

Kafka Connect Notes

By

Naveen Kumar M

<https://www.linkedin.com/in/naveen-kumar-m-5a6960119/>

Table of Contents

About the course	4
Course Objectives	4
Course Structure	4
Kafka Connect Overview	5
Why Kafka Connect	5
Kafka Connect and Streams Architecture Design	6
Kafka Connect Concepts	6
High Level	6
Kafka Connect Concepts	7
Standalone vs Distributed Mode	8
Distributed Mode in Details	8
Docker	9
Kafka Connect Source	10
Standalone Mode	10
Standalone Mode Demo	11
Distributed Mode	15
List of Available Connectors	18
Twitter Source Connector in Distributed Mode	19
What we learned	26
Kafka Connector Sink	27
Architecture	27
Elastic Search Sink Connector	27
ElasticSearchSink in Distributed Mode	28
Kafka Connect RESP API	33
Setting up the command line	33
Get Worker information	33
List Connectors available on a Worker	34
Ask about Active Connectors	35
Get information about a Connector Tasks and Config	35
Get Connector Status	36
Pause / Resume a Connector (no response if the call is succesful)	36
Get Connector Configuration	37
Delete our Connector	37
Create a new Connector	37
Update Connector configuration	38
JDBC Sink Connector	38
PostgreSQL	38

Distributed Mode	39
<i>Writing Own Connector</i>	42
Config Definitions	42
Connector	43
Writing a Schema	44
Data Model	46
Getting Data from Github API	47
Source Partition, Source Offsets	48
Source Task	49
Building and running a connector in standalone mode	50
Deploying Connector in Landoop Cluster	51
<i>Development guide for Connector</i>	53
<i>Guideline for Developing Connector</i>	53
<i>Setting Up Kafka Connect in Production</i>	54

About the course

Course Objectives

Course Objectives



- We're going to build a pipeline to ingest data from Twitter in real time and sink them into ElasticSearch and PostgreSQL
- Learn about Kafka Connect:
 - Kafka Connect Architecture
 - Standalone, Distributed Mode
 - Source Connectors Configuration
 - Sink Connectors Configuration
 - Kafka Connect UI
 - Kafka Connect REST API



Course Structure



Course Structure

1. Kafka Connect Concepts
2. Setup and Launch Kafka Connect Cluster using Docker Compose
- 3. Kafka Connect Source API – Hands on**
 - Get data from a file into a Kafka Topic (FileStreamSourceConnector)
 - Get data from Twitter into a Kafka Topic (TwitterSourceConnector)
- 4. Kafka Connect Sink API – Hands on**
 - Store the data from Kafka into Elastic Search (ElasticSearchSinkConnector)
 - Store the data from Kafka into PostgreSQL (JDBCSinkConnector)

(This course does not cover Avro as it isn't part of the native Kafka APIs)



Why Kafka Connect and Streams

- Four Common Kafka Use Cases:

Source => Kafka
Kafka => Kafka
Kafka => Sink
Kafka => App

Producer API
Consumer, Producer API
Consumer API
Consumer API

Kafka Connect Source
Kafka Streams
Kafka Connect Sink

- Simplify and improve getting data in and out of Kafka
- Simplify transforming data within Kafka without relying on external libs

Kafka Connect Source

- Kafka Connect API, put all the data from source to Kafka.

Kafka Streams

- Does transformation on Kafka topic.

Kafka Connect Sink

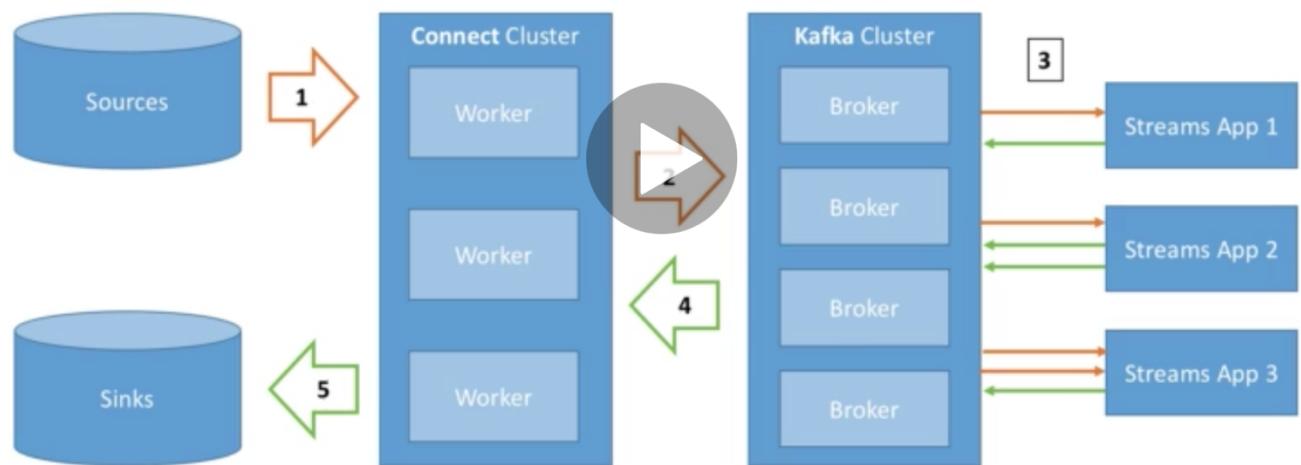
- Getting data out of Kafka and put it into target.

Why Kafka Connect



- Programmers always want to import data from the same sources:
Databases, JDBC, Couchbase, GoldenGate, SAP HANA, Blockchain, Cassandra, DynamoDB, FTP, IOT, MongoDB, MQTT, RethinkDB, Salesforce, Solr, SQS, Twitter, etc...
- Programmers always want to store data in the same sinks:
 - S3, ElasticSearch, HDFS, JDBC, SAP HANA, DocumentDB, Cassandra, DynamoDB, HBase, MongoDB, Redis, Solr, Splunk, Twitter
- It is tough to achieve Fault Tolerance, Idempotence, Distribution, Ordering
- Other programmers may already have done a very good job!

Kafka Connect and Streams Architecture Design



Steps

- Connect cluster made of workers. Pull the data from source and push it into Kafka cluster.
- Stream pull the data and does the transformation and put it back to Kafka cluster.

Kafka Connect Concepts

High Level



Kafka Connect – High level

- Source Connectors to get data from Common Data Sources
- Sink Connectors to publish that data in Common Data Stores
- Make it easy for non-experienced dev to quickly get their data reliably into Kafka
- Part of your ETL pipeline
- Scaling made easy from small pipelines to company-wide pipelines
- Re-usable code!

Kafka Connect Concepts

Kafka Connect - Concepts

- Kafka Connect Cluster has multiple loaded **Connectors**
 - Each connector is a re-usable piece of code (java jars)
 - Many connectors exist in the open source world, leverage them!
- Connectors + **User Configuration => Tasks**
 - A task is linked to a connector configuration
 - A job configuration may spawn multiple tasks
- Tasks are executed by Kafka Connect **Workers** (servers)
 - A worker is a single java process
 - A worker can be standalone or in a cluster

Key Notes:

- Connectors are re-usable piece of code.
- Connectors + User configuration gives us tasks.
- Task linked to user configuration.
- Job is a combination of multiple tasks.
- Task are executed by Kafka connect workers.

Kafka Connect Workers

Standalone vs Distributed Mode

- **Standalone:**

- A single process runs your connectors and tasks
- Configuration is bundled with your process
- Very easy to get started with, [useful for development and testing](#)
- Not fault tolerant, no scalability, hard to monitor

- **Distributed:**

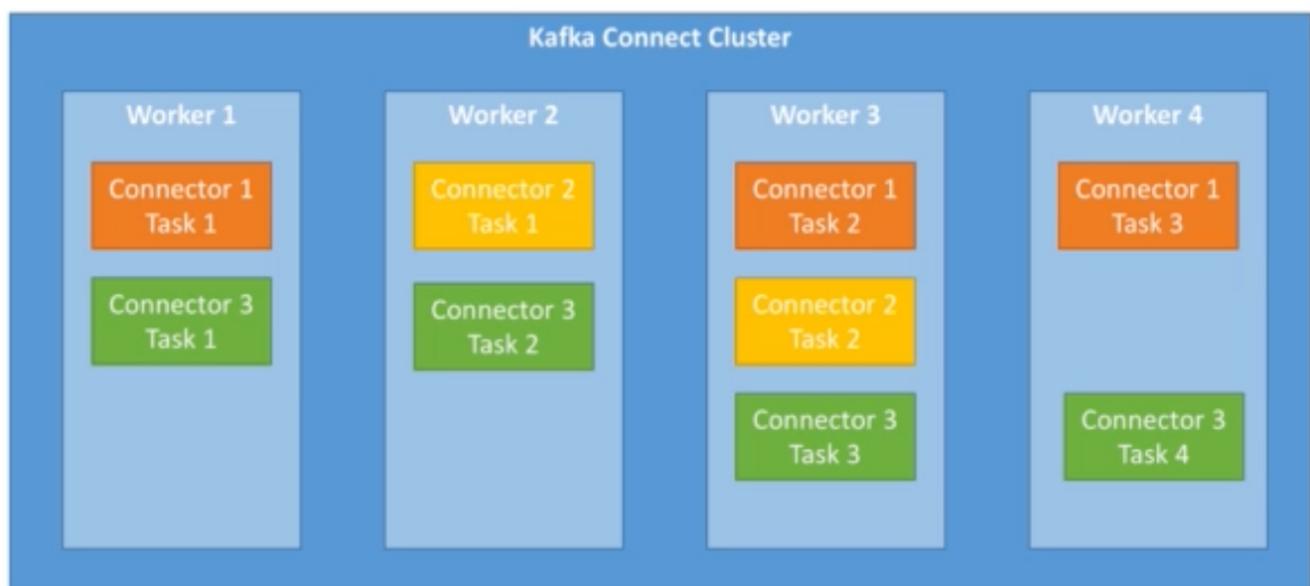
- Multiple workers run your connectors and tasks
- Configuration is submitted using a REST API
- Easy to scale, and fault tolerant (rebalancing in case a worker dies)
- [Useful for production deployment of connectors](#)

Distributed Mode in Details

We have connector's tasks spread across in workers.

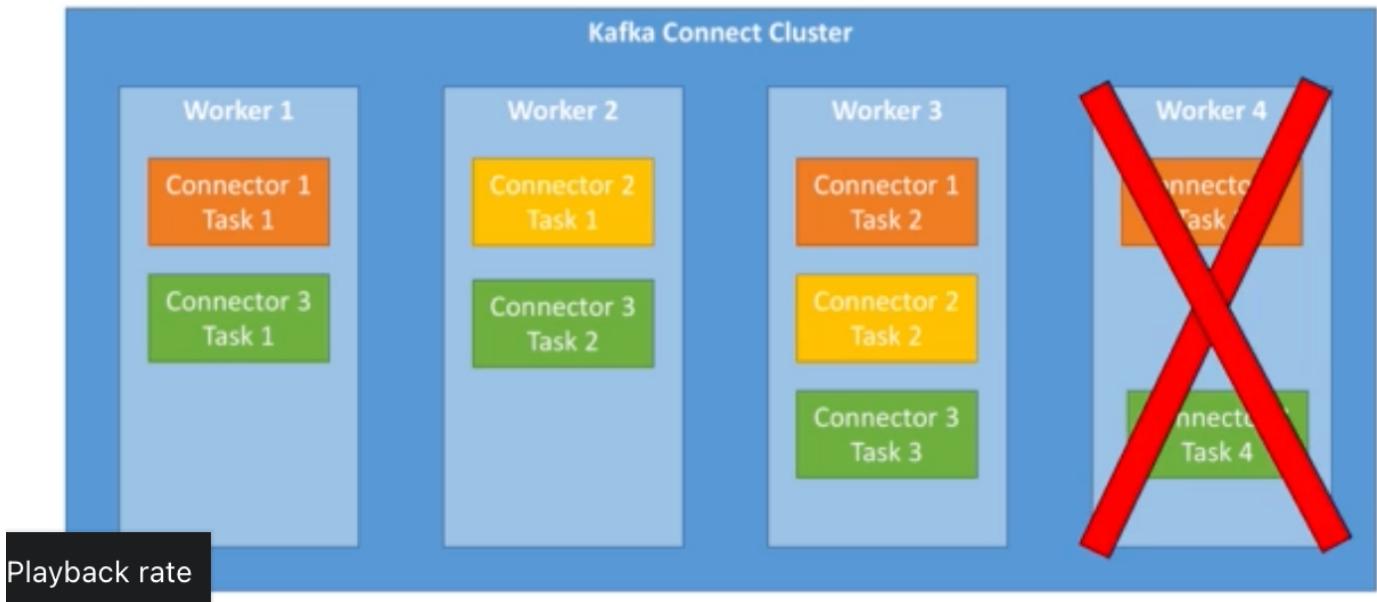
Kafka Connect Cluster

Distributed Architecture in details

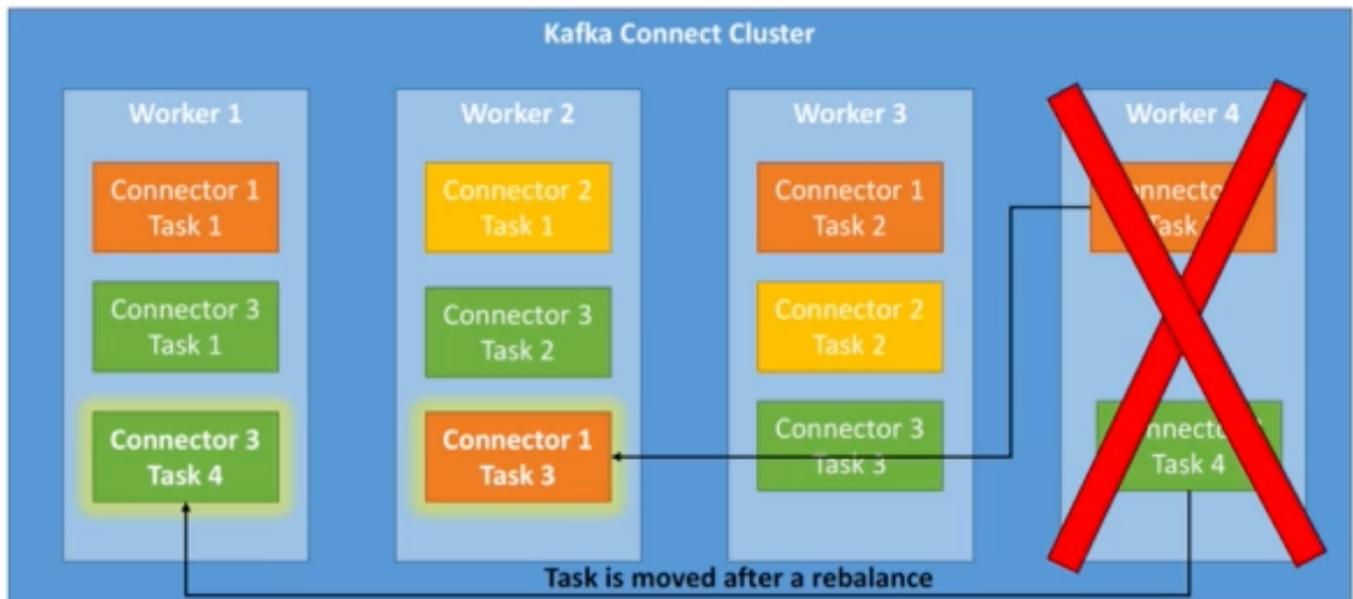


Unfortunately, one worker dies due to network error.

