Recommendations for Developers Using Confluent Cloud™

Written by: Gwen Shapira

June 2018

Table of Contents

Introduction **3**

Understanding the Confluent Cloud Service **4**

Sizing Your Cloud Cluster **4**

- Throughput 4
- Retention and Compaction 5
- Availability 6

Architecture Considerations for Cloud Success **6**

- Cloud Native Design 6
- Resilience 6
- Microservices 7
- Serverless Architecture 7

Using Confluent Platform with Confluent Cloud **8**

- Clients 8
  - ZooKeeper and the ClientAdmin API. 8

Monitor the Applications **8**

- Client Metrics 8
- End-to-end Stream Monitoring 9
- Consumer Offsets 9

Security and Compliance **9**

- Encryption, Authentication, and Authorization 9
- Enterprise Security 10
- Regulatory Compliance 10

More than one Cluster — Multi-region and Multi-Cloud deployments **10**

Summary **10**

Introduction

Confluent Cloud is a fully managed Apache Kafka as a Service in the public cloud. Using Confluent Cloud allows developers to focus on building applications, microservices and data pipelines, rather than on managing the underlying infrastructure. Confluent Cloud can serve as a convenient starting point for developers writing their first Kafka-based application, as well as a reliable and performant production platform.

But — just because we run Kafka for you, doesn't mean your job is done!

In this white paper, we offer recommendations and best practices for designing data architectures that will work well with Confluent Cloud. In addition to discussing integration with Confluent Cloud itself, we tackle the special challenges and benefits of running software in Cloud environments and offer guidance that will help you plan your way to a reliable, performant, and cost-effective deployment.

Understanding the Confluent Cloud Service

Confluent Cloud is designed to bring you the reliability, speed, and scalability of Apache Kafka® without the operational overhead of running it. Confluent Cloud is a fully managed data streaming service hosted on multiple regions of public cloud providers.

Confluent Cloud puts Confluent's extensive knowledge and experience managing large-scale streaming data environments into practice as a fully managed service. It significantly reduces the operational complexity of implementing and managing a reliable and secure enterprise streaming platform.

Since the goal of Confluent Cloud is to relieve organizations and developers from the challenges of operating a streaming platform at scale, Confluent Cloud abstracts away the details of operating the platform — no more choosing instance types, storage options, network optimizations and number of nodes. In true Serverless fashion, you just need to understand your data requirements and Confluent Cloud will be as elastic as your workload.

The billing is inclusive of all compute, network and storage costs charged by the cloud provider, as well as outbound data transfer from the Confluent Cloud cluster. There is no charge per message. Confluent also handles patches and upgrades, so your Confluent Cloud upgrades will always include the most recent features and bug fixes. Since Kafka supports rolling upgrades and both forward and backward compatibility, most upgrades are transparent and will have no impact on your client applications. We can upgrade the brokers on Confluent Cloud and any clients applications will just keep on working, with the version they currently have. No changes necessary. We will communicate upgrades when there is impact, for example security patches (see the support portal for Cloud).

Confluent Cloud has two plans:

- Confluent Cloud Enterprise is production-scale platform for enterprise applications that require 24x7 support, SLAs and advanced security features.
- Confluent Cloud Professional provides a self-service provisioning model for developers and organizations who are getting started on their Kafka journey.

Sizing Your Cloud Cluster

Confluent Cloud has capacity parameters you can customize and which impact the price you will pay. Which means you need to do some capacity planning. The parameters you can modify are throughput, retention and availability.

Throughput

Throughput is often the most challenging aspect to plan for, and since it impacts pricing, it is worth taking the time to do well. Here are a few suggestions:

- **Start small:** If you are planning to use Confluent Cloud Professional for development (i.e. instead of installing Kafka on your laptop), go with the smallest tier.
- **Enable compression:** In order to maximize throughput at lower cost, enable compression on the producers writing to Confluent Cloud (using compression.type in the producer configuration) and choose the throughput tier based on compressed size. Snappy compression has very low overhead and will lower the throughput compared to uncompressed. It is like a cloud freebie.
- **Batch events:** With Apache Kafka clients version 0.11 and higher, the message format is much more efficient when sending message batches. Confluent Cloud always runs the latest Kafka version, so take advantage of the new efficiency improvements! Test your workload with different batch.size and latency.ms settings on your producers to send more messages without having to pay more for throughput.

