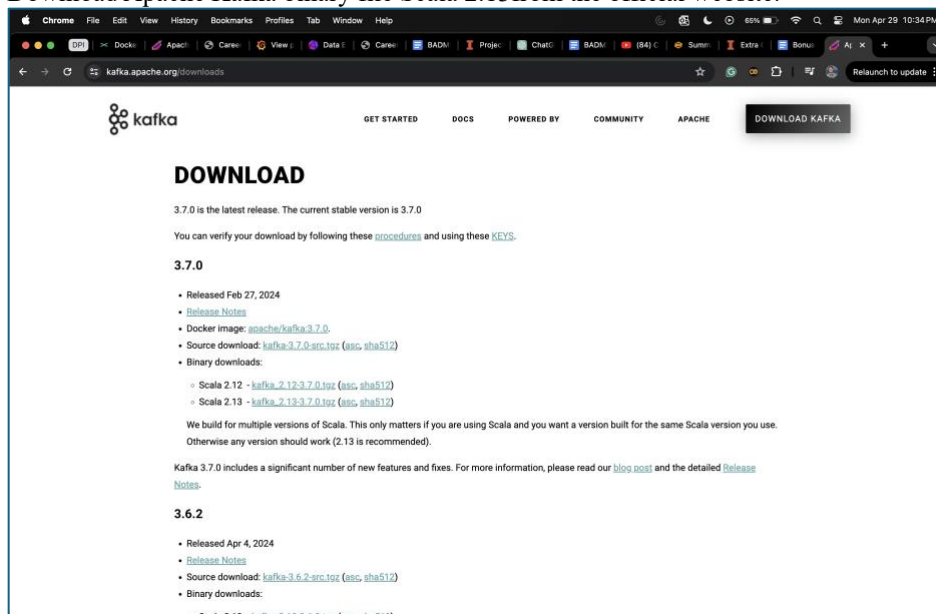
Install Zookeeper:

Go to the Apache Zookeeper releases page to download the latest stable release of Zookeeper. Look for the binary archive, which typically has an extension `.tar.gz`. Extract the file and have it ready.



Install Kafka:

Download Apache Kafka binary file Scala 2.13 from the official website.



Once the file is downloaded place the file in a folder. Extract the file in the same folder by giving the following command on the terminal.

```
tar -xvf kafka_2.13-3.7.0.tar.gz
```

Kafka Setup:

Next objective is to create a virtual environment (venv) so that the Kafka server can be run in it.

```
PROBLEMS 174 OUTPUT DEBUG CONSOLE TERMINAL PORTS
> ▾ TERMINAL
❏ (base) panshulsaraswat@Panshuls-MBP BDI % python -m venv bdiec
(base) panshulsaraswat@Panshuls-MBP BDI % source bdiec/bin/activate

(bdiec) (base) panshulsaraswat@Panshuls-MBP BDI % █
```

After the successful creation of the environment venv navigate to the extracted Kafka folder in the venv.

```
(bdiec) (base) panshulsaraswat@Panshuls-MBP BDI % cd kafka_2.13-3.7.0
(bdiec) (base) panshulsaraswat@Panshuls-MBP kafka_2.13-3.7.0 % █
```

Now we run the below script to launch the zookeeper which is required to run Kafka.

bin/zookeeper-server-start.sh config/zookeeper.properties

After the above command is run in the terminal successfully, below results will be displayed indicating the zookeeper service is running

```
> ▾ TERMINAL
❏ [2024-04-29 22:48:10,027] INFO Using org.apache.zookeeper.server.NIOServerCnxnFactory as server connection factory (org.apac
he.zookeeper.server.ServerCnxnFactory)
[2024-04-29 22:48:10,027] WARN maxCnxns is not configured, using default value 0. (org.apache.zookeeper.server.ServerCnxnFac
tory)
[2024-04-29 22:48:10,028] INFO Configuring NIO connection handler with 10s sessionless connection timeout, 2 selector thread
(s), 20 worker threads, and 64 kB direct buffers. (org.apache.zookeeper.server.NIOServerCnxnFactory)
[2024-04-29 22:48:10,033] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServerCnxnFactory)
[2024-04-29 22:48:10,045] INFO Using org.apache.zookeeper.server.watch.WatchManager as watch manager (org.apache.zookeeper.s
erver.watch.WatchManagerFactory)
[2024-04-29 22:48:10,045] INFO Using org.apache.zookeeper.server.watch.WatchManager as watch manager (org.apache.zookeeper.s
erver.watch.WatchManagerFactory)
[2024-04-29 22:48:10,045] INFO zookeeper.snapshotSizeFactor = 0.33 (org.apache.zookeeper.server.ZKDatabase)
[2024-04-29 22:48:10,045] INFO zookeeper.commitLogCount=500 (org.apache.zookeeper.server.ZKDatabase)
[2024-04-29 22:48:10,047] INFO zookeeper.snapshot.compression.method = CHECKED (org.apache.zookeeper.server.persistence.Snap
Stream)
[2024-04-29 22:48:10,049] INFO Reading snapshot /tmp/zookeeper/version-2/snapshot.1a9 (org.apache.zookeeper.server.persisten
ce.FileSnap)
[2024-04-29 22:48:10,053] INFO The digest in the snapshot has digest version of 2, with zxid as 0x1a9, and digest value as 3
95655149635 (org.apache.zookeeper.server.DataTree)
[2024-04-29 22:48:10,062] INFO 18 txns loaded in 5 ms (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2024-04-29 22:48:10,062] INFO Snapshot loaded in 17 ms, highest zxid is 0x1bb, digest is 403991666305 (org.apache.zookeeper
.server.ZKDatabase)
[2024-04-29 22:48:10,062] INFO Snapshotting: 0x1bb to /tmp/zookeeper/version-2/snapshot.1bb (org.apache.zookeeper.server.per
sistence.FileTxnSnapLog)
[2024-04-29 22:48:10,064] INFO Snapshot taken in 2 ms (org.apache.zookeeper.server.ZooKeeperServer)
[2024-04-29 22:48:10,069] INFO PrepRequestProcessor (sid:0) started, reconfigEnabled=false (org.apache.zookeeper.server.Prep
RequestProcessor)
[2024-04-29 22:48:10,070] INFO zookeeper.request_throttler.shutdownTimeout = 10000 ms (org.apache.zookeeper.server.RequestTh
rottler)
[2024-04-29 22:48:10,078] INFO Using checkIntervalMs=60000 maxPerMinute=10000 maxNeverUsedIntervalMs=0 (org.apache.zookeeper
.server.ContainerManager)
[2024-04-29 22:48:10,078] INFO ZooKeeper audit is disabled. (org.apache.zookeeper.audit.ZKAuditProvider)
█
```

Note: Do not terminate the zookeeper terminal as we need to keep this service running

Next step is run a kafka server, open another terminal and navigate to extracted Kafka directory and give the below command to run a script which will launch the Kafka server.

