**Overview of Confluent Cloud Examples**

[Confluent Cloud](https://www.confluent.io/confluent-cloud) is a resilient, scalable streaming data service based on Apache Kafka®, delivered as a fully managed service. It has a web interface and local command-line interface that you can use to manage cluster resources, Kafka topics, Schema Registry, and other services.

This page describes a few resources to help you build and validate your solutions on Confluent Cloud.

**Cost to Run Examples**

Caution

All of these scripted Confluent Cloud examples use real Confluent Cloud resources and may be billable. An example may create a new Confluent Cloud environment, Kafka cluster, topics, ACLs, and service accounts, and resources that have hourly charges like connectors and ksqlDB applications. To avoid unexpected charges, carefully [evaluate the cost of resources](https://docs.confluent.io/cloud/current/billing/overview.html#cloud-billing) before you start. After you are done running a Confluent Cloud example, destroy all Confluent Cloud resources to avoid accruing hourly charges for services and verify that they have been deleted.

Confluent Cloud Free Trial

When you sign up for Confluent Cloud, a free credit is added to your billing account. This credit expires after 30 days. Use this free trial period to defray the cost of running examples. For more information, see [Deploy Free Clusters on Confluent Cloud](https://docs.confluent.io/cloud/current/get-started/free-trial.html#free-trial).

**Available Examples**

Confluent Cloud Quickstart

The [Confluent Cloud Quickstart](https://github.com/confluentinc/examples/tree/latest/cp-quickstart/) is an automated version of the [Confluent Platform Quickstart](https://docs.confluent.io/platform/current/platform-quickstart.html), but this one runs in Confluent Cloud.

A diagram of a diagram of a diagram

Description automatically generated

ccloud-stack Utility

The [ccloud-stack Utility for Confluent Cloud](https://docs.confluent.io/platform/current/tutorials/examples/ccloud/docs/ccloud-stack.html" \l "ccloud-stack) creates a stack of fully managed services in Confluent Cloud. Executed with a single command, it is a quick way to create fully managed components in Confluent Cloud, which you can then use for learning and building other demos. Do not use this in a production environment. The script uses the Confluent CLI to dynamically do the following in Confluent Cloud:

* Create a new environment.
* Create a new service account.
* Create a new Kafka cluster and associated credentials.
* Enable Confluent Cloud Schema Registry and associated credentials.
* Create a new ksqlDB app and associated credentials.
* Create ACLs with wildcard for the service account.
* Generate a local configuration file with all above connection information, useful for other demos/automation.

A group of blue icons

Description automatically generated

Tutorials

There are many more tutorials, courses, videos and more at at <https://developer.confluent.io/>. This site features full code examples using Kafka, Kafka Streams, and ksqlDB to demonstrate real use cases. You can run the tutorials locally or some with Confluent Cloud.

Client Code Examples

If you are looking for code examples of producers writing to and consumers reading from Confluent Cloud, or producers and consumers using Avro with Confluent Schema Registry, refer to [Code Examples for Apache Kafka](https://docs.confluent.io/platform/current/tutorials/examples/clients/docs/clients-all-examples.html#clients-all-examples). It provides client examples written in various programming languages.

A screenshot of a phone

Description automatically generated

Confluent CLI

The [Confluent CLI tutorial](https://docs.confluent.io/platform/current/tutorials/examples/ccloud/docs/beginner-cloud.html#ccloud-cli-tutorial) is a fully scripted example that shows users how to interact with Confluent Cloud using the Confluent CLI. It steps through the following workflow:

* Create a new environment and specify it as the default.
* Create a new Kafka cluster and specify it as the default.
* Create a user key/secret pair and specify it as the default.
* Produce and consume with the Confluent CLI.
* Create a service account key/secret pair.
* Run a Java producer: before and after ACLs.
* Run a Java producer: showcase a Prefix ACL.
* Run Connect and kafka-connect-datagen connector with permissions.
* Run a Java consumer: showcase a Wildcard ACL.
* Delete the API key, service account, Kafka topics, Kafka cluster, environment, and the log files.