Kafka Connect Cluster Distributed Architecture in details



Now Kafka Connect Cluster does the automatic rebalancing. So, tasks exist in failed worker node moved to working worker node.

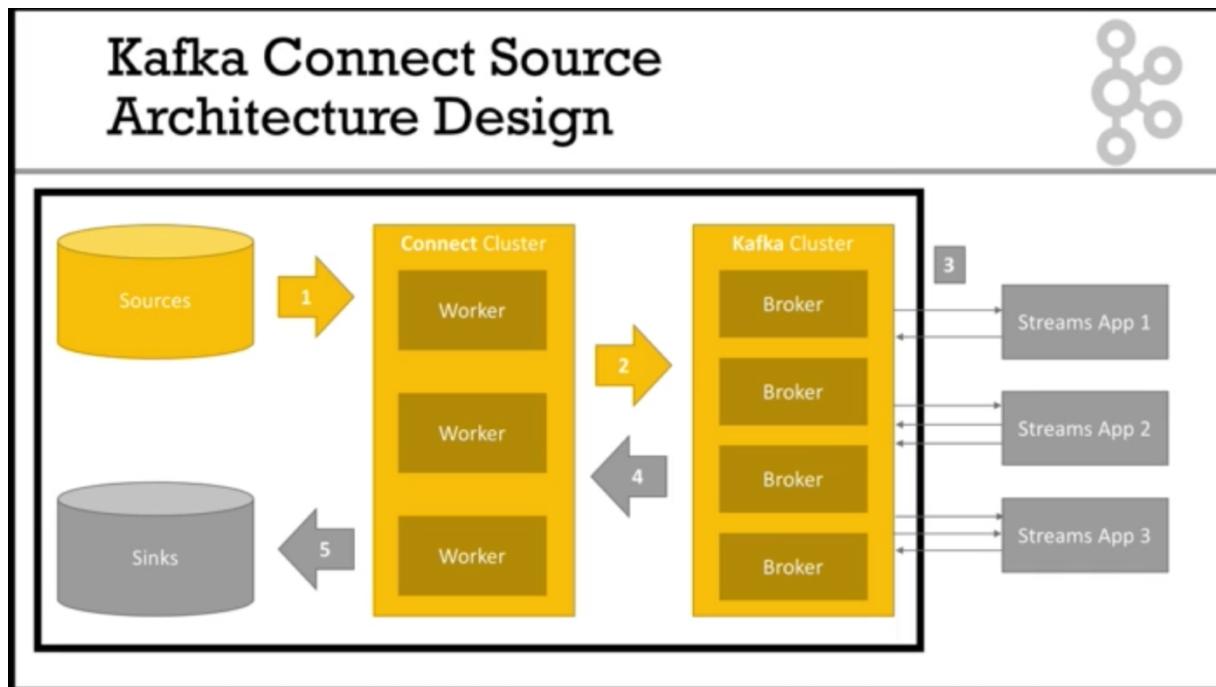


Docker

- Docker allows you to run containers.
- Containers are basically applications that have already been configured for you and that require no additional setup.

- It's the only way to ensure that an application will run the same way on every computer, and that's why we use it in this course.

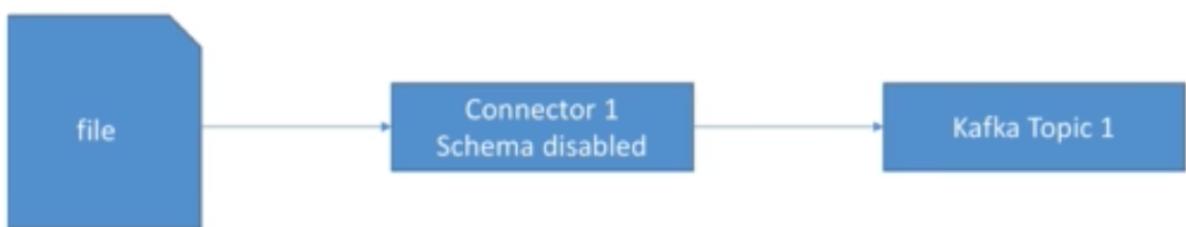
Kafka Connect Source



Standalone Mode

Example: FileStreamSourceConnector STANDALONE MODE

- Goal:
 - Read a file and load the content directly into Kafka
 - Run in a connector in **standalone mode** (useful for development)



- Learning:
 - Understand how to configure a connector in standalone mode
 - Get a first feel for Kafka Connect Standalone

Standalone Mode Demo

Step 1: docker-compose up kafka-cluster

```
docker-compose.yml      kafka-connect-tutorial-sinks.sh      kafka-connect-tutorial-sources.sh x
kafka-connect-tutorial-sources.sh
1  #!/bin/bash
2
3  # Make sure you change the ADV_HOST variable in docker-compose.yml
4  # if you are using docker Toolbox
5
6  # 1) Source connectors
7  # Start our kafka cluster
8  docker-compose up kafka-cluster
9  # Wait 2 minutes for the kafka cluster to be started
10
11 #####
12 # A) FileStreamSourceConnector in standalone mode
13 # Look at the source/demo-1/worker.properties file and edit bootstrap
14 # Look at the source/demo-1/file-stream-demo.properties file
15 # Look at the demo-file.txt file
16
17 # We start a hosted tools, mapped on our code
18 # Linux / Mac
19 docker run --rm -it -v "${pwd}":/tutorial --net=host landoop/fast-data-dev:cp3.3.0 bash
20 # Windows Command Line:
21 docker run --rm -it -v %cd%:/tutorial --net=host landoop/fast-data-dev:cp3.3.0 bash
22 # Windows Powershell:
23 docker run --rm -it -v ${PWD}:/tutorial --net=host landoop/fast-data-dev:cp3.3.0 bash

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE
/Users/naveen.kumar1/.zshrc:68: no matches found: load?
+ code3 docker-compose up kafka-cluster
Starting code3_kafka-cluster_1 ... done
Attaching to code3_kafka-cluster_1
kafka-cluster_1  Setting advertised host to 127.0.0.1.
kafka-cluster_1  Operating system RAM available is 1526 MiB, which is less than the lowest
kafka-cluster_1  recommended of 5120 MiB. Your system performance may be seriously impacted.
kafka-cluster_1  Starting services.
kafka-cluster_1  This is landoop's fast-data-dev. Kafka 0.11.0.0, Confluent OSS 3.3.0.
kafka-cluster_1  You may visit http://127.0.0.1:3030 in about a minute.
kafka-cluster_1  2021-09-11 07:12:31,393 CRIT Supervisor running as root (no user in config file)
kafka-cluster_1  2021-09-11 07:12:31,393 WARN Included extra file "/etc/supervisord.d/01-zookeeper.conf" during parsing
kafka-cluster_1  2021-09-11 07:12:31,393 WARN Included extra file "/etc/supervisord.d/02-broker.conf" during parsing
kafka-cluster_1  2021-09-11 07:12:31,393 WARN Included extra file "/etc/supervisord.d/03-schema-registry.conf" during parsing
kafka-cluster_1  2021-09-11 07:12:31,393 WARN Included extra file "/etc/supervisord.d/04-rest-proxy.conf" during parsing
kafka-cluster_1  2021-09-11 07:12:31,393 WARN Included extra file "/etc/supervisord.d/05-connect-distributed.conf" during parsing
kafka-cluster_1  2021-09-11 07:12:31,393 WARN Included extra file "/etc/supervisord.d/06-caddy.conf" during parsing
kafka-cluster_1  2021-09-11 07:12:31,393 WARN Included extra file "/etc/supervisord.d/07-smoke-tests.conf" during parsing
kafka-cluster_1  2021-09-11 07:12:31,393 WARN Included extra file "/etc/supervisord.d/08-logs-to-kafka.conf" during parsing
kafka-cluster_1  2021-09-11 07:12:31,393 WARN Included extra file "/etc/supervisord.d/99-supervisord-sample-data.conf" during parsing
kafka-cluster_1  2021-09-11 07:12:31,397 INFO supervisord started with pid 7
```

Step 2: Configuring property files.

Set up for infrastructure.

```
source > demo-1 > ⚙ worker.properties
1  # from more information, visit: http://docs.confluent.io/3.2.0/connect
2  bootstrap.servers=127.0.0.1:9092
3  key.converter=org.apache.kafka.connect.json.JsonConverter
4  key.converter.schemas.enable=false
5  value.converter=org.apache.kafka.connect.json.JsonConverter
6  value.converter.schemas.enable=false
7  # we always leave the internal key to JsonConverter
8  internal.key.converter=org.apache.kafka.connect.json.JsonConverter
9  internal.key.converter.schemas.enable=false
10 internal.value.converter=org.apache.kafka.connect.json.JsonConverter
11 internal.value.converter.schemas.enable=false
12 # Rest API
13 rest.port=8086
14 rest.host.name=127.0.0.1
15 # this config is only for standalone workers
16 offset.storage.file.filename=standalone.offsets
17 offset.flush.interval.ms=10000
18
```

It is required only for standalone mode. For distributed mode worker.properties is not required.

Set up for connector

```
source > demo-1 > ⚙ file-stream-demo-standalone.properties
1 # These are standard kafka connect parameters, need for ALL connectors
2 name=file-stream-demo-standalone
3 connector.class=org.apache.kafka.connect.file.FileStreamSourceConnector
4 tasks.max=1
5 # Parameters can be found here: https://github.com/apache/kafka/blob/trunk
6 file=demo-file.txt
7 topic=demo-1-standalone
8
```

Step 3: Run the container.

```
docker run --rm -it -v "$(pwd)":/tutorial --net=host landoop/fast-data-dev:cp3.3.0 bash
```

--rm – is used when you need the container to be deleted after the task for it is complete.

-it – Starts the container in the interactive mode (hence -it flag) that allows you to interact with /bin/bash of the container. That means now you will have bash session inside the container, so you can ls, mkdir, or do any bash command inside the container.

```
→ code3 docker run --rm -it -v "$(pwd)":/tutorial --net=host landoop/fast-data-dev:cp3.3.0 bash
root@fast-data-dev / $ ls /tutorial
ls: cannot access '/tutorial': No such file or directory
root@fast-data-dev / $ ls /tutorial
docker-compose.yml  kafka-connect-tutorial-sinks.sh  kafka-connect-tutorial-sources.sh  setup  sink  source
root@fast-data-dev / $ █
```

Step 4: Create a Kafka topic.

```
kafka-topics --create --topic demo-1-standalone --partitions 3 --replication-factor 1 --zookeeper
127.0.0.1:2181
```

```
root@fast-data-dev / $ kafka-topics --create --topic demo-1-standalone --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:2181
Created topic "demo-1-standalone".
```

Under the demo-1 folder we have a property file required for connector standalone.

```
root@fast-data-dev / $ cd /tutorial/source/demo-1
root@fast-data-dev demo-1 $ ls
demo-file.txt  file-stream-demo-standalone.properties  worker.properties
root@fast-data-dev demo-1 $ █
```

Step 5: Run standalone connector.

```
connect-standalone worker.properties file-stream-demo-standalone.properties
```

```

root@fast-data-dev demo-1 $ connect-standalone worker.properties file-stream-demo-standalone.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/confluent/share/java/kafka/sl4j-log4j12-1.7.25.jar!/org/sl4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/confluent/share/java/kafka-connect-hdfs/sl4j-log4j12-1.7.5.jar!/org/sl4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/confluent/share/java/kafka-connect-twitter/kafka-connect-twitter-0.1-master-af63e4c-cp3.3.0-jar-with-dependencies.jar!/org/sl4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

```

Since we redirected to the property files folder. We could able to pass the property file directly without giving the folder path.

```

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

ssl.keymanager.algorithm = SunX509
ssl.keystore.location = null
ssl.keystore.password = null
ssl.keystore.type = JKS
ssl.protocol = TLS
ssl.provider = null
ssl.secure.random.implementation = null
ssl.trustmanager.algorithm = PKIX
ssl.truststore.location = null
ssl.truststore.password = null
ssl.truststore.type = JKS
transaction.timeout.ms = 60000
transactional.id = null
value.serializer = class org.apache.kafka.common.serialization.ByteArraySerializer
(org.apache.kafka.clients.producer.ProducerConfig:223)
[2021-09-11 09:40:18,952] INFO Kafka version : 0.11.0.0-cp1 (org.apache.kafka.common.utils.AppInfoParser:83)
[2021-09-11 09:40:18,969] INFO Kafka commitId : 5cadaa94d0a69e0d (org.apache.kafka.common.utils.AppInfoParser:84)
[2021-09-11 09:40:20,565] INFO Source task WorkerSourceTask{id=file-stream-demo-standalone-0} finished initialization and start (org.apache.kafka.connect.runtime.WorkerSourceTask:143)
[2021-09-11 09:40:21,067] INFO Created connector file-stream-demo-standalone (org.apache.kafka.connect.cli.ConnectStandalone:91)

```

Step 6: Checking the output in UI

TOPIC	PARTITIONS	CONFIGURATION
backblaze_smart	2 Partitions × 1 Replication 2configs	...
logs_broker	1 Partitions × 1 Replication	...
nyc_yellow_taxi_trip_data	1 Partitions × 1 Replication 2configs	...
demo-1-standalone	3 Partitions × 1 Replication	...
reddit_posts	5 Partitions × 1 Replication 2configs	avro
sea_vessel_position_reports	3 Partitions × 1 Replication 2configs	avro

demo-1-standalone

DATA	PARTITIONS	CONFIGURATION
Topic is empty	3	

Topic is empty because our file is empty.