bin/kafka-server-start.sh config/server.properties

Once the command is run successfully, below results will be shown indicating Kafka server is now setup and is now ready for data to be sent to it.

```
> ▾ TERMINAL
00, supportedProtocols=List(range)) in group console-consumer-82932 with generation 1. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,986] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-35 in 29 milliseconds for epoch 0, of which 28 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,986] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-5 in 29 milliseconds for epoch 0, of which 29 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,987] INFO Loaded member MemberMetadata(memberId=console-consumer-5860c7d1-6335-4088-8e74-7b57f7be4cbba, groupId=None, clientId=console-consumer, clientHost=/192.168.50.124, sessionTimeoutMs=45000, rebalanceTimeoutMs=30000, supportedProtocols=List(range)) in group console-consumer-55397 with generation 1. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,987] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-20 in 30 milliseconds for epoch 0, of which 29 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,987] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-27 in 30 milliseconds for epoch 0, of which 30 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,987] INFO Loaded member MemberMetadata(memberId=console-consumer-b85c8b62-79a3-4f9d-9689-9e42304fd3dc, groupId=None, clientId=console-consumer, clientHost=/192.168.50.124, sessionTimeoutMs=45000, rebalanceTimeoutMs=30000, supportedProtocols=List(range)) in group console-consumer-5756 with generation 1. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,988] INFO Loaded member MemberMetadata(memberId=console-consumer-9ed50f98-e5ff-4fb9-b0f2-974d62bbca26, groupId=None, clientId=console-consumer, clientHost=/10.192.106.183, sessionTimeoutMs=45000, rebalanceTimeoutMs=30000, supportedProtocols=List(range)) in group console-consumer-95377 with generation 1. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,988] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-42 in 31 milliseconds for epoch 0, of which 30 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,988] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-12 in 31 milliseconds for epoch 0, of which 31 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,988] INFO Loaded member MemberMetadata(memberId=console-consumer-6dd32251-4179-457d-b62a-838fa430a9bd, groupId=None, clientId=console-consumer, clientHost=/10.192.106.183, sessionTimeoutMs=45000, rebalanceTimeoutMs=30000, supportedProtocols=List(range)) in group console-consumer-26192 with generation 1. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,988] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-21 in 31 milliseconds for epoch 0, of which 31 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,988] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-36 in 31 milliseconds for epoch 0, of which 31 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,988] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-6 in 31 milliseconds for epoch 0, of which 31 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,989] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-43 in 32 milliseconds for epoch 0, of which 31 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,989] INFO Loaded member MemberMetadata(memberId=console-consumer-6bf9f2c3-d1e7-4932-affd-fd10b55837e3, groupId=None, clientId=console-consumer, clientHost=/192.168.1.136, sessionTimeoutMs=45000, rebalanceTimeoutMs=30000, supportedProtocols=List(range)) in group console-consumer-38993 with generation 1. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,989] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-13 in 32 milliseconds for epoch 0, of which 32 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2024-04-29 22:55:49,989] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-28 in 32 milliseconds for epoch 0, of which 32 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
```

In an another terminal deactivate virtual environment venv, as this terminal will be later used to run a python file.

Create a python script in which you provide the data source and the Kafka topic for it to be written to.

The kafka-python library, which is a Kafka client for Python, allows for easy interaction with the Kafka system. This library can be installed using below command.

pip install kafka-python

Below is the script which used to send NBA data from NBA api. This code also contains a scheduler which ensures that the data is updated every 60 seconds in the Kafka topic/server. The scheduler can be removed as well depending on the use case.

Code:

```
from kafka import KafkaProducer
import json
import time
from nba_api.live.nba.endpoints import scoreboard
from apscheduler.schedulers.blocking import BlockingScheduler
import logging

# Set up logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Kafka Configuration
kafka_broker = 'localhost:9092'
kafka_topic = 'nba-scoreboard-1' <topic name>

# Create a Kafka producer
producer = KafkaProducer(
    bootstrap_servers=[kafka_broker],
    value_serializer=lambda x: json.dumps(x).encode('utf-8')
)

def fetch_nba_scores():
    try:
        # Fetch today's NBA scoreboard
        games = scoreboard.ScoreBoard()
        data = games.get_dict() # Fetch the games data as a dictionary
        return data
    except Exception as e:
        logger.error(f"Failed to fetch NBA scores: {e}")
        return None

def send_data_to_kafka(data):
    try:
        if data:
            producer.send(kafka_topic, data)
            producer.flush()
            logger.info("Data sent to Kafka successfully.")
        else:
            logger.info("No data to send.")
    except Exception as e:
        logger.error(f"Failed to send data to Kafka: {e}")

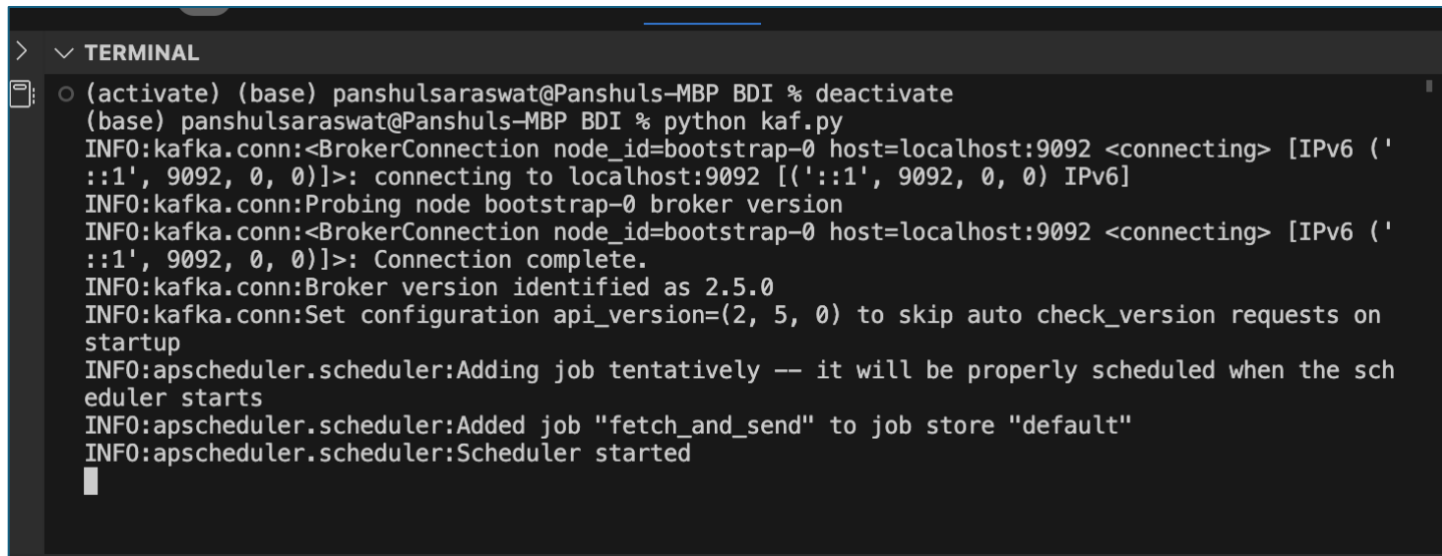
def fetch_and_send():
    data = fetch_nba_scores() # Fetch data from NBA API
    send_data_to_kafka(data) # Send the data to Kafka

def main():
    scheduler = BlockingScheduler()
    scheduler.add_job(fetch_and_send, 'interval', minutes=1) # Adjust the interval as needed
    try:
        scheduler.start()
    except (KeyboardInterrupt, SystemExit):
        pass
```

```
if __name__ == "__main__":
    main()
```

Save the python file and run in the terminal as shown below.

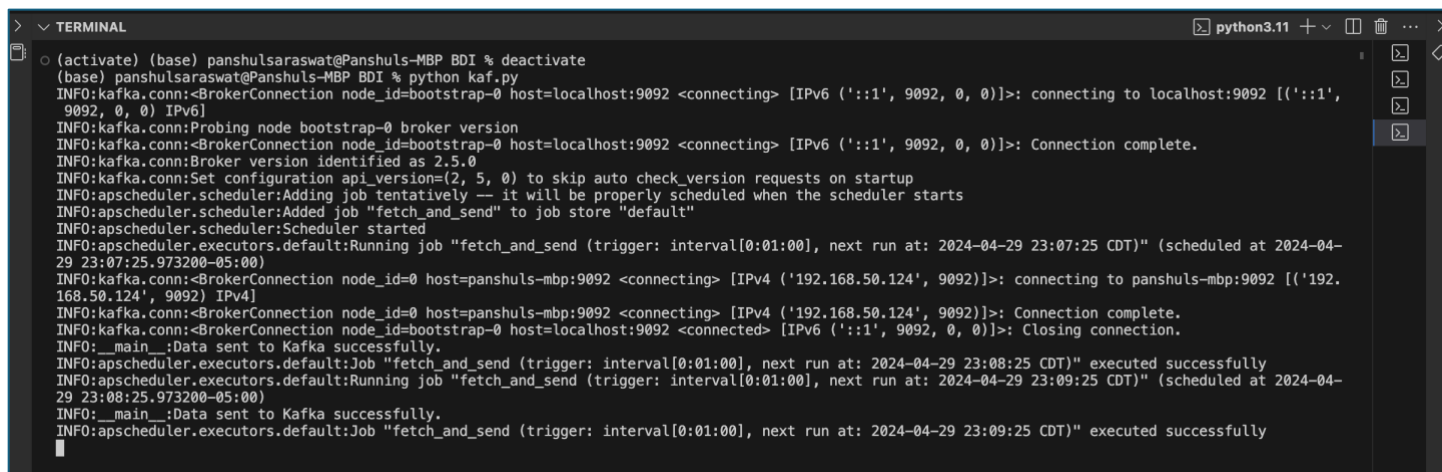
Python kaf.py



```
> ▼ TERMINAL
○ (activate) (base) panshulsaraswat@Panshuls-MBP BDI % deactivate
(base) panshulsaraswat@Panshuls-MBP BDI % python kaf.py
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=localhost:9092 <connecting> [IPv6 ('::1', 9092, 0, 0)]>: connecting to localhost:9092 [('::1', 9092, 0, 0) IPv6]
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=localhost:9092 <connecting> [IPv6 ('::1', 9092, 0, 0)]>: Connection complete.
INFO:kafka.conn:Broker version identified as 2.5.0
INFO:kafka.conn:Set configuration api_version=(2, 5, 0) to skip auto check_version requests on startup
INFO:apscheduler.scheduler:Adding job tentatively -- it will be properly scheduled when the scheduler starts
INFO:apscheduler.scheduler:Added job "fetch_and_send" to job store "default"
INFO:apscheduler.scheduler:Scheduler started
```

