aws marketplace

EBOOK

# Architecting event-driven API-first platforms to build Everything-as-a-Service

How to bring platform thinking into your internal business domain to help developers compose applications faster

aws marketplace

ARCHITECTING EVENT-DRIVEN API-FIRST PLATFORMS
TO BUILD EVERYTHING-AS-A-SERVICE

# Table of contents

aws marketplace

# Getting up to speed with cloud-native

Implementing microservices and cloud-native architectures has been a goal for many organizations who hope to increase speed and agility. But achieving this goal also introduces additional complexity, as cloud-native applications are decoupled and distributed.

In this ebook, we'll offer guidance for architecting an effective Everything-as-a-Service approach to building cloud-native applications and infrastructure. You'll learn the value of adopting a strategy that considers database, application, and API management from the onset, and explore implementing asynchronous communications using messaging for event-driven microservices.

## Modernization is a business priority today

To delight customers and win new business, organizations need to build reliable, scalable, and secure applications. That means adopting new technologies, practices, and consuming services as APIs.

As an application development professional, your goal is to deliver business value fast. Modern applications help achieve this goal by separating and decoupling the monolith into smaller functional services—or **microservices**—that focus on one thing and do it well. Each microservice often has its own data store and can be deployed and scaled independently. They represent the real world, where service boundaries equal business boundaries.

This has forced organizations to evolve by giving engineering teams the autonomy to architect, develop, deploy, and maintain each microservice. With this approach, you end up with the ability to make decisions very quickly because your decisions only impact individual services. After all, innovation requires change. You can learn faster by making lots of little changes to drive incremental innovation, rather than waiting to take one giant leap.

## Increasing the speed of innovation

Modern applications were born out of a necessity to deliver smaller features faster to customers. While this directly addresses only the application architecture aspect, it requires other teams to build and execute in a similar manner to be successful. In order to continuously deliver features, there is a need for all cross-functional teams to operate as a single team—a strategy referred to as One Team.

Each type of change will need its own fully automated delivery pipeline—for example, application, infrastructure, configurations, feature flags and OS patching will either need their own pipeline or need to be part of the Continuous Delivery (CD) pipeline. And capabilities like test automation and security testing need to be integrated into the pipeline so there is a high degree of confidence for changes that flow through the pipeline are ready to be deployed into production.

### Key aspects of modern applications:

- Use independently scalable microservices such as serverless and containers
- Connect through APIs
- Deliver updates continuously
- Adapt quickly to change
- Scale globally
- Are fault tolerant
- Carefully manage state and persistence
- Have security built in

## What is cloud-native? Why does it matter?

Cloud-native is an evolving term. The vast amount of software that's being built today needs a place to run and all the components and processes required to build an application need to fit together and work cohesively as a system.

The Cloud Native Computing Foundation (CNCF) definition states:

***Cloud-native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds.***

This definition has to broadly apply to everyone, but not everyone has the same capabilities. This is known as the lowest common denominator problem. It is where you try and appeal to a broader group and their capabilities, but in doing so you also need to limit the capabilities that can be leveraged.

Amazon Web Services (AWS) goes many steps further by providing a broad set of capabilities that belong to a family called serverless. Serverless technologies are more than just AWS Lambda—these services remove the heavy lifting associated with running, managing, and maintaining servers. This lets you focus on core business logic and quickly adding value.

aws marketplace

# Everything-as-a-Service (EaaS) for cloud-native development

As we cover the different capabilities your organization needs to acquire to go fully cloud-native, it's useful to view each one as a step in a journey.

The map below is a model for how organizations typically evolve their cloud-native understanding. As your organization or team moves from stage to stage, you are gaining capabilities that make releasing new features and functionality faster, better, and cheaper. In the following sections, we'll be focusing on the capability of Everything-as-a-Service.

