How to Ingest Streaming Data from Kafka to Apache Pinot™

By: Barkha Herman

May 30th, 2022 • 9 min read

We previously walked through getting started with Apache Pinot™ using batch data, and now we will learn how to ingest streaming data using Apache Kafka® topics.

As the story goes, Apache Pinot was created at LinkedIn to provide a platform that could ingest a high number of incoming events (kafka) and provide “fresh” (sub second) analytics to a large number (20+ million) of users, fast (sub second latency). So, really, consuming events is part of the reason why Pinot was created.

The obligatory “What is Apache Pinot and StarTree?” section

Pinot is a real-time, distributed, open source, and free-to-use OLAP datastore, purpose-built to provide ultra low-latency analytics at extremely high throughput. It is open source and free to use.

How does StarTree come in? StarTree offers a fully managed version of the Apache Pinot real-time analytics system , plus other tools around it that you can try for free. The system includes StarTree Dataset Manager and StarTree ThirdEye, a UI based data ingestion tool, and a real-time anomaly detection and root cause analysis tool, respectively.

How to install Kafka alongside Pinot

Prerequisite

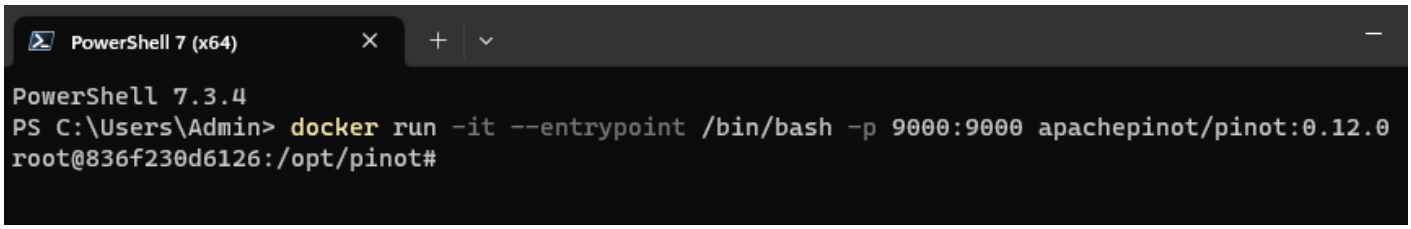
Complete the steps outlined in the [introduction to Apache Pinot](#).

Step 1: Install Kafka on your Pinot Docker image

Make sure you have completed the first article in the series.

We will be installing Apache Kafka onto our already existing Pinot docker image. To start the Docker image, run the following command:

docker run -it --entrypoint /bin/bash -p 9000:9000
apachepinot.docker.scarf.sh/apachepinot/pinot:0.12.0



We want to override the ENTRYPOINT and run Bash script within the Docker image. If you already have a container running, you can skip this step. I tend to tear down containers after use, so in my case, I created a brand new container.

Now, start each of the components one at a time like we did in the previous session:

bin/pinot-admin.sh StartZookeeper &

bin/pinot-admin.sh StartController &

bin/pinot-admin.sh StartBroker &

bin/pinot-admin.sh StartServer &

Run each of the commands one at a time. The & allows you to continue using the same Bash shell session. If you like, you can create different shells for each service:

