# Configure CDC in Azure Database for MySQL – Flexible Server by running Zookeeper, Kafka, Debezium on Docker containers

**Change data capture** (**CDC**) is a set of software design patterns to monitor row-level changes in database tables in response to insert, update, and delete operations.

Debezium is a distributed platform that turns your existing databases into event streams, so applications can see and respond immediately to each row-level change in the databases. Debezium is built on top of [Apache Kafka](http://kafka.apache.org/) and provides [Kafka Connect](http://kafka.apache.org/documentation.html#connect) compatible connectors that monitor specific database management systems. The Debezium MySQL connector reads the binlog, produces change events for row-level INSERT, UPDATE, and DELETE operations, and emits the change events to Kafka topics. Client applications read those Kafka topics.

[Docker](https://docs.docker.com/) is an open source containerization technology for building and containerizing your applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker’s methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

This tutorial is a quickstart to build a CDC system for Azure Database for MySQL – Flexible Server by running Zookeeper, Kafka, Debezium in Docker containers.

## Prerequisites

Before you begin completing the process outlined in this post, ensure that you have:

* An [Azure VM running Linux](https://docs.microsoft.com/azure/virtual-machines/linux/quick-create-portal).
* An [Instance of Azure Database for MySQL - Flexible Server](https://docs.microsoft.com/azure/mysql/flexible-server/quickstart-create-server-portal).

## Run Docker containers

### Install Docker

This tutorial deploys the Docker engine on Ubuntu 20.04. For installation steps, see <https://docs.docker.com/engine/install/ubuntu>.

**Note:** To deploy on other platforms, see [Install Docker Engine](https://docs.docker.com/engine/install/).

### Run ZooKeeper

The first step to start is ZooKeeper. Start a container with ZooKeeper by running:

sudo docker run -it --name zookeeper -p 2181:2181 -p 2888:2888 -p 3888:3888 quay.io/debezium/zookeeper:2.1

You should see the output of ZooKeeper in your terminal:

A computer code on a black background

Description automatically generated with low confidence

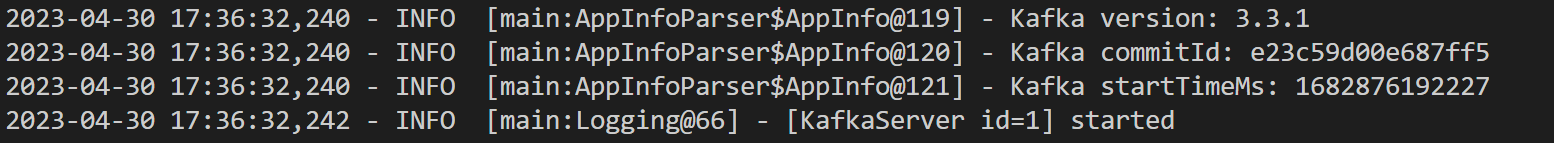
### Run Kafka

After starting Zookeeper, you can run Kafka in a new container by running:

sudo docker run -it --name kafka -p 9092:9092 --link zookeeper:zookeeper quay.io/debezium/kafka:2.1

This command runs a new container using version 2.1 of the quay.io/debezium/kafka image.

You should see the output of Kafka in your terminal:



### Run Kafka connector

After we have both Zookeeper and Kafka running, we can start the Kafka Connect service. It exposes a REST API to manage the Debezium MySQL connector.

sudo docker run -it --name connect -p 8083:8083 -e GROUP\_ID=1 -e CONFIG\_STORAGE\_TOPIC=my\_connect\_configs -e OFFSET\_STORAGE\_TOPIC=my\_connect\_offsets -e STATUS\_STORAGE\_TOPIC=my\_connect\_statuses --link kafka:kafka --link zookeeper:zookeeper quay.io/debezium/connect:2.1

### (Optional) Create a MySQL User

When you have the containers, you may need a user in Azure Database for MySQL – Flexible Server to connect with. If you already have one, this step can be skipped.