- **Estimate average and peak throughput:** It is likely that your workload is variable, and you may be tempted to size your cluster based on your peak throughput. However, with this choice you are likely to over-pay. If you tell us you have 2 MB/s write throughput, we'll assume that over a week you'll write 1.2TB and you will be charged for storage accordingly. However, this calculation is only correct if your average throughput over the course of a week is 2 MB/s. If you only use 2 MB/s at peak hours, and your average is 1 MB/s, you won't need all the storage that will be allocated (and charged for). As a rule of thumb — estimate both peak throughput and average over your entire retention period (see below) If your peak throughput is less than 3 times the average, plan capacity based your average. If your peak throughput is more than 3 times the average, plan capacity based on a third of the peak throughput. If you consistently exceed the throughput available for your cluster we will contact you about your subscription. If you exceed even our over-provisioned limit, this will cause back-pressure and producer retries. As long as your applications handle producer back-pressure correctly, you will not lose any events. See below for details on client monitoring.
- **Estimate fan-out:** It is important to plan for fan-out, i.e. the number of consumer groups that will read from each topic. Each event is produced once, but can be read any number of times. In a well-constructed event-driven architecture, you expect a number of consumers for each event — this is the entire point of the architecture. So when you provision a cluster, pay special attention to the "bytes out" limits and make sure it takes fan-out into account and that there is sufficient bandwidth for all consumers.
- **Take intermediate topics into account:** If you have Kafka Streams, KSQL or Control Center applications connected to Confluent Cloud, those applications may create intermediate and internal topics — topics that are not accessed by your application directly but are necessary for data processing and recovery. Since these applications stream events through these topics, they add throughput. As you estimate the cluster capacity, make sure you also account for throughput on these topics.

As you plan capacity, keep in mind that Confluent Cloud Professional has three available pre-set throughput options: 1MB, 2MB and 5MB. Use the advice above to find which tier is the best fit for you. The three pre-set throughput options assume a ratio of two consumers to each producer. This is a great fit when you are deploying a cluster for your first 1-3 applications. For example, you can deploy one microservice producing events that are then consumed by another service and also stream the same events to Amazon S3. Our Enterprise plan will be useful once your platform evolves beyond the first few use-cases.

For Confluent Cloud Enterprise, our sales team will help you with capacity planning when pricing the cluster. Use the guide above to provide them with accurate information.

Retention and Compaction

Kafka is typically used with time-based retention policies where it stores all messages for a set amount of time — whether they were consumed or not. This is one of the reasons Kafka scales and maintains its performance as the number of consumers grows.

Confluent Cloud Enterprise customers choose the retention period for the cluster, and this can be as high as your use-case requires. We recommend configuring a minimum of three days — this means that in case of unexpected application issues, you have up to three days to resolve the issues before you risk losing data as it expires from Kafka. Three days means you can wait until Monday to handle issues and rest assured your data will still be around. Keep in mind that storage is one of the factors that affect the price of Confluent Cloud, so if you are not sure about your exact requirements, start with a lower plausible estimate and adjust as you go.

In addition to time-based retention, Kafka's **log compaction** feature is available for Confluent Cloud. If you enable log compaction for a topic, Confluent Cloud will store the latest event for every key in the topic, for as long as the topic exists. A compacted topic can be seen as a snapshot: it is useful for restoring state for an in-memory service, a persistent data store, or reloading a cache. It allows downstream consumers to restore their state.

Confluent Cloud Professional storage limits are expressed in storage size rather than number of days. You can choose between 0.5, 1.5 and 5.0 TB. This storage is allocated to all topics in the cluster, both with time-based retention and compacted topics and it is based on actual data size. Confluent Cloud handles replication behind the scenes so you actually get three times the amount you chose. If you always thought of your data requirements in terms of days, you'll need to calculate how this translated to storage size (Each 1MB/s writes stored for seven days implies around 600GB of storage).

Availability

All Confluent Cloud clusters are highly available, with three replicas per partition. Confluent Cloud Enterprise customers can choose a high durability option of placing each replica in a separate availability zone, which means your cluster will survive failure of an entire availability zone. This is recommended for critical use-cases and Confluent also provides an SLA if you choose the high durability option.

Architectural Considerations for Cloud Success

Confluent Cloud provides fully managed Apache Kafka, which means that Confluent is responsible for the availability, reliability and performance of your Kafka clusters. You are still responsible for the applications and microservices that you develop and deploy in the Cloud. Since we develop, test and deploy all our applications in a cloud environment, we have recommendations for running event-driven applications in the cloud. Read on!