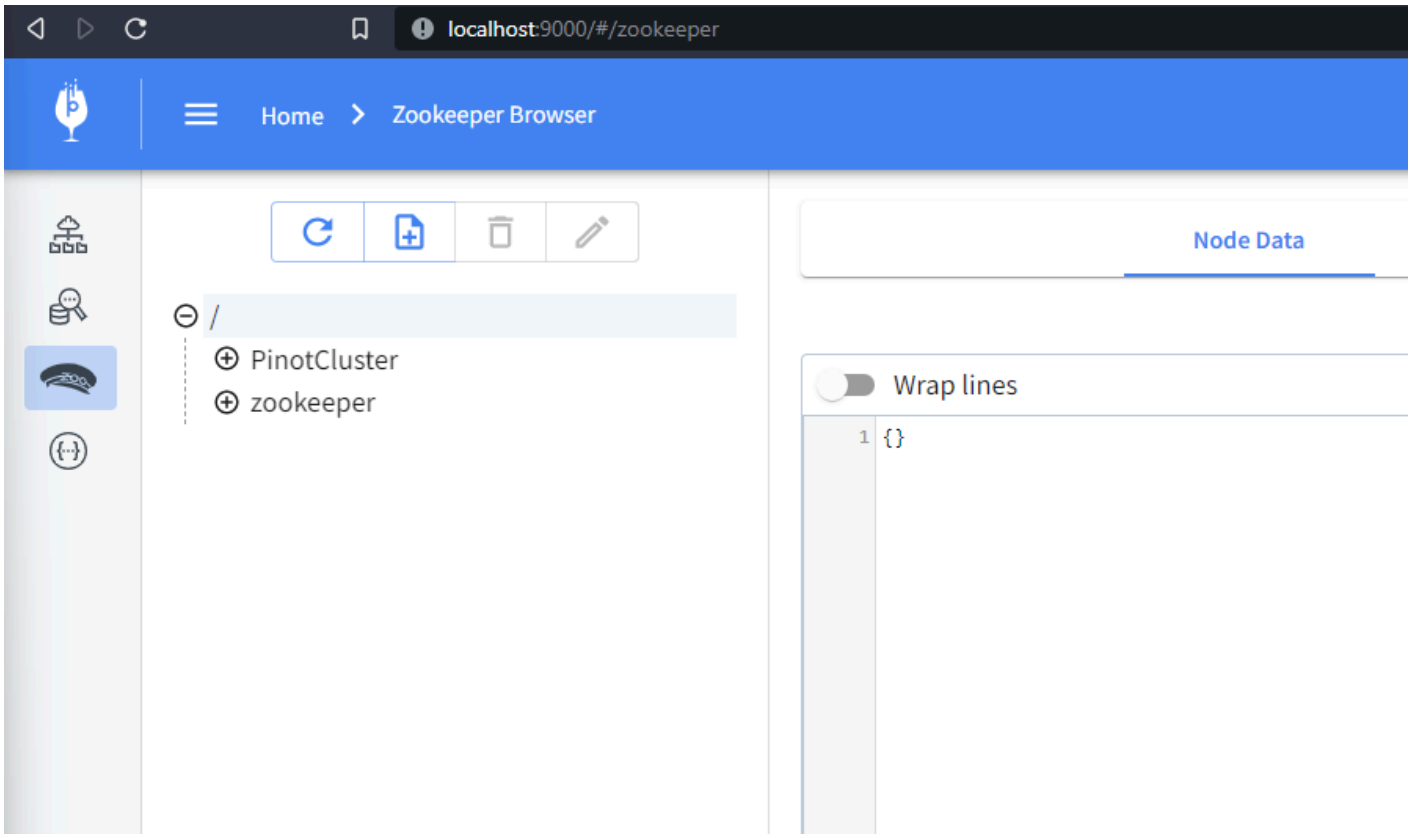
1. Get the container ID by running docker ps

2. Run `` `docker exec -it DOCKER_CONTAINER_ID bash` `` where DOCKER_CONTAINER_ID is the ID received from step 1.
3. Run the pinot-admin.sh command to start the desired service

It should look like this:

```
PS C:\Users\Admin> docker ps
CONTAINER ID   IMAGE                          COMMAND       CREATED
S                                    NAMES
836f230d6126   apachepinot/pinot:0.12.0   "/bin/bash"   About a minute ago
-8099/tcp, 0.0.0.0:9000->9000/tcp   blissful_jackson
PS C:\Users\Admin> docker exec -it 836f230d6126 bash
root@836f230d6126:/opt/pinot# bin/pinot-admin.sh StartZookeeper
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/pinot/lib/pinot-all-0.12.0-jar-with-de
```

You can now browse to http://localhost:9000/#/zookeeper to see the running cluster:



Step 2: Install Kafka on the Docker container

Next, let's install Kafka. We will be installing Kafka on the existing docker container. For this step, download the TAR file, extract the contents, and start Kafka.

Apache Kafka is an open source software platform that provides a unified, high-throughput, low-latency platform for handling real-time data feeds.

