Install KSQL with Docker

You can deploy KSQL by using Docker containers. Starting with Confluent Platform 4.1.2, Confluent maintains images at Docker Hub (https://hub.docker.com/u/confluentinc) for KSQL Server (https://hub.docker.com/r/confluentinc/cp-ksql-server/) and the KSQL command-line interface (CLI) (https://hub.docker.com/r/confluentinc/cp-ksql-cli/).

KSQL runs separately from your Apache Kafka® cluster, so you specify the IP addresses of the cluster's bootstrap servers when you start a container for KSQL Server. To set up Confluent Platform by using containers, see Confluent Platform Quick Start (Docker) (../../../quickstart/ce-docker-quickstart.html#ce-docker-quickstart).

Use the following settings to start containers that run KSQL in various configurations.

- KSQL Headless Server Settings (Production)
- KSQL Headless Server with Interceptors Settings (Production)
- KSQL Interactive Server Settings (Development)
- KSQL Interactive Server with Interceptors Settings (Development)
- Connect KSQL Server to a Secure Kafka Cluster, Like Confluent Cloud
- Configure a KSQL Server by Using Java System Properties
- View KSQL Server Logs
- Enable the Processing Log
- Connect KSQL CLI to a Dockerized KSQL Server
- Start KSQL CLI With a Provided Configuration File
- Connect KSQL CLI to a KSQL Server Running on Another Host (Cloud)

- Wait for a Particular Phrase in a Container's Log
- Run Custom Code Before Launching a Container's Program
- Execute a KSQL script in the KSQL CLI

Scale Your KSQL Server Deployment

You can scale KSQL by adding more capacity per server (vertically) or by adding more servers (horizontally). Also, you can scale KSQL clusters during live operations without loss of data. For more information, see Scaling KSQL (../capacity-planning.html#ksql-capacity-planning-scaling).

Assign Configuration Settings in the Docker Run Command

You can dynamically pass configuration settings into containers by using environment variables. When you start a container, set up the configuration with the `-e` or `--env` flags in the `docker run` command.

For a complete list of KSQL parameters, see KSQL Configuration Parameter Reference (server-config/config-reference.html#ksql-param-reference).

In most cases, to assign a KSQL configuration parameter in a container, you prepend the parameter name with `KSQL_` and substitute the underscore character for periods. For example, to assign the `ksql.queries.file` setting in your `docker run` command, specify:

Also, you can set configuration options by using the `KSQL_OPTS` environment variable. For example, to assign the `ksql.queries.file` setting in your `docker run` command, specify:

```
-e KSQL_OPTS="-Dksql.queries.file=/path/in/container/queries.sql"
```

Properties set with `KSQL_OPTS` take precedence over values specified in the KSQL configuration file. For more information, see Setting KSQL Server Parameters (server-config/index.html#set-ksql-server-properties).

KSQL Server

The following commands show how to run KSQL Server in a container.

KSQL Headless Server Settings (Production)

You can deploy KSQL Server into production in a non-interactive, or *headless*, mode. In headless mode, interactive use of the KSQL cluster is disabled, and you configure KSQL Server with a predefined `.sql` file and the `KSQL_KSQL_QUERIES_FILE` setting. For more information, see Non-interactive (Headless) KSQL Usage (server-config/index.html#restrict-ksql-interactive).

Use the following command to run a headless, standalone KSQL Server instance in a container:

```
s.sql \
  confluentinc/cp-ksql-server:5.3.2
```

`KSQL_BOOTSTRAP_SERVERS`

A list of hosts for establishing the initial connection to the Kafka cluster.

`KSQL_KSQL_SERVICE_ID`

The service ID of the KSQL server, which is used as the prefix for the internal topics created by KSQL.

`KSQL_KSQL_QUERIES_FILE`

A file that specifies predefined KSQL queries.