Cloud Native Design

Keep in mind that not just the Kafka cluster is running in the Cloud, but your applications as well. Of course, you still have applications running in the on-prem data center. You don't need to "lift and shift" them to the cloud just because Kafka is in the cloud, but it is also not recommended to connect on-premise applications to cloud data stores directly. In order to connect on-premises applications to Confluent Cloud, we recommend the "Bridge to Cloud" pattern in which you use Confluent Replicator to stream data between on-premise Kafka clusters and Confluent Cloud.

However, we can assume that some applications will migrate to the cloud over time, and that new applications will be developed on the cloud from the ground up.

We've noticed that developers who run applications in the cloud for the first time are often surprised by the volatility of the cloud environment. IPs can change, certificates can expire, servers are restarted, entire instances sometimes disappear, network packets are lost more frequently than they do in most data centers, and performance of each instance can change without warning. While it is always a good idea to plan for change, when you are running applications in the cloud, this is mandatory.

Planning for change isn't the same as planning for failure. Since cloud environments are built for frequent change, a cloud-native architecture lets you use the volatile environment to your advantage.

Being cloud-native involves many concepts and moving parts, and while a complete organizational transformation is outside the scope of this paper, there are several principles that are especially important to keep in mind as you build and deploy cloud applications with Confluent Cloud clusters:

Resilience

The key to successful cloud deployments is to build resilient applications. Applications that handle the volatility of cloud environments gracefully. At Confluent, we test all our platform components extensively in cloud environments. This includes everything from unit tests, integration tests, long running tests, failure injection tests and random failure injection tests (also known as "chaos engineering"). The range of unexpected events we and our customers encounter at random are quite impressive - a Kafka client that didn't support leader election completely, a failure to renew security certificates and even a key SaaS provider who failed to renew their domain registration, causing any number of API calls to fail.

We highly recommend investing in extensive failure testing as well as long running tests for your cloud applications. But preparing for failure is more than just new tests. It is a way of thinking that starts at design and ends with engineering culture. If you are migrating to the cloud and to microservices, you'll want to embrace some new ideas. This presentation by Adrian Cockroft, currently at AWS and formerly at Netflix will be a good place to start.

Microservices

Microservices architectures build applications as a collection of distributed, loosely-coupled services. This works well in a cloud environment where the cloud providers themselves give you access to distributed services. Data storage in its many forms is typically handled by external services — whether Confluent Cloud's managed Kafka, AWS's DynamoDB and S3, Google Cloud Platform's BigQuery and so on. This means that the microservices that make up your application can be stateless and rely on other services to handle their state.

Building your application to speak to Kafka with stateless microservices allows you to build more resilient applications, since loss of a service instance doesn't cause loss of data, and processing can instantly move to another instance of the microservice. And it is far easier to scale components automatically by deploying additional instances of microservices.

Serverless Architecture

Serverless architectures are architectures that either rely extensively on 3rd party services that are exposed only via API calls or ephemeral functions reacting to events (FaaS or Lambda). Serverless architectures combine 3rd party services and FaaS with other microservices to provide the functionality required from the application. Confluent Cloud is a good way to integrate microservices with Functions and other APIs.

Serverless services need to conform to 3 important criteria:

1. **Scale on-demand.** Serverless services should automatically scale up when there is higher demand (more events, more API calls) and scale down when demand is lower.
2. **Price per call.** Serverless services are priced per-event or per-API call. There are no fixed costs.
3. **No operations.** Other than calling an API or configuring the function there should be no operational overhead. Scaling up and down, failure recovery and upgrades should all happen without the user intervention.

Confluent Cloud isn't serverless at the moment. While it does handle scaling, failures and upgrades without user intervention, the pricing model doesn't match the serverless model.

However, there are a few ways you can use Confluent Cloud in order to implement a Serverless Architecture:

- **Event Broker:** event broker is Kafka's oldest use-case, but for serverless architectures it is critical. Most cloud providers have built-in integration between their own FaaS and the rest of their services, but suppose that you use Azure IOT hub to collect device events and then want to use a Google Function with its Tensorflow training model to react to these events. Azure IOT does not natively trigger events in Google Functions. But with the large Kafka Connect ecosystem, it is rather easy to get events from Azure IOT to Confluent Cloud in Azure, replicate to Confluent Cloud in GCP and have Google Function react to those events.
- **KSQL:** KSQL servers allow you to create long-running, auto-scaling streams transformations by publishing SQL to a REST API. This gives you a nice serverless experience with SQL language, which is well suited for data transformations and stream processing, rather than the more limited Lambda API.