Use the following command to download the Kafka image:

```
cd ..
curl https://downloads.apache.org/kafka/3.4.0/kafka_2.12-3.4.0.tgz --output kafka
```

It should look this:

```
root@836f230d6126:/opt/pinot# cd ..
root@836f230d6126:/opt# curl https://downloads.apache.org/kafka/3.4.0/kafka_2.12-3.4.0.tgz --output kafka.tgz --output k
afka.tgz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  101M  100  101M    0     0  5203k      0  0:00:19  0:00:19 --:--:-- 6657k
root@836f230d6126:/opt#
```

Note that we've changed the directory to keep the Kafka folder separate from the Pinot folder.

Now, let's expand the downloaded TAR file, rename the folder for convenience, and delete the downloaded file:

```
tar -xvf kafka.tgz
mv kafka_2.12-3.4.0 kafka
rm -rf kafka.tgz
```

It should look like this:

```
root@836f230d6126:/opt# tar -xvf kafka.tgz
kafka_2.12-3.4.0/
kafka_2.12-3.4.0/LICENSE
kafka_2.12-3.4.0/NOTICE
kafka_2.12-3.4.0/bin/
kafka_2.12-3.4.0/bin/kafka-delete-records.sh
```

```
root@836f230d6126:/opt# mv kafka_2.12-3.4.0 kafka
root@836f230d6126:/opt# rm -rf kafka.tgz
root@836f230d6126:/opt# ls
kafka  pinot
root@836f230d6126:/opt#
```

Now, Kafka and Pinot reside locally on our Docker container with Pinot up and running. Let's run the Kafka service. Kafka will use the existing ZooKeeper for configuration management.

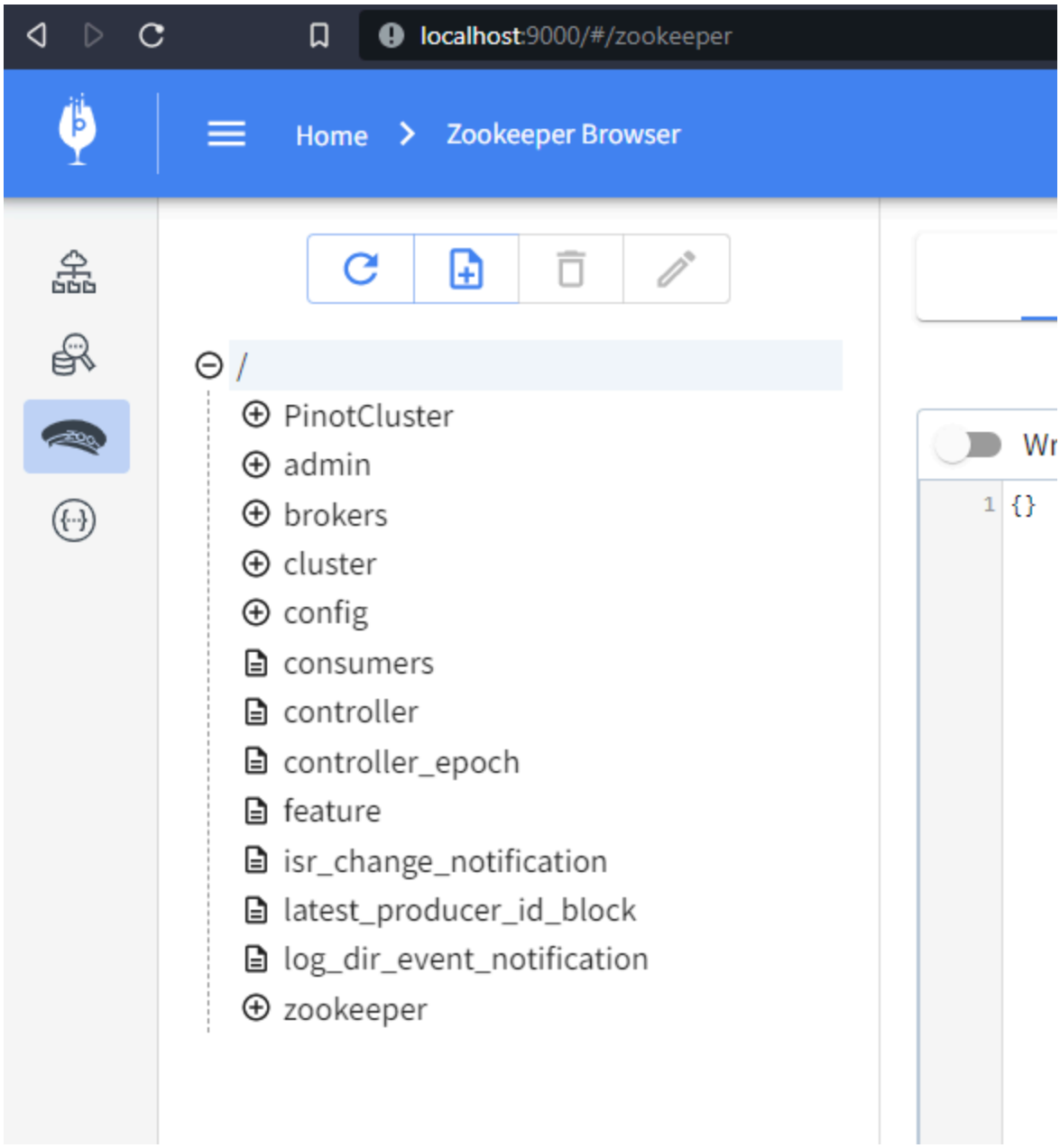
Use the following command to run Kafka:

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

It should look like this:

```
root@836f230d6126:/opt# cd kafka
root@836f230d6126:/opt/kafka# ./bin/kafka-server-start.sh config/server.properties
[2023-04-21 14:43:46,867] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistratio
n$)
[2023-04-21 14:43:47,541] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TL
S renegotiation (org.apache.zookeeper.common.X509Util)
```

To verify that Kafka is running, let's look at our ZooKeeper configs by browsing to http://localhost:9000/#/zookeeper:



You may have to refresh the page and find many more configuration items appear thanexpectedt. These are Kafka configurations.

Step 3: Ingest data into Kafka

In this step, we will ingest data into Kafka. We will be using Wikipedia events since they are easily accessible. We will use a node script to ingest the Wikipedia events, then add them to a Kafka Topic.

Let's first create some folders like this:

cd /opt

mkdir realtime

cd realtime

mkdir events

It should look like this:

```
root@836f230d6126:/opt/pinot# cd /opt
root@836f230d6126:/opt# mkdir realtime
root@836f230d6126:/opt# cd realtime
root@836f230d6126:/opt/realtime# mkdir events
root@836f230d6126:/opt/realtime#
```

You may have to start a new PowerShell window and connect to Docker for this. Now, let's install Node.js and any dependencies we might need for the event consumption script:

```
curl -fsSL https://deb.nodesource.com/setup_14.x | bash -
apt install nodejs
```

Node.js takes a few minutes to install. Next, we will create a script to consume the events called wikievents.js. Cut and paste the following code to this file:

```
var EventSource = require('eventsource');
var fs = require('fs');
var path = require('path');
const { Kafka } = require('kafkajs');

var url = 'https://stream.wikimedia.org/v2/stream/recentchange';

const kafka = new Kafka({
    clientId: 'wikievents',
    brokers: ['localhost:9092']
});

const producer = kafka.producer();
```

```
async function start() {
    await producer.connect();
    startEvents();
}

function startEvents() {
    console.log(`Connecting to EventStreams at ${url}`);
    var eventSource = new EventSource(url);

    eventSource.onopen = function () {
        console.log('--- Opened connection.');
    };

    eventSource.onerror = function (event) {
        console.error('--- Encountered error', event);
    };

    eventSource.onmessage = async function (event) {
        const data = JSON.parse(event.data);
        const eventPath = path.join(__dirname, './events', data.wiki);
        fs.existsSync(eventPath) || fs.mkdirSync(eventPath);
        fs.writeFileSync(path.join(eventPath, data.meta.id + '.json'), event.data
        await producer.send({
            topic: 'wikipedia-events',
            messages: [
                {
                    key: data.meta.id,
                    value: event.data
                }
            ]
        });
    };
}

start();
```

You can use vi to create the file and save it. You can also use Docker Desktop to edit the file.