There is an example:

CREATE USER 'debezium'@'%' IDENTIFIED BY '123456';  
​  
GRANT SELECT, RELOAD, SHOW DATABASES, REPLICATION SLAVE, REPLICATION CLIENT ON \*.\* TO 'debezium'@'%';  
​  
flush privileges;

## Register the MySQL connector

By registering the Debezium MySQL connector, the connector will start monitoring the MySQL database server’s binlog. When a row in the database changes, Debezium generates a change event.

1. Configure and start the Debezium MySQL source connector

{  
 "name": "azuremysql-connector",    
 "config": {    
   "connector.class": "io.debezium.connector.mysql.MySqlConnector",  
   "tasks.max": "1",    
   "database.hostname": "{YOUR-SERVER-NAME}.mysql.database.azure.com",    
   "database.port": "3306",  
   "database.user": "{USERNAME}",  
   "database.password": "{PASSWORD}",  
   "database.server.id": "100",    
   "topic.prefix": "dbserver",    
   "database.include.list": "school",    
   "schema.history.internal.kafka.bootstrap.servers": "kafka:9092",    
   "schema.history.internal.kafka.topic": "schemahistory.school"    
}  
}

You may need to use your own properties:

* {SERVER-NAME}: Replace with the name of the Azure Database for MySQL instance
* {USER-NAME}: Replace with your username for the database.
* {PASSWORD}: Replace with the password for the specified database user account.

For more information, see [MySQL connector configuration properties](https://debezium.io/documentation/reference/2.1/connectors/mysql.html#mysql-connector-properties).

1. Use the curl command to register the MySQL connector

curl -i -X POST -H "Accept:application/json" -H "Content-Type:application/json" localhost:8083/connectors/ -d '{ "name": "azuremysql-connector", "config": { "connector.class": "io.debezium.connector.mysql.MySqlConnector", "tasks.max": "1", "database.hostname": "{YOUR-SERVER-NAME}.mysql.database.azure.com", "database.port": "3306", "database.user": "{USERNAME}", "database.password": "{PASSWORD}", "database.server.id": "100", "topic.prefix": "dbserver", "database.include.list": "school", "schema.history.internal.kafka.bootstrap.servers": "kafka:9092", "schema.history.internal.kafka.topic": "schemahistory.school" } }'

You should see the response in the terminal:

HTTP/1.1 201 Created  
Date: Sun, 30 Apr 2023 16:53:43 GMT  
Location: http://localhost:8083/connectors/azuremysql-connector  
Content-Type: application/json  
Content-Length: 529  
Server: Jetty(9.4.48.v20220622)  
​  
{"name":"azuremysql-connector","config":{"connector.class":"io.debezium.connector.mysql.MySqlConnector","tasks.max":"1","database.hostname":"{YOUR-SERVER-NAME}.mysql.database.azure.com","database.port":"3306","database.user":"{USERNAME}","database.password":"{PASSWORD}","database.server.id":"100","topic.prefix":"dbserver","database.include.list":"school","schema.history.internal.kafka.bootstrap.servers":"kafka:9092","schema.history.internal.kafka.topic":"schemahistory.school","name":"azuremysql-connector"},"tasks":[],"type":"source"}

1. Use the command below to see if the connector is included.

curl -H "Accept:application/json" localhost:8083/connectors/

["azuremysql-connector"]

## View change data capture

After registering the MySQL connector, it starts to monitoring the database for data change events. We can start to view the events.

1. Start the watch-topic utility to watch dbserver.school.MyUsers topic

By viewing the dbserver.school.MyUsers topic, you can see how the MySQL connector captured events in the database.

sudo docker run -it --name watcher --link zookeeper:zookeeper --link kafka:kafka quay.io/debezium/kafka:2.1 watch-topic -a -k dbserver.school.MyUsers

The events are key-value pairs in the topic.

1. The key has two parts, a schema and a payload. You could see the MyUsers table's primary key (PersonID) has a value of "1".