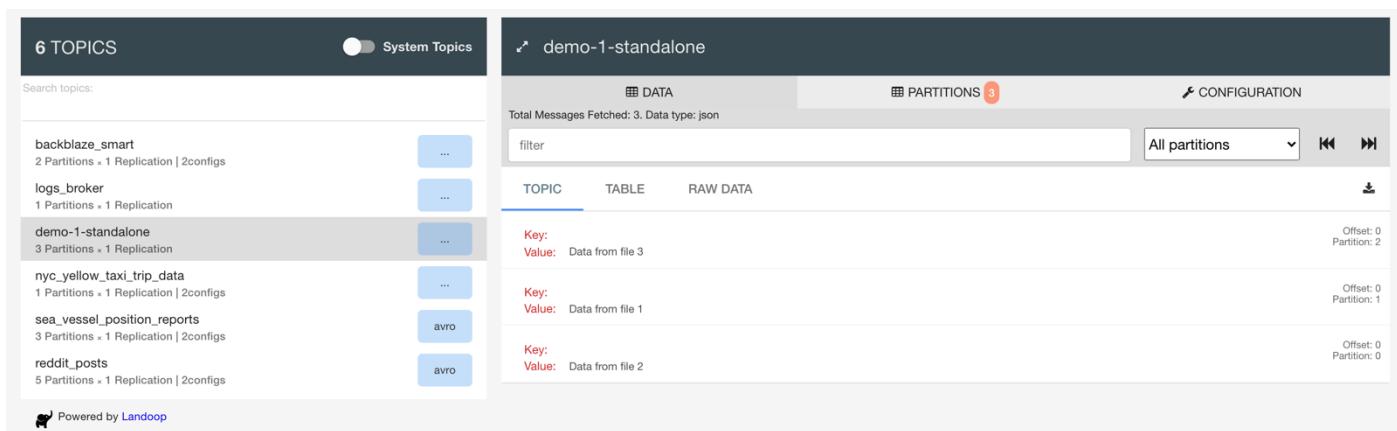
```
source > demo-1 >  demo-file.txt
```

```
1
```

Let's put some data in file check the topic data.

```
source > demo-1 >  demo-file.txt
```

```
1 Data from file 1  
2 Data from file 2  
3 Data from file 3  
4
```



The screenshot shows the Landoop Kafka UI interface. On the left, there is a list of topics: backblaze_smart, logs_broker, demo-1-standalone, nyc_yellow_taxi_trip_data, sea_vessel_position_reports, and reddit_posts. The 'demo-1-standalone' topic is selected and expanded. On the right, the 'demo-1-standalone' topic details page is displayed. It shows three partitions (3). The first partition has an offset of 0 and a partition of 2. The second partition has an offset of 0 and a partition of 1. The third partition has an offset of 0 and a partition of 0. The 'DATA' tab is selected, showing the following data entries:

TOPIC	TABLE	RAW DATA	Offset	Partition
demo-1-standalone		Key: Value: Data from file 3	0	2
demo-1-standalone		Key: Value: Data from file 1	0	1
demo-1-standalone		Key: Value: Data from file 2	0	0

It created the standalone.offsets file to store the offset.

```

CODE3
source > demo-1 > standalone.offsets
1   000 sr0 java.util.HashMap 000 '0 0 F0
2     loadFactorI0    thresholdxp7@00000 w000 000 ur0 [B00 0 T0 00x00<["file-stream-demo-standalone", {"filename": "demo-file.txt"}]uq0~0 000 {"posi

```

CODE3

- > setup
- > sink
- < source
 - demo-1
 - demo-file.txt
 - file-stream-demo-standalone.pro...
 - standalone.offsets**
 - worker.properties
 - demo-2
 - demo-3
 - docker-compose.yml
 - kafka-connect-tutorial-sinks.sh
 - kafka-connect-tutorial-sources.sh

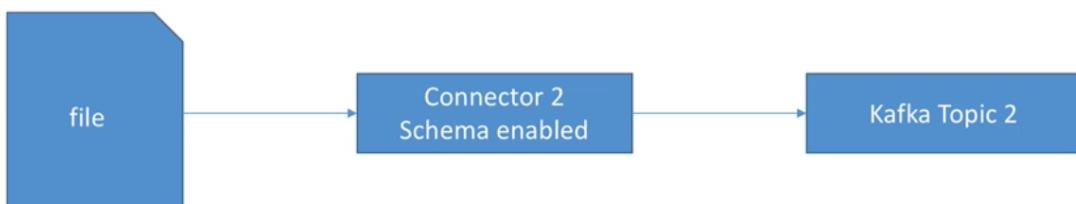
Distributed Mode

Example: FileStreamSourceConnector DISTRIBUTED MODE



- Goal:

- Read a file and load the content directly into Kafka
- Run in **distributed mode** on our already set-up Kafka Connect Cluster



- Learning:

- Understand how to configure a connector in distributed mode
- Get a first feel for Kafka Connect Cluster
- Understand the schema configuration option

Step 1: Run the landoop tool and create a topic.

```
docker run --rm -it --net=host landoop/fast-data-dev:cp3.3.0 bash
```

```
→ code3 docker run --rm -it --net=host landoop/fast-data-dev:cp3.3.0 bash
root@fast-data-dev / $
```

Create a topic named demo-2-distributed.

```
kafka-topics --create --topic demo-2-distributed --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:2181
```

```
root@fast-data-dev / $ kafka-topics --create --topic demo-2-distributed --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:2181
Created topic "demo-2-distributed".
root@fast-data-dev / $
```

As we can see that topic we had created, reflected in the UI.

The screenshot shows the Kafka Topics UI. On the left, a list of topics is displayed with their details. One topic, 'demo-2-distributed', is highlighted with a blue oval. On the right, a detailed view of the 'demo-2-distributed' topic is shown, with tabs for DATA, PARTITIONS (3), and CONFIGURATION. The DATA tab shows a message: 'Topic is empty'. The PARTITIONS tab has a red notification badge with the number '3'. The CONFIGURATION tab is visible at the top right.

Step 2: Create a connector cluster.

From the dashboard, go to connector UI and click 'NEW' button for creating a new connector.

The screenshot shows the Kafka Connect UI. On the left, a list of connectors is shown with one named 'logs-broker'. A 'NEW' button is highlighted with a yellow oval. On the right, a 'Create New Connector' dialog is open. It shows a configuration snippet for a 'File' connector. The 'File' icon is selected. Below the code, there are two checkboxes: 'Show cURL command' and 'Show Optional fields'. At the bottom, there are two red error messages: 'Config "file" requires a value' and 'Config "topic" requires a value'. A 'CREATE' button is located at the bottom right.

Next, provide the connector properties and click on create.

The screenshot shows the Kafka Connect UI. On the left, a list of connectors is shown with one named 'logs-broker'. A 'NEW' button is visible. On the right, a 'Create New Connector' dialog is open. The 'File' icon is selected. The configuration code now includes specific properties: 'home=file-stream-demo-distributed', 'connector.class=org.apache.kafka.connect.file.FileStreamSourceConnector', 'tasks.max=1', 'file=demo-file.txt', 'topic=demo-2-distributed'. Below the code, the 'Show Optional fields' checkbox is checked. At the bottom, a 'CREATE' button is highlighted with a red oval. There are also some warning messages at the very bottom: 'Kafka Connect: Configuration is valid.', 'Warning: Config "key.converter.schemas.enable" is unknown', and 'Warning: Config "value.converter.schemas.enable" is unknown'.

Now our connector created successfully.

The screenshot shows the Kafka Connect dashboard with the following details:

- 2 Connectors**: The dashboard header indicates there are 2 connectors.
- Dashboard**: A central area showing metrics:
 - SINK CONNECTORS**: 0
 - SOURCE CONNECTORS**: 2
 - TOPICS USED BY CONNECTORS**: 2
- Search connectors**: A search bar with placeholder "Search connectors". Below it, two connectors are listed:
 - file-stream-demo-distributed**: Status 1x, with a "File" icon and a "Logs" icon.
 - logs-broker**: Status 1x, with a "File" icon and a "Logs" icon.
- System Information**:
 - Kafka Connect : /api/kafka-connect
 - Kafka Connect Version : 0.11.0.0-cp1
 - Kafka Connect UI Version : 0.9.2
- Connect topology**: A diagram showing the flow from "file-stream-demo-distributed" to "demo-2-distributed" and from "logs-broker" to "logs_broker".
- Powered by LANDOOP**: A small logo at the bottom left.

Step 3: Creating a source file inside the Kafka Connect Cluster.

Identifying docker container id.

docker ps

```
+ code3 docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
 NAMES
545ee6112c13        landoop/fast-data-dev:cp3.3.0   "/usr/local/bin/dumb..."   44 hours ago       Up 29 minutes    0.0.0.0:2181->2181/tcp, :::2181->2181/tcp, 0.
.0.0.0:3030->3030/tcp, :::3030->3030/tcp, 0.0.0.0:8081-8083->8081-8083/tcp, :::8081-8083->8081-8083/tcp, 0.0.0.0:9092->9092/tcp, :::9092->9092/tcp, 0.
.0.0.0:9581-9585->9581-9585/tcp, :::9581-9585->9581-9585/tcp, 3031/tcp   code3_kafka-cluster_1
+ code3
```

Next, we are creating the file inside the container and feeding the data.

```
docker exec -it 545ee6112c13 bash
touch demo-file.txt
echo "hi" >> demo-file.txt
echo "hello" >> demo-file.txt
echo "from the other side" >> demo-file.txt
```

```
+ code3 docker exec -it 545ee6112c13 bash
root@fast-data-dev ~ $ touch demo-file.txt
root@fast-data-dev ~ $ ls
bin      connectors     dev extra-connect-jars lib      mnt      proc run      srv tmp      var
build.info  demo-file.txt etc home          media opt      root sbin sys      usr
root@fast-data-dev ~ $ echo "hi" >> demo-file.txt
root@fast-data-dev ~ $ echo "hello" >> demo-file.txt
root@fast-data-dev ~ $ echo "from the other side" >> demo-file.txt
root@fast-data-dev ~ $
```

Lp 48 Col 1

Step 3: Checking the topic to see the feed data.

KAFKA TOPICS

7 TOPICS System Topics

Search topics:

backblaze_smart	2 Partitions × 1 Replication 2configs	...
demo-2-distributed	3 Partitions × 1 Replication	...
logs_broker	1 Partitions × 1 Replication	...
demo-1-standalone	3 Partitions × 1 Replication	...
nyc_yellow_taxi_trip_data	1 Partitions × 1 Replication 2configs	...
sea_vessel_position_reports	3 Partitions × 1 Replication 2configs	avro
reddit_posts	5 Partitions × 1 Replication 2configs	avro

demo-2-distributed

DATA PARTITIONS 3 CONFIGURATION

Total Messages Fetched: 3. Data type: json

filter All partitions

TOPIC	TABLE	RAW DATA
		<ul style="list-style-type: none"> ▶ Key: { schema: null, payload: null } ▶ Value: { schema: [object Object], payload: hi }
		<ul style="list-style-type: none"> ▶ Key: { schema: null, payload: null } ▶ Value: { schema: [object Object], payload: hello }
		<ul style="list-style-type: none"> ▶ Key: { schema: null, payload: null } ▶ Value: { schema: [object Object], payload: from the other side }

Offset: 0 Partition: 0
Offset: 1 Partition: 0
Offset: 2 Partition: 0

Powered by Landoop

Step 4: Checking output for validating the schema configuration.

As per our connector's configuration, we can see the data in json format.

```
docker run --rm -it --net=host landoop/fast-data-dev:cp3.3.0 bash
kafka-console-consumer --topic demo-2-distributed --from-beginning --bootstrap-server 127.0.0.1:9092
```

```
root@fast-data-dev / $ exit
exit
→ code3 docker run --rm -it --net=host landoop/fast-data-dev:cp3.3.0 bash
root@fast-data-dev / $ kafka-console-consumer --topic demo-2-distributed --from-beginning --bootstrap-server 127.0.0.1:9092
{"schema": {"type": "string", "optional": false}, "payload": "hi"}
{"schema": {"type": "string", "optional": false}, "payload": "hello"}
{"schema": {"type": "string", "optional": false}, "payload": "from the other side"}
```

List of Available Connectors

List of available connectors

- There are many source and sinks connectors
- The best place to find them is the Confluent website:

<https://www.confluent.io/hub/>

- The second best place to find them is Google 😊

Connectors in Landoop distribution



- Many connectors are bundled with the Landoop Docker image so you can use them right away
- If you want to add your own, you need to map your volumes to your Landoop Docker image on start-up.
- More information:

<https://github.com/Landoop/fast-data-dev#enable-additional-connectors>

Twitter Source Connector in Distributed Mode

TwitterSourceConnector



- Goal:

- Gather data from Twitter in Kafka Connect Distributed mode



- Learning:

- Gather real data using <https://github.com/Eneco/kafka-connect-twitter>

Step 1: Create a twitter developer account and create an app.

Developer Portal

Docs Community Updates Support

Dashboard

Projects & Apps

Overview

Kafka Pet Project

Products NEW

Account

Standard

Kafka Pet Project >

MONTHLY TWEET CAP USAGE 0%
0 Tweets pulled of 500,000 Resets on October 4 at 00:00 UTC

+Add App

Spaces endpoints are here!
Help people discover Spaces they'll love and make sure hosts reach the broadest audience possible.
Get Spaces endpoints

Opinions, we all have them...
So let's hear yours. Take the annual developer survey.

PRIVACY COOKIES TWITTER TERMS & CONDITIONS DEVELOPER POLICY & TERMS © 2021 TWITTER INC. FOLLOW @TWITTERDEV SUBSCRIBE TO DEVELOPER NEWS



Name your App

1 App name 2 Keys & Tokens

Apps are where you get your access **keys & tokens**, plus set permissions. You can find them within your Projects.

Kafka Pet Project

15

Back

Next

For security, this will be the last time we'll display these. If something happens, you can always regenerate them. [Learn more](#)

API Key ⓘ

gPCOPTOR7IGYH7GkFyNY8QqxC

[Copy](#) **API Key Secret** ⓘ

nSgzgjNMfFqG4SOurlb2vKkgI6mYHj3zZUf0caoN0Kb71o0RuS

[Copy](#) **Bearer Token** ⓘAAAAAAAAAAAAAAAANG5TgEAAAAAZqetv6IRKacQydHu%2B
L2SdClc0B8%3D8UQ2X10tY5cptNxCR3CUIN6QfTA8lolpPbSoclC2JgP
ybw2vux[Copy](#)

Next, setup your App.