A blue and black square with a white square and a black background

Description automatically generated

Observability for Kafka Clients to Confluent Cloud

The [observability for Apache Kafka clients to Confluent Cloud example](https://docs.confluent.io/platform/current/tutorials/examples/ccloud-observability/docs/index.html#ccloud-observability-index) showcases which client metrics to monitor with various failure scenarios and dashboards. The clients are running against Confluent Cloud. The example creates a Confluent Cloud cluster, Java producers and consumers, Prometheus, Grafana, and various exporters. The same principles can be applied to any other time-series database or visualization technology and non-java clients–they generally offer similar metrics.

A blue and white computer screen with a exclamation mark

Description automatically generated

Cloud ETL

The [cloud ETL example](https://docs.confluent.io/platform/current/tutorials/examples/cloud-etl/docs/index.html#cloud-etl) showcases a cloud ETL solution leveraging all fully-managed services on Confluent Cloud. Using the Confluent CLI, the example creates a source connector that reads data from an AWS Kinesis stream into Confluent Cloud, then a Confluent Cloud ksqlDB application processes that data, and then a sink connector writes the output data into cloud storage in the provider of your choice (GCP GCS, AWS S3, or Azure Blob).

A diagram of a cloud computing network

Description automatically generated

On-Premises Kafka to Confluent Cloud

The [hybrid cloud example](https://docs.confluent.io/platform/current/tutorials/cp-demo/docs/index.html#cp-demo) and playbook showcase a hybrid Kafka deployment: one cluster is a self-managed cluster running locally, the other is a Confluent Cloud cluster. Replicator copies the on-prem data to Confluent Cloud so that stream processing can happen in the cloud.

A diagram of a cloud computing process

Description automatically generated

Microservices in the Cloud

The [microservices cloud example](https://docs.confluent.io/platform/current/tutorials/examples/microservices-orders/docs/index.html#tutorial-microservices-orders) showcases an order management workflow targeting Confluent Cloud. Microservices are deployed locally on Docker, and they are configured to use a Kafka cluster, ksqlDB, and Confluent Schema Registry in Confluent Cloud. Kafka Connect is also deployed locally on Docker, and it runs a SQL source connector to produce to Confluent Cloud and a Elasticsearch sink connector to consume from Confluent Cloud.

A diagram of a diagram

Description automatically generated

**Build Your Own Cloud Demo**

ccloud-stack Utility

The [ccloud-stack Utility for Confluent Cloud](https://docs.confluent.io/platform/current/tutorials/examples/ccloud/docs/ccloud-stack.html" \l "ccloud-stack) creates a stack of fully managed services in Confluent Cloud. Executed with a single command, it is a quick way to create fully managed components in Confluent Cloud, which you can then use for learning and building other demos. Do not use this in a production environment. The script uses the Confluent CLI to dynamically do the following in Confluent Cloud:

* Create a new environment.
* Create a new service account.
* Create a new Kafka cluster and associated credentials.
* Enable Confluent Cloud Schema Registry and associated credentials.
* Create a new ksqlDB app and associated credentials.
* Create ACLs with wildcard for the service account.
* Generate a local configuration file with all above connection information, useful for other demos/automation.

A group of blue icons

Description automatically generated

Autogenerate Configurations to connect to Confluent Cloud

The *configuration generation script <auto-generate-configs>* reads a configuration file and auto-generates delta configurations for all Confluent Platform components and clients. Use these per-component configurations for Confluent Platform components and clients connecting to Confluent Cloud:

* Confluent Platform Components:
  + Schema Registry
  + ksqlDB Data Generator
  + ksqlDB
  + Confluent Replicator
  + Confluent Control Center
  + Kafka Connect
  + Kafka connector
  + Kafka command line tools
* Kafka Clients:
  + Java (Producer/Consumer)
  + Java (Streams)
  + Python
  + .NET
  + Go
  + Node.js
  + C++
* OS:
  + ENV file

Self Managed Components to Confluent Cloud

This [Docker-based environment](https://github.com/confluentinc/cp-all-in-one/tree/latest/cp-all-in-one-cloud) can be used with Confluent Cloud. The docker-compose.yml launches all services in Confluent Platform (except for the Kafka brokers), runs them in containers on localhost, and automatically configures them to connect to Confluent Cloud. Using this as a foundation, you can then add any connectors or applications.

A diagram of a computer program

Description automatically generated

Put It All Together

You can chain these utilities to build your own hybrid examples that span on-prem and Confluent Cloud, where some self-managed components run on-prem and fully-managed services run in Confluent Cloud.