KSQL Headless Server with Interceptors Settings (Production)

Confluent Platform supports pluggable *interceptors* to examine and modify incoming and outgoing records. Specify interceptor classes by assigning the `KSQL_PRODUCER_INTERCEPTOR_CLASSES` and `KSQL_CONSUMER_INTERCEPTOR_CLASSES` settings. For more info on interceptor classes, see Confluent Monitoring Interceptors (../../../control-center/installation/clients.html#controlcenter-clients).

Use the following command to run a headless, standalone KSQL Server with the specified interceptor classes in a container:

```
docker run -d \
  -v /path/on/host:/path/in/container/ \
  -e KSQL_BOOTSTRAP_SERVERS=localhost:9092 \
  -e KSQL_KSQL_SERVICE_ID=ksql_standalone_2_ \
  -e KSQL_PRODUCER_INTERCEPTOR_CLASSES=io.confluent.monitoring.clients.interceptor.MonitoringProducerInterceptor \
  -e KSQL_CONSUMER_INTERCEPTOR_CLASSES=io.confluent.monitoring.clients.interceptor.MonitoringConsumerInterceptor \
  -e KSQL_KSQL_QUERIES_FILE=/path/in/container/queries.sql \
  confluentinc/cp-ksql-server:5.3.2
```

A list of hosts for establishing the initial connection to the Kafka cluster.

`KSQL_KSQL_SERVICE_ID`

The service ID of the KSQL server, which is used as the prefix for the internal topics created by KSQL.

`KSQL_KSQL_QUERIES_FILE`

A file that specifies predefined KSQL queries.

`KSQL_PRODUCER_INTERCEPTOR_CLASSES`

A list of fully qualified class names for producer interceptors.

`KSQL_CONSUMER_INTERCEPTOR_CLASSES`

A list of fully qualified class names for consumer interceptors.

KSQL Interactive Server Settings (Development)

Develop your KSQL applications by using the KSQL command-line interface (CLI), or the graphical interface in Confluent Control Center, or both together.

Run a KSQL Server that enables manual interaction by using the KSQL CLI:

```
docker run -d \
  -p 127.0.0.1:8088:8088 \
  -e KSQL_BOOTSTRAP_SERVERS=localhost:9092 \
  -e KSQL_LISTENERS=http://0.0.0.0:8088/ \
  -e KSQL_KSQL_SERVICE_ID=ksql_service_2_ \
  confluentinc/cp-ksql-server:5.3.2
```

`KSQL_BOOTSTRAP_SERVERS`

A list of hosts for establishing the initial connection to the Kafka cluster.

`KSQL_KSQL_SERVICE_ID`

`KSQL_LISTENERS`

A list of URIs, including the protocol, that the broker listens on.

In interactive mode, a KSQL CLI instance running outside of Docker can connect to the KSQL server running in Docker.

KSQL Interactive Server with Interceptors Settings (Development)

Run a KSQL Server with interceptors that enables manual interaction by using the KSQL CLI:

```
docker run -d \
  -p 127.0.0.1:8088:8088 \
  -e KSQL_BOOTSTRAP_SERVERS=localhost:9092 \
  -e KSQL_LISTENERS=http://0.0.0.0:8088/ \
  -e KSQL_KSQL_SERVICE_ID=ksql_service_3_ \
  -e KSQL_PRODUCER_INTERCEPTOR_CLASSES=io.confluent.monitoring.clients.interceptor.MonitoringProducerInterceptor \
  -e KSQL_CONSUMER_INTERCEPTOR_CLASSES=io.confluent.monitoring.clients.interceptor.MonitoringConsumerInterceptor \
  confluentinc/cp-ksql-server:5.3.2
```

`KSQL_BOOTSTRAP_SERVERS`

A list of hosts for establishing the initial connection to the Kafka cluster.

`KSQL_KSQL_SERVICE_ID`

The service ID of the KSQL server, which is used as the prefix for the internal topics created by KSQL.

`KSQL_LISTENERS`

A list of URIs, including the protocol, that the broker listens on.