Once the data is sent successfully you can see a print statement in the terminal:

INFO: __main__:Data sent to Kafka successfully.



```
> ▼ TERMINAL
○ (activate) (base) panshulsaraswat@Panshuls-MBP BDI % deactivate
(base) panshulsaraswat@Panshuls-MBP BDI % python kaf.py
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=localhost:9092 <connecting> [IPv6 ('::1', 9092, 0, 0)]>: connecting to localhost:9092 [('::1', 9092, 0, 0) IPv6]
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=localhost:9092 <connecting> [IPv6 ('::1', 9092, 0, 0)]>: Connection complete.
INFO:kafka.conn:Broker version identified as 2.5.0
INFO:kafka.conn:Set configuration api_version=(2, 5, 0) to skip auto check_version requests on startup
INFO:apscheduler.scheduler:Adding job tentatively -- it will be properly scheduled when the scheduler starts
INFO:apscheduler.scheduler:Added job "fetch_and_send" to job store "default"
INFO:apscheduler.scheduler:Scheduler started
INFO:apscheduler.executors.default:Running job "fetch_and_send (trigger: interval[0:01:00], next run at: 2024-04-29 23:07:25 CDT)" (scheduled at 2024-04-29 23:07:25.973200-05:00)
INFO:kafka.conn:<BrokerConnection node_id=0 host=panshuls-mbp:9092 <connecting> [IPv4 ('192.168.50.124', 9092)]>: connecting to panshuls-mbp:9092 [('192.168.50.124', 9092) IPv4]
INFO:kafka.conn:<BrokerConnection node_id=0 host=panshuls-mbp:9092 <connecting> [IPv4 ('192.168.50.124', 9092)]>: Connection complete.
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=localhost:9092 <connected> [IPv6 ('::1', 9092, 0, 0)]>: Closing connection.
INFO: __main__:Data sent to Kafka successfully.
INFO:apscheduler.executors.default:Job "fetch_and_send (trigger: interval[0:01:00], next run at: 2024-04-29 23:08:25 CDT)" executed successfully
INFO:apscheduler.executors.default:Running job "fetch_and_send (trigger: interval[0:01:00], next run at: 2024-04-29 23:08:25 CDT)" (scheduled at 2024-04-29 23:08:25.973200-05:00)
INFO: __main__:Data sent to Kafka successfully.
INFO:apscheduler.executors.default:Job "fetch_and_send (trigger: interval[0:01:00], next run at: 2024-04-29 23:09:25 CDT)" executed successfully
```

Viewing the data that is sent to Kafka:

Open a new terminal and ensure that you are in the same venv and then navigate to the Kafka folder.

Use the below command to run a script that shows the data that is there in the Kafka topic from beginning.

```
bin/kafka-console-consumer.sh --topic <your_topic_name> --from-beginning --bootstrap-server localhost:9092
```

Below you can see a screenshot of the NBA data that was sent to Kafka.


```
PROBLEMS 177 OUTPUT DEBUG CONSOLE TERMINAL PORTS
> TERMINAL
java - kafka_2.13-3.7.0
0429/OKCNO0P", "gameStatus": 3, "gameStatusText": "Final", "period": 4, "gameClock": "", "gameTimeUTC": "2024-04-30T00:30:00Z", "gameEt": "2024-04-29T20:30:00Z", "regulationPeriods": 4, "ifNecessary": false, "seriesGameNumber": "Game 4", "gameLabel": "West - First Round", "gameSubLabel": "Game 4", "seriesText": "OKC wins 4-0", "seriesConference": "West", "poRoundDesc": "First Round", "gameSubtype": "", "homeTeam": {"teamId": 1610612740, "teamName": "Pelicans", "teamCity": "New Orleans", "teamTricode": "NOP", "wins": 0, "losses": 4, "score": 89, "seed": 8, "inBonus": null, "timeoutsRemaining": 0, "periods": [{"period": 1, "periodType": "REGULAR", "score": 21}, {"period": 2, "periodType": "REGULAR", "score": 22}, {"period": 3, "periodType": "REGULAR", "score": 28}, {"period": 4, "periodType": "REGULAR", "score": 18}], "awayTeam": {"teamId": 1610612760, "teamName": "Thunder", "teamCity": "Oklahoma City", "teamTricode": "OKC", "wins": 4, "losses": 0, "score": 97, "seed": 1, "inBonus": null, "timeoutsRemaining": 1, "periods": [{"period": 1, "periodType": "REGULAR", "score": 21}, {"period": 2, "periodType": "REGULAR", "score": 23}, {"period": 3, "periodType": "REGULAR", "score": 26}, {"period": 4, "periodType": "REGULAR", "score": 27}], "gameLeaders": {"homeLeaders": {"personId": 202685, "name": "Jonas Valanciunas", "jerseyNum": "17", "position": "C", "teamTricode": "NOP", "playerSlug": null, "points": 19, "rebounds": 13, "assists": 1}, "awayLeaders": {"personId": 1628983, "name": "Shai Gilgeous-Alexander", "jerseyNum": "2", "position": "G", "teamTricode": "OKC", "playerSlug": null, "points": 24, "rebounds": 10, "assists": 3}}, "pbOdds": {"team": null, "odds": 0.0, "suspended": 0}}}, {"meta": {"version": 1, "request": "https://nba-prod-us-east-1-mediaops-s3.amazonaws.com/NBA/liveData/scoreboard/todaysScoreboard_00.json", "time": "2024-04-30 12:12:13.1213", "code": 200}, "scoreboard": {"gameDate": "2024-04-29", "leagueId": "00", "leagueName": "National Basketball Association", "gameId": "0042300155", "gameCode": "20240429/LALDEN", "gameStatus": 2, "gameStatusText": "Q4 4:07", "period": 4, "gameClock": "PT04M07.00S", "gameTimeUTC": "2024-04-30T02:00:00Z", "gameEt": "2024-04-29T22:00:00-04:00", "regulationPeriods": 4, "ifNecessary": false, "seriesGameNumber": "Game 5", "gameLabel": null, "gameSubLabel": null, "seriesText": "DEN leads 3-1", "seriesConference": "West", "poRoundDesc": "First Round", "gameSubtype": "", "homeTeam": {"teamId": 1610612743, "teamName": "Nuggets", "teamCity": "Denver", "teamTricode": "DEN", "wins": 3, "losses": 1, "score": 97, "seed": 2, "inBonus": null, "timeoutsRemaining": 2, "periods": [{"period": 1, "periodType": "REGULAR", "score": 28}, {"period": 2, "periodType": "REGULAR", "score": 22}, {"period": 3, "periodType": "REGULAR", "score": 31}, {"period": 4, "periodType": "REGULAR", "score": 16}], "awayTeam": {"teamId": 1610612747, "teamName": "Lakers", "teamCity": "Los Angeles", "teamTricode": "LAL", "wins": 1, "losses": 3, "score": 95, "seed": 7, "inBonus": null, "timeoutsRemaining": 2, "periods": [{"period": 1, "periodType": "REGULAR", "score": 24}, {"period": 2, "periodType": "REGULAR", "score": 29}, {"period": 3, "periodType": "REGULAR", "score": 26}, {"period": 4, "periodType": "REGULAR", "score": 16}], "gameLeaders": {"homeLeaders": {"personId": 1629008, "name": "Michael Porter Jr.", "jerseyNum": "1", "position": "F", "teamTricode": "DEN", "playerSlug": "michael-porter-jr", "points": 26, "rebounds": 4, "assists": 1}, "awayLeaders": {"personId": 2544, "name": "LeBron James", "jerseyNum": "23", "position": "F", "teamTricode": "LAL", "playerSlug": "lebron-james", "points": 26, "rebounds": 7, "assists": 10}}, "pbOdds": {"team": null, "odds": 0.0, "suspended": 0}}, {"gameId": "0042300104", "gameCode": "20240429/BOSMIA", "gameStatus": 3, "gameStatusText": "Final", "period": 4, "gameClock": "", "gameTimeUTC": "2024-04-29T23:30:00Z", "gameEt": "2024-04-29T19:30:00Z", "regulationPeriods": 4, "ifNecessary": false, "seriesGameNumber": "Game 4", "gameLabel": "East - First Round", "gameSubLabel": "Game 4", "seriesText": "BOS leads 3-1", "seriesConference": "East", "poRoundDesc": "First Round", "gameSubtype": "", "homeTeam": {"teamId": 1610612748, "teamName": "Heat", "teamCity": "Miami", "teamTricode": "MIA", "wins": 1, "losses": 3, "score": 88, "seed": 8, "inBonus": null, "timeoutsRemaining": 0, "periods": [{"period": 1, "periodType": "REGULAR", "score": 24}, {"period": 2, "periodType": "REGULAR", "score": 12}, {"period": 3, "periodType": "REGULAR", "score": 23}, {"period": 4, "periodType": "REGULAR", "score": 29}], "awayTeam": {"teamId": 1610612738, "teamName": "Celtics", "teamCity": "Boston", "teamTricode": "BOS", "wins": 3, "losses": 1, "score": 102, "seed": 1, "inBonus": null, "timeoutsRemaining": 1, "periods": [{"period": 1, "periodType": "REGULAR", "score": 34}, {"period": 2, "periodType": "REGULAR", "score": 19}, {"period": 3, "periodType": "REGULAR", "score": 28}, {"period": 4, "periodType": "REGULAR", "score": 21}], "gameLeaders": {"homeLeaders": {"personId": 1628389, "name": "Bam Adebayo", "jerseyNum": "13", "position": "C-F", "teamTricode": "MIA", "playerSlug": null, "points": 25, "rebounds": 17, "assists": 5}, "awayLeaders": {"personId": 1628401, "name": "Derrick White", "jerseyNum": "9", "position": "G", "teamTricode": "BOS", "playerSlug": null, "points": 38, "rebounds": 4, "assists": 3}}, "pbOdds": {"team": null, "odds": 0.0, "suspended": 0}}, {"gameId": "0042300144", "gameCode": "20240429/OKCNO0P", "gameStatus": 3, "gameStatusText": "Final", "period": 4, "gameClock": "", "gameTimeUTC": "2024-04-30T00:30:00Z", "gameEt": "2024-04-29T20:30:00Z", "regulationPeriods": 4, "ifNecessary": false, "seriesGameNumber": "Game 4", "gameLabel": "West - First Round", "gameSubLabel": "Game 4", "seriesText": "OKC wins 4-0", "seriesConference": "West", "poRoundDesc": "First Round", "gameSubtype": "", "homeTeam": {"teamId": 1610612740, "teamName": "Pelicans", "teamCity": "New Orleans", "teamTricode": "NOP", "wins": 0, "losses": 4, "score": 89, "seed": 8, "inBonus": null, "timeoutsRemaining": 0, "periods": [{"period": 1, "periodType": "REGULAR", "score": 21}, {"period": 2, "periodType": "REGULAR", "score": 22}, {"period": 3, "periodType": "REGULAR", "score": 28}, {"period": 4, "periodType": "REGULAR", "score": 18}], "awayTeam": {"teamId": 1610612760, "teamName": "Thunder", "teamCity": "Oklahoma City", "teamTricode": "OKC", "wins": 4, "losses": 0, "score": 97, "seed": 1, "inBonus": null, "timeoutsRemaining": 1, "periods": [{"period": 1, "periodType": "REGULAR", "score": 21}, {"period": 2, "periodType": "REGULAR", "score": 23}, {"period": 3, "periodType": "REGULAR", "score": 26}, {"period": 4, "periodType": "REGULAR", "score": 27}], "gameLeaders": {"homeLeaders": {"personId": 202685, "name": "Jonas Valanciunas", "jerseyNum": "17", "position": "C", "teamTricode": "NOP", "playerSlug": null, "points": 19, "rebounds": 13, "assists": 1}, "awayLeaders": {"personId": 1628983, "name": "Shai Gilgeous-Alexander", "jerseyNum": "2", "position": "G", "teamTricode": "OKC", "playerSlug": null, "points": 24, "rebounds": 10, "assists": 3}}, "pbOdds": {"team": null, "odds": 0.0, "suspended": 0}}}]
```