[Go to dashboard](#)[App settings](#)

Dashboard

Standard

[Kafka Pet Project >](#)

MONTHLY TWEET CAP USAGE ⓘ

0 Tweets pulled of 500,000

0%

Resets on October 4 at 00:00 UTC

PROJECT APP



Kafka Pet Project



Kafka Pet Project

Settings Keys and tokens

Consumer Keys ⓘ

API Key and Secret

Regenerate

Authentication Tokens ⓘ

Bearer Token

Generated September 14, 2021

Regenerate

Revoke

Access Token and Secret

For @mnavveensmn

Generate

New Access Token and Secret

Did you save your API Key and Key Secret?

For security, this will be the last time we'll display these. If something happens, you can always regenerate them. [Learn more](#)

API Key

gPCOPTOR7IGYH7GkFyNY8QqxC

Copy 

API Key Secret

nSgzgjNMfFqG4SOurlb2vKkgI6mYHj3zZUf0caoN0Kb71o
0RuS

Copy 

Bearer Token

AAAAAAAAAAAAAAAAANG5TgEAAAАЗqetv6IRKa
AAAAAAAAAAAAAAAAANG5TgEAAAАЗqetv6IRKa

Cancel

Yes, I saved them

Step 2: Configure a property for 'TwitterSourceConnector' configuration.

```

source > demo-3 > ⚙ source-twitter-distributed.properties
1 # Basic configuration for our connector
2 name=source-twitter-distributed
3 connector.class=com.eneco.trading.kafka.connect.twitter.TwitterSourceConnector
4 tasks.max=1
5 topic=demo-3-twitter
6 key.converter=org.apache.kafka.connect.json.JsonConverter
7 key.converter.schemas.enable=true
8 value.converter=org.apache.kafka.connect.json.JsonConverter
9 value.converter.schemas.enable=true
10 # Twitter connector specific configuration
11 twitter.consumerkey=gPCOPTOR7lGYH7GkFyNY8Qqx
12 twitter.consumersecret=nSgzgjNMfFqG4S0urlb2vKkgI6mYHj3zZUf0caoN0Kb71o0RuS
13 twitter.token=1434035052732583938-y8gN6T6fqHTN1Yg8XGn18Gs0UVKgLV
14 twitter.secret=MqcWZnKcvYojisAdpH6J8JQGLLBnHIKnoVxnneJmSgBNu
15 track.terms=programming,java,kafka,scala
16 language=en
17

```

Step 3: Create a topic named “demo-3-twitter”

```

→ code3 docker run --rm -it --net=host landoop/fast-data-dev:cp3.3.0 bash
root@fast-data-dev / $ kafka-topics --create --topic demo-3-twitter --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:2181
Created topic "demo-3-twitter".

```

Step 4: Create a new Twitter connector in Landoop UI.

In Landoop main page, click ‘ENTER’ in the Connector card.

Click on ‘NEW’.

KAFKA CONNECT

2 Connectors

NEW

Dashboard

SINK CONNECTORS 0

SOURCE CONNECTORS 2

TOPICS USED BY CONNECTORS 2

Connect topology

Powered by LANDOOP

Click on 'Twitter'.

KAFKA CONNECT

2 Connectors

NEW

New Connector

Search

Sources

Sinks

Twitter Use the Twitter API to stream data into Kafka

Yahoo Finance Stream stock and currency exchange rates into Kafka

File Tail files or folders and stream data into Kafka

Blockchain Get blockchain.info data into Kafka

Jdbc Stream data from SQL server into Kafka

Cassandra Extract Cassandra data using the CQL driver into Kafka

Bloomberg

InfluxDB Store Kafka data into InfluxDB

MongoDB Write Kafka data into MongoDB

HazelCast Store Kafka data into HazelCast (RingBuffer)

Jdbc Store Kafka data into SQL

Amazon S3

Paste the properties created in the step 2.

KAFKA CONNECT

2 Connectors

NEW

Create New Connector

Twitter → Kafka

class: com.eneeco.trading.kafka.connect.twitter.TwitterSourceConnector

```

1 name=TwitterSourceConnector
2 connector.class=com.eneeco.trading.kafka.connect.twitter.TwitterSourceConnector
3 tasks.max=1
4 twitter.consumerkey=
5 twitter.consumersecret=
6 twitter.token=
7 twitter.secret=

```

HINTS

Show cURL command Show Optional fields

CREATE

Config "twitter.consumerkey" requires a value
 Config "twitter.consumersecret" requires a value
 Config "twitter.token" requires a value
 Config "twitter.secret" requires a value
 Required config "topic" is not there

Powered by LANDOOP

Click on 'CREATE'.

KAFKA CONNECT

2 Connectors

NEW

Search connectors

file-stream-demo-distributed	✓ 1x	<input type="button" value="File"/> → <input type="button" value="SS"/>
logs-broker	✓ 1x	<input type="button" value="File"/> → <input type="button" value="SS"/>

Kafka Connect : /api/kafka-connect
Kafka Connect Version : 0.11.0.0-cp1
Kafka Connect UI Version : 0.9.2

Create New Connector

HINTS

Twitter → Kafka
 class: com.eneco.trading.kafka.connect.twitter.TwitterSourceConnector

```

1 name=source-twitter-distributed
2 connector.class=com.eneco.trading.kafka.connect.twitter.TwitterSourceConnector
3 tasks.max=1
4 topic=demo-3-twitter
5 key.converter=org.apache.kafka.connect.json.JsonConverter
6 key.converter.schemas.enable=true
7 value.converter=org.apache.kafka.connect.json.JsonConverter
8 value.converter.schemas.enable=true
9 twitter.consumerkey=gPCOPTOR7LGYH7GKfNY8QqxC
10 twitter.consumersecret=nSzgjNMFFg4q4SDur1b2vKkgI6mYHj3zZUF0caoN0Kb71o0RuS
11 twitter.token=1434035052732583938-ybgM6T6fqHTN1YgXKn18G50UVkLV
12 twitter.secret=MqCNzNkcv0j1SadpHGJ8J0GLLBHIIkn0VxneJM5gBNu
13 track.terms=programming,java,kafka,scala
14 language=en
15
  
```

Show cURL command Show Optional fields

CREATE

Powered by [LANDOOP](#)

Now connector is created.

KAFKA CONNECT

3 Connectors

NEW

Search connectors

file-stream-demo-distributed	✓ 1x	<input type="button" value="File"/> → <input type="button" value="SS"/>
logs-broker	✓ 1x	<input type="button" value="File"/> → <input type="button" value="SS"/>
source-twitter-distributed	✓ 1x	Twitter → <input type="button" value="SS"/>

Kafka Connect : /api/kafka-connect
Kafka Connect Version : 0.11.0.0-cp1
Kafka Connect UI Version : 0.9.2

Dashboard

EXPORT CONFIG

SINK CONNECTORS: 0

SOURCE CONNECTORS: 3

TOPICS USED BY CONNECTORS: 3

Connect topology

```

graph TD
    S1[source-twitter-distributed] --- D1[demo-3-twitter]
    S2[file-stream-demo-distributed] --- D1
    S3[logs-broker] --- D1
  
```

Powered by [LANDOOP](#)

Step 5: Checking for topic in UI.

As we can see, we are getting the data from twitter to the specified topic.

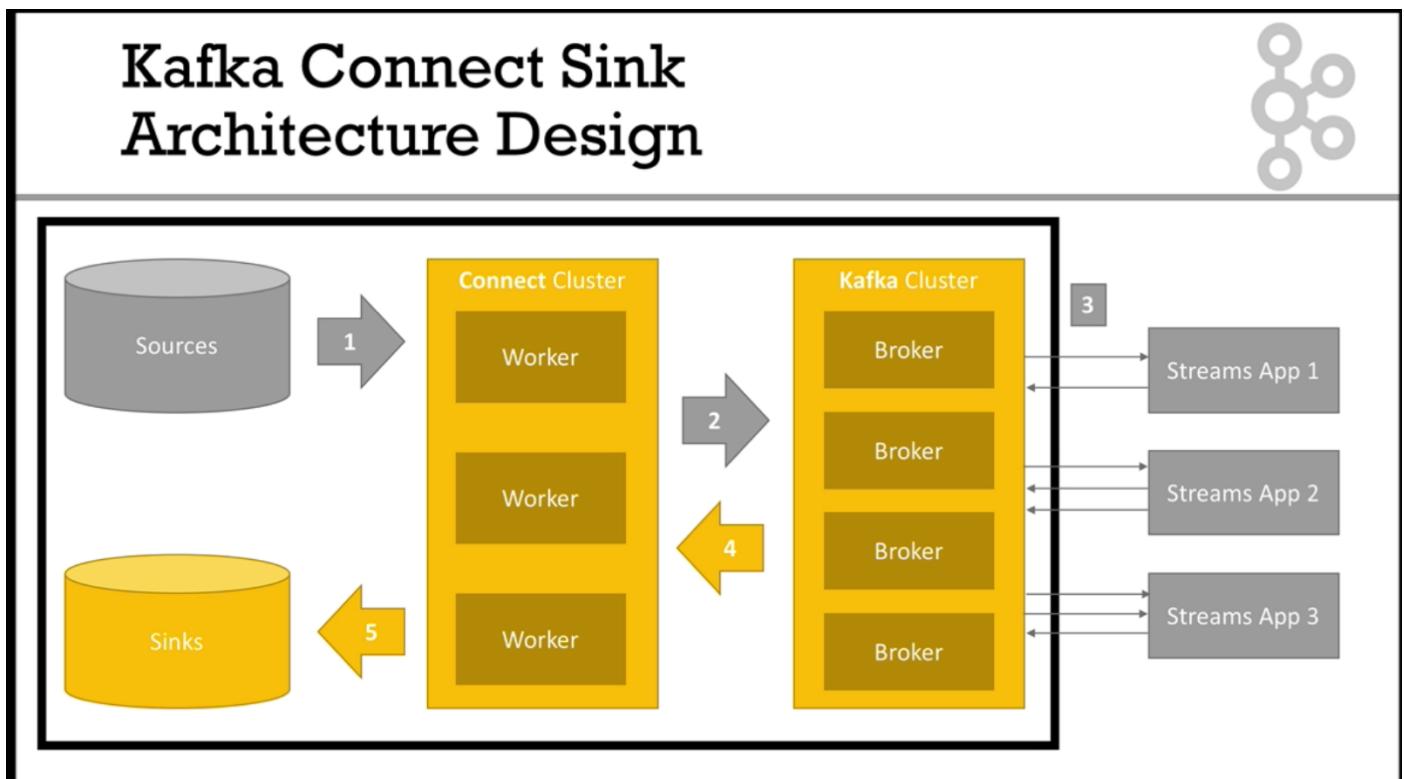
What we learned

What we learned

- How to run Kafka connectors in
 - Standalone mode (good for development)
 - Distributed mode (good for production)
 - Where to find Kafka connectors
 - How to pull data from Twitter directly into a Kafka topic
 - Now, you can go online and learn how to use other connectors!

Kafka Connector Sink

Architecture



Elastic Search Sink Connector

ElasticSearch

- We're going to start an ElasticSearch instance to Sink the data to
 - We will use Docker to start our ElasticSearch instance
 - This will serve as our Sink for our first Sink Connector
-
- ElasticSearch is an easy way to store json data and search across it
 - Used by many companies around the world to power search engines and advanced analytics capabilities
 - No knowledge of ElasticSearch is required for this course



Example: ElasticSearchSink DISTRIBUTED MODE

- Goal:

- Start an ElasticSearch instance using Docker
- Sink a topic with multiple partitions to ElasticSearch
- Run in distributed mode with multiple tasks



- Learning:

- Learn about the `tasks.max` parameter
- Understand how Sink Connectors work

Step 1: Running Docker File

```

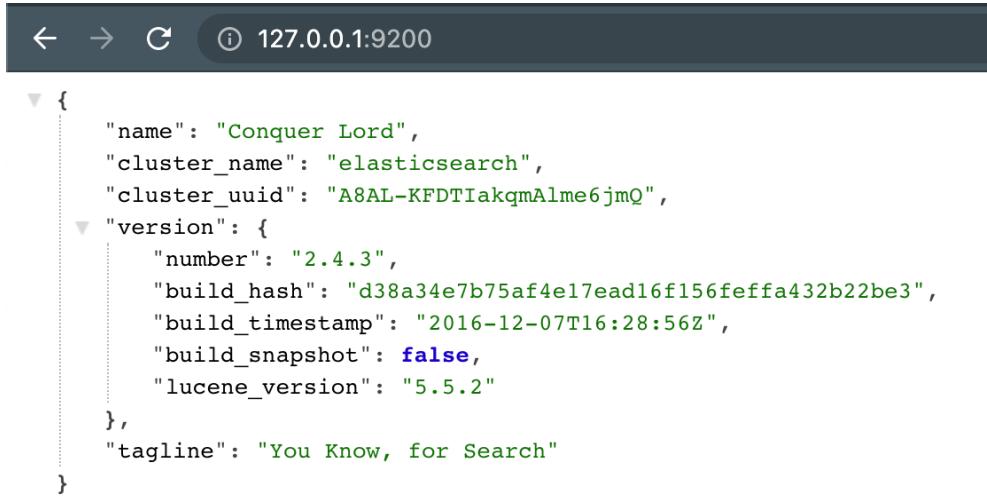
1  docker-compose.yml
2
3  version: '2'
4
5  services:
6    # this is our kafka cluster.
7    kafka-cluster:
8      image: landoop/fast-data-dev:cp3.3.0
9      environment:
10        ADV_HOST: 127.0.0.1          # Change to 192.168.99.100 if using Docker Toolbox
11        RUNTESTS: 0                # Disable Running tests so the cluster starts faster
12      ports:
13        - 2181:2181                 # Zookeeper
14        - 3030:3030                 # Landoop UI
15        - 8081-8083:8081-8083     # REST Proxy, Schema Registry, Kafka Connect ports
16        - 9581-9585:9581-9585     # JMX Ports
17        - 9092:9092                 # Kafka Broker
18
19    # we will use elasticsearch as one of our sinks.
20    # This configuration allows you to start elasticsearch
21    elasticsearch:
22      image: itzg/elasticsearch:2.4.3
23      environment:
24        PLUGINS: appbaseio/dejavu
25        OPTS: -Dindex.number_of_shards=1 -Dindex.number_of_replicas=0
26      ports:
27        - "9200:9200"
28
29    # we will use postgres as one of our sinks.
30    # This configuration allows you to start postgres
31    postgres:
32      image: postgres:9.5-alpine
33      environment:
34        POSTGRES_USER: postgres      # define credentials
35        POSTGRES_PASSWORD: postgres # define credentials
36        POSTGRES_DB: postgres      # define database
37      ports:
38        - 5432:5432                 # Postgres port
  
```

```
docker-compose up kafka-cluster elasticsearch postgres
```

```
→ code3 docker-compose up kafka-cluster elasticsearch postgres
Pulling elasticsearch (itzg/elasticsearch:2.4.3)...
2.4.3: Pulling from itzg/elasticsearch
b7f33cc0b48e: Pull complete
43a564ae36a3: Pull complete
efb75a810eee: Downloading [=====] 8.936MB/39.67MB
b0aeda70fb4d: Downloading [=====] 4.172MB/27.35MB
d99b5af0dd89: Downloading [==>] 1.407MB/27.34MB
8c3b2ac50d38: Waiting
8e45dd41801a: Waiting
```

Step 2: Checking whether ElasticSearch working fine or not.

