CONFLUENT

Microsoft Azure

TECHNICAL OVERVIEW

Confluent Data Warehouse Modernization with Microsoft Azure

Introduction

Organizations are increasingly looking to migrate their analytics data from existing on-prem data analytics platforms (Teradata, Cloudera, etc.) to cloud based data warehouses (DW). This process however can be long, arduous, and expensive. Some companies have mission critical ETL jobs running that are very expensive to migrate directly into a cloud DW, or cannot be suddenly disrupted. Other companies are required to work across multiple cloud platforms.

Migrating your on-prem data warehouse or connecting DWs across clouds to unlock analytics/ ML innovation doesn't need to be a multi year lift and shift. With Confluent's data warehouse modernization solution, enterprises can create a bridge across clouds and on-prem environments to start moving data immediately while unlocking additional value. Pairing Confluent with **Microsoft Azure** helps you:

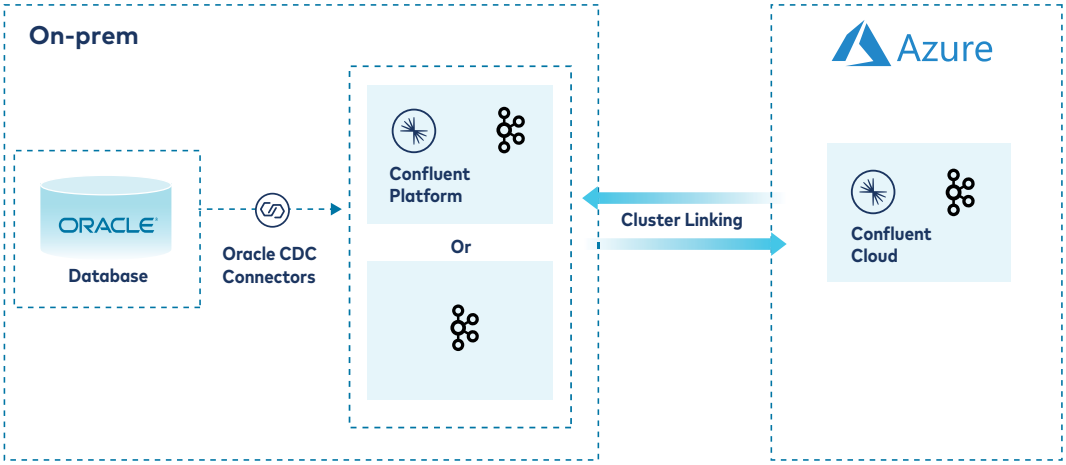
1. Reduce the TCO and time-to-value of hybrid and multicloud data pipelines powering real-time ETL for your data warehouse
2. Unlock next gen event streaming capabilities at cloud scale to power new analytics and real-time apps
3. Connect to any app, DW, or data, no matter where it lives, to get more data to and from your DW

Furthermore, Confluent enables businesses to **build an integrated data analytics platform with Microsoft Azure** with unified security, billing, and management - all accessible via Azure Marketplace.

This document details different ways to integrate data to cloud via Confluent, deployment approach, security considerations, as well as integrating on prem database with Confluent Cloud, and accompanies Confluent's Data Warehousing and Modernization study. If you are looking to understand the business value of our solution, please feel free to visit our Data Warehousing and Modernization webpage.

Migrate/Consolidate On-prem data to the cloud quickly, securely with Confluent Cloud

Confluent provides two primary mechanisms for sending data to and from the cloud. Data already in Apache Kafka® can be securely copied from Kafka on-premises, either Apache Kafka OSS or Confluent Platform, to Confluent Cloud via Cluster Linking. Alternatively, data can be streamed into Kafka using a Connector on Kafka Connect or a Fully Managed Connector on Confluent Cloud.