# Applying a more practical approach to new products and services

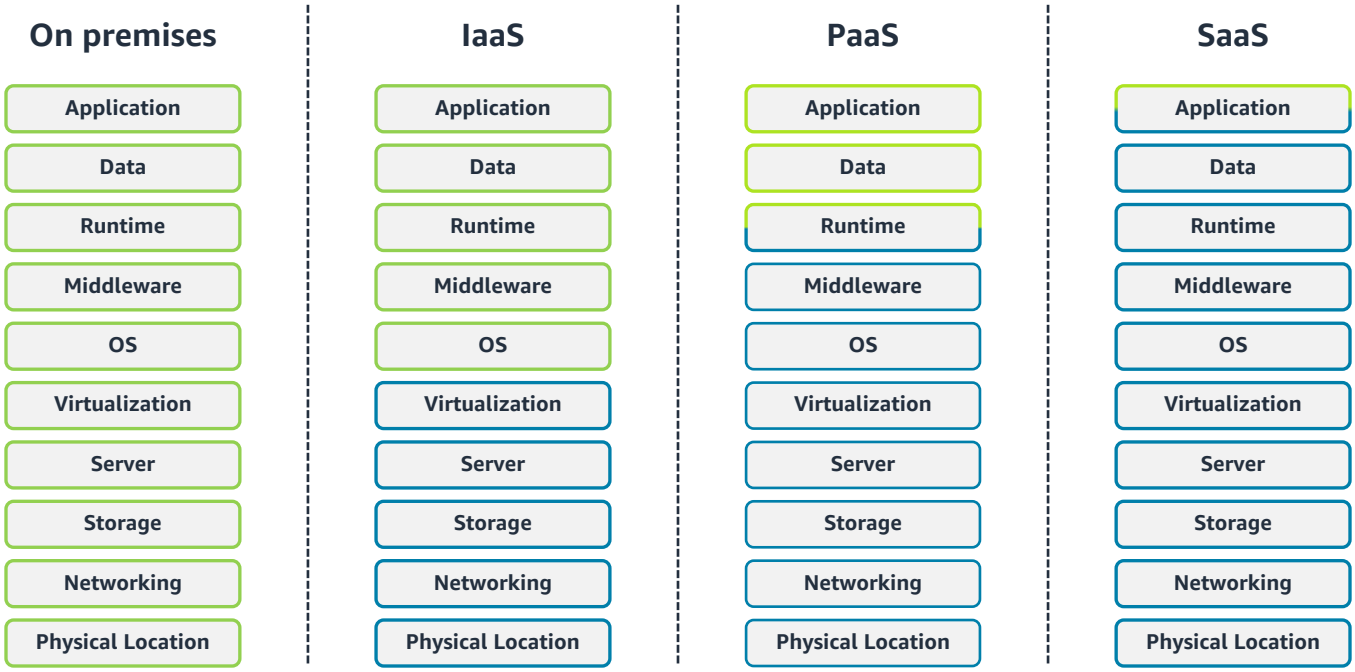
Developers have long known that it's more advantageous to assemble or compose applications as opposed to building everything from scratch. For example, in Java to do some mathematical calculations developers simply import the Math class and use the required methods like sqrt, min, max, etc. It would be a lot of work to create the Math class from scratch! The time required to build using lower-level constructs is much greater than taking an existing library or module and reusing it.

Besides time, when developers leverage existing constructs, most of the issues and kinks have been worked out of them and they are usually going to be much more efficient than something built from scratch.

While we build applications with this knowledge, we tend not to extend this principle into new products or services. What this means is that when we are building products or services, we start with lower-level infrastructure components and add applications to support the functionality we need. The problem with this approach is we end up duplicating efforts across the organization, which affects cost and time to market, and adds additional operational overhead to support the same functionality across multiple teams.

# Evolution of the “as-a-Service” model

Let’s talk about the evolution and mindset shift in relation to the problem we just considered in the previous section. The diagram below illustrates your progression as you move from an on-premises to a SaaS model. By making this transition, you are effectively reducing the undifferentiated heavy lifting of maintaining and managing infrastructure and certain pieces of software.




As you move from an on-premises model to an **Infrastructure as a Service (IaaS)** model, you gain the same  flexibility as if it were your infrastructure—minus the overhead of managing the premises, power, security, virtualization, storage, networking, and other aspects of app development.

**Platform as a Service (PaaS)** further builds on the IaaS layer by adding abstractions that can be leveraged by development teams. PaaS hides the complexities of dealing with operating systems, middleware, and runtimes—and allows developers and operators to focus on writing and supporting the application rather than the infrastructure.

Finally, **Software as a Service (SaaS)** refers to applications that are fully managed by the provider. This means that organizations can purchase software licenses and immediately get to work without any development, management of infrastructure, or installation. You can begin realizing value very quickly, often without the involvement of IT.

aws marketplace

# Benefits of Everything-as-a-Service

Cloud-native and DevOps turn the way developers approach and solve problems upside down. There is no longer a need to manage and maintain servers as was once necessary. Servers, patching, configuration, and infrastructure should all be treated as fungible components that don't change in different environments—new ones are simply deployed and a switch from the old to the new is orchestrated. This is all made possible with Cloud-Native, as everything developers need to consume is accessible through an API.