<http://127.0.0.1:9200/>



```
{
  "status": "green",
  "shards_at_risk": 0,
  "shards_total": 0
}
```

```
{
  "name": "Conquer Lord",
  "cluster_name": "elasticsearch",
  "cluster_uuid": "A8AL-KFDTIakqmAlme6jmQ",
  "version": {
    "number": "2.4.3",
    "build_hash": "d38a34e7b75af4e17ead16f156feffa432b22be3",
    "build_timestamp": "2016-12-07T16:28:56Z",
    "build_snapshot": false,
    "lucene_version": "5.5.2"
  },
  "tagline": "You Know, for Search"
}
```

Step 3: Prepare properties for Elastic Search Sink Connector

```
sink > demo-elasticsearch > ⚙ sink-elastic-twitter-distributed.properties
1 # Basic configuration for our connector
2 name=sink-elastic-twitter-distributed
3 connector.class=io.confluent.connect.elasticsearch.ElasticsearchSinkConnector
4 # We can have parallelism here so we have two tasks!
5 tasks.max=2
6 topics=demo-3-twitter
7 # the input topic has a schema, so we enable schemas conversion here too
8 key.converter=org.apache.kafka.connect.json.JsonConverter
9 key.converter.schemas.enable=true
10 value.converter=org.apache.kafka.connect.json.JsonConverter
11 value.converter.schemas.enable=true
12 # ElasticSearch connector specific configuration
13 # # http://docs.confluent.io/3.3.0/connect/connect-elasticsearch/docs/configuration_options.html
14 connection.url=http://elasticsearch:9200
15 type.name=kafka-connect
16 # because our keys from the twitter feed are null, we have key.ignore=true
17 key.ignore=true
18 # some (dummy) settings we need to add to ensure the configuration is accepted
19 # The bug is tracked at https://github.com/Landoop/fast-data-dev/issues/42
20 topic.index.map="demo-3-twitter:index1"
21 topic.key.ignore=true
22 topic.schema.ignore=true
23
```

Step 4: Create a new Elastic Search Sink Connector in landoop UI.

The screenshot shows the Kafka Connect UI interface. At the top, there's a header 'KAFKA CONNECT' and a navigation bar with '3 Connectors' and a 'NEW' button (circled in red). Below this is a search bar labeled 'Search'. On the left, under 'Sources', there are three connectors: 'file-stream-demo-distributed', 'logs-broker', and 'source-twitter-distributed'. On the right, under 'Sinks', there is a list of various connectors: Twitter, Yahoo Finance, File, Blockchain, Jdbc, Cassandra, Bloomberg, Elastic Search, Cassandra, InfluxDB, MongoDB, HazelCast, Jdbc, and Amazon S3. The 'ElasticSearch (confluent)' connector is circled in red at the bottom of the sink list. The status bar at the bottom shows 'Kafka Connect : /api/kafka-connect', 'Kafka Connect Version : 0.11.0.0-cp1', and 'Kafka Connect UI Version : 0.9.2'.

Paste the properties created in step 3. And click on click on create.

The screenshot shows the 'Create New Connector' dialog. The title is 'Create New Connector' with a 'HINTS' link. The configuration code is:

```

1 name=sink-elasticsearch-twitter-distributed
2 connector.class=io.confluent.connect.elasticsearch.ElasticsearchSinkConnector
3 tasks.max=2
4 topics=demo-3-twitter
5 key.converter=org.apache.kafka.connect.json.JsonConverter
6 key.converter.schemas.enable=true
7 value.converter=org.apache.kafka.connect.json.JsonConverter
8 value.converter.schemas.enable=true
9 connection.url=http://elasticsearch:9200
10 type.name=kafka-connect
11 key.ignore=true
12 topic.index.map="demo-3-twitter:index1"
13 topic.key.ignore=true
14 topic.schema.ignore=true
15

```

Below the code, there are two checkboxes: 'Show cURL command' and 'Show Optional fields'. At the bottom, it says 'Kafka Connect: Configuration is valid.' and 'Warning: Config "key.converter.schemas.enable" is unknown' and 'Warning: Config "value.converter.schemas.enable" is unknown'. There are 'CREATE' and 'Cancel' buttons at the bottom right.

KAFKA CONNECT

4 Connectors

NEW

Search connectors

- file-stream-demo-distributed: 1x → [File]
- logs-broker: 1x → [File]
- sink-elastic-twitter-distributed: 2x → [ElasticSearch (confluent)]**
- source-twitter-distributed: 1x → [Twitter] → [File]

Kafka Connect : /api/kafka-connect
Kafka Connect Version : 0.11.0.0-cp1
Kafka Connect UI Version : 0.9.2

sink-elastic-twitter-distributed

Kafka → ElasticSearch (confluent) is RUNNING

TASKS: 0 [127.0.0.1:8083] | 1 [127.0.0.1:8083]

TOPICS

CONFIGURATION

EDIT

```

1+ {
2   "connector.class": "io.confluent.connect.elasticsearch.ElasticsearchSinkConnector",
3   "type.name": "kafka-connect",
4   "tasks.max": "2",
5   "topics": "demo-3-twitter",
6   "key.ignore": "true",
7   "key.converter.schemas.enable": "true",
8   "topic.index.map": "demo-3-twitter:index1",
9   "topic.key.ignore": "true",
10  "value.converter.schemas.enable": "true",
11  "name": "sink-elastic-twitter-distributed",
12  "value.converter": "org.apache.kafka.connect.json.JsonConverter",
13  "connection.url": "http://elasticsearch:9200",
14  "key.converter": "org.apache.kafka.connect.json.JsonConverter",
15  "topic.schema.ignore": "true"
16 }

```

Step 5: Checking twitter data in Elasticsearch UI.

http://127.0.0.1:9200/_plugin/dejavu/



Dejavu
The Missing Web UI for Elasticsearch

Query View demo-3-twitter http://127.0.0.1:9200

Types Queries Reload Showing 20 of total 2599

	type / id	id	created_at	
<input checked="" type="checkbox"/> kafka-connect	[...] kafka-connect / demo-...	1437663038593802200	2021-09-14T06:22:33.000+0000	...
	[...] kafka-connect / demo-...	1437663042284707800	2021-09-14T06:22:34.000+0000	...
	[...] kafka-connect / demo-...	1437663045640147000	2021-09-14T06:22:35.000+0000	...
	[...] kafka-connect / demo-...	1437663050333626400	2021-09-14T06:22:36.000+0000	...

user

```

{
  "id": 1319903481486950400,
  "name": "TensorFlow Bot",
  "screen_name": "bot_tensorflow",
  "location": null,
  "verified": false,
  "friends_count": 1,
  "followers_count": 141,
  "statuses_count": 27579
}

```

RT @AmitChampaneri1: ⚡ insight! #EthicalAI Application Pyramid via @ingliuori 🇺🇸 #BigData #Analytics #DataScience #MachineLearning #IoT #IoT #PyTorch #Py...

RT @gp_pulipaka: Top 10 Machine Learning #Books That You Should Read Before You Start With It. #BigData #Analytics #DataScience #IoT #IoT...

RT @VeilleCyber3: How the NationalScienceFoundation is taking on #fairness in #AI https://t.co/hdZqTOPF2S #programming #Data #DataScienc...

Step 6: Querying from Dejavu UI.

Creating a query.

Add Query

Name: Only Retweet

Type (aka table): kafka-connect

Query body (JSON)

```

1 v {
2 v   "query":{
3 v     "term" : {
4 v       "is_retweet" : true
5 v     }
6 v   }
7 v }
```

Add

Add Query

Name: High Friends Count

Type (aka table): kafka-connect

Query body (JSON)

```

1 v {
2 v   "query":{
3 v     "range" : {
4 v       "user.friends_count" : {
5 v         "gt" : 500
6 v       }
7 v     }
8 v   }
9 v }
```

Add

Query View

demo-3-twitter

http://127.0.0.1:9200

	type / id	id	created_at	user	text
Only Retweet	[...]	1437663050333626400	2021-09-14T06:22:36.000+0000	[...]	RT @Doxaxone: #AWS Announces Amazon S3 Plugin for #PyTorch. #BigData #Analytics #DataScience #AI #MachineLearning #IoT #IoT #Python #RSta...
High Friends Count	[...]	1437663068184526800	2021-09-14T06:22:40.000+0000	[...]	RT @gp_pulipaka: Top 10 Machine Learning #Books That You Should Read Before You Start With It. #BigData #Analytics #DataScience #IoT #IoT...
	[...]	1437663099046203400	2021-09-14T06:22:47.000+0000	[...]	Five Best Programming Language for Mobile App Development - https://t.co/vErTPAOv best programming language, a... https://t.co/USIcc0i0Yr...
	[...]	1437663117765316600	2021-09-14T06:22:52.000+0000	[...]	RT @BR_Softech_AU: Let me tell you best 5 Reasons why #Go @golang is vastly implemented in DevOps. https://t.co/r9rxrk7r #GoLang #...



Kafka Connect REST API

- All the actions performed by Landoop Kafka Connect UI are actually triggering REST API calls to Kafka Connect.
- Let's learn about those
(<http://docs.confluent.io/3.2.0/connect/managing.html#common-rest-examples>)

- | | |
|--|--|
| 1. Get Worker information
2. List Connectors available on a Worker
3. Ask about Active Connectors
4. Get information about a Connector Tasks and Config | 5. Get Connector Status
6. Pause / Resume a Connector
7. Delete our Connector
8. Create a new Connector
9. Update Connector configuration
10. Get Connector Configuration |
|--|--|

Setting up the command line

```
# let's start a command line to all have linux commands
docker run --rm -it --net=host landoop/fast-data-dev:cp3.3.0 bash
# Install jq to pretty print json
apk update && apk add jq
```

```
→ code3 docker run --rm -it --net=host landoop/fast-data-dev:cp3.3.0 bash
root@fast-data-dev / $ apk update && apk add jq
fetch http://dl-cdn.alpinelinux.org/alpine/v3.6/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.6/community/x86_64/APKINDEX.tar.gz
v3.6.5-44-gda55e27396 [http://dl-cdn.alpinelinux.org/alpine/v3.6/main]
v3.6.5-34-gf0ba0b43d5 [http://dl-cdn.alpinelinux.org/alpine/v3.6/community]
OK: 8468 distinct packages available
(1/2) Installing oniguruma (6.3.0-r0)
(2/2) Installing jq (1.5-r4)
Executing busybox-1.26.2-r7.trigger
OK: 140 MiB in 63 packages
root@fast-data-dev / $
```

Get Worker information

```
curl -s 127.0.0.1:8083/ | jq
```

```
root@fast-data-dev / $ curl -s 127.0.0.1:8083/ | jq
{
  "version": "0.11.0.0-cp1",
  "commit": "5cadaa94d0a69e0d"
}
root@fast-data-dev / $
```

4 Connectors

NEW

Search connectors

file-stream-demo-distributed	✓ 1x	File →	File
logs-broker	✓ 1x	File →	File
sink-elastic-twitter-distributed	✓ 2x	File →	ElasticSearch (confluent)
source-twitter-distributed	✓ 1x	Twitter →	Twitter

Kafka Connect : /api/kafka-connect

Kafka Connect Version : 0.11.0.0-cp1

Kafka Connect UI Version : 0.9.2

List Connectors available on a Worker

```
curl -s 127.0.0.1:8083/connector-plugins | jq
```

It will list all the connectors.

```
root@fast-data-dev / $ curl -s 127.0.0.1:8083/connector-plugins | jq
[{"class": "com.datamountaineer.streamreactor.connect.azure.documentdb.sink.DocumentDbSinkConnector", "type": "unknown", "version": "null"}, {"class": "com.datamountaineer.streamreactor.connect.blockchain.source.BlockchainSourceConnector", "type": "source", "version": "null"}, {"class": "com.datamountaineer.streamreactor.connect.bloomberg.BloombergSourceConnector", "type": "source", "version": "null"}, {"class": "com.datamountaineer.streamreactor.connect.cassandra.sink.CassandraSinkConnector", "type": "unknown", "version": "1"}, {"class": "com.datamountaineer.streamreactor.connect.cassandra.source.CassandraSourceConnector", "type": "source", "version": "null"}, {"class": "com.datamountaineer.streamreactor.connect.coap.sink.CoapSinkConnector", "type": "sink", "version": "null"}, {"class": "com.datamountaineer.streamreactor.connect.coap.source.CoapSourceConnector", "type": "source", "version": "null"}, {"class": "com.datamountaineer.streamreactor.connect.druid.DruidSinkConnector", "type": "sink", "version": "null"}]
```

New Connector

Search

Sources	Sinks
 Twitter Use the Twitter API to stream data into Kafka	 Elastic Search Write data from Kafka to Elastic Search
 Yahoo Finance Stream stock and currency exchange rates into Kafka	 Cassandra Store Kafka data into Cassandra
 File Tail files or folders and stream data into Kafka	 InfluxDB Store Kafka data into InfluxDB
 Blockchain Get blockchain.info data into Kafka	 MongoDB Write Kafka data into MongoDB
 Jdbc Stream data from SQL server into Kafka	 HazelCast Store Kafka data into HazelCast (RingBuffer)
 Cassandra Extract Cassandra data using the CQL driver into Kafka	 Jdbc Store Kafka data into SQL
 Bloomberg Use the Bloomberg API to stream data into Kafka	 Amazon S3 Store Kafka data into Amazon S3

Ask about Active Connectors

```
curl -s 127.0.0.1:8083/connectors | jq
```

```
root@fast-data-dev / $ curl -s 127.0.0.1:8083/connectors | jq
[
  "sink-elastic-twitter-distributed",
  "source-twitter-distributed",
  "logs-broker",
  "file-stream-demo-distributed"
]
```