This Completes Apache Kafka setup.

Task 2: Data Processing, and Storing

Setup Apache Spark:

PySpark is used to process the data streaming from Kafka. It handles reading, transforming, and extracting useful information from the Kafka topics. The PySpark library in Python, installed from the below command.

```
pip install pyspark
```

Spark-Kafka connector allows PySpark to read from and write to Kafka topics directly. This is crucial for integrating Kafka streams with Spark processing.

Spark-SQL-Kafka-0-10 Package must be specified in your Spark session to connect Spark with Kafka. It's included with Spark but needs to be explicitly enabled with configuration options when initializing the Spark session:

```
.config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10_2.12:<spark_version>")
```

Replace <spark_version> with your actual Spark version.

Setup Influx DB:

InfluxDB is a time-series database designed to handle high write and query loads. It is particularly useful in scenarios involving real-time analytics.

InfluxDB Client for Python: `influxdb_client` library, allows you to connect, write, and query data from an InfluxDB instance. This can be installed using following command.

pip install influxdb-client

Creating a bucket in InfluxDB can be done through the InfluxDB UI (User Interface). Open your web browser and navigate to the InfluxDB instance. The default URL is usually <http://localhost:8086>

Once Logged in we need to create and setup bucket in the influx db where the data is stored. As explained above install the dependency if not installed earlier.

The screenshot shows the 'Setting Up Python' wizard in the InfluxDB UI. The left sidebar contains a vertical list of steps: Overview, Install Dependencies (highlighted with a blue circle), Get Token, Initialize Client, Write Data, Execute a Simple Query, Execute an Aggregate Query, and Finish. The main content area is titled 'Install Dependencies' and includes the instruction: 'First, you need to install the `influxdb-client` module. Run the command below in your terminal.' Below this is a code block containing the command `pip3 install influxdb-client` and a 'COPY TO CLIPBOARD' button. A note states: 'You'll need to have **Python 3** installed.' At the bottom right are 'PREVIOUS' and 'NEXT' buttons.

Create and save the token, later to be used to find the bucket

The screenshot shows the 'Setting Up Python' wizard in the InfluxDB UI, now at the 'Tokens' step. The left sidebar shows the 'Get Token' step highlighted with a blue circle. The main content area is titled 'Tokens' and includes the instruction: 'InfluxDB Cloud uses Tokens to authenticate API access. We've created an all-access token for you for this set up process.' Below this is a code block containing the command `export INFLUXDB_TOKEN=QNWpt05eTK4-m0HV90BB-GRD5LFPGq271yABWh8ofZwKcEw6XI8auQ4_LToe2abosa4lZJ2T` and a 'COPY TO CLIPBOARD' button. An information icon (i) is followed by a note: 'Creating an all-access token is not the best security practice! We recommend you delete this token in the [Tokens page](#) after setting up, and create your own token with a specific set of permissions later.' At the bottom right are 'PREVIOUS' and 'NEXT' buttons.

Now we create organization and set up server url as they are needed to have an initial connection, here 'UIUC' is the name given to organization.

The screenshot shows the 'Setting Up Python' tutorial interface. On the left is a vertical sidebar with icons and a progress list: Overview, Install Dependencies, Get Token, Initialize Client (highlighted with a gear icon), Write Data, Execute a Simple Query, Execute an Aggregate Query, and Finish. The main content area is titled 'Initialize Client'. It instructs the user to run a command in their terminal to open the interactive Python shell: `python3`. Below this is a 'COPY TO CLIPBOARD' button. Then, it asks the user to paste the following code after the prompt (`>>>`) and press Enter. The code is:

```
import influxdb_client, os, time
from influxdb_client import InfluxDBClient, Point, WritePrecision
from influxdb_client.client.write_api import SYNCHRONOUS

token = os.environ.get("INFLUXDB_TOKEN")
org = "UIUC"
url = "http://localhost:8086"

write_client = influxdb_client.InfluxDBClient(url=url, token=token, org=org)
```

 Another 'COPY TO CLIPBOARD' button is below the code. At the bottom, a note states: 'Here, we initialize the token, organization info, and server url that are needed to set up the initial connection to InfluxDB. The client connection is then established with the `InfluxDBClient` initialization.'