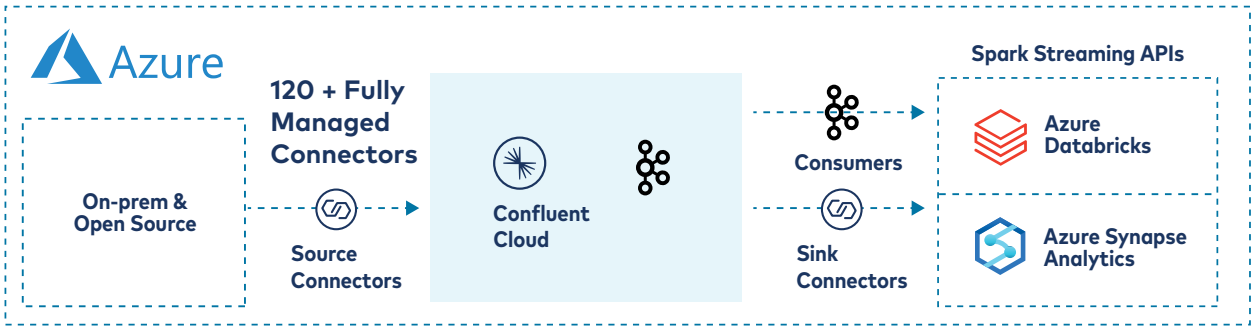


Data Transformation & Filtering

Data can be processed in flight with ksqlDB and Confluent Cloud fully managed connectors. Enabling the reduction of throughput and overall data pipeline total cost of ownership.

Extend Advanced Analytics Capabilities

Connectors available within the Kafka ecosystem, allows developers to easily and securely add data from open source, on-prem data sources to cloud native analytics projects via language APIs for .NET, C# and REST. Data in Kafka can be easily accessed by data engineers and developers and allows integrated analysis leveraging big data tools, such as Spark, leveraging languages such as Python and R. Access to event-level data enables business stakeholders and analysts to perform real-time data analysis and queries on event level data. Confluent Cloud's suite of 120+ Fully Managed Connectors allows enterprises to quickly configure connectors to popular cloud native services reducing operational overhead.



CASE STUDY: BLUE MARIPOSA—RETAILER

Let's look at how a theoretical retailer, we'll call them Blue Mariposa, sends on-premises data from Oracle database to Azure for data consolidation and advanced analytics with Azure Synapse and Azure Databricks.

This document will walk through the technical considerations and recommendations when implementing the following steps:

Step 1: Prepare the Confluent Environment

- Plan your Confluent installation
- Install Confluent Platform on an on-premise Kubernetes environment
- Prepare the Oracle DW for CDC
- Extract customer data from Oracle DW using the Oracle CDC Source Premium Connector

Step 2: Data Migration/Consolidation—On-Prem to Cloud

- Stream data from Confluent Platform to Confluent Cloud with Cluster Linking

Step 3: Data Transformation & Filtering

- Join data from multiple sources and filter before sending to Confluent Cloud with ksqlDB

Step 4: Modernize with Additional Data Sources and Real-time Advanced Analytics

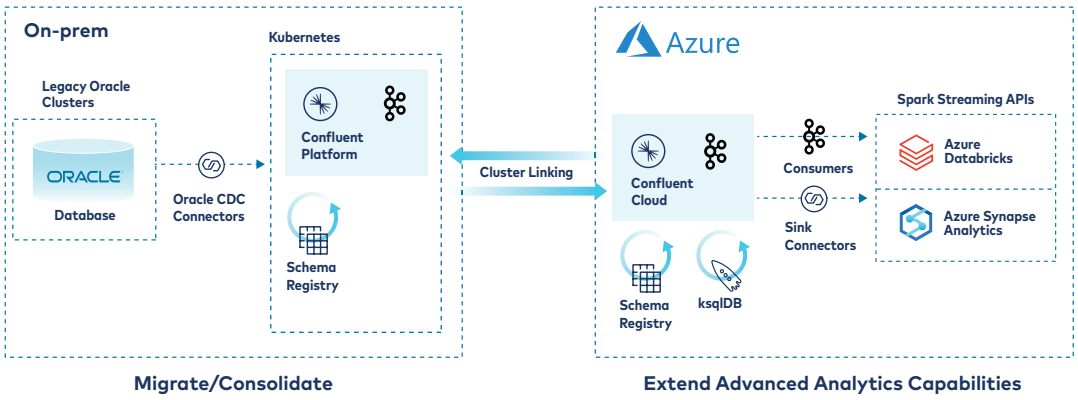
- Load the cleaned and merged data into Synapse, using the fully managed Synapse Sink Connector
- Analyze the cleaned and merged data with Synapse or Databricks, using the Spark Structured Streaming APIs

SOLUTION DIAGRAM

TToday, most enterprise data warehouses are installed on-premises, and in this scenario, we are walking through installing Confluent Platform on-premise, to initiate a data pipeline to the cloud. We will need to install Confluent Platform on-premises, as well as create a Dedicated cluster on Confluent Cloud. We will need networking access between the Kubernetes cluster and a host with Java, Confluent Platform components, the Confluent CLI and Confluent Cloud CLI to allow us to programmatically interact with clusters on both the on-prem and cloud based instances via the Confluent APIs. Access to the Control Center UI can be enabled by either port-forwarding or configuring a load balancer with the external IP address and ports of the Control Center service.

