Connecting Kafka Connect to Confluent Cloud

You can configure a local Connect cluster backed by a source Apache Kafka® cluster in Confluent Cloud.

cluster into a local file.

Prerequisites

- Access to Confluent Cloud.
- Confluent Cloud CLI installed and configured.
- curl
- jq

Create the Topics in Cloud Cluster

1. Create a `page_visits` topic as follows:

```
ccloud kafka topic create --partitions 1 page_visits
```

Important

You must manually create topics for source connectors to write to.

2. Produce records to the topic named `page_visits`. Press `Ctrl+C` to exit.

```
Starting Kafka Producer. ^C to exit
foo
bar
baz
^C
```

Your output should resemble:

```
{"field1": "hello", "field2": 1}
{"field1": "hello", "field2": 2}
{"field1": "hello", "field2": 3}
{"field1": "hello", "field2": 4}
{"field1": "hello", "field2": 5}
{"field1": "hello", "field2": 6}
^D
```

3. Verify that they can be consumed:

```
ccloud kafka topic consume -b page_visits
```

Your output should resemble:

```
{"field1": "hello", "field2": 1}
{"field1": "hello", "field2": 2}
{"field1": "hello", "field2": 3}
{"field1": "hello", "field2": 4}
{"field1": "hello", "field2": 5}
{"field1": "hello", "field2": 6}
^D
```

Set up a Connect Worker with Confluent Cloud

Download the latest ZIP or TAR distribution of Confluent Platform fromhttps://www.confluent.io/download/. Follow the instructions based on whether you are using a Standalone Cluster or Distributed Cluster.

Replace `<cloud-bootstrap-servers>`, `<api-key>`, and `<api-secret>` with appropriate values from your Kafka cluster setup.

Standalone Cluster

1. Create `my-connect-standalone.properties` in the config directory, whose contents look like the following (note the security configs with `consumer.*` and `producer.*` prefixes).

```
cat etc/my-connect-standalone.properties
bootstrap.servers=<cloud-bootstrap-servers>

# The converters specify the format of data in Kafka and how to translate it into Connect data. Every Connect user wi
ll
# need to configure these based on the format they want their data in when loaded from or stored into Kafka
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
# Converter-specific settings can be passed in by prefixing the Converter's setting with the converter you want to ap
ply
# it to
key.converter.schemas.enable=false
value.converter.schemas.enable=false

# The internal converter used for offsets and config data is configurable and must be specified, but most users will

# always want to use the built-in default. Offset and config data is never visible outside of Kafka Connect in this f
ormat.
internal.key.converter=org.apache.kafka.connect.json.JsonConverter
internal.value.converter=org.apache.kafka.connect.json.JsonConverter
internal.key.converter.schemas.enable=false
internal.value.converter.schemas.enable=false

# Store offsets on local filesystem
offset.storage.file.filename=/tmp/connect.offsets
# Flush much faster than normal, which is useful for testing/debugging
offset.flush.interval.ms=10000

ssl.endpoint.identification.algorithm=https
sasl.mechanism=PLAIN
request.timeout.ms=20000
retry.backoff.ms=500
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
username="<api-key>" password="<api-secret>";
security.protocol=SASL_SSL

consumer.ssl.endpoint.identification.algorithm=https
consumer.sasl.mechanism=PLAIN
consumer.request.timeout.ms=20000
consumer.retry.backoff.ms=500
consumer.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
username="<api-key>" password="<api-secret>";
consumer.security.protocol=SASL_SSL

producer.ssl.endpoint.identification.algorithm=https
producer.sasl.mechanism=PLAIN
producer.request.timeout.ms=20000
producer.retry.backoff.ms=500
producer.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
username="<api-key>" password="<api-secret>";
producer.security.protocol=SASL_SSL
```

2. (Optional) Add the configs to `my-connect-standalone.properties` to connect to Confluent Cloud Schema Registry per the example in connect-ccloud.delta on GitHub at ccloud/examples/template_delta_configs.

```
# Confluent Schema Registry for Kafka Connect
value.converter=io.confluent.connect.avro.AvroConverter
value.converter.basic.auth.credentials.source=USER_INFO
value.converter.schema.registry.basic.auth.user.info=<SCHEMA_REGISTRY_API_KEY>:<SCHEMA_REGISTRY_API_SECRET>
value.converter.schema.registry.url=https://<SCHEMA_REGISTRY_ENDPOINT>
```