Click on create bucket, below are the 2 buckets created in this process, once done test data can be sent to check, and below images shows the same setup

The screenshot shows the 'Setting Up Python' tutorial interface at the 'Write Data' step. The progress list on the left now highlights 'Write Data' with a pencil icon. The main content area is titled 'BUCKET' and features a '+ CREATE BUCKET' button. Below this, two buckets are listed: 'n_score' and 'nba'. The instruction says: 'Run the following code in your Python shell:'. The code is:

```
bucket="<BUCKET>"

write_api = client.write_api(write_options=SYNCHRONOUS)

for value in range(5):
    point = (
        Point("measurement1")
        .tag("tagname1", "tagvalue1")
        .field("field1", value)
    )
    write_api.write(bucket=bucket, org="UIUC", record=point)
    time.sleep(1) # separate points by 1 second
```

 A 'COPY TO CLIPBOARD' button is at the bottom.

u

↑

↗

📁

🔧


📅

🔔

⚙️

🔍

📄



Setting Up

Python

🕒 5 minutes

- Overview
- Install Dependencies
- Get Token
- Initialize Client
- Write Data
- Execute a Simple Query
- Execute an Aggregate Query
- Finish

```
|> range(start: -10m)
```

In this query, we are looking for data points within the last 10 minutes with a measurement of "measurement1".

Let's use that Flux query in our Python code!

Run the following:

```
query_api = client.query_api()

query = """from(bucket: "<BUCKET>")
|> range(start: -10m)
|> filter(fn: (r) => r._measurement == "measurement1")"""
tables = query_api.query(query, org="UIUC")

for table in tables:
    for record in table.records:
        print(record)
```

COPY TO CLIPBOARD

PREVIOUSNEXT

u

↑

↗

📁

🔧


📅

🔔

⚙️

🔍

📄



Setting Up

Python

🕒 5 minutes

- Overview
- Install Dependencies
- Get Token
- Initialize Client
- Write Data
- Execute a Simple Query
- Execute an Aggregate Query
- Finish

```
|> range(start: -10m)
```

In this example, we use the `mean()` function to calculate the average value of data points in the last 10 minutes.

Run the following:

```
query_api = client.query_api()

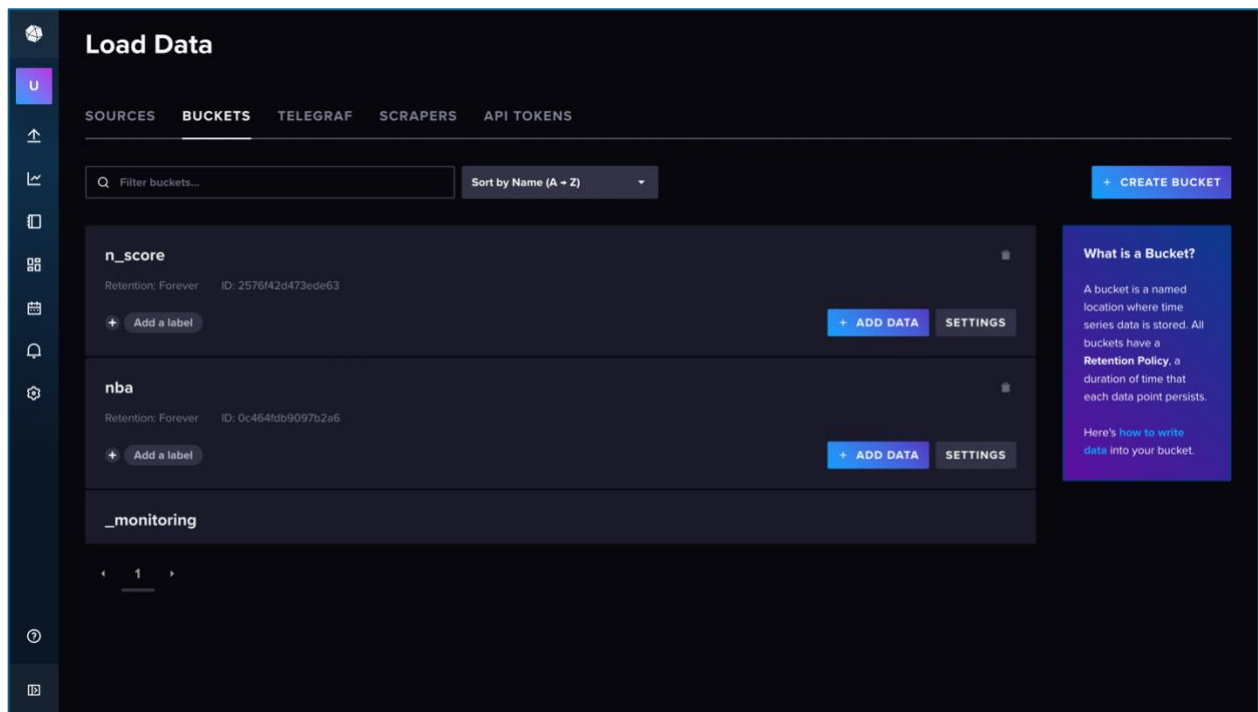
query = """from(bucket: "<BUCKET>")
|> range(start: -10m)
|> filter(fn: (r) => r._measurement == "measurement1")
|> mean()"""
tables = query_api.query(query, org="UIUC")

for table in tables:
    for record in table.records:
        print(record)
```

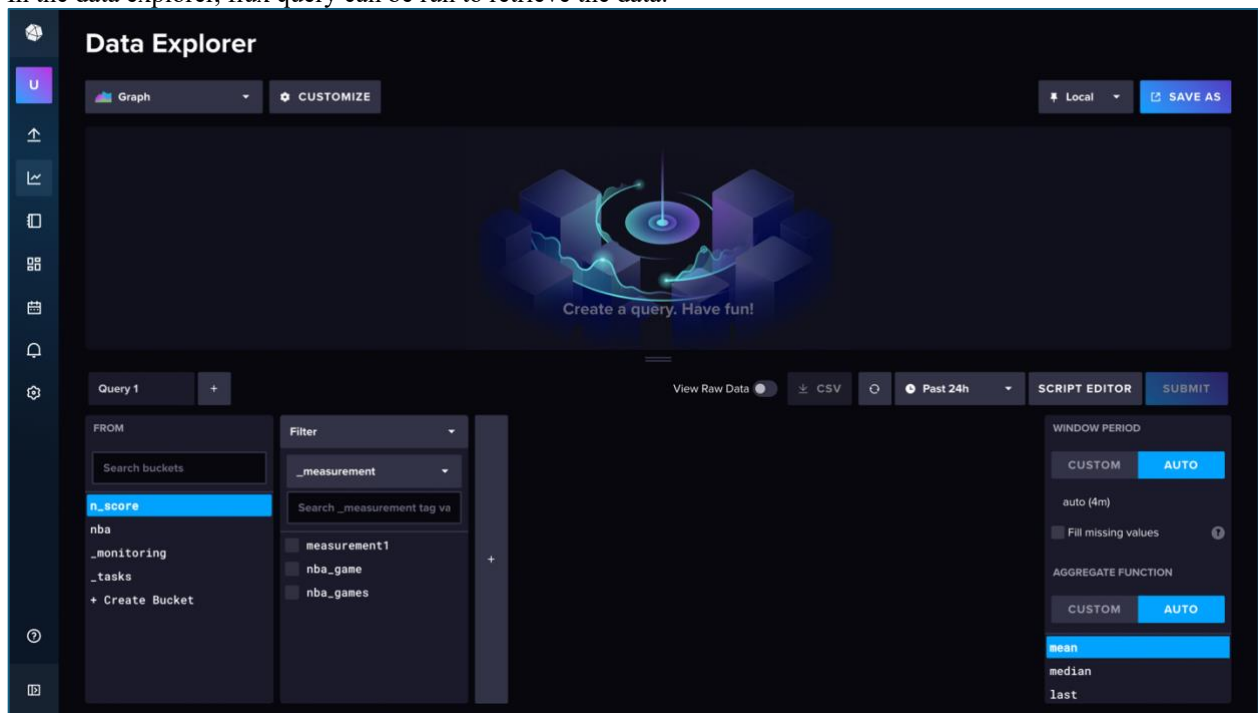
COPY TO CLIPBOARD

PREVIOUSNEXT

This will return the mean of the five values. $((0+1+2+3+4) / 5 = 2)$



In the data explorer, flux query can be run to retrieve the data.