On-prem to Azure Pipeline

Oracle CDC Connector to Synase Analytics and Azure Databricks



This solution has been built using the following components:

COMPONENT	DESCRIPTION
Confluent Platform 7.0	Confluent Platform addresses requirements of modern platform streaming applications. It includes: Confluent Control Center, for end-to-end monitoring and management. Confluent Replicator, for managing multi-datacenter deployments. Confluent Auto Data Balancer, for optimizing resource utilization and easy scalability. Tiered Storage, for unlimited retention. Multi-Region Clusters, for high availability. Schema Validation, for data governance. Kubernetes Operator, for containerized installation and operation. Ansible Playbooks and Templates, for non-containerized installation and operation. And Role-Based Access Control (RBAC), Structured Audit Logs, and Secret Protection, for enterprise grade security.
Confluent Cloud Dedicated	Confluent Cloud is fully managed Kafka with no infrastructure to provision, monitor, or manage. Load is automatically distributed across brokers, consumer groups automatically rebalance when a consumer is added or removed, the state stores used by applications using the Kafka Streams APIs are automatically backed up to Confluent Cloud, and failures are automatically mitigated. Basic and Standard clusters are multi-tenant clusters. Dedicated runs on per-customer dedicated compute resources and supports the most features and custom options.
Confluent CLI 1.39.1	Confluent CLI enables management of Confluent Platform components
Confluent Cloud CLI 1.25	Confluent Cloud command line interface (CCloud CLI) allows developers to create, deploy and manage Confluent components via the command line.
Confluent for Kubernetes 1.21.2	Confluent for Kubernetes (CFK) is a cloud-native control plane for deploying and managing Confluent in your private cloud environment. CFK leverages Kubernetes-native API approach to configure, deploy, and manage Confluent Platform components.

Preparing the Confluent Environment

Plan your Confluent Installation

Choosing the right deployment model is critical for the success and scalability of the Confluent event streaming platform. You want to provide the right hardware (and cloud instances) for each use case to ensure that the system reliably provides high-throughput and low-latency data streams. You can access the Confluent Platform Reference Architecture for considerations, guidelines and recommendations for deploying Apache Kafka and Confluent Platform in Production. Once you have settled on a deployment model, you can leverage the Sizing Calculator for Apache Kafka and Confluent Platform. Here you can obtain suggestions for VM types and storage for Kafka components, based on anticipated throughput, read fanout and retention, as well as calculate how many partitions a single topic will need.

Confluent Cloud Security Controls

Data security is a primary concern when it is transported in and out of Confluent Cloud as well as when the data is persisted to disk. Confluent provides industry standard and audited protection mechanisms to ensure customers can confidently store data in Confluent Cloud. To further protect data, network-level isolation is available through VPC/VNet isolation and private networking options.

Encryption in Transit

Encryption using TLS 1.2 is required for all client connections to Confluent Cloud and HTTP Strict Transport Security (HSTS) is enabled.

Encryption at Rest Data at rest uses essentially the same default transparent AES-256 based disk encryption across AWS, Google Cloud, and Azure. The transparent disk encryption is well suited for Kafka since Kafka serializes data into raw bytes before it is being persisted to disk.

Encryption at Rest

Data at rest uses essentially the same default transparent AES-256 based disk encryption as Azure. The transparent disk encryption is well suited for Kafka since Kafka serializes data into raw bytes before it is being persisted to disk.

Encryption Key Management

Confluent Cloud uses one master key per account/project/tenant using the default cloud provider disk encryption mechanism described above. For Dedicated clusters, Confluent supports Self-Managed Keys, a.k.a. bring-your-own-key (BYOK) encryptions, in AWS or Google Cloud.

Secure Networks with Azure VNet Peering or Azure Private Link

VNet peering between Confluent Cloud can be established by providing Confluent with the CIDR range and adding the peering via the Confluent Cloud UI with parameters for Tenant ID, Subscription ID, VNet Resource Group Name and VNet Name. Confluent Cloud Console components, like topic management and ksqlDB, are set up with private endpoints that are not publicly reachable. You must configure internal access to these components.