Get information about a Connector Tasks and Config

```
curl -s 127.0.0.1:8083/connectors/source-twitter-distributed/tasks | jq
```

```
root@fast-data-dev / $ curl -s 127.0.0.1:8083/connectors/source-twitter-distributed/tasks | jq
[
  {
    "id": {
      "connector": "source-twitter-distributed",
      "task": 0
    },
    "config": {
      "connector.class": "com.eneco.trading.kafka.connect.twitter.TwitterSourceConnector",
      "twitter.token": "1434035052732583938-y8gN6T6fqHTN1Yg8XGn18Gs0UVKgLV",
      "tasks.max": "1",
      "twitter.secret": "MqcWZnKcvYojiSadpH6J8JQGLLBnHIKnoVxnneJmSgBNu",
      "track.terms": "programming,java,kafka,scala",
      "language": "en",
      "key.converter.schemas.enable": "true",
      "task.class": "com.eneco.trading.kafka.connect.twitter.TwitterSourceTask",
      "value.converter.schemas.enable": "true",
      "name": "source-twitter-distributed",
      "topic": "demo-3-twitter",
      "twitter.consumersecret": "nSgzgjNMfFqG4S0urlb2vKkgI6mYHj3zZUf0caoN0Kb71o0RuS",
      "value.converter": "org.apache.kafka.connect.json.JsonConverter",
      "key.converter": "org.apache.kafka.connect.json.JsonConverter",
      "twitter.consumerkey": "gPCOPT0R7lGYH7GkFyNY8QqxC"
    }
  }
]
```

Get Connector Status

```
curl -s 127.0.0.1:8083/connectors/file-stream-demo-distributed/status | jq
```

```
root@fast-data-dev / $ curl -s 127.0.0.1:8083/connectors/file-stream-demo-distributed/status | jq
{
  "name": "file-stream-demo-distributed",
  "connector": {
    "state": "RUNNING",
    "worker_id": "127.0.0.1:8083"
  },
  "tasks": [
    {
      "state": "RUNNING",
      "id": 0,
      "worker_id": "127.0.0.1:8083"
    }
  ]
}
root@fast-data-dev / $
```

Pause / Resume a Connector (no response if the call is successful)

```
curl -s -X PUT 127.0.0.1:8083/connectors/file-stream-demo-distributed/pause
```

```
curl -s -X PUT 127.0.0.1:8083/connectors/file-stream-demo-distributed/resume
```

```
root@fast-data-dev / $ curl -s -X PUT 127.0.0.1:8083/connectors/file-stream-demo-distributed/pause
root@fast-data-dev / $ curl -s 127.0.0.1:8083/connectors/file-stream-demo-distributed/status | jq
{
  "name": "file-stream-demo-distributed",
  "connector": {
    "state": "PAUSED",
    "worker_id": "127.0.0.1:8083"
  },
  "tasks": [
    {
      "state": "PAUSED",
      "id": 0,
      "worker_id": "127.0.0.1:8083"
    }
  ]
}
root@fast-data-dev / $
```

```
root@fast-data-dev / $ curl -s -X PUT 127.0.0.1:8083/connectors/file-stream-demo-distributed/resume
root@fast-data-dev / $ curl -s 127.0.0.1:8083/connectors/file-stream-demo-distributed/status | jq
{
  "name": "file-stream-demo-distributed",
  "connector": {
    "state": "RUNNING",
    "worker_id": "127.0.0.1:8083"
  },
  "tasks": [
    {
      "state": "RUNNING",
      "id": 0,
      "worker_id": "127.0.0.1:8083"
    }
  ]
}
root@fast-data-dev / $
```

Get Connector Configuration

```
curl -s 127.0.0.1:8083/connectors/file-stream-demo-distributed | jq
```

```
root@fast-data-dev / $ curl -s 127.0.0.1:8083/connectors/file-stream-demo-distributed | jq
{
  "name": "file-stream-demo-distributed",
  "config": {
    "connector.class": "org.apache.kafka.connect.file.FileStreamSourceConnector",
    "key.converter.schemas.enable": "true",
    "file": "demo-file.txt",
    "tasks.max": "1",
    "value.converter.schemas.enable": "true",
    "name": "file-stream-demo-distributed",
    "topic": "demo-2-distributed",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "key.converter": "org.apache.kafka.connect.json.JsonConverter"
  },
  "tasks": [
    {
      "connector": "file-stream-demo-distributed",
      "task": 0
    }
  ]
}
root@fast-data-dev / $
```

Delete our Connector

```
curl -s -X DELETE 127.0.0.1:8083/connectors/file-stream-demo-distributed
```

```
root@fast-data-dev / $ curl -s -X DELETE 127.0.0.1:8083/connectors/file-stream-demo-distributed
root@fast-data-dev / $ curl -s 127.0.0.1:8083/connectors/file-stream-demo-distributed | jq
{
  "error_code": 404,
  "message": "Connector file-stream-demo-distributed not found"
}
root@fast-data-dev / $
```

Create a new Connector

```
curl -s -X POST -H "Content-Type: application/json" --data '{"name": "file-stream-demo-distributed",
"config": {"connector.class": "org.apache.kafka.connect.file.FileStreamSourceConnector", "key.converter.schemas.enable": "true", "file": "demo-file.txt", "tasks.max": "1", "value.converter.schemas.enable": "true", "name": "file-stream-demo-distributed", "topic": "demo-2-distributed", "value.converter": "org.apache.kafka.connect.json.JsonConverter", "key.converter": "org.apache.kafka.connect.json.JsonConverter"} }' http://127.0.0.1:8083/connectors | jq
```

```
root@fast-data-dev / $ curl -s -X POST -H "Content-Type: application/json" --data '{"name": "file-stream-demo-distributed", "config":{"connector.class": "org.apache.kafka.connect.file.FileStreamSourceConnector", "key.converter.schemas.enable": "true", "file": "demo-file.txt", "tasks.max": "1", "value.converter.schemas.enable": "true", "name": "file-stream-demo-distributed", "topic": "demo-2-distributed", "value.converter": "org.apache.kafka.connect.json.JsonConverter"}, "key.converter": "org.apache.kafka.connect.json.JsonConverter"}' http://127.0.0.1:8083/connectors | jq
{
  "name": "file-stream-demo-distributed",
  "config": {
    "connector.class": "org.apache.kafka.connect.file.FileStreamSourceConnector",
    "key.converter.schemas.enable": "true",
    "file": "demo-file.txt",
    "tasks.max": "1",
    "value.converter.schemas.enable": "true",
    "name": "file-stream-demo-distributed",
    "topic": "demo-2-distributed",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "key.converter": "org.apache.kafka.connect.json.JsonConverter"
  },
  "tasks": [
    {
      "connector": "file-stream-demo-distributed",
      "task": 0
    }
  ]
}
```

Update Connector configuration

```
curl -s -X PUT -H "Content-Type: application/json" --data
'{"connector.class": "org.apache.kafka.connect.file.FileStreamSourceConnector", "key.converter.schemas.enable": "true", "file": "demo-file.txt", "tasks.max": "2", "value.converter.schemas.enable": "true", "name": "file-stream-demo-distributed", "topic": "demo-2-distributed", "value.converter": "org.apache.kafka.connect.json.JsonConverter", "key.converter": "org.apache.kafka.connect.json.JsonConverter"}' 127.0.0.1:8083/connectors/file-stream-demo-distributed/config | jq
```

```
root@fast-data-dev / $ curl -s -X PUT -H "Content-Type: application/json" --data '{"connector.class": "org.apache.kafka.connect.file.FileStreamSourceConnector", "key.converter.schemas.enable": "true", "file": "demo-file.txt", "tasks.max": "2", "value.converter.schemas.enable": "true", "name": "file-stream-demo-distributed", "topic": "demo-2-distributed", "value.converter": "org.apache.kafka.connect.json.JsonConverter", "key.converter": "org.apache.kafka.connect.json.JsonConverter"}' 127.0.0.1:8083/connectors/file-stream-demo-distributed/config | jq
{
  "name": "file-stream-demo-distributed",
  "config": {
    "connector.class": "org.apache.kafka.connect.file.FileStreamSourceConnector",
    "key.converter.schemas.enable": "true",
    "file": "demo-file.txt",
    "tasks.max": "2",
    "value.converter.schemas.enable": "true",
    "name": "file-stream-demo-distributed",
    "topic": "demo-2-distributed",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "key.converter": "org.apache.kafka.connect.json.JsonConverter"
  },
  "tasks": [
    {
      "connector": "file-stream-demo-distributed",
      "task": 0
    }
  ]
}
root@fast-data-dev / $
```

JDBC Sink Connector

PostgreSQL

PostgreSQL



- We're going to start an PostgreSQL instance to Sink the data to
- We will use Docker to start our PostgreSQL instance
- This will serve as our Sink for our second Sink Connector

- PostgreSQL is a very popular relational database
- Used by many companies around the world to back their website or run BI reports onto
- No knowledge of PostgreSQL is required for this course

Distributed Mode

Example: JDBCSSinkConnector DISTRIBUTED MODE



- Goal:
 - Start an PostgreSQL instance using Docker
 - Run in distributed mode with multiple tasks



- Learning:
 - Learn about the JDBC Sink Connector

Step 1: Creating a JDBCSSinkConnector in landoop UI.

```

sink > demo-postgres > ⚙ sink-postgres-twitter-distributed.properties
1 # Basic configuration for our connector
2 name=sink-postgres-twitter-distributed
3 connector.class=io.confluent.connect.jdbc.JdbcSinkConnector
4 # We can have parallelism here so we have two tasks!
5 tasks.max=1
6 topics=demo-3-twitter
7 # the input topic has a schema, so we enable schemas conversion here too
8 key.converter=org.apache.kafka.connect.json.JsonConverter
9 key.converter.schemas.enable=true
10 value.converter=org.apache.kafka.connect.json.JsonConverter
11 value.converter.schemas.enable=true
12 # JDBCSSink connector specific configuration
13 # http://docs.confluent.io/3.2.0/connect/connect-jdbc/docs/sink_config_options.html
14 connection.url=jdbc:postgresql://postgres:5432/postgres
15 connection.user=postgres
16 connection.password=postgres
17 insert.mode=upsert
18 # we want the primary key to be offset + partition
19 pk.mode=kafka
20 # default value but I want to highlight it:
21 pk.fields=__connect_topic,__connect_partition,__connect_offset
22 fields.whitelist=id,created_at,text,lang,is_retweet
23 auto.create=true
24 auto.evolve=true
25

```

KAFKA CONNECT

4 Connectors NEW

Create New Connector

Kafka → Jdbc
class: io.confluent.connect.jdbc.JdbcSinkConnector

```

11 value.converter.schemas.enable=true
12 # JDBCSSink connector specific configuration
13 # http://docs.confluent.io/3.2.0/connect/connect-jdbc/docs/sink_config_options.html
14 connection.url=jdbc:postgresql://postgres:5432/postgres
15 connection.user=postgres
16 connection.password=postgres
17 insert.mode=upsert
18 # we want the primary key to be offset + partition
19 pk.mode=kafka
20 # default value but I want to highlight it:
21 pk.fields=__connect_topic,__connect_partition,__connect_offset
22 fields.whitelist=id,created_at,text,lang,is_retweet
23 auto.create=true
24 auto.evolve=true
25

```

Show cURL command Show Optional fields

CREATE

Kafka Connect: /api/kafka-connect
Kafka Connect Version: 0.11.0.0-cp1
Kafka Connect UI Version: 0.9.2

Powered by [LANDOOP](#)

KAFKA CONNECT

5 Connectors NEW

sink-postgres-twitter-distributed IS RUNNING PAUSE RESTART DELETE

Kafka → Jdbc

TASKS 0 [127.0.0.1:8083] **TOPICS** demo-3-twitter

CONFIGURATION EDIT

```

1 {
2   "connector.class": "io.confluent.connect.jdbc.JdbcSinkConnector",
3   "connection.password": "postgres",
4   "tasks.max": "1",
5   "topics": "demo-3-twitter",
6   "key.converter.schemas.enable": "true",
7   "fields.whitelist": "id,created_at,text,lang,is_retweet",
8   "auto.evolve": "true",
9   "connection.user": "postgres",
10  "value.converter.schemas.enable": "true",
11  "name": "sink-postgres-twitter-distributed",
12  "pk.mode": "true",
13  "value.converter": "org.apache.kafka.connect.json.JsonConverter",
14  "connection.url": "jdbc:postgresql://postgres:5432/postgres",
15  "insert.mode": "upsert",
16  "key.converter": "org.apache.kafka.connect.json.JsonConverter",
17  "pk.mode": "kafka",
18  "pk.fields": "__connect_topic,__connect_partition,__connect_offset"
19 }

```

Step 2: Checking the twitter data in PostgreSQL.

