Design Considerations for Cloud-Native Data Systems

JUL 26, 2021 READ TIME: 8 MIN

Twenty years ago, the data warehouses of choice were Oracle and Teradata. Since then, growth and innovation has shifted to the cloud, and a new generation of data systems have emerged. While this has been true for a few years now, the recent success of technologies like Snowflake reminds us how much growth and value cloud-native data systems bring. Contrary to what the old joke about cloud computing says, BigQuery isn't just Oracle running on someone else's computer. The user experience is completely different, designing and building the service is different, and those differences demonstrate that as companies shift workloads to the cloud they also shift their expectations, and will vote with their wallets for solutions that align better with these new expectations.

We summarize this shift in expectations by using the term *cloud native*. Being cloud native can mean different things in different contexts. The Twelve-Factor App is a methodology commonly used to explain what cloud native means (although the paper uses the term *software as a service*), but it refers mostly to development and deployment practices of those who write the applications, not what the users of applications expect. When we look at cloud-native data systems, and try to generalize from services as diverse as Snowflake, BigQuery, Aurora, DynamoDB, and S3, we see several common capabilities that users now expect to see in cloud-native data platforms:

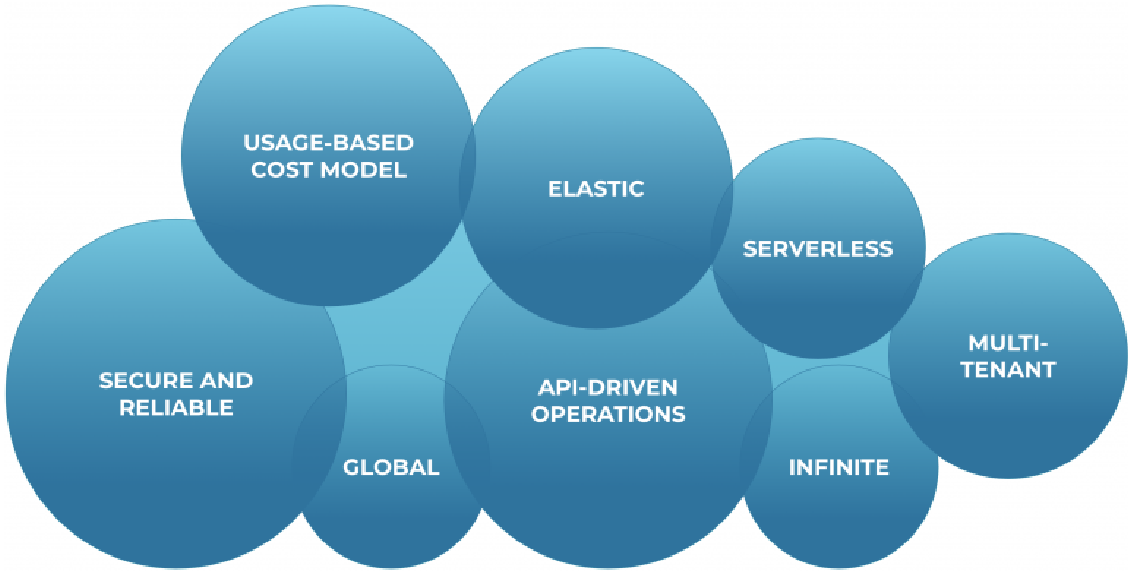
- **Elasticity:** Services are expected to scale up and down with a single click or API call at maximum, and preferably scale automatically based on policies.
- **Infinite scale:** While nothing is truly infinite, cloud-native services appear infinite to their users. Thanks to elastic scaling and behind-the-scenes capacity planning, only the most extreme use cases will ever notice that the service has scalability limits.