Using Confluent Platform with Confluent Cloud

We are aiming to provide all the services available as part of the Confluent Platform as part of the managed Confluent Cloud. Currently, we include Apache Kafka and we will add the rest of the platform throughout 2018. Services that are not provided as part of Confluent Cloud can be self-hosted on the same cloud provider as Confluent Cloud and configured to use the Confluent Cloud Kafka cluster. Services that can be self-hosted include all services that are part of Confluent Cloud and also Confluent KSQL, Confluent Schema Registry, Confluent REST Proxy, Kafka Connect (including all connectors), Confluent Replicator and Confluent Control Center.

Confluent Platform documentation contains full instructions on how to connect every component in the platform to Confluent Cloud.

Clients

Confluent Cloud clusters use the well known Kafka protocol. This means that any Kafka client that supports version 0.9.0 and higher of the Kafka protocol (i.e. does not rely on Apache ZooKeeper), should work well in Confluent Cloud. Our users use the Apache Kafka Java clients, librdkafka, kafka-python, confluent-python, confluent-go, Sarama client for Go, ruby-kafka client by Zendesk, kafka-node client, and many more.

Confluent supports the Apache Kafka Java clients, Kafka Streams APIs, and clients for C, C++, .Net, Python and Go. Other clients, and the requisite support, can be sourced from the community.

Apache ZooKeeper and the ClientAdmin API.

If you are used to developing applications on-premises, you may be tempted to access Apache Zookeeper directly. Confluent Cloud gives you access to Kafka, but does not provide direct access to ZooKeeper. If your application requires its Zookeeper for its own functionality, you'll have to manage it on your own, but it will not be the same ZooKeeper that Confluent Cloud uses and therefore you won't be able to see brokers, topics and other Kafka ZNodes.

Confluent Cloud runs the latest version of Apache Kafka, which provides an AdminClient API. Confluent Cloud CLI uses the API to do topic management, and you can use the API from your applications as well.

There are few client API requests that are disallowed on Confluent Cloud: Adding partitions to existing topics and managing (create, delete, describe) access control lists. We are planning on enabling them at a future date.

Monitor the Applications

You no longer need to monitor Apache Kafka brokers, since we manage it for you, but you are still responsible for the availability and performance of your event-driven applications. Naturally, you are already monitoring all your applications, but applications that use Apache Kafka as consumers or producers of events should be monitored in three specific ways:

Client Metrics

Supported Kafka clients (see below) and the Kafka Streams API include a pluggable interface for reporting performance and availability metrics. The Java clients include a JMX reporter, so you can monitor them using any monitoring tool that supports JMX. Make sure you review the available metrics (https://kafka.apache.org/documentation/#selector_monitoring) and set up monitoring when you are deploying your applications to the cloud. Many cloud monitoring products provide both a web-based GUI and REST API to configure the metric collections and alerts. While getting started with GUI is easy, cloud environments are built for frequent change and constantly configuring metrics in the GUI is error-prone and time-consuming. We recommend automating the metrics configuration as part of the application deployment process.

For clients using Confluent Cloud, make sure you monitor the “throttling” metrics: ‘produce-throttle-time-avg’ and ‘fetch-throttle-time-avg’. These metrics will increase when the Confluent Cloud cluster is applying back-pressure to stay within the throughput limits of your price tier. Contact us if these numbers are consistently high.

End-to-end Stream Monitoring

Confluent Control Center is part of Confluent Enterprise, and a license to use it is included with Confluent Cloud Enterprise subscription. Currently, you'll self-host Control Center and run it yourself. Use Confluent Control Center to monitor end-to-end data flow of your applications — from producer to consumer. Control Center can indicate under-consumption ("lost" events), over-consumption (duplicates) and high latency between producer and consumer. In the Confluent Cloud environment, Control Center will not show broker health metrics, but if you configure interceptors with your Confluent Cloud bootstrap brokers, Control Center will monitor, report and alert on the end-to-end stream monitoring data. Note that Control Center creates multiple topics and both produces and consumes events from the cluster, so this will contribute to your cluster size calculations.

In the future, the web interface for Confluent Cloud will include a dashboard with throughput and latency information for your cluster, as well as additional management capabilities currently provided by the Confluent Cloud CLI.

Consumer Offsets

For applications that consume events from Confluent Cloud, the most important metric to monitor is consumer lag. That is - the difference, in number of messages, between the last message produced to a specific partition and the last message an application consumed from that partition. In other words — how much does the application lag behind the latest events it should consume?