`KSQL_PRODUCER_INTERCEPTOR_CLASSES`

A list of fully qualified class names for producer interceptors.

`KSQL_CONSUMER_INTERCEPTOR_CLASSES`

interceptors (../../../control-center/installation/clients.html#controlcenter-clients).

In interactive mode, a CLI instance running outside of Docker can connect to the server running in Docker.

Connect KSQL Server to a Secure Kafka Cluster, Like Confluent Cloud

KSQL Server runs outside of your Kafka clusters, so you need specify in the container environment how KSQL Server connects with a Kafka cluster.

Run a KSQL Server that uses a secure connection to a Kafka cluster:

```
docker run -d \
  -p 127.0.0.1:8088:8088 \
  -e KSQL_BOOTSTRAP_SERVERS=REMOVED_SERVER1:9092,REMOVED_SERVER2:9093,REMOVED_SERVER3:9094 \
  -e KSQL_LISTENERS=http://0.0.0.0:8088/ \
  -e KSQL_KSQL_SERVICE_ID=default_ \
  -e KSQL_KSQL_SINK_REPLICAS=3 \
  -e KSQL_KSQL_STREAMS_REPLICATION_FACTOR=3 \
  -e KSQL_SECURITY_PROTOCOL=SASL_SSL \
  -e KSQL_SASL_MECHANISM=PLAIN \
  -e KSQL_SASL_JAAS_CONFIG="org.apache.kafka.common.security.plain.PlainLoginModule required username=\"<username>\" password=\"<strong-password>\";" \
  confluentinc/cp-ksql-server:5.3.2
```

`KSQL_BOOTSTRAP_SERVERS`

A list of hosts for establishing the initial connection to the Kafka cluster.

`KSQL_KSQL_SERVICE_ID`

The service ID of the KSQL server, which is used as the prefix for the internal topics created by KSQL.

`KSQL_LISTENERS`

A list of URIs, including the protocol, that the broker listens on.

`KSQL_KSQL_STREAMS_REPLICATION_FACTOR`
The replication factor for internal topics, the command topic, and output topics.

`KSQL_SECURITY_PROTOCOL`
The protocol that your Kafka cluster uses for security.

`KSQL_SASL_MECHANISM`
The SASL mechanism that your Kafka cluster uses for security.

`KSQL_SASL_JAAS_CONFIG`
The Java Authentication and Authorization Service (JAAS) configuration.

Learn about KSQL Security (server-config/security.html#ksql-security).

Configure a KSQL Server by Using Java System Properties

Use the `KSQL_OPTS` environment variable to assign configuration settings by using Java system properties. Prepend the KSQL setting name with `-D`. For example, to set the KSQL service identifier in the `docker run` command, use:

```
-e KSQL_OPTS="-Dksql.service.id=<your-service-id>"
```

Run a KSQL Server with a configuration that's defined by Java properties:

```
docker run -d \
  -v /path/on/host:/path/in/container/ \
  -e KSQL_BOOTSTRAP_SERVERS=localhost:9092 \
  -e KSQL_OPTS="-Dksql.service.id=ksql_service_3_  -D
ksql.queries.file=/path/in/container/queries.sql" \
  confluentinc/cp-ksql-server:5.3.2
```

cluster.

`KSQL_OPTS`
A space-separated list of Java options.

The previous example assigns two settings, `ksql.service.id` and `ksql.queries.file`. Specify more configuration settings by adding them in the `KSQL_OPTS` line. Remember to prepend each setting name with `-D`.

View KSQL Server Logs

Use the `docker logs` command to view KSQL logs that are generated from within the container:

```
docker logs -f <container-id>
```

Your output should resemble:

```
[2019-01-16 23:43:05,591] INFO stream-thread [_confluent-ksql-default_transient_1507119262168861890_1527205385485-71c8a94c-abe9-45ba-91f5-69a762ec5c1d-StreamThread-17] Starting (org.apache.kafka.streams.processor.internals.StreamThread:713)
...
```

Enable the Processing Log

KSQL emits a log of record processing events, called the processing log, to help you debug KSQL queries. For more information, see KSQL Processing Log (../developer-guide/processing-log.html#ksql-processing-log).