To level that up even more, what if we thought of higher-level services as APIs and solved problems by composing services? If we do that, we have Everything as a Service. A few examples of an organizational-level service that is consumed include logging service, data service, analytics service, reporting service, and so on. In an organization, if you want to build a new service you would simply consume other services in the organization and build on top of those.

**Another way to think of this is building a platform that is specific to the needs of the organization that can be consumed via self-service APIs.**

# Rethinking development with cloud-native

Developing software today is like assembling a car with pre-built parts. Recent research highlights that only 10 to 30 percent of code is custom and that 70 to 90 percent is open-source.* You can start by exploring whether any services from Independent Software Vendors (ISVs) will fit your needs.

A best practice would be to adhere to the following order:

1. **Explore the services offered by your cloud provider.**
   If you run into any limitations such as cost or capabilities, try the next step.
2. **Explore ISV services.**
   If you can't find what you are looking for or run into limitations, try Step 3.
3. **Explore internal services that are built by your own organization.**
   This is especially true for large enterprise organizations.

Only after you have exhausted all these options without solving your problem would you then attempt to build a customized solution.

For example, if you need to process online payments for your e-commerce website, you simply use a service acquired from an ISV such as Stripe. Or if you want to optimize delivery routes and reduce fuel consumption, you can use Amazon Location Service. This style of development lets you focus on the business logic so you can delight your customers by rapidly delivering new features. We will talk about few more examples and patterns in a later section.

* https://www.linuxfoundation.org/blog/blog/a-summary-of-census-ii-open-source-software-application-libraries-the-world-depends-on

# Platform as a Service

It’s important to clearly understand what a platform is in the context of a cloud-native world. In cloud-native, “platform” means anything that is not part of your business domain. So, in a cloud-native world a foundational concept is to consume everything that’s not part of your business domain as a service. This may already seem intuitive, but in reality organizations seem to be trying to rebuild pieces of the platform.

Let’s look at an example. Say, you are developing an application that needs to send emails. With this need in mind, it’s not uncommon for an organization to achieve this by standing up an email server or servers, build a team to run, manage, and maintain them, and call it a service. Then to use the service, they would open a ticket to the email team requesting access to send email. This scenario may sound like it fits into the realm of Everything-as-a-Service, but in reality, it’s an anti-pattern.

For starters, a service shouldn’t require tickets and manual approvals to start using it. Consumption should be self-service. The other point, which is a big one, is that the organization shouldn't be running, managing, and maintaining any platform services. In cloud-native, virtually all platform services are provided by AWS or software and services vendors.

What should be provided as a service are business domain-specific services. Between AWS and AWS Partners, you are covered when it comes to platform services. There are databases, messaging buses, queuing systems, API gateways, caching systems, storage, testing, security, and many other services available. These platform services are very robust and can scale, plus they are built on a time-proven foundation so you don’t have to build from scratch and learn as you go.

**The takeaway here is don’t rebuild what’s already available as a service but do expose your business domain as microservices that others within the organization can consume as a service.**

# The different types of “as a Service”

Fundamentally, the concept of Everything-as-a-Service is to expose services that other teams within the organization can consume. Examples include data services, event streaming services, testing services, and others. Think of this as abstraction and separation of concerns.

## DevOps as a Service

From an organizational level, AWS recommends that DevOps be provided as a service that other teams can consume. DevOps in this context would mean build services, pipelines, deployment systems, Observability, and the repository. You should not necessarily build these out, but they should be composed as a collection of services that can span multiple teams.