{  
 "schema": {  
 "type": "struct",  
 "fields": [  
 {  
 "type": "int32",  
 "optional": false,  
 "field": "PersonID"  
 }  
 ],  
 "optional": false,  
 "name": "dbserver.school.MyUsers.Key"  
 },  
 "payload": {  
 "PersonID": 1  
 }  
}

1. The value shows the rows of the table.

{  
 "schema": {  
 "type": "struct",  
 "fields": [  
 {  
 "type": "struct",  
 "fields": [  
 {  
 "type": "int32",  
 "optional": false,  
 "field": "PersonID"  
 },  
 {  
 "type": "string",  
 "optional": true,  
 "field": "LastName"  
 },  
 {  
 "type": "string",  
 "optional": true,  
 "field": "FirstName"  
 },  
 {  
 "type": "string",  
 "optional": true,  
 "field": "Address"  
 },  
 {  
 "type": "string",  
 "optional": true,  
 "field": "City"  
 }  
 ],  
 "optional": true,  
 "name": "dbserver.school.MyUsers.Value",  
 "field": "before"  
 },  
 {  
 "type": "struct",  
 "fields": [  
 {  
 "type": "int32",  
 "optional": false,  
 "field": "PersonID"  
 },  
 {  
 "type": "string",  
 "optional": true,  
 "field": "LastName"  
 },  
 {  
 "type": "string",  
 "optional": true,  
 "field": "FirstName"  
 },  
 {  
 "type": "string",  
 "optional": true,  
 "field": "Address"  
 },  
 {  
 "type": "string",  
 "optional": true,  
 "field": "City"  
 }  
 ],  
 "optional": true,  
 "name": "dbserver.school.MyUsers.Value",  
 "field": "after"  
 },  
 {  
 "type": "struct",  
 "fields": [  
 {  
 "type": "string",  
 "optional": false,  
 "field": "version"  
 },  
 {  
 "type": "string",  
 "optional": false,  
 "field": "connector"  
 },  
 {  
 "type": "string",  
 "optional": false,  
 "field": "name"  
 },  
 {  
 "type": "int64",  
 "optional": false,  
 "field": "ts\_ms"  
 },  
 {  
 "type": "string",  
 "optional": true,  
 "name": "io.debezium.data.Enum",  
 "version": 1,  
 "parameters": {  
 "allowed": "true,last,false,incremental"  
 },  
 "default": "false",  
 "field": "snapshot"  
 },  
 {  
 "type": "string",  
 "optional": false,  
 "field": "db"  
 },  
 {  
 "type": "string",  
 "optional": true,  
 "field": "sequence"  
 },  
 {  
 "type": "string",  
 "optional": true,  
 "field": "table"  
 },  
 {  
 "type": "int64",  
 "optional": false,  
 "field": "server\_id"  
 },  
 {  
 "type": "string",  
 "optional": true,  
 "field": "gtid"  
 },  
 {  
 "type": "string",  
 "optional": false,  
 "field": "file"  
 },  
 {  
 "type": "int64",  
 "optional": false,  
 "field": "pos"  
 },  
 {  
 "type": "int32",  
 "optional": false,  
 "field": "row"  
 },  
 {  
 "type": "int64",  
 "optional": true,  
 "field": "thread"  
 },  
 {  
 "type": "string",  
 "optional": true,  
 "field": "query"  
 }  
 ],  
 "optional": false,  
 "name": "io.debezium.connector.mysql.Source",  
 "field": "source"  
 },  
 {  
 "type": "string",  
 "optional": false,  
 "field": "op"  
 },  
 {  
 "type": "int64",  
 "optional": true,  
 "field": "ts\_ms"  
 },  
 {  
 "type": "struct",  
 "fields": [  
 {  
 "type": "string",  
 "optional": false,  
 "field": "id"  
 },  
 {  
 "type": "int64",  
 "optional": false,  
 "field": "total\_order"  
 },  
 {  
 "type": "int64",  
 "optional": false,  
 "field": "data\_collection\_order"  
 }  
 ],  
 "optional": true,  
 "name": "event.block",  
 "version": 1,  
 "field": "transaction"  
 }  
 ],  
 "optional": false,  
 "name": "dbserver.school.MyUsers.Envelope",  
 "version": 1  
 },  
 "payload": {  
 "before": null,  
 "after": {  
 "PersonID": 1,  
 "LastName": "Paris",  
 "FirstName": "Annie",  
 "Address": "ad",  
 "City": "shanghai"  
 },  
 "source": {  
 "version": "2.1.4.Final",  
 "connector": "mysql",  
 "name": "dbserver",  
 "ts\_ms": 1682909260000,  
 "snapshot": "false",  
 "db": "school",  
 "sequence": null,  
 "table": "myusers",  
 "server\_id": 2619929678,  
 "gtid": "76c7f119-e77a-11ed-915a-000d3a8e5fc2:15",  
 "file": "mysql-bin.000004",  
 "pos": 4525,  
 "row": 0,  
 "thread": 34,  
 "query": null  
 },  
 "op": "c",  
 "ts\_ms": 1682909260599,  
 "transaction": null  
 }  
}