Assign the following configuration settings to enable the processing log.

```
KSQL_KSQL_LOGGING_PROCESSING_TOPIC_AUTO_CREATE: "tru
e"
KSQL_KSQL_LOGGING_PROCESSING_STREAM_AUTO_CREATE: "tru
e"
```

KSQL Command-line Interface (CLI)

Develop the KSQL queries and statements for your real-time streaming applications by using the KSQL CLI, or the graphical interface in Confluent Control Center, or both together. The KSQL CLI connects to a running KSQL Server instance to enable inspecting Kafka topics and creating KSQL streams and tables. For more information, see Configuring KSQL CLI (cli-config.html#install-cli-config).

The following commands show how to run the KSQL CLI in a container and connect to a KSQL Server.

Connect KSQL CLI to a Dockerized KSQL Server

Run a KSQL CLI instance in a container and connect to a KSQL Server that's running in a different container.

```
# Run KSQL Server.
docker run -d -p 10.0.0.11:8088:8088 \
  -e KSQL_BOOTSTRAP_SERVERS=localhost:9092 \
  -e KSQL_OPTS="-Dksql.service.id=ksql_service_3_  -D
listeners=http://0.0.0.0:8088/" \
  confluentinc/cp-ksql-server:5.3.2

# Connect the KSQL CLI to the server.
docker run -it confluentinc/cp-ksql-cli http://10.0.
0.11:8088
```

cluster.

`KSQL_OPTS`
A space-separated list of Java options.

The Docker network created by KSQL Server enables you to connect with a dockerized KSQL CLI.

Start KSQL CLI With a Provided Configuration File

Set up a a KSQL CLI instance by using a configuration file, and run it in a container:

```
# Assume KSQL Server is running.
# Ensure that the configuration file exists.
ls /path/on/host/ksql-cli.properties

docker run -it \
  -v /path/on/host/:/path/in/container  \
  confluentinc/cp-ksql-cli:5.3.2 http://10.0.0.11:8088 \
  --config-file /path/in/container/ksql-cli.properties
```

Connect KSQL CLI to a KSQL Server Running on Another Host (Cloud)

Run a KSQL CLI instance in a container and connect to a remote KSQL Server host:

```
docker run -it confluentinc/cp-ksql-cli:5.3.2 \
  http://ec2-blah.us-blah.compute.amazonaws.com:8080
```

Your output should resemble:

```
Having trouble? Type 'help' (case-insensitive) for a
 rundown of how things work!

ksql>
```

Interact With KSQL Running in a Docker Container

You can communicate with KSQL Server and the KSQL CLI when they run in Docker containers. The following examples show common tasks with KSQL processes that run in containers.

- Wait for an HTTP Endpoint to Be Available
- Wait for a Particular Phrase in a Container's Log
- Run Custom Code Before Launching a Container's Program
- Execute a KSQL script in the KSQL CLI

Wait for an HTTP Endpoint to Be Available

Sometimes, a container reports its state as `up` before it's actually running. In this case, the docker-compose `depends_on` dependencies aren't sufficient. For a service that exposes an HTTP endpoint, like KSQL Server, you can force a script to wait before running a client that requires the service to be ready and available.

Use the following bash commands to wait for KSQL Server to be available:

```
  echo -e $(date) "KSQL Server HTTP state: " $(curl -
s -o /dev/null -w %{http_code} http://<ksql-server-ip
-address>:8088/) " (waiting for 200)"
  sleep 5
done
```

This script pings the KSQL Server at `<ksql-server-ip-address>:8088` every five seconds, until it receives an HTTP 200 response.

Note

The previous script doesn't work with "headless" deployments of KSQL Server, because headless deployments don't have a REST API server.