Server Information

Connection Test
Successfully connected

Name	Database Type	SSL	
postgres	PostgreSQL	<input type="checkbox"/>	
Server Address			
127.0.0.1	5432	Domain <input type="button" value="..."/>	
User	Password	Database/Keyspace	Schema
postgres	*****	postgres	Schema <input type="button" value="..."/>
<input type="checkbox"/> SSH Tunnel			
<input type="button" value="Test"/>		<input type="button" value="Cancel"/> Ø	<input type="button" value="Save"/> ✓

PostgreSQL 9.5.25 : Kafka-Postgre : postgres : public.demo-3-twitter

Items	Queries	History	__connect_topic	__connect_partition	__connect_offset	created_at	id	text	lang
			demo-3-twitter	0	0	2021-09-14T05:52:41.000+0000	1437655523562856448	RT @TheRamoliya: The growing pain of caches in m...	en
			demo-3-twitter	0	1	2021-09-14T05:52:42.000+0000	1437655527547416576	RT @bluejak3: IPS Community Suite 4.6.6 Free...	en
			demo-3-twitter	0	2	2021-09-14T05:52:45.000+0000	1437655538742054914	RT @sayalook: Things change over time !...	en
			demo-3-twitter	0	3	2021-09-14T05:52:47.000+0000	1437655545931042821	RT @TheRamoliya: The growing pain of caches in m...	en
			demo-3-twitter	0	4	2021-09-14T05:52:50.000+0000	1437655560682459137	RT @sayalook: Things change over time !...	en
			demo-3-twitter	0	5	2021-09-14T05:53:00.000+0000	143765559865606144	RT @TheRamoliya: The growing pain of caches in m...	en
			demo-3-twitter	0	6	2021-09-14T05:53:01.000+0000	1437655604311568394	RT @_ArifChaudhary: Login in programming,...	en
			demo-3-twitter	0	7	2021-09-14T05:53:04.000+0000	1437655618706345985	RT @TBrownSanjo: @tinyStormado @CACorrections...	en
			demo-3-twitter	0	8	2021-09-14T05:53:05.000+0000	1437655623521353730	#softwaredevelopment #webdevelopment #react #...	en
			demo-3-twitter	0	9	2021-09-14T05:53:08.000+0000	1437655636267782150	#cron #backend #programming #softwareengineeri...	en
			demo-3-twitter	0	10	2021-09-14T05:53:11.000+0000	1437655647064035331	RT @TheRamoliya: The growing pain of caches in m...	en
			demo-3-twitter	0	11	2021-09-14T05:53:12.000+0000	1437655650914471937	RT @TheRamoliya: The growing pain of caches in m...	en
			demo-3-twitter	0	12	2021-09-14T05:53:18.000+0000	1437655676034060290	@Iwariof Seems I did forget Python, right. And th...	en
			demo-3-twitter	0	13	2021-09-14T05:53:29.000+0000	1437655722980900865	RT @usapaypalsell: Full Verified PayPal Account Visi...	en
			demo-3-twitter	0	14	2021-09-14T05:53:34.000+0000	1437655743679893504	RT @New081082: Things change time ...	en
			demo-3-twitter	0	15	2021-09-14T05:53:34.000+0000	1437655746187980802	RT @ekbarth3great: #softwaredevelopment #webd...	en
			demo-3-twitter	0	16	2021-09-14T05:53:35.000+0000	1437655747089747970	More than 50 Current Openings nd the list is growi...	en
			demo-3-twitter	0	17	2021-09-14T05:53:37.000+0000	1437655758338867206	RT @ekbarth3great: #cron #backend #programmin...	en
			demo-3-twitter	0	18	2021-09-14T05:53:39.000+0000	1437655767025340419	RT @New081082: Things change time ...	en
			demo-3-twitter	0	19	2021-09-14T05:53:42.000+0000	143765577833992197	RT @exemLPP: More than 50 Current Openings nd t...	en
			demo-3-twitter	0	20	2021-09-14T05:53:44.000+0000	1437655786751152138	RT @TheRamoliya: The growing pain of caches in m...	en
			demo-3-twitter	0	21	2021-09-14T05:53:44.000+0000	1437655786793152514	RT @ekbarth3great: #softwaredevelopment #webd...	en
			demo-3-twitter	0	22	2021-09-14T05:53:44.000+0000	1437655786835038209	RT @usapaypalsell: Full Verified PayPal Account Visi...	en
			demo-3-twitter	0	23	2021-09-14T05:53:44.000+0000	1437655786835034117	RT @TheDailyTechTa1: Short introduction to Rabbit...	en
			demo-3-twitter	0	24	2021-09-14T05:53:44.000+0000	1437655786747056129	RT @abhayparashar_p: #TioBe's top 10 programmi...	en
			demo-3-twitter	0	25	2021-09-14T05:53:44.000+0000	1437655786868682752	RT @New081082: Things change time ...	en
			demo-3-twitter	0	26	2021-09-14T05:53:44.000+0000	1437655786881273856	RT @PDH_SciTechNews: #Programming PHP maint...	en
			demo-3-twitter	0	27	2021-09-14T05:53:44.000+0000	1437655786830934021	RT @healthyfitness09: Full Verified PayPal Account...	en
			demo-3-twitter	0	28	2021-09-14T05:53:47.000+0000	1437655798847614976	RT @PDH_SciTechNews: #Programming PHP maint...	en
			demo-3-twitter	0	29	2021-09-14T05:53:51.000+0000	1437655814475448325	RT @healthyfitness09: Full Verified PayPal Account...	en

Section goals

- Understand how connectors are made, using the [GitHubSourceConnector](#) as an example
 - Dependencies
 - ConfigDef
 - Connector
 - Schema & Struct
 - Source Partition & Source Offsets
 - Task
- Learn how to deploy your connectors (or any connector)
 - Package Jars
 - Run jars in standalone mode
 - Deploy Jars

Config Definitions



-
- Config Def are the way to communicate to the user how you want your Configuration to be
 - You can have mandatory or optional parameters (with default), and validate the types and constraints of your parameters. You should also document the parameters

```
public static final String OWNER_CONFIG = "github.owner";
private static final String OWNER_DOC = "Owner of the repository you'd like to follow";

public static ConfigDef conf() {
    return new ConfigDef()
        .define(OWNER_CONFIG, Type.STRING, Importance.HIGH, OWNER_DOC)
        .define(BATCH_SIZE_CONFIG, Type.INT, 100, new BatchSizeValidator(), Importance.LOW, BATCH_SIZE_DOC)
```

```

package com.simplesteph.kafka;
import ...

public class GitHubSourceConnectorConfig extends AbstractConfig {
    public static final String TOPIC_CONFIG = "topic";
    private static final String TOPIC_DOC = "Topic to write to";

    public static final String OWNER_CONFIG = "github.owner";
    private static final String OWNER_DOC = "Owner of the repository you'd like to follow";

    public static final String REPO_CONFIG = "github.repo";
    private static final String REPO_DOC = "Repository you'd like to follow";

    public static final String SINCE_CONFIG = "since.timestamp";
    private static final String SINCE_DOC =
        "Only issues updated at or after this time are returned.\n" +
        "  + This is a timestamp in ISO 8601 format: YYYY-MM-DDTHH:MM:SSZ.\n" +
        "  + Defaults to a year from first launch.";

    public static final String BATCH_SIZE_CONFIG = "batch.size";
    private static final String BATCH_SIZE_DOC = "Number of data points to retrieve at a time. Defaults to 100 (max value)";

    public static final String AUTH_USERNAME_CONFIG = "auth.username";
    private static final String AUTH_USERNAME_DOC = "Optional Username to authenticate calls";

    public static final String AUTH_PASSWORD_CONFIG = "auth.password";
    private static final String AUTH_PASSWORD_DOC = "Optional Password to authenticate calls";

    public GitHubSourceConnectorConfig(ConfigDef config, Map<String, String> parsedConfig) {
        super(config, parsedConfig);
    }
}

```

Connector

Connector



- That's the class of your Kafka Connector that is referenced from the configs. It should load the config, and create a few tasks.

```

public String version()
public void start(Map<String, String> map)
public Class<? extends Task> taskClass()
public List<Map<String, String>> taskConfigs(int i)
public void stop()
public ConfigDef config()

```

The screenshot shows the IntelliJ IDEA interface with the file `GitHubSourceConnector.java` open. The code implements the `SourceConnector` interface, handling configuration, start, task class, task configurations, stop, and config retrieval. The code uses `GitHubSourceConnectorConfig` and `GitHubSourceTask` from the `com.simplesteph.kafka` package. The project structure on the left includes `src`, `main`, `java`, and `com.simplesteph.kafka` packages.

```
package com.simplesteph.kafka;
import ...
public class GitHubSourceConnector extends SourceConnector {
    private static Logger log = LoggerFactory.getLogger(GitHubSourceConnector.class);
    private GitHubSourceConnectorConfig config;
    @Override
    public String version() { return VersionUtil.getVersion(); }
    @Override
    public void start(Map<String, String> map) { config = new GitHubSourceConnectorConfig(map); }
    @Override
    public Class<? extends Task> taskClass() { return GitHubSourceTask.class; }
    @Override
    public List<Map<String, String>> taskConfigs(int i) {
        // Define the individual task configurations that will be executed.
        ArrayList<Map<String, String>> configs = new ArrayList<>(initialCapacity: 1);
        configs.add(config.originalsStrings());
        return configs;
    }
    @Override
    public void stop() {
        // Do things that are necessary to stop your connector.
        // nothing is necessary to stop for this connector
    }
    @Override
    public ConfigDef config() { return GitHubSourceConnectorConfig.config(); }
}
```

Start() function map argument values

```
name=GitHubSourceConnectorDemo
tasks.max=1
connector.class=com.simplesteph.kafka.GitHubSourceConnector
topic=github-issues
github.owner=kubernetes
github.repo=kubernetes
since.timestamp=2017-01-01T00:00:00Z
# I heavily recommend you set those two fields:
# auth.username=your_username
# auth.password=your_password
```

Writing a Schema

Schemas



- **Schema** is an abstraction that allow you to define how your data structure will look like. It is using primitive types, and then the Kafka Connect framework will convert it into JSON or Avro as needed.
- It is necessary to correctly design your schema before you program

```
public static Schema USER_SCHEMA =
SchemaBuilder.struct().name(SCHEMA_VALUE_USER)
.version(1)
.field(USER_URL_FIELD, Schema.STRING_SCHEMA)
.field(USER_ID_FIELD, Schema.INT32_SCHEMA)
.field(USER_LOGIN_FIELD, Schema.STRING_SCHEMA)
.build();
```

```
// Key Schema
public static Schema KEY_SCHEMA = SchemaBuilder.struct().name(SCHEMA_KEY)
.version(1)
.field(OWNER_FIELD, Schema.STRING_SCHEMA)
.field(REPOSITORY_FIELD, Schema.STRING_SCHEMA)
.field(NUMBER_FIELD, Schema.INT32_SCHEMA)
.build();

// Value Schema
public static Schema USER_SCHEMA = SchemaBuilder.struct().name(SCHEMA_VALUE_USER)
.version(1)
.field(USER_URL_FIELD, Schema.STRING_SCHEMA)
.field(USER_ID_FIELD, Schema.INT32_SCHEMA)
.field(USER_LOGIN_FIELD, Schema.STRING_SCHEMA)
.build();

// optional schema
public static Schema PR_SCHEMA = SchemaBuilder.struct().name(SCHEMA_VALUE_PR)
.version(1)
.field(PR_URL_FIELD, Schema.STRING_SCHEMA)
.field(PR_HTML_URL_FIELD, Schema.STRING_SCHEMA)
.optional()
.build();

public static Schema VALUE_SCHEMA = SchemaBuilder.struct().name(SCHEMA_VALUE_ISSUE)
.version(1)
.field(URL_FIELD, Schema.STRING_SCHEMA)
.field(TITLE_FIELD, Schema.STRING_SCHEMA)
.field(CREATED_AT_FIELD, Schema.INT64_SCHEMA)
.field(UPDATED_AT_FIELD, Schema.INT64_SCHEMA)
.field(NUMBER_FIELD, Schema.INT32_SCHEMA)
.field(STATE_FIELD, Schema.STRING_SCHEMA)
.field(USER_FIELD, USER_SCHEMA) // mandatory
.field(PR_FIELD, PR_SCHEMA)    // optional
.build();
```



Data Model

- It is always really good to have your data exist in classes in a “model” package.
- I modelled Issue, User and Pull Request as POJOs

```
public class PullRequest {  
  
    private String url;  
    private String htmlUrl;  
}
```

Screenshot of an IDE showing the project structure and code for the `User` class.

Project Structure:

- Project: kafka-connect-github-source-1.1
- src/main/java/com/simplesteph/kafka/model
- com.simplesteph.kafka.model.User.java

User.java Code:

```
package com.simplesteph.kafka.model;  
  
import ...  
  
public class User {  
    private String login;  
    private Integer id;  
    private String avatarUrl;  
    private String gravatarId;  
    private String url;  
    private String htmlUrl;  
    private String followersUrl;  
    private String followingUrl;  
    private String gistsUrl;  
    private String starredUrl;  
    private String subscriptionsUrl;  
    private String organizationsUrl;  
    private String reposUrl;  
    private String eventsUrl;  
    private String receivedEventsUrl;  
    private String type;  
    private Boolean siteAdmin;  
    private Map<String, Object> additionalProperties = new HashMap<>();  
  
    /**  
     * No args constructor for use in serialization  
     */  
    public User() {}  
  
    /**  
     * @param eventsUrl  
     * @param additionalProperties  
     */  
}
```

Screenshot of an IDE showing the implementation of the `fromJson` static method for the `User` class.

Code:

```
public static User fromJson(JSONObject jsonObject) {  
    User user = new User();  
    user.setUrl(jsonObject.getString(USER_URL_FIELD));  
    user.setHtmlUrl(jsonObject.getString(USER_HTML_URL_FIELD));  
    user.setId(jsonObject.getInt(USER_ID_FIELD));  
    user.setLogin(jsonObject.getString(USER_LOGIN_FIELD));  
    return user;  
}
```

```

@Test
public void canParseJson(){
    User user = User.fromJson(userJson);
    assertEquals(user.getId(),(Integer) 20851561);
    assertEquals(user.getUrl(), actual: "https://api.github.com/users/simplesteph");
    assertEquals(user.getLogin(), actual: "simplesteph");
}

```

Getting Data from Github API

Getting Data from GitHub API



- I query the GitHub API using the Unirest Library, that we wrap in our own client
- I implemented the concepts of
 - Dynamic URL
 - Pagination
 - Rate Throttling
 - Error Handling

```

public class GitHubAPIHttpClient{
    protected JSONArray getNextIssues() throws InterruptedException
    protected HttpResponse<JsonNode> getNextIssuesAPI() throws UnirestException
    protected String constructUrl()
}

```

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "kafka-connect-github-source-1.1". It contains a "src" directory with "main" and "java" sub-directories. The "java" directory contains several classes: GitHubAPIHttpClient, GitHubSchemas, GitHubSourceConnector, GitHubSourceConnectorConfig, GitHubSourceTask, and VersionUtil. There are also test classes: GitHubSourceConnectorTest, GitHubSourceTaskTest, and UserTest.
- Code Editor:** The main editor window displays the "GitHubAPIHttpClient.java" file. The code defines a class that interacts with the GitHub API using the Unirest library. It includes methods for getting the next issues, performing an API call, and constructing URLs. It also handles rate limiting by tracking X-RateLimit, X-RateRemaining, and X-RateReset headers.
- Toolbars and Status Bar:** The top bar shows standard IntelliJ icons like file, edit, and search. The status bar at the bottom right indicates the current time (19:14), file type (LF), encoding (UTF-8), and code space usage (4 spaces).