From this set up below are the things to be noted:

influxdb_token = "CGJpFEeqwj1XiZe0HXFyT38_UOCuxrX-qEhIx7RyA4KY8L5HICN-pvvdghxStvXuoN56RmrIfW0N8JGyU2B1MA=="

influxdb_org = "UIUC"

```
influxdb_bucket = "n_score"
```

```
influxdb_url = http://localhost:8086
```

Python Shell Code:

Once all the above set up is done, create a new python script which consumes the data from our kafka topic consumer and defines schema for our data to convert from json to tabular format and send it to our influx db at every minute interval, below is the code:

[Yesterday 9:23 PM] Saraswat, Panshul

Spark code :

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, from_json, explode
from pyspark.sql.types import *
from influxdb_client import InfluxDBClient, Point, WriteOptions

# InfluxDB Client Configuration
url = "http://localhost:8086"
token = "CGJpFEeqwj1XiZe0HxFyT38_UOCuxrX-qEhIx7RyA4KY8L5HICN-
pvvdghxStvXuoN56RmrIfW0N8JGyU2B1MA=="
org = "UIUC"
bucket = "n_score"
client = InfluxDBClient(url=url, token=token, org=org)
write_api = client.write_api(write_options=WriteOptions(batch_size=1000, flush_interval=10_000))

# Define the schema based on the nested JSON structure
schema = StructType([
    StructField("meta", StructType([
        StructField("version", IntegerType()),
        StructField("request", StringType()),
        StructField("time", StringType()),
        StructField("code", IntegerType())
    ])),
    StructField("scoreboard", StructType([
        StructField("gameDate", StringType()),
        StructField("leagueId", StringType()),
        StructField("leagueName", StringType()),
        StructField("games", ArrayType(StructType([
            StructField("gameId", StringType()),
            StructField("gameCode", StringType()),
            StructField("gameStatus", IntegerType()),
            StructField("gameStatusText", StringType()),
            StructField("period", IntegerType()),
            StructField("gameClock", StringType()),
            StructField("gameTimeUTC", StringType()),
            StructField("gameEt", StringType()),
            StructField("regulationPeriods", IntegerType()),
            StructField("ifNecessary", BooleanType()),
            StructField("seriesGameNumber", StringType()),
            StructField("gameLabel", StringType()),
            StructField("gameSubLabel", StringType()),
            StructField("seriesText", StringType()),
            StructField("seriesConference", StringType()),
```

```

StructField("poRoundDesc", StringType()),
StructField("gameSubtype", StringType()),
StructField("homeTeam", StructType([
    StructField("teamId", IntegerType()),
    StructField("teamName", StringType()),
    StructField("teamCity", StringType()),
    StructField("teamTricode", StringType()),
    StructField("wins", IntegerType()),
    StructField("losses", IntegerType()),
    StructField("score", IntegerType()),
    StructField("seed", IntegerType()),
    StructField("inBonus", StringType()),
    StructField("timeoutsRemaining", IntegerType()),
    StructField("periods", ArrayType(StructType([
        StructField("period", IntegerType()),
        StructField("periodType", StringType()),
        StructField("score", IntegerType())
    ])))
])),
StructField("awayTeam", StructType([
    StructField("teamId", IntegerType()),
    StructField("teamName", StringType()),
    StructField("teamCity", StringType()),
    StructField("teamTricode", StringType()),
    StructField("wins", IntegerType()),
    StructField("losses", IntegerType()),
    StructField("score", IntegerType()),
    StructField("seed", IntegerType()),
    StructField("inBonus", StringType()),
    StructField("timeoutsRemaining", IntegerType()),
    StructField("periods", ArrayType(StructType([
        StructField("period", IntegerType()),
        StructField("periodType", StringType()),
        StructField("score", IntegerType())
    ])))
])),
StructField("gameLeaders", StructType([
    StructField("homeLeaders", StructType([
        StructField("personId", IntegerType()),
        StructField("name", StringType()),
        StructField("jerseyNum", StringType()),
        StructField("position", StringType()),
        StructField("teamTricode", StringType()),
        StructField("playerSlug", StringType()),
        StructField("points", IntegerType()),
        StructField("rebounds", IntegerType()),
        StructField("assists", IntegerType())
    ])),
    StructField("awayLeaders", StructType([
        StructField("personId", IntegerType()),
        StructField("name", StringType()),
        StructField("jerseyNum", StringType()),
        StructField("position", StringType()),
        StructField("teamTricode", StringType()),
        StructField("playerSlug", StringType()),
        StructField("points", IntegerType()),
    ]))
]),

```



```

        StructField("rebounds", IntegerType()),
        StructField("assists", IntegerType())
    ))
   )),
    StructField("pbOdds", StructType([
        StructField("team", StringType()),
        StructField("odds", DoubleType()),
        StructField("suspended", IntegerType())
    ]))
    ]))
    ])
)

```

Initialize Spark Session

```
spark = SparkSession.builder.appName("NBA Scores Kafka Consumer").getOrCreate()
```

Read from Kafka

```
df = spark.readStream.format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "nba-scoreboard-1") \
    .load()

```

Select message value and convert from bytes to string

```
df = df.selectExpr("CAST(value AS STRING)")
```

Parse the JSON string using the defined schema

```
parsed_df = df.select(from_json(col("value"), schema).alias("data"))
```

Explode the games array into individual game records

```
games_df = parsed_df.select(explode(col("data.scoreboard.games")).alias("game"))
```

Select relevant fields for processing

```
flattened_df = games_df.select(
    col("game.gameId").alias("gameId"),
    col("game.gameCode").alias("gameCode"),
    col("game.gameStatusText").alias("gameStatusText"),
    col("game.homeTeam.teamName").alias("homeTeamName"),
    col("game.homeTeam.score").alias("homeScore"),
    col("game.awayTeam.teamName").alias("awayTeamName"),
    col("game.awayTeam.score").alias("awayScore"),
    col("game.period").alias("period"),

```

```
)
col("game.gameTimeUTC").alias("gameTimeUTC")
)
```

Function to write each batch of DataFrame to InfluxDB

```
def write_to_influx(batch_df, epoch_id):
```

```
    points = []
    for row in batch_df.collect():
        point = Point("nba_games") \
            .tag("gameId", row.gameId) \
            .tag("gameCode", row.gameCode) \
            .field("gameStatusText", row.gameStatusText) \
            .field("homeTeamName", row.homeTeamName) \
            .field("homeScore", int(row.homeScore)) \
            .field("awayTeamName", row.awayTeamName) \
            .field("awayScore", int(row.awayScore)) \
            .field("period", int(row.period)) \
            .time(row.gameTimeUTC)
        points.append(point)
    write_api.write(bucket=bucket, record=points)
```

Write the stream to InfluxDB

```
query = flattened_df.writeStream.outputMode("append") \
    .foreachBatch(write_to_influx) \
    .start()
```

```
query.awaitTermination()
```

Save the python file, and in new terminal run the spark job from the below command.

spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.1 spark.py