There are two ways to monitor consumer lag in Confluent Cloud:

- Monitor `records-lag-max` metric from the Java Consumer. The benefit of this approach is that it is fairly straight forward to collect this information. The drawback is that not every consumer in every language publishes that metric, and that since it is published by the consumer, the metric is not available if the consumer isn't available.
- Confluent Control Center provides consumer lag monitoring by querying the brokers directly. This is similar to the second approach above, but includes a graphic visualization of the lag and its trends and built-in alerts. Right now Control Center is supported and included in price of Cloud Enterprise, but it is self-managed. In the future, we will run and manage it for you, and give you access to view metrics and define alerts.

Security and Compliance

Encryption, Authentication, and Authorization

We require TLS/SSL encryption and SASL_PLAIN authentication for all connections to Confluent Cloud. We make security both default and easy so this mostly means just copy-pasting few lines of configuration from Confluent Cloud UI to the configuration of a client.

When you create a Confluent Cloud cluster, a pair of authentication keys is created for the cluster and you'll be asked to download and store them. Make sure you store them securely, because they are essentially the password to your cluster, and try not to lose them because we can't re-create lost keys.

You can create and revoke additional keys through the web interface. Consider creating one API key per application, that way you can revoke access to specific applications without affecting others.

Confluent Cloud does not yet support ACLs. While we can issue multiple keys to the same cluster, at this point all keys have admin access to all the topics in the cluster. In the future you will be able to limit access for specific keys.

All data is stored on secured infrastructure that is allocated by the public cloud provider, with access control restricted to Confluent engineers inside a Confluent controlled VPC. Confluent Cloud uses encrypted storage by default, so you have encryption of data at rest — this means that even if a malicious user somehow gains access to the physical disks with the data, they'll be unable to decipher it.

Enterprise Security

Confluent Cloud Enterprise is a private cluster and all Amazon resources (EC2 instances, EBS volumes, etc) are allocated specifically for this single cluster. There is no shared data from other customers in an Enterprise cluster. Confluent Cloud Enterprise provides VPC Peering as optional security feature.

Regulatory Compliance

Confluent maintains a secure, private environment within AWS with a number of internal security controls. Confluent Cloud and Confluent Platform are SOC-2 compliant

Your industry may be governed by other regulations such as HIPAA, GDPR or PCI-DSS. Confluent Cloud is in the process of achieving compliance with these regulations, but, as of the time of publication of this document, it had not achieved certification. Check the documentation for an up-to-date list of the compliance status of Confluent Cloud (link). As a Confluent Cloud user, it is your responsibility to encrypt your sensitive data *before* it is sent to Confluent and decrypt the data by downstream services and applications, in order to achieve end-to-end encryption. If you own the encryption keys, Confluent employees cannot see your data.

To remain in compliance, do not send Confluent Cloud un-encrypted PHI, EU PII or PCI data.

More than one Cluster — Multi-region and Multi-Cloud deployments

There are few good reasons why you'll want Kafka clusters on both on-prem and cloud environments, multiple cloud environments or multiple regions of the same cloud provider:

1. Cloud migrations normally last over a year, sometimes significantly longer. During the migration you'll want to run Kafka on-prem and in the cloud and stream events between these two environments as a way to keep them in-sync.
2. Some companies have long-term strategy to run on-prem and in cloud, sometimes for regulatory reasons. In this case you'll also run Kafka on-prem and in the cloud and stream events between these two environments as a way to keep them in-sync.
3. We discussed resilience earlier in this paper. If your disaster plans require protection from an event where an entire region of a cloud provider fails, you'll want to run a cluster in each region. If you are also concerned about an entire cloud provider experiencing failures, you'll want to run a cluster in each cloud provider. Confluent Cloud makes this easy by letting you easily create and manage clusters in any cloud provider and any region from the same interface and with the same tools.

In all those cases, just managing the clusters isn't enough, you'll need to replicate some or all data between the clusters. Our documentation will guide you on how to replicate data between Confluent Cloud clusters. Both Confluent Replicator and Apache Kafka Mirror Maker can replicate data between Confluent Cloud clusters (and from on-premise clusters as well). We recommend using Confluent Replicator for its usability and support for topic renaming and event filtering.

Summary

Confluent Cloud is managed Kafka, which generally means that you have fewer decisions to make and less to worry about. Everything you know about Kafka still applies, but you can skip the maintenance and focus on the strategic parts of your architecture. In this document, we highlighted areas in which building on the cloud in general and Confluent Cloud specifically is different and provide specific recommendations toward building data architectures in this environment. You've learned that capacity planning and monitoring are still important, but in very different ways and you've learned what Cloud Native means for event-driven streams architectures.