Source partition, Source Offsets

- Source Partition allows Kafka Connect to know which source you've been reading
- Source Offsets allow Kafka Connect to Track until when you've been reading for the Source Partition you chose
- **They are different than partition and offsets for Kafka.**
- Source Partition and Source Offsets are for Kafka Connect Source

```
private Map<String, String> sourcePartition() {
    Map<String, String> map = new HashMap<>();
    map.put(OWNER_FIELD, config.getOwnerConfig());
    map.put(REPOSITORY_FIELD, config.getRepoConfig());
    return map;
}
```

kafka-connect-github-source-1.1 – GitHubSourceTask.java

```
Project src/main/java/com/simplesteph/kafka GitHubSourceTask
kafka-connect-github-source-1.1 [kafka-con
  > .idea
  > config
    > GitHubSourceConnectorExample.properties
    > worker.properties
  > offsets
  > src
    > main
      > assembly
      > java
        > com.simplesteph.kafka
          > model
          > utils
          > Validators
            GitHubAPIClient
            GitHubSchemas
            GitHubSourceConnector
            GitHubSourceConnectorConfig
            GitHubSourceTask
            VersionUtil
        > test
          > java
            > com.simplesteph.kafka
              > model
                IssueTest
                UserTest
              > utils
                GitHubSourceConnectorConfig
                GitHubSourceConnectorTest
                GitHubSourceTaskTest
      > .gitignore
  > buildRecordValue(issue),
    issue.getUpdatedAt().toEpochMilli();

  @Override
  public void stop() {
    // Do whatever is required to stop your task.
}

  private Map<String, String> sourcePartition() {
    Map<String, String> map = new HashMap<>();
    map.put(OWNER_FIELD, config.getOwnerConfig());
    map.put(REPOSITORY_FIELD, config.getRepoConfig());
    return map;
}

  private Map<String, String> sourceOffset(Instant updatedAt) {
    Map<String, String> map = new HashMap<>();
    map.put(UPDATED_AT_FIELD, DateUtils.MaxInstant(updatedAt, nextQuerySince).toString());
    map.put(NEXT_PAGE_FIELD, nextPageToVisit.toString());
    return map;
}

  private Struct buildRecordKey(Issue issue){
    // Key Schema
    Struct key = new Struct(KEY_SCHEMA)
      .put(OWNER_FIELD, config.getOwnerConfig())
      .put(REPOSITORY_FIELD, config.getRepoConfig())
      .put(NUMBER_FIELD, issue.getNumber());
    return key;
}
```

Task



- Our Task is what does the actual job
- It's supposed to initialize, then find where to resume from, and finally poll the source for records

```
public class GitHubSourceTask extends SourceTask {
    public String version()
    public void start(Map<String, String> map)
    public List<SourceRecord> poll() throws
        InterruptedException
    public void stop()
}
```

kafka-connect-github-source-1.1 > src > main > java > com > simplesteph > kafka > GitHubSourceTask

```
package com.simplesteph.kafka;
import ...

public class GitHubSourceTask extends SourceTask {
    private static final Logger log = LoggerFactory.getLogger(GitHubSourceTask.class);
    public GitHubSourceConnectorConfig config;

    protected Instant nextQuerySince;
    protected Integer lastIssueNumber;
    protected Integer nextPageToVisit = 1;
    protected Instant lastUpdatedAt;

    GitHubAPIHttpClient gitHubHttpApiClient;

    @Override
    public String version() { return VersionUtil.getVersion(); }

    @Override
    public void start(Map<String, String> map) {
        //Do things here that are required to start your task. This could be open a connection to a database, etc.
        config = new GitHubSourceConnectorConfig(map);
        initializeLastVariables();
        gitHubHttpApiClient = new GitHubAPIHttpClient(config);
    }

    private void initializeLastVariables(){
        Map<String, Object> lastSourceOffset = null;
        lastSourceOffset = context.offsetStorageReader().offset(sourcePartition());
        if( lastSourceOffset == null){
            // we haven't fetched anything yet, so we initialize to 7 days ago
            nextQuerySince = config.getSince();
            lastIssueNumber = -1;
        } else {
            Object updatedAt = lastSourceOffset.get(UPDATED_AT_FIELD);
            lastIssueNumber = Integer.parseInt(lastSourceOffset.get(NUMBER_FIELD));
        }
    }
}
```

Building and running a connector in standalone mode

- Building: `mvn clean package`
 - Running (see `./run.sh` – only for linux / mac users):
 1. Export classpath:
 - `export CLASSPATH=$(find target/ -type f -name '*.jar'| grep '\-package' | tr '\n' ':')`
 2. Build the Dockerfile:
 - `docker build . -t simplesteph/kafka-connect-source-github:1.0`
 3. Run the docker image:
 - `docker run -e CLASSPATH=$CLASSPATH \
--net=host --rm -t \
-v $(pwd)/offsets:/kafka-connect-source-github/offsets \
simplesteph/kafka-connect-source-github:1.0`

The screenshot shows the IntelliJ IDEA interface with the project 'kafka-connect-github-source-1.1' open. The 'Project' tool window on the left lists files like .idea, config, offsets, src/main, src/test, target, .gitignore, build.sh, Dockerfile, end-to-end.sh, LICENSE, pom.xml, README.md, and run.sh. The 'Terminal' tab at the bottom shows the output of running the script:

```
[INFO] -----
[INFO] Total time: 7.802 s
[INFO] Finished at: 2021-09-17T14:56:56+05:30
[INFO] -----
[INFO] -----
[INFO] + [+] Building 9.1s (9/9) FINISHED
--> [Internal] load build definition from Dockerfile
--> > transferring dockerfile: 120B
--> [internal] load .dockerignore
[INFO] -----
[INFO] Run | TODO | Problems | Terminal | Build | Dependencies | Event Log
Build completed successfully in 4 sec, 73 ms (today 13:33)
```

Deploying Connector in Landoop Cluster

Deploying your Connectors



1. Extract the jars
2. Mount the jars into your Landoop Image
3. Use the Connect UI to launch the connector!

Step 1: Running the Kafka Cluster using landoop image in docker.

```
docker run -it --rm -p 2181:2181 -p 3030:3030 -p 8081:8081 -p 8082:8082 -p 8083:8083 -p 9092:9092 -e ADV_HOST=127.0.0.1 -e RUNTESTS=0 -v "$(pwd)":/connectors/kafka-connect-github-source landoop/fast-data-dev
```

Here mounting to the "connectors/kafka-connect-github-source" folder is mandatory.

```

→ kafka-connect-github-source docker run -it --rm -p 2181:2181 -p 3030:3030 -p 8081:8081 -p 8082:8082 -p 8083:8083 -p 9092:9092 -e ADV_HOST=127.0.0.1 -e RUNTESTS=0 -v "$(pwd)":/connectors/kafka-connect-github-source landoop/fast-data-dev
Setting advertised host to 127.0.0.1.
Operating system RAM available is 3088 MiB, which is less than the lowest recommended of 4096 MiB. Your system performance may be seriously impacted.
Starting services.
This is Lenses.io's fast-data-dev. Kafka 2.6.2-L0 (Lenses.io's Kafka Distribution).
You may visit http://127.0.0.1:3030 in about a minute.
2021-09-17 14:44:11,045 INFO Included extra file "/etc/supervisord.d/01-zookeeper.conf" during parsing
2021-09-17 14:44:11,045 INFO Included extra file "/etc/supervisord.d/02-broker.conf" during parsing
2021-09-17 14:44:11,045 INFO Included extra file "/etc/supervisord.d/03-schema-registry.conf" during parsing
2021-09-17 14:44:11,045 INFO Included extra file "/etc/supervisord.d/04-rest-proxy.conf" during parsing
2021-09-17 14:44:11,045 INFO Included extra file "/etc/supervisord.d/05-connect-distributed.conf" during parsing
2021-09-17 14:44:11,045 INFO Included extra file "/etc/supervisord.d/06-caddy.conf" during parsing

```

Step 2: Creating a new topic

docker run --rm -it --net=host landoop/fast-data-dev bash

kafka-topics --create --topic github-issues --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:2181

```

→ kafka-connect-github-source docker run --rm -it --net=host landoop/fast-data-dev bash
root@fast-data-dev / $
root@fast-data-dev / $ kafka-topics --create --topic github-issues --partitions 3 --replication-factor 1 --zookeeper 127.0.0.1:2181
1
Created topic github-issues.
root@fast-data-dev / $

```

The screenshot shows the Kafka Topics UI at the URL 127.0.0.1:3030/kafka-topics-ui/#/cluster/fast-data-dev/topic/n/github-issues/. The interface has a header 'KAFKA TOPICS'. On the left, there is a list of '7 TOPICS' including 'logs_broker', 'nyc_yellow_taxi_trip_data', 'github-issues', 'telecom_italia_data', 'backblaze_smart', 'telecom_italia_grid', and 'sea_vessel_position_reports'. The 'github-issues' topic is selected, and its details are shown on the right. The 'DATA' tab is active, showing the message 'Topic is empty'. The 'PARTITIONS' tab shows a value of 3 with a red notification badge. The 'CONFIGURATION' tab shows a value of 26 with a red notification badge. At the bottom, it says 'Powered by Landoop'.

Step 2: Creating a new connector UI landoop UI.

All the JARs are loaded in target mount directory in the docker run command. So that we could able to see the GitHubSourceConnector in Connector New UI in landoop.

And the we created a github source source connector.

The screenshot shows the Kafka Connect UI interface. On the left, there's a list of connectors: GitHubSourceConnector (1x) and logs-broker (1x). The GitHubSourceConnector is selected. On the right, the detailed configuration for the GitHubSourceConnector is shown. It has a green status bar at the top indicating "GitHub → Kafka IS RUNNING". Below this, under "TASKS", there is one task running on port 127.0.0.1:8083. Under "TOPICS", there is one topic named "github-issues". At the bottom, the "CONFIGURATION" section displays the following properties:

```

1 connector.class=com.simplesteph.kafka.GitHubSourceConnector
2 github.owner=kubernetes
3 tasks.max=1
4 topic=github-issues
5 github.repo=kubernetes

```

Step 3: Checking the topic data.

The screenshot shows the Kafka Topics UI. On the left, it lists 7 topics: nyc_yellow_taxi_trip_data, github-issues, logs_broker, telecom_italia_data, backblaze_smart, telecom_italia_grid, and sea_vessel_position_reports. The "github-issues" topic is selected and expanded. The right side of the screen shows the details for the "github-issues" topic. It has 3 partitions. The "DATA" tab is active, showing 100 total messages fetched. The "PARTITIONS" tab shows partition 0 with 3 messages. The "CONFIGURATION" tab shows 26 configurations. Below these tabs, the "TOPIC" column lists the 3 messages in the topic. Each message is represented by a key-value pair. The "RAW DATA" column shows the raw Avro data for each message. The "OFFSET" and "PARTITION" columns show the offset and partition for each message. The messages are as follows:

- Offset: 0 Partition: 0


```

Key: { owner: kubernetes, repository: kubernetes, number: 94857 }
Value: { url: https://api.github.com/repos/kubernetes/kubernetes/issues/94857, title: No access to events if only the API group 'events.k8s.io' is specified in a role, created_at: 2018-07-10T14:45:00Z, updated_at: 2018-07-10T14:45:00Z, closed_at: null, state: open, assignee: null, labels: [ ], comments: 0, reactions: 0, milestones: [ ], pull_request: null }
      
```
- Offset: 1 Partition: 0


```

Key: { owner: kubernetes, repository: kubernetes, number: 89948 }
Value: { url: https://api.github.com/repos/kubernetes/kubernetes/issues/89948, title: IKS | Node host name is not equal between kubernetes API and node's os, created_at: 2018-06-27T14:45:00Z, updated_at: 2018-06-27T14:45:00Z, closed_at: null, state: open, assignee: null, labels: [ ], comments: 0, reactions: 0, milestones: [ ], pull_request: null }
      
```
- Offset: 2 Partition: 0


```

Key: { owner: kubernetes, repository: kubernetes, number: 88628 }
Value: { url: https://api.github.com/repos/kubernetes/kubernetes/issues/88628, title: [WIP] Enhance workloads upgrade tests, created_at: 2018-06-26T14:45:00Z, updated_at: 2018-06-26T14:45:00Z, closed_at: null, state: open, assignee: null, labels: [ ], comments: 0, reactions: 0, milestones: [ ], pull_request: null }
      
```
- Offset: 3 Partition: 0


```

Key: { owner: kubernetes, repository: kubernetes, number: 74878 }
Value: { url: https://api.github.com/repos/kubernetes/kubernetes/issues/74878, title: "dockerc ps" no containers in this node, but "kubectl get po" has running pods., created_at: 2018-06-25T14:45:00Z, updated_at: 2018-06-25T14:45:00Z, closed_at: null, state: open, assignee: null, labels: [ ], comments: 0, reactions: 0, milestones: [ ], pull_request: null }
      
```
- Offset: 4 Partition: 0


```

Key: { owner: kubernetes, repository: kubernetes, number: 94868 }
Value: { url: https://api.github.com/repos/kubernetes/kubernetes/issues/94868, title: Adding cheftako to CCM owners., created_at: 2018-07-09T14:45:00Z, updated_at: 2018-07-09T14:45:00Z, closed_at: null, state: open, assignee: null, labels: [ ], comments: 0, reactions: 0, milestones: [ ], pull_request: null }
      
```
- Offset: 5 Partition: 0


```

Key: { owner: kubernetes, repository: kubernetes, number: 94854 }
Value: { url: https://api.github.com/repos/kubernetes/kubernetes/issues/94854, title: test(apply); deflake run_kubectl_apply_tests[round 3], created_at: 2018-07-09T14:45:00Z, updated_at: 2018-07-09T14:45:00Z, closed_at: null, state: open, assignee: null, labels: [ ], comments: 0, reactions: 0, milestones: [ ], pull_request: null }
      
```
- Offset: 6 Partition: 0


```

Key: { owner: kubernetes, repository: kubernetes, number: 94865 }
Value: { url: https://api.github.com/repos/kubernetes/kubernetes/issues/94865, title: kubectl patch service account not having any effect, created_at: 2018-07-09T14:45:00Z, updated_at: 2018-07-09T14:45:00Z, closed_at: null, state: open, assignee: null, labels: [ ], comments: 0, reactions: 0, milestones: [ ], pull_request: null }
      
```

Development guide for Connector

<https://docs.confluent.io/3.2.0/connect/devguide.html>

Guideline for Developing Connector

<https://www.confluent.io/wp-content/uploads/Partner-Dev-Guide-for-Kafka-Connect.pdf?x18424>

<https://gist.github.com/jcustenborder/b9b1518cc794e1c1895c3da7abbe9c08#schema-types>

Setting up Kafka Connect in Production (1/2)

Here is what we know

- We know how to use connectors
- We know how to create connectors
- We know how to add connectors
- We know how to use the Landoop Kafka Connect UI

All of that is great, but you may have realised this is only for development purposes.

... so how do we do things in production?

Setting up Kafka Connect in Production (2/2)

To setup Kafka Connect in production

1. Download Kafka Binaries
2. Set up the connect-distributed.properties as needed
3. Add your jars where needed (plugins.path or classpath)
4. And launch Kafka Connect!
5. (optional) Setup the Landoop Kafka Connect UI

You will be able to interact with Kafka Connect using the REST API, or the UI if you set it up