WRITTEN BY

Gwen Shapira

Engineering Manager, Confluent

- **Resiliency:** Cloud-native data services are architected for high availability in the face of most failures. Most boast SLAs of 99.95% (4.5 hours downtime per year) and higher, but in reality customers expect much higher availability.
- **Multi-tenancy:** Most cloud-native services take advantage of the economy of scale and manageability that multi-tenancy offers. The best user experience is achieved in services like S3 where the service is delivered as a high-level abstraction (requests and queries rather than CPUs) and the tenants are well isolated, so that users have no reason to be aware that the same physical system provides services to additional tenants. In other cases, like AWS Aurora, users do purchase dedicated low-level compute resources like CPUs and memory—but the provider is still able to share underlying infrastructure like network and storage.
- **Pay per use:** Many cloud-native data systems have a pay-per-use model where users pay for resources they use rather than resources provisioned. Resources used can be high-level resources, such as put/get requests, or lower level such as CPU and memory used. The key difference is that unlike on-prem data systems, users don't pay simply for licensing cores that they may or may not use.
- **Cost-effectiveness:** Comparing costs between fully managed cloud services and on-prem or self-managed services is not trivial, but the elasticity of cloud-native data systems combined with a pay-per-use model means that it is possible to run "right-sized" systems without having to worry about over-provisioning, capacity planning, and paying for unused resources. Multi-tenant models allow service providers to offer the service for much lower cost than any self-managed service could achieve. For many use cases, adopting the service would not be feasible without the low usage for a low-cost option.
- **Global:** Cloud-native data systems often include global replication as a built-in part of the product, something that users can enable with a checkbox or one API call, rather than a problem that they need to solve on their own.



What is a cloud-native data system?

Apache Kafka®, from its earliest days, had a goal to run at company scale. To us, this meant that instead of siloing workloads into separate clusters, we aimed to make Kafka scalable enough and multi-tenant enough that one could safely run the entire business on a single Kafka cluster. Our early investment in elasticity, scalability, and multi-tenancy is now bearing fruit as we take Kafka to its next stage in evolution—a cloud-native data system, operated as a fully managed service.

This blog series presents some of the more recent work done to enhance Kafka's cloud-native capabilities in the context of Confluent Cloud, with a focus on some of the key capabilities listed above: elasticity, multi-tenancy, scalability, resiliency, cost-effectiveness, and global replication.

While writing these blogs, the audience we've had in mind was other developers working on modern data systems and are looking at making these systems cloud-native. But you'll notice as you read these posts that the learnings are far more general. If you're building microservices based on an event log, running a system that has to stretch over globally, have customers that demand strict, industry-leading performance and availability SLAs, or you cannot ever, ever lose data, many of the techniques and architectural decisions we applied across the thousands of clusters in Confluent Cloud are likely to prove useful.

Our journey is not unique since every cloud-native data system and every company offering managed data stores as a service will need to provide similar capabilities. We believe that the techniques we've developed while building Confluent Cloud can be generalized to all cloud-native data systems.

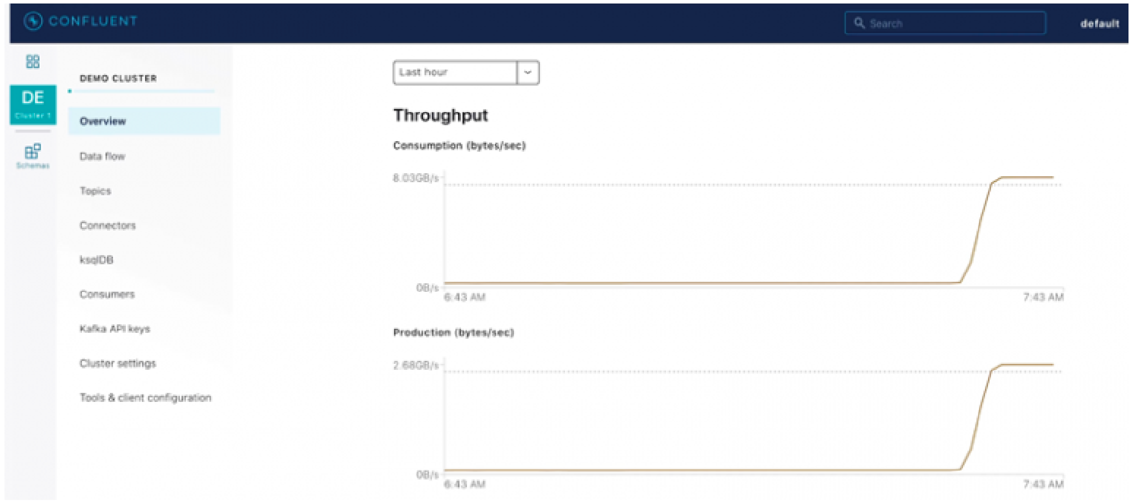
Elasticity

While adding new brokers to Kafka is straightforward, Confluent Cloud faced a new challenge: The ability to allow a customer to expand any cluster with a click, across a fleet with thousands of clusters. Just adding brokers is not enough—we needed to shift proportional load to the new brokers, within a few minutes and with no impact on existing client traffic.