Azure Private Link offers one-way connectivity from your VNet to a Confluent Cloud cluster, ensuring secure access by limiting traffic origination from within your VNet, without any coordination of CIDR block ranges. Private Link connections to Confluent Cloud can only be made from VNets in registered customer Azure subscriptions. With Azure Private Link, Confluent exposes service alias(es) for each new cluster, for which customers can create corresponding private endpoints in their own Azure VNets. Some Confluent Cloud Console components, like topic management, use cluster endpoints that are not publicly reachable when Azure Private Link is enabled for a cluster. Unlike VNet peering, Azure Private Link does not require the use of a proxy to forward traffic from your browser through your VNet to the Confluent Cloud cluster. You must configure your network to route requests for these components over the Private Link connection (see also, private endpoint DNS configuration). A Dedicated Cluster is required for all private networking options.

Compliance

Confluent maintains a number of compliance certifications and supports industry security standards, such as SOC 1, 2, and 3, ISO 27001, PCI DSS, CSA Star Level 1 and HIPAA . For additional information or to contact the compliance team please refer to Confluent's Trust and Security Page. Confluent's Cloud Data Processing Addendum addresses both CCPA and GDPR requirements and you can view how Confluent handles personal data via Confluent's Privacy Policy.

Install Confluent Platform to an on-premises Kubernetes environment

The Confluent for Kubernetes Quickstart is available as a starting point to install the following Confluent Platform components: Confluent Operator (1), Connect (1), Control Center (1), Brokers (3), Schema registry (1) and ZooKeepers (3). The confluent-platform.yaml file can be easily updated with configuration overrides to customize Confluent Platform components for bespoke installations. For example, this deployment will require Kafka server overrides to enable cluster linking as well as Connect overrides to deploy a Docker image of Connect stored within a cloud hosted container registry service. More complex scenarios are available via Confluent Kubernetes-Examples github repo.

To prepare for deployment, verify Kubernetes environment supports persistent volume claims and install kubectl and Helm 3. To deploy, create a namespace 'Confluent' to host pods, then set it as your default namespace. First, install Confluent operator using Helm, then Confluent-for-Kubernetes, then apply Confluent Platform customizations via the confluent-platform.yaml.

The available Helm 3 charts reference the namespace 'confluent', create namespace confluent and set current context.

```
kubectl create namespace confluent
kubectl config set-context —-current --namespace confluent
```

Add the Confluent Helm packages to Helm repo

```
helm repo add confluentinc https://packages.confluent.io/helm
helm repo update
```

Deploy Confluent Operator Pod

```
helm upgrade --install confluent-operator confluentinc/confluent-for-kubernetes
```

Deploy Confluent Platform Pods

```
kubectl apply -f
https://raw.githubusercontent.com/confluentinc/confluent-kubernetes-examples/master/quickstart-deploy/confluent-platform.yaml
```

Verify Pods are deployed

```
kubectl get pods
```

Post installation, verify connectivity with your host with Confluent components, and access Control Center via port-forwarding or load balancer.

Prepare the Oracle database for CDC

Please refer to Database prerequisites for guidelines.

Extract customer data from Oracle DB using the Oracle CDC Source Premium Connector

Confluent's ecosystem of 120+ connectors unlock data from a variety of sources, whether on premises or multi-cloud, and proprietary or open source. On premises relational database systems often hold highly critical enterprise transaction workloads. Organizations often find that they want to use the data that it stores elsewhere, such as their analytics platform or for driving real-time applications. Change data capture (CDC) solves this challenge by efficiently identifying and capturing data that has been added to, updated, or removed from tables.

Kafka Connect and the ecosystem of available connectors makes change data available to the rest of the organization. Whether in single node (standalone) or scaled to an organization-wide service (distributed), Connect provides lower time to production. Fully managed connectors hosted on Confluent Cloud abstract away all the operational burden of Connect cluster management. Most fully managed connectors can be configured and launched within minutes, with simply the connection details, credentials and a few additional parameters.