You could use insert, update, delete statements to see the CDC events.

Connect to your Azure Database for MySQL database by running the following command:

mysql -h {SERVER-NAME}.mysql.database.azure.com -u mysqladmin –p

### Insert

In MySQL client command line, run the following statement:

mysql> insert into MyUsers values(5,'Taylor','Lucy','5 Hawk street','Paris');

In the terminal, you could see there is one more record occurs.

The payload of the key is {"PersonID":5}

{  
 "schema": {  
...  
 },  
 "payload": {  
 "PersonID": 5  
 }  
}

The payload of the value shows the row we have inserted,

"before": null,

"after": {"PersonID":5,"LastName":"Taylor","FirstName":"Lucy","Address":"5 Hawk street","City":"Paris"}

{  
 "schema": {  
...  
  
 },  
 "payload": {  
 "before": null,  
 "after": {  
 "PersonID": 5,  
 "LastName": "Taylor",  
 "FirstName": "Lucy",  
 "Address": "5 Hawk street",  
 "City": "Paris"  
 },  
...  
 }  
}

### Update

In MySQL client command line, run the following statement:

mysql> UPDATE MyUsers SET LastName='Lee' WHERE PersonID=1;

In the terminal, you could see there is one more record occurs.

The payload of the key is {"PersonID":1}

{  
 "schema": {  
...  
 },  
 "payload": {  
 "PersonID": 1  
 }  
}

The payload of the value shows the row updated,

"before":{"PersonID":1,"LastName":"Paris","FirstName":"Annie","Address":"ad","City":"shanghai"},

"after":{"PersonID":1,"LastName":"Lee","FirstName":"Annie","Address":"ad","City":"shanghai"}

{  
 "schema": {  
...  
 },  
 "payload": {  
 "before": {  
 "PersonID": 1,  
 "LastName": "Paris",  
 "FirstName": "Annie",  
 "Address": "ad",  
 "City": "shanghai"  
 },  
 "after": {  
 "PersonID": 1,  
 "LastName": "Lee",  
 "FirstName": "Annie",  
 "Address": "ad",  
 "City": "shanghai"  
 },  
...  
 }  
}

### Delete

In MySQL client command line, run the following statement:

mysql> DELETE FROM MyUsers WHERE PersonID = 3;

In the terminal, you could see there is one more record occurs.

The payload of the key is {"PersonID":3}

{  
 "schema": {  
...  
 },  
 "payload": {  
 "PersonID": 3  
 }  
}

The payload of the value shows the deleted row,

"before": {"PersonID":3,"LastName":"Hills","FirstName":"Bob","Address":"adc","City":"London"},

"after": null

{  
 "schema": {  
...  
 },  
 "payload": {  
 "before": {  
 "PersonID": 3,  
 "LastName": "Hills",  
 "FirstName": "Bob",  
 "Address": "adc",  
 "City": "London"  
 },  
 "after": null,  
...  
 }  
}

## Clean up

When you would like to clean up your application, we could easily remove the Docker containers.

sudo docker rm watcher connect kafka zookeeper