Making Apache Kafka Serverless: Lessons From Confluent Cloud, introduces the Confluent Cloud control plane, discusses a few key control plane services, highlights the way Kafka is used as the underlying event log behind the control plane, and details the integration between the control plane and the Kafka services provided to our customers that allows Kafka to shift load between brokers as required.

The use of event-driven microservices and choreography patterns to build our control plane allowed different engineering teams at Confluent to own their logic in the control plane, while creating unified end-to-end workflows that deliver elasticity to our users. Creating this system on top of an event log makes it resilient to failures, recoverable, and auditable. We use Kafka's best practices to run thousands of Kafka clusters.

The blog also briefly explores two key capabilities in Kafka related to managing broker load that make this level of elasticity possible. The first feature is Self Balancing Clusters (SBC), a proprietary addition to Confluent Server that allows Confluent Cloud clusters to automatically detect load and shift it to less utilized brokers. The second is tiered storage, which makes it possible to shift load while moving a minimal amount of data.



Performance and scalability

Next we'll discuss performance, scalability, and the relationship between these aspects of a system. Confluent Cloud's approach to performance is based on the idea that by understanding user workloads well and optimizing Kafka accordingly, we can deliver better performance on cost-effective infrastructure.

Speed, Scale, and Storage: Our Journey from Apache Kafka to Performance in Confluent Cloud, shares three useful principles for systems engineers who are trying to design a high-performance cloud-native data system: Know your users and optimize for their workloads, infrastructure matters, and you can't improve what you don't see.

Beyond the specific principles, the overarching theme is performance optimization. In order to serve a wide range of customer workloads, you need to build a system that is scalable in many dimensions. You need to optimize and align constraints across an entire stack—from hardware, through the operating system, to your application. And, in managed cloud-native systems, there has to be a tight feedback loop with production—production observability, ability to detect, analyze, and learn from performance or scalability-related incidents, and ability to rapidly roll out improvements.

Multi-tenancy

Multi-tenant capabilities differentiate truly cloud-native systems from older generation software that is managed on someone else's computer. Systems that are built from the ground up to be cloud-native software-as-a-service offerings are typically built as multi-tenant systems, which introduces economies of scale to the service.

Multi-tenant systems are deployed both on-prem, for centralized management and cost-efficiency, and in the cloud. But the expectations are different because on-prem systems are rarely expected to be transparently elastic. It is common for teams that run multi-tenant clusters on-prem to spend significant time understanding the workload of each tenant and then to provision clusters with spare capacity. In cloud-native systems, operators of the system do not have this luxury—the system has to automatically and transparently respond to new workloads, both by elastic scaling and by isolating tenants from noisy neighbors.

From On-Prem to Cloud-Native: Multi-Tenancy in Confluent Cloud discusses Confluent Cloud's approach to solving these challenges, introduces the three pillars of multi-tenancy (access isolation, namespace isolation, and performance isolation), and discusses how, since Kafka was originally designed to be a company-scale system, Apache Kafka already contains the basic building blocks to run as a multi-tenant system.

The blog post addresses how these basic building blocks are used to create the multi-tenant products in Confluent Cloud, focusing especially on performance isolation, elasticity, and the importance of adjusting quotas on the fly based on changes in available resources and workloads.

Storage Durability Audit

The series concludes with a blog post that introduces a high-level capability—the Storage Durability Audit. On-prem deployments are often limited by the overhead of introducing additional services. Cloud-native services, built to run as resilient managed services in an elastic cloud environment, take advantage of the robust infrastructure already built to deploy and deliver reliable services in order to deliver added-value higher level capabilities that on-prem would meet resistance in the form of a long process to deploy additional services.

Confluent's Storage Durability Audit is an example of such higher level capability. Auditing data integrity is important in any data storage system and complements the investment in backup and restore—without integrity audits, restores will only happen on catastrophic failures or when the consumers of the data discover discrepancies. Yet despite the importance of such systems, they are rarely implemented in on-prem deployments due to the additional complexity they introduce.