```
> ~ TERMINAL
-4f11-9545-bb3065de251c/commits/0 using temp file file:/private/var/folders/xt/sjlwfv795tgfwl993wgvjhh0000gn/T/temporary-76f89b1f-2538-4f11-9545-bb3065de251c/commits/0.c33b564d-d8b1-469f-9fda-fb1bd0bbf8be.tmp
24/05/01 21:47:07 INFO CheckpointFileManager: Renamed temp file file:/private/var/folders/xt/sjlwfv795tgfwl993wgvjhh0000gn/T/temporary-76f89b1f-2538-4f11-9545-bb3065de251c/commits/0.c33b564d-d8b1-469f-9fda-fb1bd0bbf8be.tmp to file:/private/var/folders/xt/sjlwfv795tgfwl993wgvjhh0000gn/T/temporary-76f89b1f-2538-4f11-9545-bb3065de251c/commits/0
24/05/01 21:47:07 INFO MicroBatchExecution: Streaming query made progress: {
  "id" : "013e11ec-dad2-4a0f-bf3a-51f08c5057df",
  "runId" : "28b14bde-6292-4930-bf53-bb68bf70c7e2",
  "name" : null,
  "timestamp" : "2024-05-02T02:47:06.069Z",
  "batchId" : 0,
  "numInputRows" : 0,
  "inputRowsPerSecond" : 0.0,
  "processedRowsPerSecond" : 0.0,
  "durationMs" : {
    "addBatch" : 434,
    "commitOffsets" : 25,
    "getBatch" : 10,
    "latestOffset" : 366,
    "queryPlanning" : 222,
    "triggerExecution" : 1098,
    "waitCommit" : 28
  },
  "stateOperators" : [ ],
  "sources" : [ {
    "description" : "KafkaV2[Subscribe[nba-scoreboard-1]]",
    "startOffset" : null,
    "endOffset" : {
      "nba-scoreboard-1" : {
        "0" : 1189
      }
    },
    "latestOffset" : null,
    "numInputRows" : 0,
    "inputRowsPerSecond" : 0.0,
    "processedRowsPerSecond" : 0.0
  } ],
  "sink" : {
    "description" : "ForeachBatchSink",
    "numOutputRows" : -1
  }
}
```

Here while sending the data, we look for numoutputRows= -1 which means that the data was successfully sent to InfluxDB

Task 3: Visualize with Grafana

Setup Grafana:

Install and run Grafana by running below 2 commands in terminal

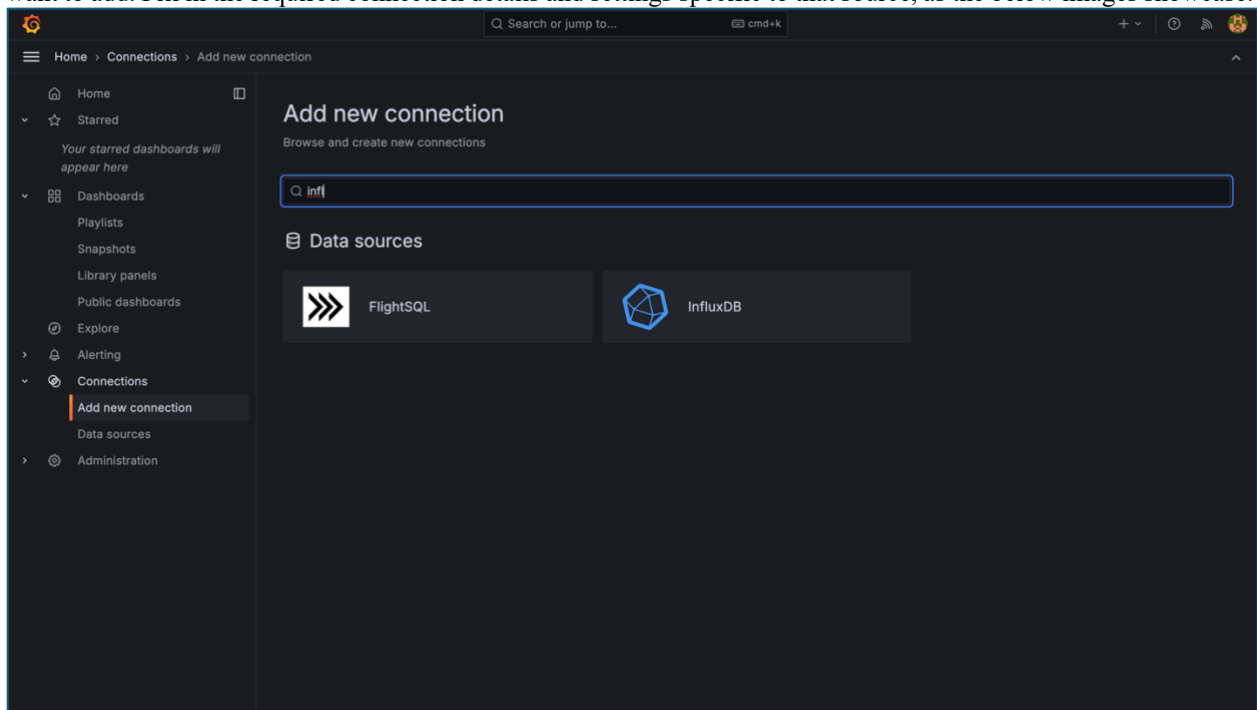
```
brew install grafana
```

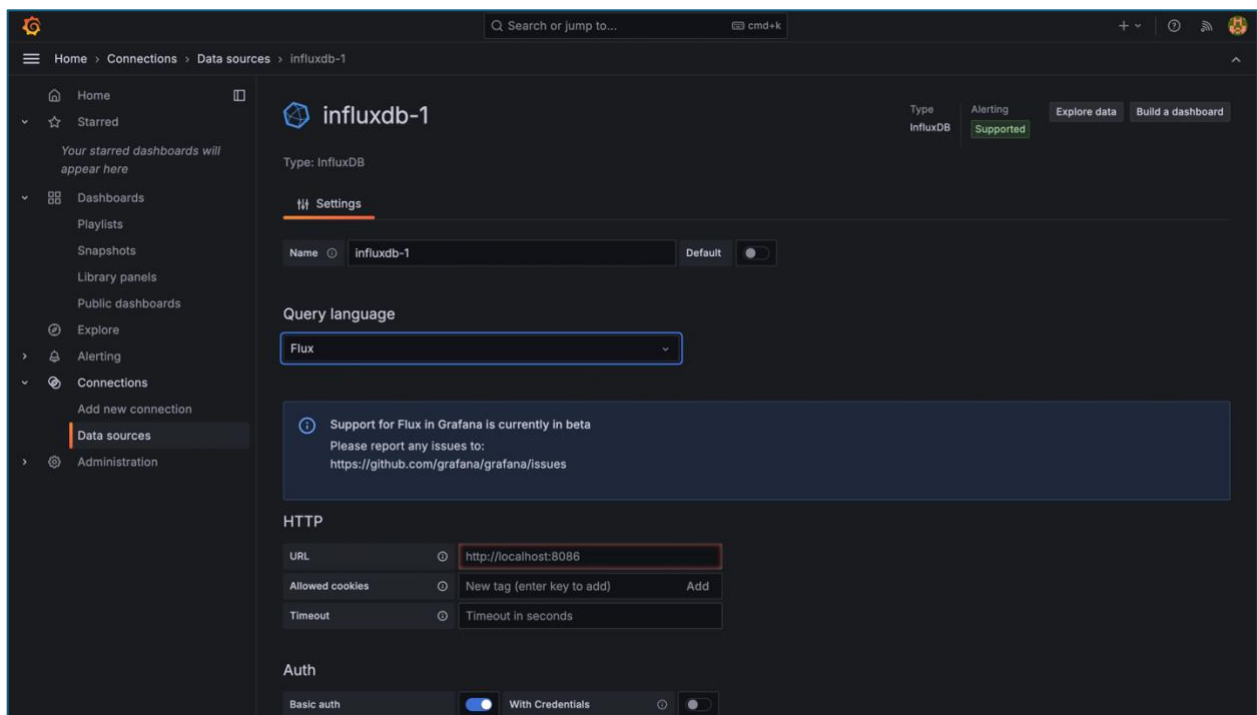
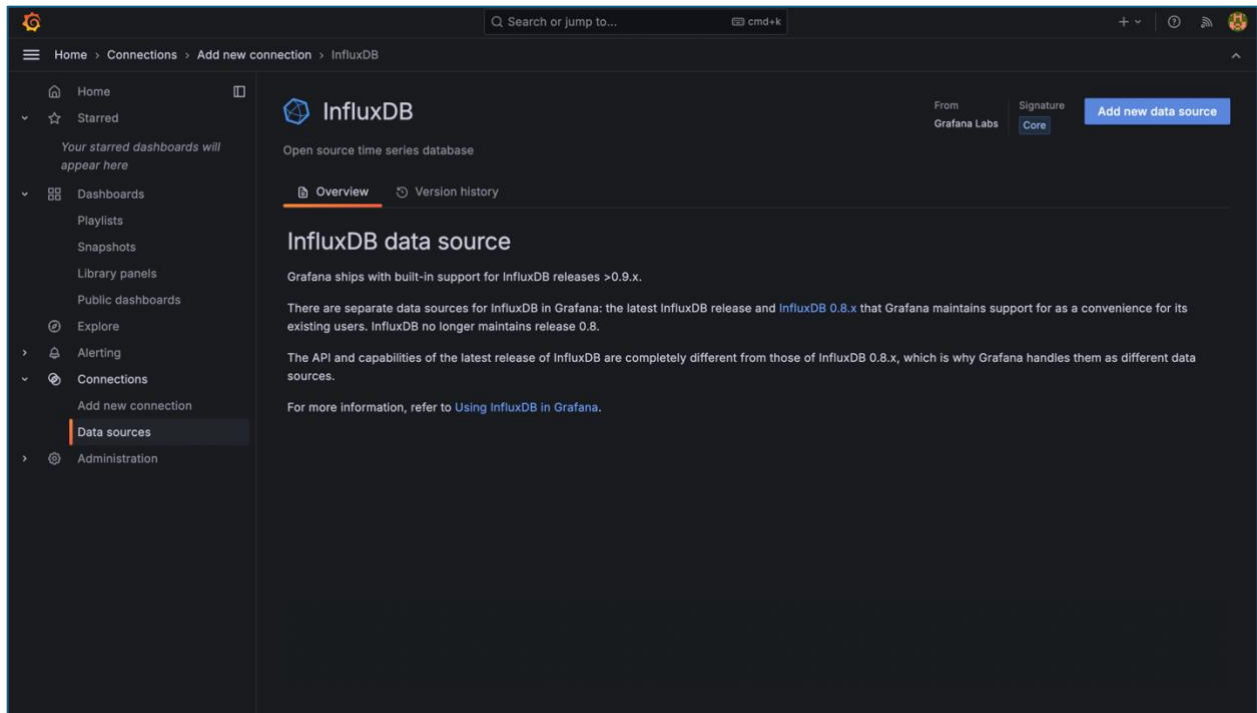