To install the two modules referenced in the file above, kafkajs and eventsource, run the following command:

```
npm i eventsource kafkajs
```

Let's run the program. This will result in the download of many files, so I recommend running the program for just a few minutes. You can stop the run by using Ctrl-C.

```
node wikievents.js
```

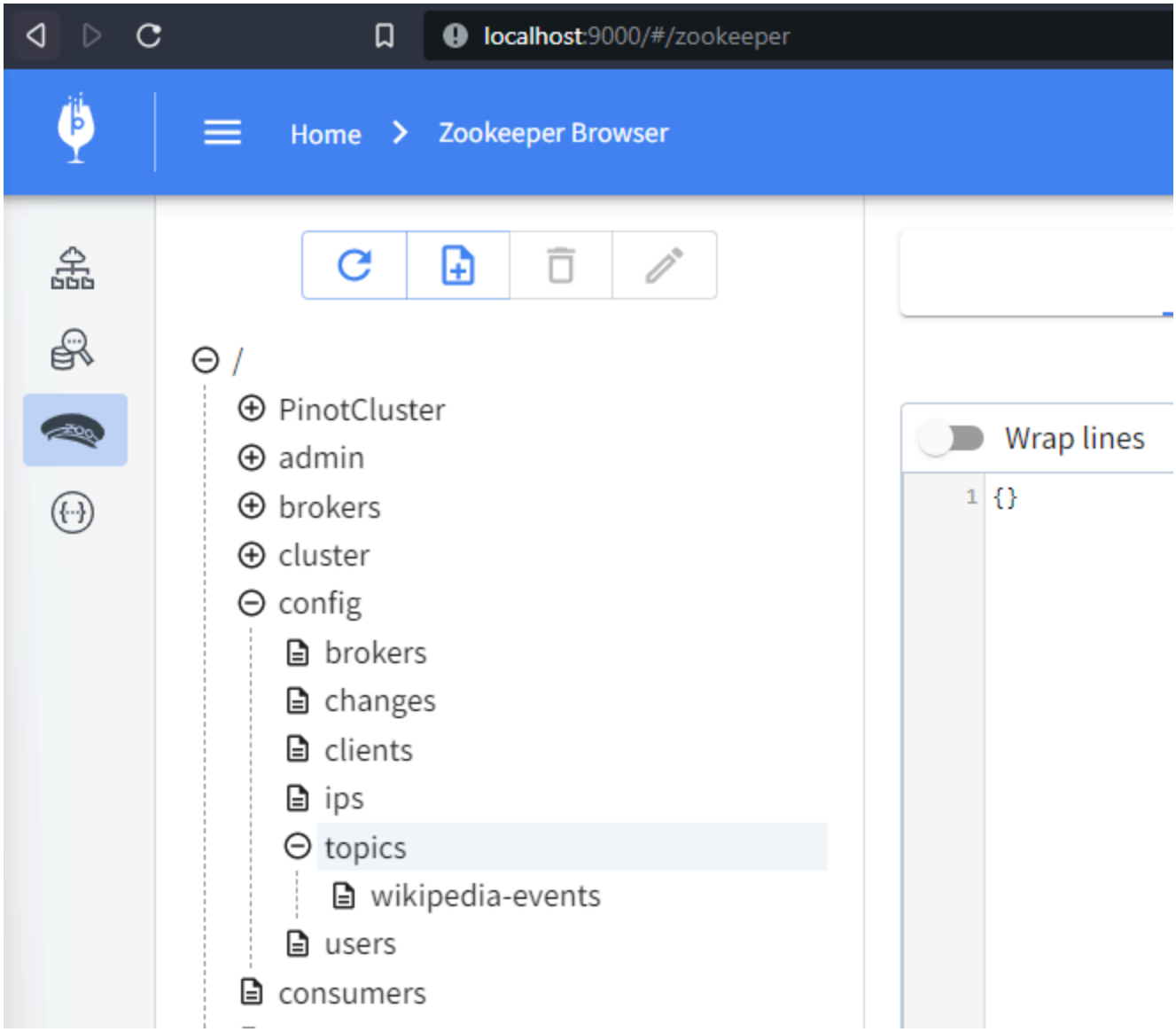
Use Ctrl-C to stop the program. Navigate to the events folder to see some new folders created with the various language events downloaded from Wikipedia.

```
root@836f230d6126:/opt/realtime# node wikievents.js
{"level":"WARN","timestamp":"2023-04-21T15:31:28.328Z","logger":"kafkajs","message":"KafkaJS v2.0.0 swi
tched default partitioner. To retain the same partitioning behavior as in previous versions, create the
 producer with the option \"createPartitioner: Partitioners.LegacyPartitioner\". See the migration guid
e at https://kafka.js.org/docs/migration-guide-v2.0.0#producer-new-default-partitioner for details. Sil
ence this warning by setting the environment variable \"KAFKAJS_NO_PARTITIONER_WARNING=1\""}
Connecting to EventStreams at https://stream.wikimedia.org/v2/stream/recentchange
--- Opened connection.
{"level":"ERROR","timestamp":"2023-04-21T15:31:29.691Z","logger":"kafkajs","message":"[Connection] Resp
onse Metadata(key: 3, version: 6)","broker":"localhost:9092","clientId":"wikievents","error":"There is
no leader for this topic-partition as we are in the middle of a leadership election","correlationId":1,
"size":93}
{"level":"ERROR","timestamp":"2023-04-21T15:31:29.981Z","logger":"kafkajs","message":"[Connection] Resp
onse Metadata(key: 3, version: 6)","broker":"localhost:9092","clientId":"wikievents","error":"There is
no leader for this topic-partition as we are in the middle of a leadership election","correlationId":2,
"size":93}
^C
root@836f230d6126:/opt/realtime# ls
events  node_modules  package-lock.json  wikievents.js
root@836f230d6126:/opt/realtime# cd events
root@836f230d6126:/opt/realtime/events# ls
commonswiki  enwiki        etwikiquote  idwiki         mediawikiwiki  ruwiki        zhwiki
cswiki       enwiktionary  euwiki       incubatorwiki  nowiki         specieswiki
dewiki       eswiktionary  frwiki       jawiki         plwiki         wikidatawiki
```