3. Create `my-file-sink.properties` in the config directory, whose contents look like the following (note the security configs with `consumer.*` prefix):

```
cat ./etc/my-file-sink.properties
name=my-file-sink
connector.class=org.apache.kafka.connect.file.FileStreamSinkConnector
tasks.max=1
topics=page_visits
file=my_file.txt
```

Important

You must include the following configuration properties if you are using a self-managed connector that requires an enterprise license.

```
confluent.topic.bootstrap.servers=<cloud-bootstrap-servers>
confluent.topic.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule \
required username="<CLUSTER_API_KEY>" password="<CLUSTER_API_SECRET>";
confluent.topic.security.protocol=SASL_SSL
confluent.topic.sasl.mechanism=PLAIN
```

Important

You must include the following configuration properties if you are using a self-managed connector that uses Reporter to write response back to Kafka (for example, the Azure Functions Sink connector or the Google Cloud Functions Sink connector) .

```
reporter.admin.bootstrap.servers=<cloud-bootstrap-servers>
reporter.admin.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule \
required username="<CLUSTER_API_KEY>" password="<CLUSTER_API_SECRET>";
reporter.admin.security.protocol=SASL_SSL
reporter.admin.sasl.mechanism=PLAIN

reporter.producer.bootstrap.servers=<cloud-bootstrap-servers>
reporter.producer.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule \
required username="<CLUSTER_API_KEY>" password="<CLUSTER_API_SECRET>";
reporter.producer.security.protocol=SASL_SSL
reporter.producer.sasl.mechanism=PLAIN
```

4. Run the `connect-standalone` script with the filenames as arguments:

```
./bin/connect-standalone  ./etc/my-connect-standalone.properties ./etc/my-file-sink.properties
```

This should start a connect worker on your machine which will consume the records produced earlier using the `ccloud` command. If you tail the contents of `my_file.txt`, it should resemble the following:

```
tail -f my_file.txt
{"field1": "hello", "field2": 1}
{"field1": "hello", "field2": 2}
{"field1": "hello", "field2": 3}
{"field1": "hello", "field2": 4}
{"field1": "hello", "field2": 5}
{"field1": "hello", "field2": 6}
```

Distributed Cluster

1. Create `connect-distributed` in the config directory, whose contents look like the following (note the security configs with `consumer.*` and `producer.*` prefixes).

```
cat etc/my-connect-distributed.properties
bootstrap.servers=<cloud-bootstrap-servers>

group.id=connect-cluster

key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=false
value.converter.schemas.enable=false

internal.key.converter=org.apache.kafka.connect.json.JsonConverter
internal.value.converter=org.apache.kafka.connect.json.JsonConverter
internal.key.converter.schemas.enable=false
internal.value.converter.schemas.enable=false

# Connect clusters create three topics to manage offsets, configs, and status
# information. Note that these contribute towards the total partition limit quota.
offset.storage.topic=connect-offsets
offset.storage.replication.factor=3
offset.storage.partitions=3

config.storage.topic=connect-configs
config.storage.replication.factor=3

status.storage.topic=connect-status
status.storage.replication.factor=3

offset.flush.interval.ms=10000

ssl.endpoint.identification.algorithm=https
sasl.mechanism=PLAIN
request.timeout.ms=20000
retry.backoff.ms=500
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
username="<api-key>" password="<api-secret>";
security.protocol=SASL_SSL

consumer.ssl.endpoint.identification.algorithm=https
consumer.sasl.mechanism=PLAIN
consumer.request.timeout.ms=20000
consumer.retry.backoff.ms=500
consumer.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
username="<api-key>" password="<api-secret>";
consumer.security.protocol=SASL_SSL

producer.ssl.endpoint.identification.algorithm=https
producer.sasl.mechanism=PLAIN
producer.request.timeout.ms=20000
producer.retry.backoff.ms=500
producer.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
username="<api-key>" password="<api-secret>";
producer.security.protocol=SASL_SSL
```