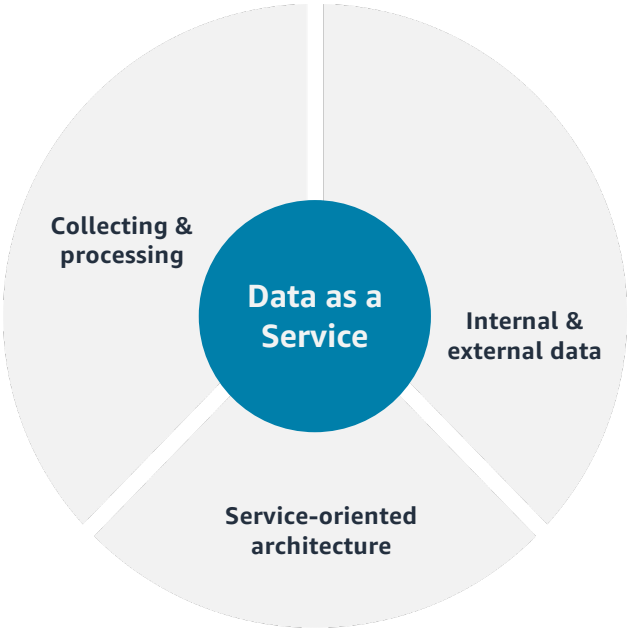
From the services just mentioned, build services, deployment systems, and Observability could be upstream services that the DevOps services are composed of. For example, you get all the above-mentioned capabilities with AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy, AWS CodePipeline for your CI/CD needs, and Amazon CloudWatch for Observability. All these services are fully managed by AWS and you don’t need to do the heavy lifting of maintaining and managing them.

## Data as a Service

Data silos tend to arise naturally in large companies because departments and teams often have their own goals, priorities, and IT budgets.

But organizations of any size can end up with siloed data if they are not intentional about addressing it. Data as a Service is a way to break down data silos by democratizing data. Data democracy does not mean every employee needs to have access to every bit of data that an organization has. Instead, it's an ongoing process of enabling everybody in an organization, irrespective of their technical know-how, to work with data comfortably, to feel confident talking about it, and as a result, make data-informed decisions and build customer experiences powered by data.

Having Data as a service enables this, as everyone will have a central place to go to consume the data they need regardless of whether they are a developer creating an app or a business user that wants to create a dashboard.




## Testing as a Service

Testing as a Service is a model in which testing activities associated with some of an organization's business activities are performed by a service rather than named employees or a QA department. In aligning with the theme of Everything-as-a-Service, this should be consumed via APIs that are instrumented throughout the DevOps pipeline.

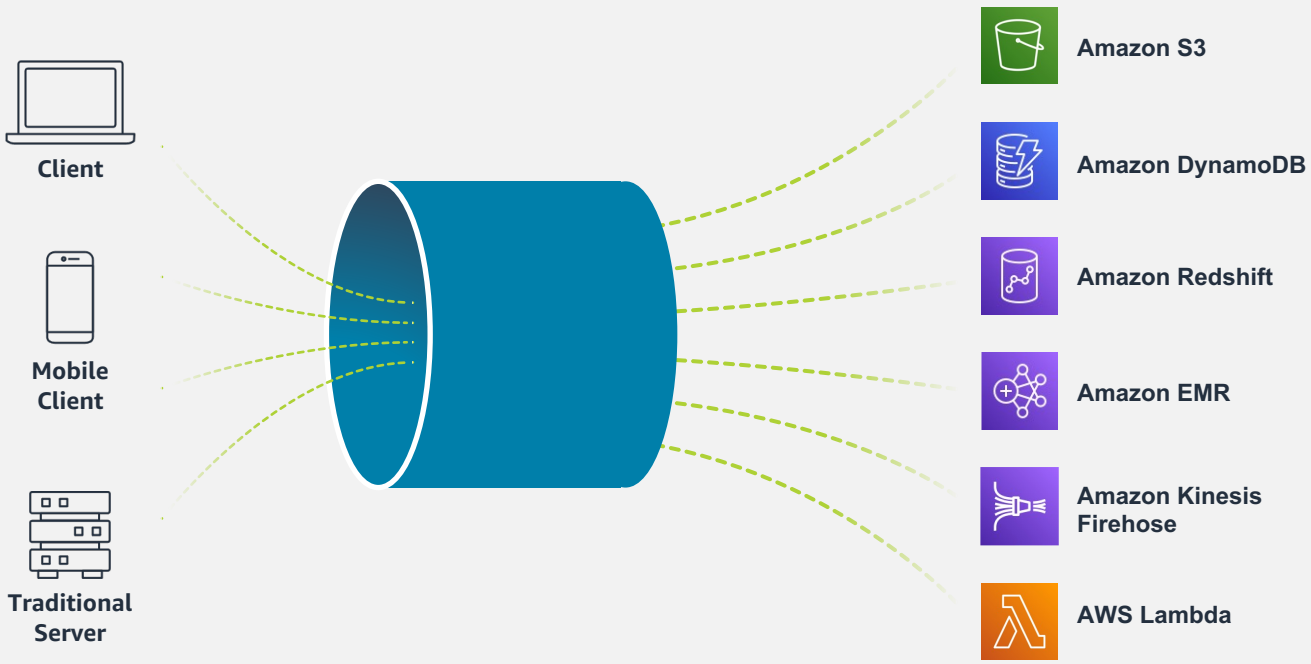
Another way to think about this is that, if your organization uses a specific test tool or suite, you want to think about consuming that tool as a service. The team exposing that service should be abstracting away all the heavy lifting associated with managing and maintaining that test suite. Ideally, all the tests should be easy to define within the repository where the source code is stored.