Navigate to the enwiki folder and review some of the downloaded JSON files.

```
root@836f230d6126:/opt/realtime/events# cd enwiki
root@836f230d6126:/opt/realtime/events/enwiki# ls
027b6819-bcdd-48e7-b473-06c00be066d0.json  6787c85b-dc52-40b4-8afb-c3b146cc6e4b.json
1ac125aa-141c-4d3d-b7ba-e3af1c04d5dc.json  6bcfe27a-b3de-4844-aaae-a4488dcd87cf.json
2ead0473-3d4d-4c41-89f7-a648c0e62dde.json  6f8d7e49-5431-43c2-b85b-9a7e5a69e8f8.json
4255954c-3d3a-4c34-b756-6fcfbedb12dc.json  edb53ab8-7a69-4ff7-83b9-2c695b2b4974.json
4299fbcd-70ce-4608-96ea-ac6128340366.json  fa5eaa49-b9e7-4908-a3ff-b3c22d02a1b2.json
594e08bb-65d7-41d8-957e-f26372a7c3ab.json
root@836f230d6126:/opt/realtime/events/enwiki#
```

At http://localhost:9000/#/zookeeper, you can find the Kafka topic by locating the ZooKeeper config and expanding config > topics. You may have to refresh your browser.



Here, you should see the wikipedia-events topic that we created using the Node.js script. So far, so good.

Step 4: Connect Kafka to Pinot

With Kafka installed and configured to receive events, we can connect it to Pinot.

To create a real-time table in Pinot that can consume the Kafka topic, create a schema and a configuration table. The schema configuration is very much like the schema that

we created for our batch example. You can use vi to create a file named realtime.schema.json and cut and paste the content below.

Here's the JSON for the wikievents schema:

```
{
    "schemaName": "wikievents",
    "dimensionFieldSpecs": [
        {
            "name": "id",
            "dataType": "STRING"
        },
        {
            "name": "wiki",
            "dataType": "STRING"
        },
        {
            "name": "user",
            "dataType": "STRING"
        },
        {
            "name": "title",
            "dataType": "STRING"
        },
        {
            "name": "comment",
            "dataType": "STRING"
        },
        {
            "name": "stream",
            "dataType": "STRING"
        },
        {
            "name": "domain",
            "dataType": "STRING"
        },
        {
            "name": "topic",
            "dataType": "STRING"
        },
```

```
        {
            "name": "type",
            "dataType": "STRING"
        },
        {
            "name": "metaJson",
            "dataType": "STRING"
        }
    ],
    "dateTimeFieldSpecs": [
        {
            "name": "timestamp",
            "dataType": "LONG",
            "format": "1:MILLISECONDS:EPOCH",
            "granularity": "1:MILLISECONDS"
        }
    ]
}
```