The Oracle CDC Source premium connector is a self-managed connector. Self-managed connectors can be manually installed on a Connect instance that you maintain yourself. To add additional connectors to your Confluent on Kubernetes clusters, connectors can be downloaded and added to the default Connect Docker image, then included during Confluent Platform deploy. This can be done by creating a Dockerfile, building the docker image: `docker build -t <name of image>`, and copying the Docker image to a container registry service where it can be referenced via the confluent-platform.yaml file, spec → image → application→ `<url of repo>`.

```
spec:
  replicas: 1
  image:
    application: cfltazureexamples.azurecr.io/oracle-cdc-dwm:v1
```

Once the Connect instance is up, a new connector tile will be available for selection within the Confluent Control Center UI. After selecting, the connector can be configured by adding in connector parameters similar to below, or you can import the configuration file from an existing running connector.

```
name = OracleCdcSourceConnectorConnector_0
connector.class = io.confluent.connect.oracle.cdc.OracleCdcSourceConnector
tasks.max = 1
key.converter = org.apache.kafka.connect.storage.StringConverter
value.converter = io.confluent.connect.avro.AvroConverter
topic.creation.groups = redo
oracle.server = dwm-demo.cjg294eidfaj.us-west-2.rds.amazonaws.com
oracle.port = 1521
oracle.sid = ORCL
oracle.username = <insert username>
oracle.password = <insert password>
start.from = SNAPSHOT
redo.log.row.fetch.size = 1
max.batch.size = 100
lob.topic.name.template = ${databaseName}.${schemaName}.${tableName}.${columnName}
table.inclusion.regex = ORCL.ADMIN.*
table.topic.name.template = ${databaseName}.${schemaName}.${tableName}
topic.creation.default.partitions = 5
topic.creation.redo.partitions = 1
topic.creation.redo.include = oracle-redo-log-topic
topic.creation.default.replication.factor = 3
topic.creation.redo.retention.ms = 1209600000
topic.creation.redo.replication.factor = 3
topic.creation.default.cleanup.policy = compact
topic.creation.redo.cleanup.policy = delete
value.converter.schema.registry.url = http://schemaregistry.confluent.svc.cluster.local:8081
```

There are a few config parameters to highlight:

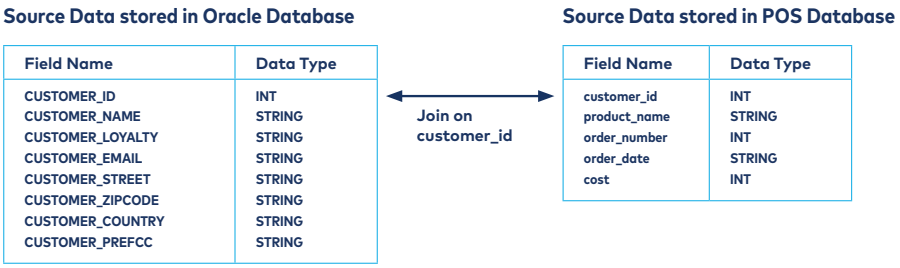
- `table.inclusion.regex` identifies the regular expression that identifies tables that this connector will capture
- `table.topic.name.template` specifies the rule for the names of the topics to which the events are written
- Since we have a few tables, including a CLOB data type, we also use `lob.topic.name.template` to specify the names of the topics where LOB values are written
- We also have a few columns with NUMERIC data types in Oracle Database, and with `"numeric.mapping":"best_fit"`; the connector will store them as an integer, a float, or a double in Kafka topics rather than as arbitrarily high precision numbers

The last group of configuration parameters uses the functionality added in Apache Kafka 2.6 to define how Connect creates topics to which the source connector writes. The following parameters create a redo log topic called oracle-redo-log-topic with one partition and create other topics (used for table-specific change events) with five partitions.

When the connector is running, it logs that it cannot find a redo log topic if there are no changes (INSERT, UPDATE, or DELETE) in the database five minutes after it completes a snapshot. To prevent this from happening, you can `increase redo.log.startup.polling.limit.ms`, or you can create a redo log topic before running the connector with `config.cleanup.policy=delete`.

Data Transformation & Filtering

ETL tools are usually required to map source to destination data and perform data transformation. ksqlDB offers a single solution for collecting streams of data, enriching them, and serving queries on new derived streams and tables. Developers can build real-time applications with the same ease and familiarity as building traditional apps on a relational database—all through a familiar, lightweight SQL syntax. ksqlDB is built on top of Kafka Streams, a lightweight, powerful Java library for enriching, transforming, and processing real-time streams of data. Having Kafka Streams at its core means ksqlDB is built on well-designed and easily understood layers of abstractions.



After pulling data from separate sources into Confluent. Data can be joined from both sources and pre-filtered using ksqlDB to reduce the amount of data being sent to the cloud and reduce ingress/egress expenses. Example queries enriching order data with customer details by joining on customer_id below.