## Event Streaming as a Service

Event streaming is a sequence of continuous data points that originate from multiple sources like transactions, IoT data, business metrics and operational metrics. Each data point from a system is referred to as an event—a fundamental unit for stream processing.

Events have a repeating and evolving nature; hence the ongoing delivery of events is referred to as a stream. As illustrated in the graphic below, event stream processing (ESP) is when you combine streams of data together for real-time data delivery, which opens new capabilities like real-time data processing and analytics. This is where the true power of event streaming lies.

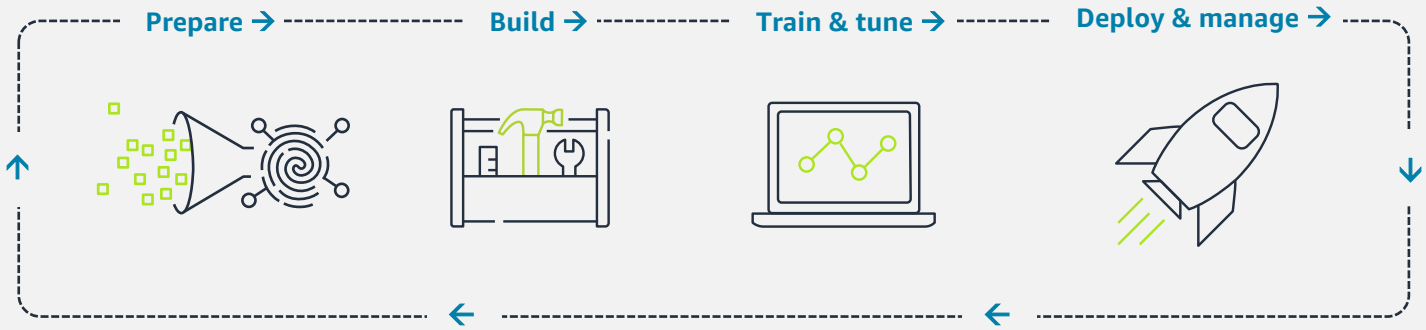
An anti-pattern often seen in organizations is that every team that needs a streaming service ends up building one. This is something you should avoid by leveraging a fully managed SaaS on AWS. For example, Confluent has reimagined Apache Kafka for the cloud to accelerate development cycles by up to 70 percent and lower management costs by up to 60 percent. We will get into more details about this later on in the ebook.



## Machine learning (ML) as a Service

At a high level, ML workflows involve preparing data: Building ML models, training and tuning them, and finally deploying and managing those models. All this can become complex very quickly, spiral out of control and cause significant expenditure.

Amazon SageMaker lets you build, train, and deploy ML models for any use case with fully managed infrastructure, tools, and workflows. Amazon SageMaker is built on Amazon's two decades of experience developing real-world ML applications, including product recommendations, personalization, intelligent shopping, robotics, and voice-assisted devices.




## A few key features and capabilities of Amazon SageMaker to highlight:

**Amazon SageMaker Studio** is a fully integrated development environment (IDE) for ML, providing complete access, control, and visibility into each step required to build, train, and deploy models. You can quickly upload data, create new notebooks, train and tune models, move back and forth between steps to adjust experiments, compare results, and deploy models to production all in one place, making you much more productive.

**Amazon SageMaker Pipelines** is a purpose-built Continuous Integration/Continuous Delivery (CI/CD) service for ML. Using Amazon SageMaker Pipelines, you can create ML workflows with an easy-to-use Python SDK, and then visualize and manage your workflow using Amazon SageMaker Studio. You can be more efficient and scale faster by storing and reusing the workflow steps you create in SageMaker Pipelines.