Creating the table config file is where the magic happens. Use vi (or your favorite editor) to create realtime.tableconfig.json and cut and paste the following content:

```
{
    "tableName": "wikievents_REALTIME",
    "tableType": "REALTIME",
    "segmentsConfig": {
        "timeColumnName": "timestamp",
        "schemaName": "wikievents",
        "replication": "1",
        "replicasPerPartition": "1"
    },
    "tenants": {
        "broker": "DefaultTenant",
        "server": "DefaultTenant",
        "tagOverrideConfig": {}
    },
    "tableIndexConfig": {
        "invertedIndexColumns": [],
        "rangeIndexColumns": [],
```

```
        "autoGeneratedInvertedIndex": false,
        "createInvertedIndexDuringSegmentGeneration": false,
        "sortedColumn": [],
        "bloomFilterColumns": [],
        "loadMode": "MMAP",
        "streamConfigs": {
            "streamType": "kafka",
            "stream.kafka.topic.name": "wikipedia-events",
            "stream.kafka.broker.list": "localhost:9092",
            "stream.kafka.consumer.type": "lowlevel",
            "stream.kafka.consumer.prop.auto.offset.reset": "smallest",
            "stream.kafka.consumer.factory.class.name": "org.apache.pinot.plugin.
            "stream.kafka.decoder.class.name": "org.apache.pinot.plugin.stream.ka
            "realtime.segment.flush.threshold.rows": "0",
            "realtime.segment.flush.threshold.time": "24h",
            "realtime.segment.flush.segment.size": "100M"
        },
        "noDictionaryColumns": [],
        "onHeapDictionaryColumns": [],
        "varLengthDictionaryColumns": [],
        "enableDefaultStarTree": false,
        "enableDynamicStarTreeCreation": false,
        "aggregateMetrics": false,
        "nullHandlingEnabled": false
    },
    "metadata": {},
    "quota": {},
    "routing": {},
    "query": {},
    "ingestionConfig": {
        "transformConfigs": [
            {
                "columnName": "metaJson",
                "transformFunction": "JSONFORMAT(meta)"
            },
            {
                "columnName": "id",
                "transformFunction": "JSONPATH(metaJson, '$.id')"
            },
            {
```

```
                "columnName": "stream",
                "transformFunction": "JSONPATH(metaJson, '$.stream')"
            },
            {
                "columnName": "domain",
                "transformFunction": "JSONPATH(metaJson, '$.domain')"
            },
            {
                "columnName": "topic",
                "transformFunction": "JSONPATH(metaJson, '$.topic')"
            }
        ]
    },
    "isDimTable": false
}
```

Notice the section called streamConfigs, where we define the source as a Kafka stream, located at localhost:9092, and consume the topic wikipedia-events. That's all it takes to consume a Kafka Topic into Pinot.