Create Stream joining Customers and Orders data stored in Confluent Cloud

```
CREATE STREAM enriched_orders WITH
(VALUE_FORMAT='AVRO') AS SELECT c.customer_id,
c.customer_name,  c.customer_loyalty,
o.product_name, o.order_number, o.cost,
o.order_date FROM customers c INNER JOIN orders
o WITHIN 24 HOURS GRACE PERIOD 60 MINUTES
ON (o.customer_id = c.customer_id) emit changes;
```

Create Table from Enriched_Orders Stream

```
CREATE TABLE orders_enhanced WITH
(KAFKA_TOPIC='orders_enhanced',
KEY_FORMAT+'JSON', VALUE_FORMAT+'AVRO') AS
SELECT c_customer_id, customer_name,
COUNT(ordeer_number) AS num_orders, SUM(cost)
AS total_spend,
MAX(STRINGTOTIMESTAMP(ORDER_DATE,
'MM/dd/yyyy')) AS last_ordeer FROM
ENRICHED_ORDERS  GROUP BY c_customer _id,
customer_name HAVINNG
MAX(STRINGTOTIMESTAMP(ORDER_DATE,
'MM/dd/yyyy'))
STRINGTOTIMESTAMP('01/01/2017',
'MM/dd/yyyy');
```

Replicate data from Confluent Platform to Confluent Cloud with Cluster Linking

Cluster Linking enables topic data sharing between hybrid environments, as well as expedites migration of on-premises to Confluent Cloud clusters. Consumers on the destination cluster can read from local, read-only, mirrored topics to read messages produced on the source cluster. If an original topic on the source cluster is removed for any reason, you can stop mirroring that topic, and convert it to a read/write topic on the destination. Unlike, Replicator and MirrorMaker2, Cluster Linking does not require running Connect to move messages from one cluster to another, ensuring that the offsets are preserved from one cluster to another. We call this "byte-for-byte" replication. Whatever is on the source, will be mirrored precisely on the destination cluster. This simplifies moving from an on-premises Kafka cluster to a Confluent Cloud cluster. The native offset preservation you get by leveraging Confluent Server on the brokers makes this much easier to do with Cluster Linking than with other Connect based methods. A Dedicated Cluster is required for Cluster Linking.

There are two steps involved in on-premises to Confluent Cloud cluster linking. First, a source-based cluster link will be created. Then, a mirror topic will be created on the destination cluster hosted by Confluent Cloud and associated with the cluster link. Please reach out to Confluent for implementation questions.

Load the cleaned and merged data into Synapse, using the fully managed Synapse Sink Connector

Once on premises data is in Confluent Cloud, Confluent's suite of fully managed connectors provide streamlined access to building pipelines to Azure native services. With fully managed connectors, Confluent Cloud has optimized the process for configuring connectors, such as dedicated SQL pools (formerly known as Azure SQL DW), Cosmos DB and Azure Data Lake Storage, reducing effort needed to set-up. Confluent Cloud resources can also be configured and associated with cloud platform subscriptions, and provide unified security, billing, and management. Once in Confluent Cloud, live streaming data that feeds into live streaming analytics can also be served to developers enabling them to build real-time apps or services.

Azure Synapse Fully Managed Connector on Confluent Cloud

Just released this fall, is the Fully Managed Synapse Connector. Azure Synapse Analytics provides a platform for data analysts and data scientists to analyze and combine data from multiple sources. Within the Synapse workspace, data can be sinked to dedicated SQL pools from Confluent Cloud via the fully managed Synapse sink connector. Once added to the Synapse Analytics workspace, analysts have the ability to perform Advanced Analytics and reporting on data in the Confluent pipeline. The ability to access event level data, enables event-level analytics and data exploration.