For example, you may want an easy way to run a connector not yet available in Confluent Cloud. In this case, you can run a self-managed connect worker and connector on prem and connect it to your Confluent Cloud cluster. Or perhaps you want to build a Kafka demo in Confluent Cloud and run the REST Proxy client or Confluent Control Center against it.

You can build any example with a mix of fully-managed services in Confluent Cloud and self-managed components on localhost, in a few easy steps.

1. Create a [ccloud-stack](https://docs.confluent.io/platform/current/tutorials/examples/ccloud/docs/ccloud-stack.html" \l "ccloud-stack) of fully managed services in Confluent Cloud. One of the outputs is a local configuration file with key-value pairs of the required connection values to Confluent Cloud. (If you already have provisioned your Confluent Cloud resources, you can skip this step).
2. ./ccloud\_stack\_create.sh
3. Run the *configuration generation script <auto-generate-configs>*, passing in that local configuration file (created in previous step) as input. This script generates delta configuration files for all Confluent Platform components and clients, including information for bootstrap servers, endpoints, and credentials required to connect to Confluent Cloud.
4. *# stack-configs/java-service-account-<SERVICE\_ACCOUNT\_ID>.config is generated by step above*
5. ./ccloud-generate-cp-configs.sh stack-configs/java-service-account-<SERVICE\_ACCOUNT\_ID>.config

One of the generated delta configuration files from this step is for environment variables, and it resembles [this example](https://github.com/confluentinc/examples/tree/latest/ccloud/template_delta_configs/env.delta), with credentials filled in.

export BOOTSTRAP\_SERVERS="<CCLOUD\_BOOTSTRAP\_SERVER>"

export SASL\_JAAS\_CONFIG="org.apache.kafka.common.security.plain.PlainLoginModule required username='<CCLOUD\_API\_KEY>' password='<CCLOUD\_API\_SECRET>';"

export SASL\_JAAS\_CONFIG\_PROPERTY\_FORMAT="org.apache.kafka.common.security.plain.PlainLoginModule required username='<CCLOUD\_API\_KEY>' password='<CCLOUD\_API\_SECRET>';"

export REPLICATOR\_SASL\_JAAS\_CONFIG="org.apache.kafka.common.security.plain.PlainLoginModule required username='<CCLOUD\_API\_KEY>' password='<CCLOUD\_API\_SECRET>';"

export BASIC\_AUTH\_CREDENTIALS\_SOURCE="USER\_INFO"

export SCHEMA\_REGISTRY\_BASIC\_AUTH\_USER\_INFO="<SCHEMA\_REGISTRY\_API\_KEY>:<SCHEMA\_REGISTRY\_API\_SECRET>"

export SCHEMA\_REGISTRY\_URL="https://<SCHEMA\_REGISTRY\_ENDPOINT>"

export CLOUD\_KEY="<CCLOUD\_API\_KEY>"

export CLOUD\_SECRET="<CCLOUD\_API\_SECRET>"

export KSQLDB\_ENDPOINT=""

export KSQLDB\_BASIC\_AUTH\_USER\_INFO=""

1. Source the above delta env file to export variables into the shell environment.
2. *# delta\_configs/env.delta is generated by step above*
3. source delta\_configs/env.delta
4. Run the desired Confluent Platform services locally using [this Docker-based example](https://github.com/confluentinc/cp-all-in-one/tree/latest/cp-all-in-one-cloud). The Docker Compose file launches Confluent Platform services on your localhost and uses environment variable substitution to populate the parameters with the connection values to your Confluent Cloud so that they can connect to Confluent Cloud. If you want to run a single service, you can bring up just that service.
5. docker-compose up -d <service>

In the case of running a self-managed connector locally that connects to Confluent Cloud, first add your desired connector to the base Kafka Connect Docker image as described in [Add Connectors or Software](https://docs.confluent.io/home/connect/extending.html), and then substitute that Docker image in your Docker Compose file.

1. Refer to the [library of bash functions](https://github.com/confluentinc/examples/tree/latest/utils/ccloud_library.sh) for examples on how to interact with Confluent Cloud via the Confluent CLI.

Any Confluent Cloud example uses real Confluent Cloud resources. After you are done running a Confluent Cloud example, manually verify that all Confluent Cloud resources are destroyed to avoid unexpected charges.