```
brew services start grafana
```

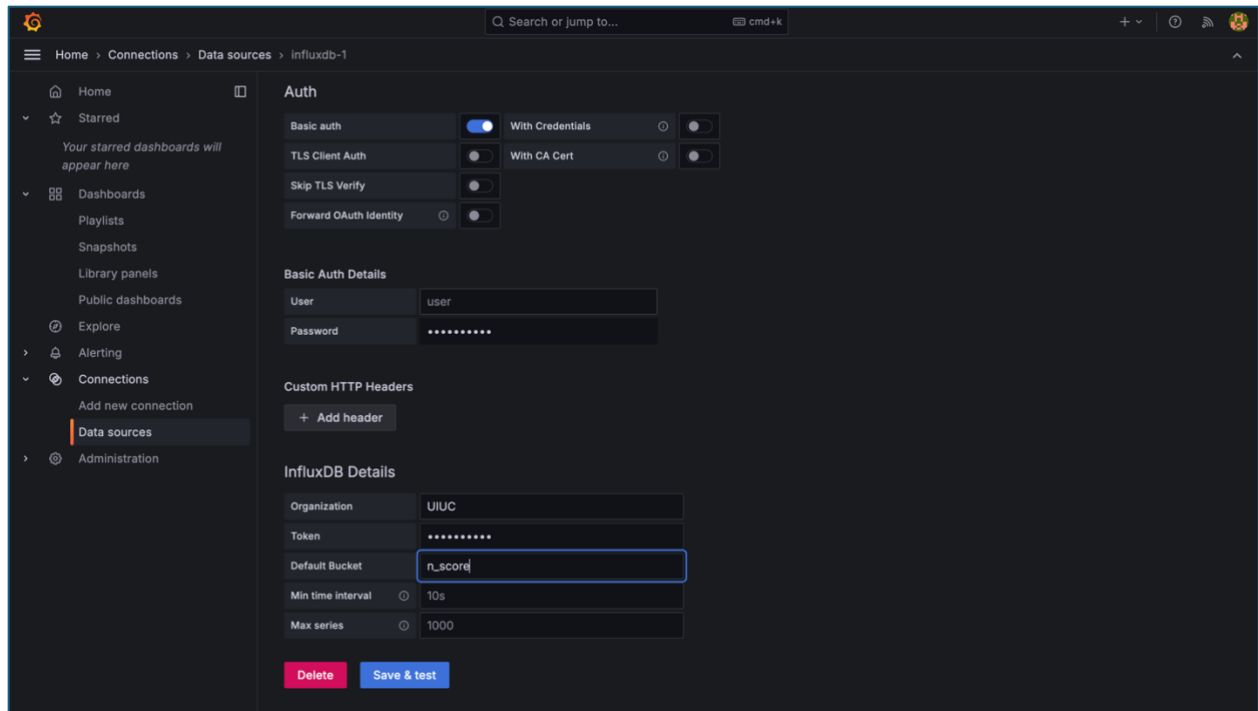
Once Grafana is installed and running, open your web browser and navigate to <http://localhost:3000/>. The default port for Grafana is 3000.

The default login credentials are Username: admin and Password: admin. Upon first login, you will be prompted to change the password.

After logging in, you can configure data sources (like InfluxDB, MySQL, PostgreSQL, etc.) that Grafana will use to pull data for visualization. Go to Configuration > Data Sources > Add data source and select the type of source you want to add. Fill in the required connection details and settings specific to that source, as the below images showcase.







Visualization:

Once your data sources are configured, you can start creating dashboards to visualize your data. Go to + Create > Dashboard and begin adding panels.

We can utilize flux query to fetch the data we need in our desired format to create and add visuals

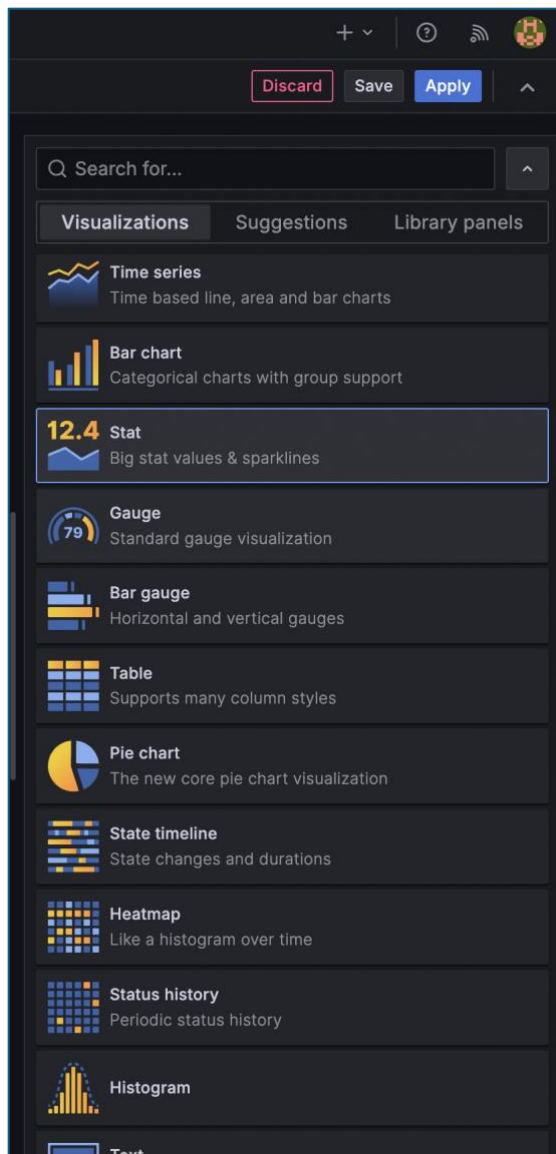
Below is the query used:

```

1 from(bucket: "n_score")
2   |> range(start: -1d)
3   |> filter(fn: (r) => r._field == "awayScore" or r._field == "homeScore" or r._field == "awayT
4   |> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn: "_value")
5   |> map(fn: (r) => ({
6     _time: r._time,
7     homeTeam: r.homeTeamName,
8     homeScore: r.homeScore,
9     awayTeam: r.awayTeamName,
10    awayScore: r.awayScore
11  })
12 )
13 |> sort(columns: ["_time"], desc: true)
14 |> limit(n: 10)
15

```

After the query is run, select the desired visualization from the right hand side and configure it to your use case.



Once done Dashboard is ready, this dashboard updates the live score of the NBA game at every one-minute interval

Clippers Mavericks homeScore

44

Celtics Heat homeScore

118

Clippers Mavericks awayScore

56

Celtics Heat awayScore

84