Topics in Confluent Cloud can be quickly loaded into dedicated SQL pools, with dynamically created tables and schema by setting `auto.create='true'` and `auto.evolve='true'`. To configure the connector within Confluent Cloud, pass in dedicated SQL pool instance details: server.name, user, database.name and credentials.

```
{
  "name": "AzureSqlDwSinkConnector_2",
  "config": {
    "topics": "inventory",
    "input.data.format": "AVRO",
    "connector.class": "AzureSqlDwSink",
    "name": "AzureSqlDwSinkConnector_2",
    "azure.sql.dw.server.name": "kafkasqlpool.database.windows.net",
    "azure.sql.dw.user": "kafkaonazure",
    "azure.sql.dw.database.name": "kafkaonazure",
    "table.name.format": "${topic}",
    "db.timezone": "UTC",
    "auto.create": "true",
    "auto.evolve": "true",
    "tasks.max": "1"
  }
}
```

CONSIDERATIONS

- Connectivity requires INSERT permissions on the Synapse instance and possibly a firewall rule between your Confluent cluster and your Synapse instance. You can view workspace networking restrictions on Networking -> Firewall rules. Options to select 'Allow Azure services and resources to access this workspace', or add client IP address/IP ranges to access.
- Primary keys are not supported, and the connector does not support updates, upserts or deletes. When auto.evolve is enabled, if a new column with a default value is added, that default value is only used for new records. Existing records will have "null" as the value for the new column. Existing records will have "null" as the value for the new column.

Analyze the cleaned and merged data with Synapse or Databricks, using the Spark Structured Streaming APIs

Data sent to Confluent Cloud can be sent to Databricks Workspaces for ad-hoc querying and advanced analysis via Spark Streaming APIs. These APIs are available from within both R or Python code, after installing the library: confluent-kafka[avro,json,protobuf]>=1.4.2. To configure storing results to Databricks Delta tables, specify the ADLS Gen2 path, a Data Bricks File System (DBFS) mount pointing to the ADLS Gen2 path, or a local DBFS location (such as dbfs:/delta/mytable and dbfs:/delta/checkpoints/mytable) where your Delta table and streaming checkpoint will be located.

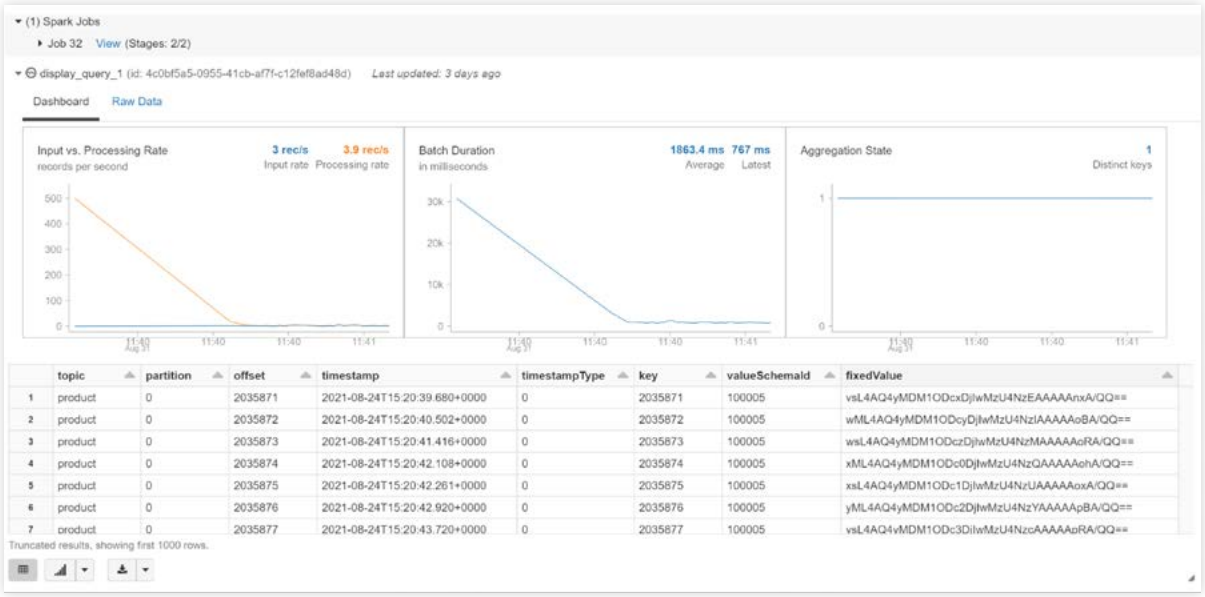
SPARK STREAMING WITH DATABRICKS EXAMPLE

```
##Define Variables
confluentBootstrapServers = "pkc-57jzz.southcentralus.azure.confluent.cloud:9092"
confluentApiKey = "<insert APIKey here>"
confluentSecret = "<insert Secret here>"
confluentRegistryApiKey = "<insert Registry APIKey here>"
confluentRegistrySecret = "<insert Registry Secret here>"
confluentTopicName = "product"
schemaRegistryUrl = "https://psrc-pg3n2.westus2.azure.confluent.cloud"

##Define Schema Registry
from confluent_kafka.schema_registry import SchemaRegistryClient
schema_registry_conf = {
    'url': schemaRegistryUrl,
    'basic.auth.user.info': '{}:{}'.format(confluentRegistryApiKey, confluentRegistrySecret)}
schema_registry_client = SchemaRegistryClient(schema_registry_conf)

##Import Library
import pyspark.sql.functions as fn
from pyspark.sql.avro.functions import from_avro
from pyspark.sql.types import StringType
binary_to_string = fn.udf(lambda x: str(int.from_bytes(x, byteorder='big')), StringType())
```

```
##Create Spark Readstream
clickstreamTestDf = (
  spark
  .readStream
  .format(“kafka”)
  .option(“kafka.bootstrap.servers”, confluentBootstrapServers)
  .option(“kafka.security.protocol”, “SASL_SSL”)
  .option(“kafka.sasl.jaas.config”, “kafkashaded.org.apache.kafka.common.security.plain.PlainLoginModule required username=’{}’ password=’{}’;”.format(confluentApiKey, confluentSecret))
  .option(“kafka.ssl.endpoint.identification.algorithm”, “https”)
  .option(“kafka.sasl.mechanism”, “PLAIN”)
  .option(“subscribe”, confluentTopicName)
  .option(“startingOffsets”, “earliest”)
  .option(“failOnDataLoss”, “false”)
  .load()
  .withColumn(‘key’, fn.col(“key”).cast(StringType()))
  .withColumn(‘fixedValue’, fn.expr(“substring(value, 6, length(value)-5)”))
  .withColumn(‘valueSchemaId’, binary_to_string(fn.expr(“substring(value, 2, 4)”)))
  .select(‘topic’, ‘partition’, ‘offset’, ‘timestamp’, ‘timestampType’, ‘key’, ‘valueSchemaId’,’fixedValue’)
)
```