Protecting Data Integrity in Confluent Cloud: Over 8 Trillion Messages Per Day explains the challenges inherent in guaranteeing data integrity with Kafka's replication and leaderelection model, Confluent's approach to detecting data anomalies in real time and at scale based on a data fragility model, and the internal details of the audit service.

On to the cloud-native series

Cloud-native data systems introduce their own requirements, and often need to be designed as such from the ground up. This blog series explores some of the key requirements for cloud-native data systems, discusses how many of them align with the concept of "company-wide scale" that served as a North Star for Apache Kafka development from its earliest days, and show how we extend the existing capabilities in Apache Kafka into the cloud-native managed service that we provide in Confluent Cloud.

We believe that we've only just begun to explore what is possible in the cloud-native data space—both from customer expectations and from the technical perspective. We hope that the lessons learned and the solutions we highlight in the series will be valuable to other engineers who are developing cloud-native data systems, and those who are adopting existing technologies to the cloud.

Other posts in this series

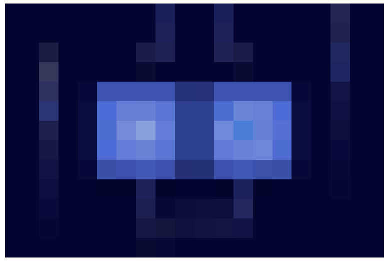
- Part 1: Making Apache Kafka Serverless: Lessons From Confluent Cloud
- Part 2: Speed, Scale, Storage: Our Journey from Apache Kafka to Performance in Confluent Cloud
- Part 3: From On-Prem to Cloud-Native: Multi-Tenancy in Confluent Cloud
- Part 4: Protecting Data Integrity in Confluent Cloud: Over 8 Trillion Messages Per Day

Gwen Shapira is a Software Enginner at Confluent. She has 15 years of experience working with code and customers to build scalable data architectures, integrating relational and big data technologies. She currently specialises in building real-time reliable data processing pipelines using Apache Kafka. Gwen is an Oracle Ace Director, an author of books including "Kafka, the Definitive Guide", and a frequent presenter at data related conferences. Gwen is also a committer on the Apache Kafka and Apache Sqoop projects.

Did you like this blog post? Share it now

Subscribe to the Confluent blog

Subscribe

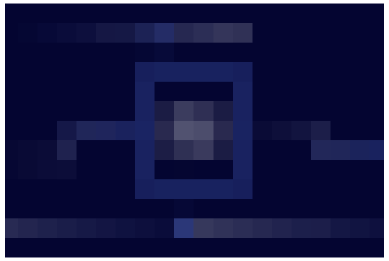


Build, Manage, and Monitor Data Streaming Applications, All Within Your Favorite IDE

SEP 17, 2024

We're excited to announce Early Access for Confluent for VS Code. This Visual Studio integration streamlines workflows, accelerates...

OLIVIA GREENE
MATTHEW SEAL
ADI POLAK



Unlock Real-Time Value from DynamoDB Data with Confluent's CDC Source Connector

AUG 27, 2024

62% of Confluent Cloud clusters run on AWS. Meanwhile, hundreds of thousands of customers are using DynamoDB. This blog...

AHMED SAEF ZAMZAM
BRAEDEN QUIRANTE
JAI VIGNESH R

Confluent Platform
Connectors
Flink
Stream Governance
Confluent Hub
Subscription
Professional Services
Training
Customers

Support
Sign Up
Log In
Cloud FAQ

Insurance
Retail and eCommerce
Automotive
Government
Gaming
Communication Service Providers
Technology
Manufacturing
Fraud Detection
Customer 360
Messaging Modernization
Streaming Data Pipelines
Event-driven Microservices
Mainframe Integration
SIEM Optimization
Hybrid and Multicloud
Internet of Things
Data Warehouse
Database

What is Kafka?
Resources
Events
Webinars
Meetups
Current: Data Streaming Event
Tutorials
Docs
Blog

Startups
Company
Careers
Partners
News
Contact
Trust and Security

Terms & Conditions | Privacy Policy | Do Not Sell My Information | Modern Slavery Policy | Cookie Settings