2. (Optional) Add the configs to `connect-distributed` to connect to Confluent Cloud Schema Registry per the example in connect-ccloud.delta on GitHub at ccloud/examples/template_delta_configs.

```
# Confluent Schema Registry for Kafka Connect
value.converter=io.confluent.connect.avro.AvroConverter
value.converter.basic.auth.credentials.source=USER_INFO
value.converter.schema.registry.basic.auth.user.info=<SCHEMA_REGISTRY_API_KEY>:<SCHEMA_REGISTRY_API_SECRET>
value.converter.schema.registry.url=https://<SCHEMA_REGISTRY_ENDPOINT>
```

3. Run Connect using the following command:

```
./bin/connect-distributed ./etc/my-connect-distributed.properties
```

To test if the workers came up correctly, you can setup another file sink as follows. Create a fil `my-file-sink.json` whose contents are as follows:

```
cat my-file-sink.json
{
  "name": "my-file-sink",
  "config": {
    "connector.class": "org.apache.kafka.connect.file.FileStreamSinkConnector",
    "tasks.max": 3,
    "topics": "page_visits",
    "file": "my_file.txt"
  }
}
```

Important

You must include the following configuration properties if you are using a self-managed connector that requires an enterprise license.

```
"confluent.topic.bootstrap.servers":"<cloud-bootstrap-servers>",
"confluent.topic.sasl.jaas.config":
"org.apache.kafka.common.security.plain.PlainLoginModule
required username=\"<CLUSTER_API_KEY>\" password=\"<CLUSTER_API_KEY>\";",
"confluent.topic.security.protocol":"SASL_SSL",
"confluent.topic.sasl.mechanism":"PLAIN"
```

Important

You must include the following configuration properties if you are using a self-managed connector that uses Reporter to write response back to Kafka (for example, the Azure Functions Sink connector or the Google Cloud Functions Sink connector) .

```
"reporter.admin.bootstrap.servers":"<cloud-bootstrap-servers>",
"reporter.admin.sasl.jaas.config":
"org.apache.kafka.common.security.plain.PlainLoginModule
required username=\"<CLUSTER_API_KEY>\" password=\"<CLUSTER_API_KEY>\";",
"reporter.admin.security.protocol":"SASL_SSL",
"reporter.admin.sasl.mechanism":"PLAIN",

"reporter.producer.bootstrap.servers":"<cloud-bootstrap-servers>",
"reporter.producer.sasl.jaas.config":
"org.apache.kafka.common.security.plain.PlainLoginModule
required username=\"<CLUSTER_API_KEY>\" password=\"<CLUSTER_API_KEY>\";",
"reporter.producer.security.protocol":"SASL_SSL",
"reporter.producer.sasl.mechanism":"PLAIN"
```

4. Post this connector config to the worker using the curl command:

```
curl -s -H "Content-Type: application/json" -X POST -d @my-file-sink.json http://localhost:8083/connectors/ | jq .
```

This should give the following response:

```
{
  "name": "my-file-sink",
  "config": {
    "connector.class": "org.apache.kafka.connect.file.FileStreamSinkConnector",
    "tasks.max": "1",
    "topics": "page_visits",
    "file": "my_file",
    "name": "my-file-sink"
  },
  "tasks": [],
  "type": null
}
```

5. Produce some records using Confluent Cloud and tail this file to check if the connectors were successfully created.

Connect to Confluent Cloud Schema Registry

(Optional) To connect to Confluent Cloud Schema Registry, add the configs per the example in connect-ccloud.delta on GitHub at ccloud/examples/template_delta_configs.

```
# Confluent Schema Registry for Kafka Connect
value.converter=io.confluent.connect.avro.AvroConverter
value.converter.basic.auth.credentials.source=USER_INFO
value.converter.schema.registry.basic.auth.user.info=<SCHEMA_REGISTRY_API_KEY>:<SCHEMA_REGISTRY_API_SECRET>
value.converter.schema.registry.url=https://<SCHEMA_REGISTRY_ENDPOINT>
```

Please report any inaccuracies on this page or suggest an edit.

7 Votes

Last updated on Oct 17, 2019.