Data within the Spark stream can be displayed dynamically from within the Databricks workspace.

```
##Display
display(clickstreamTestDf)
```



CONSIDERATIONS

Local Databricks File System (DBFS) locations are not recommended for production jobs; however, they don't require any additional authentication and enable quick development testing. For production installs, you can also look into deploying Azure Databricks workspace with Secure Cluster Connectivity, with no open ports or public IP Addresses, or within a dedicated Virtual Network (VNet).

Within Databricks, the Secrets API can be used to pass credentials into code within Azure Databricks workbooks and jobs via variables. To secure API keys, Secrets, Schema Registry API keys, Schema Registry Secrets, it is recommended to implement either an Azure Key Vault-backed scope in which secrets are stored in Azure-managed storage and encrypted with a cloud-based specific encryption key or create a Databricks-backed secret scope in which secrets are stored in Databricks-managed storage and encrypted with a cloud-based specific encryption key. In both scenarios, Credentials will be stored outside of workbook code, in a manner in which they can be accessed and used, but will show as redacted to developers. If you only have access to one key vault, it is recommended to leverage the Databricks-backed scope, as scopes referencing the same key vault will provide users access to both scopes. The Secrets utility dbutils.secrets, allows developers access to secrets via jobs and workbooks.

```
#confluentApiKey = dbutils.secrets.get(scope = "confluentDev", key = "<insert api-key>")
#confluentSecret = dbutils.secrets.get(scope = "confluentDev", key = "<insert secret")
#confluentRegistryApiKey = dbutils.secrets.get(scope = "confluentDev", key = "<insert registry-api-key>")
#confluentRegistrySecret = dbutils.secrets.get(scope = "confluentDev", key = "<insert registry-secret>")
```

A full demo of this functionality is detailed here.

For more information on Modernizing your Data Warehouse with Microsoft Azure please feel free to visit our Data Warehousing Modernization webpage or read our blog.

Resources

Start your 3-month trial of Confluent with up to $200 off on each of your first 3 monthly bills.
Try **FREE** on Azure Marketplace

Connectors on Confluent Hub

Debezium CDC Source Connector

Azure Synapse Analytics Sink Connector

Source Code on GitHub

Confluent Azure Example

—Collection of code examples maintained by Confluent SE Gianluca Natali.

Concepts, Documents and Training

Set up a Confluent Cloud Account

Set up a Confluent Platform Account