To launch the KSQL CLI in a container only after KSQL Server is available, use the following Docker Compose command:

```
docker-compose exec ksql-cli bash -c \
'echo -e "\n\n⏳ Waiting for KSQL to be available befo
re launching CLI\n"; while [ $(curl -s -o /dev/null -
w %{http_code} http://<ksql-server-ip-address>:8088/)
-eq 000 ] ; do echo -e $(date) "KSQL Server HTTP stat
e: " $(curl -s -o /dev/null -w %{http_code} http://<k
sql-server-ip-address>:8088/) " (waiting for 200)" ;
 sleep 5 ; done; ksql http://<ksql-server-ip-address
>:8088'
```

Wait for a Particular Phrase in a Container's Log

Use the `grep` command and bash process substitution (http://tldp.org/LDP/abs/html/process-sub.html) to wait until the a specific phrase occurs in the Docker Compose log:

Run Custom Code Before Launching a Container's Program

You can run custom code, like downloading a dependency or moving a file, before a KSQL process starts in a container. Use Docker Compose to overlay a change on an existing image.

Get the Container's Default Command

Discover the default command that the container runs when it launches, which is either `Entrypoint` or `Cmd`:

```
docker inspect --format='{{.Config.Entrypoint}}' confluentinc/cp-ksql-server:5.3.2
docker inspect --format='{{.Config.Cmd}}' confluentinc/cp-ksql-server:5.3.2
```

Your output should resemble:

```
[]
[/etc/confluent/docker/run]
```

In this example, the default command is `/etc/confluent/docker/run`.

Run Custom Commands Before the KSQL Process Starts

In a Docker Compose file, add the commands that you want to run before the main process starts. Use the `command` option to override the default command. In the following example, the `command` option

```
  depends_on:
    - kafka
  environment:
    KSQL_BOOTSTRAP_SERVERS: <bootstrap-server-ip>:290
92
    KSQL_LISTENERS: http://0.0.0.0:8088
  command:
    - /bin/bash
    - -c
    - |
      mkdir -p /data/maxmind
      cd /data/maxmind
      curl https://geolite.maxmind.com/download/geoi
p/database/GeoLite2-City.tar.gz | tar xz
      /etc/confluent/docker/run
```

After the `mkdir`, `cd`, `curl`, and `tar` commands run, the `/etc/confluent/docker/run` command starts the `cp-ksql-server` image with the specified settings.

Note

The literal block scalar, `- |`, enables passing multiple arguments to `command`, by indicating that the following lines are all part of the same entry.

Execute a KSQL script in the KSQL CLI

The following Docker Compose YAML runs KSQL CLI and passes it a KSQL script for execution. The manual EXIT is required. The advantage of this approach, compared with running KSQL Server headless with a queries file, is that you can still interact with KSQL, and you can pre-build the environment to a desired state.

```
    - $PWD/ksql-scripts/:/data/scripts/
  entrypoint:
    - /bin/bash
    - -c
    - |
      echo -e "\n\n⏳ Waiting for KSQL to be available
before launching CLI\n"
      while [ $$(curl -s -o /dev/null -w %{http_code}
http://<ksql-server-ip>:8088/) -eq 000 ]
      do
        echo -e $$(date) "KSQL Server HTTP state: "
 $$(curl -s -o /dev/null -w %{http_code} http://<ksql
-server-ip>:8088/) " (waiting for 200)"
        sleep 5
      done
      echo -e "\n\n-> Running KSQL commands\n"
      cat /data/scripts/my-ksql-script.sql <(echo 'EX
IT')| ksql http://<ksql-server-ip>:8088
      echo -e "\n\n-> Sleeping…\n"
      sleep infinity
```

Next Steps

- Writing Streaming Queries Against Apache Kafka® Using KSQL (Docker) (../tutorials/basics-docker.html#ksql-quickstart-docker)
- Clickstream Data Analysis Pipeline Using KSQL (Docker) (../tutorials/clickstream-docker.html#ksql-clickstream-docker)

Last updated on Jan 11, 2020.