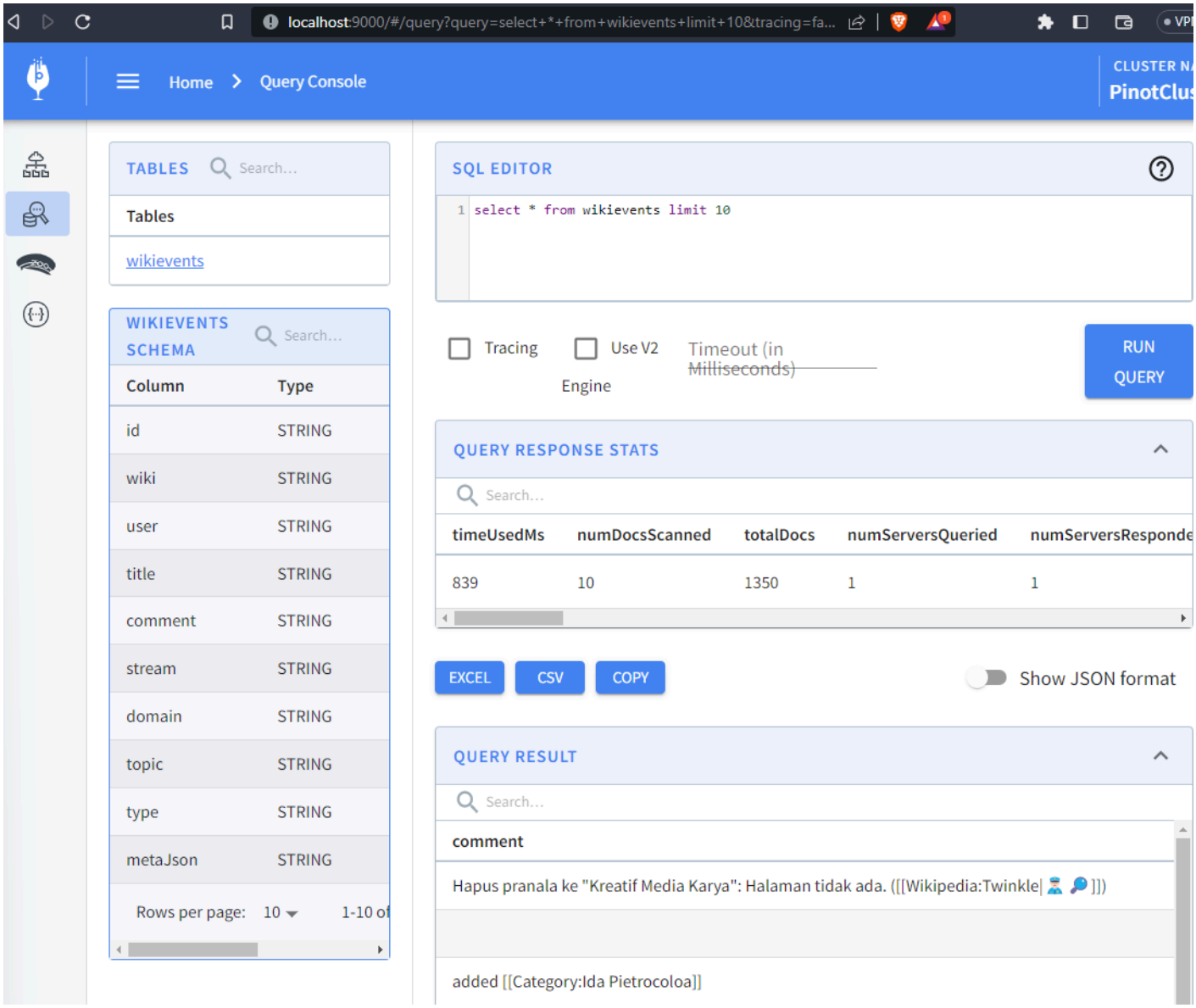
Don't believe me? Give it a try!

Create the table by running the following command:

```
/opt/pinot/bin/pinot-admin.sh AddTable -schemaFile /opt/realtime/realtime.schema.
```

Now, browse to the following location http://localhost:9000/#/tables, and you should see the newly created table. However, where's the real-time data, you say?

Run the node wikievents.js command, then query the newly created wikievents table to see the totalDocs increase in real time:



To avoid running out of space on your computer, make sure to stop the wikievents.js script when you're done :-D

Conclusion

Congratulations! Using only the table config, we simultaneously consumed Kafka topics directly into Pinot tables and queried events. We also transformed JSON to map to the Pinot table. In the transformConfigs portion of the Pinot table config file, we consumed the nested block meta into a field called metaJson. In the subsequent steps, we referenced the metaJson field with jsonPath to extract fields such as id, stream, domain, and topic.

Not only does Pinot support easy ingestion from Kafka topics, but it also provides a robust way to transform JSON to OLAP tables.

In summary, we have:

- Installed and run Kafka
- Consumed events from Wikipedia into Kafka
- Created a real-time table schema and a table in Pinot
- Streamed events from Wikipedia into Pinot tables via Kafka topics
- Run multiple queries
- Performed JSON transformations

In some upcoming blog posts, we will explore more advanced topics, such as indexes and transformations, not to mention real-time anomaly detection with ThirdEye.

In the meantime, run more queries, load more data, and don't forget to join the community Slack for support if you get stuck or would like to request a topic for me to write about—you know where to find us!

Your email address