**Amazon SageMaker Canvas** generates accurate ML predictions with no code required. Amazon SageMaker Canvas expands access to ML by providing business analysts with a visual point-and-click interface that allows them to generate accurate predictions on their own, without requiring any ML experience or having to write a single line of code.

## Contact Center as a Service

Amazon Connect is a simple-to-use, cloud-based contact center service that makes it easy to deliver better customer service at a lower cost. Amazon Connect is based on the same contact center technology used by Amazon customer service associates around the world to power millions of customer conversations. Setting up a cloud-based contact center with Amazon Connect is as easy as a few clicks in the AWS Management Console, and agents can begin taking calls within minutes.

Amazon Connect's self-service graphical interface makes it easy for non-technical users to design contact flows, manage agents, and track performance metrics—no specialized skills required. The service also makes it possible to design contact flows that adapt the caller experience, changing based on information retrieved by Amazon Connect from AWS services, like Amazon Redshift, or third-party systems, like CRM or analytics solutions. You can also build natural language contact flows using Amazon Lex, an AI service that has the same automatic speech recognition (ASR) technology and natural language understanding (NLU) that powers Amazon Alexa.

## Security as a Service

At large enterprises, security traditionally means having a final gate before releasing code to production. But often—under pressure from business and competition—releases are pushed to production accepting a certain amount of risk.

Security should be integrated at every step throughout the application development and delivery lifecycle. And this is only possible if security is offered as a service and can be triggered at any point either by the CI/CD pipeline or manually on-demand by developers.

Security services need to provide immediate feedback to developers with IDE integration while they still have the context of writing that piece of code. Security teams should only focus on configuring policies and guardrails and not worry about managing and maintaining the infrastructure and tooling.

Security services like static code analysis, static application security testing (SAST), dynamic application security testing (DAST), software composition analysis, and others are continuously evolving in ways internal security teams simply can't keep up with. Instead, leverage security services offered by AWS and AWS Partners.

# Start identifying and building your service

So, we've learned some basic concepts that apply to external platform services. But it's important to not just think about what third parties are offering as services—introspect as an organization and identify what internal solutions can be offered as a service for your teams and operate them like third-party tools.

We'll cover best practices for making this a reality in the following sections.

## Teams own everything

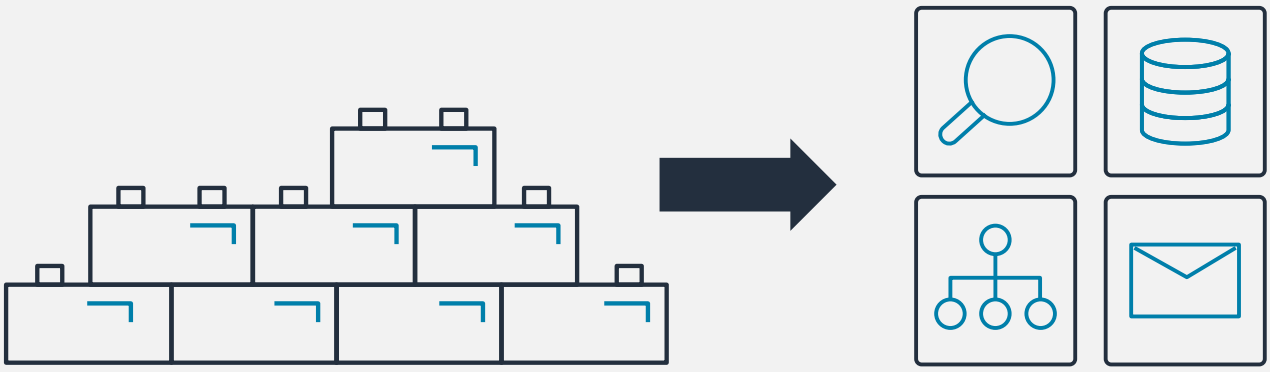
A lesson learned from Amazon's own engineering practice is the concept of a two-pizza team. A two-pizza team is small, autonomous team—small enough that it could be fed with only two pizzas. Each two-pizza team is focused on a specific product, service, or feature set, giving them more authority over a specific portion of the application. This turns developers into product owners who could quickly make decisions that affect their individual product.

**In the context of Everything-as-a-Service, each service is being built by a two-pizza team. In the context of Platform as a Service, each two-pizza team offers their service for other teams to consume.**

## Use building blocks

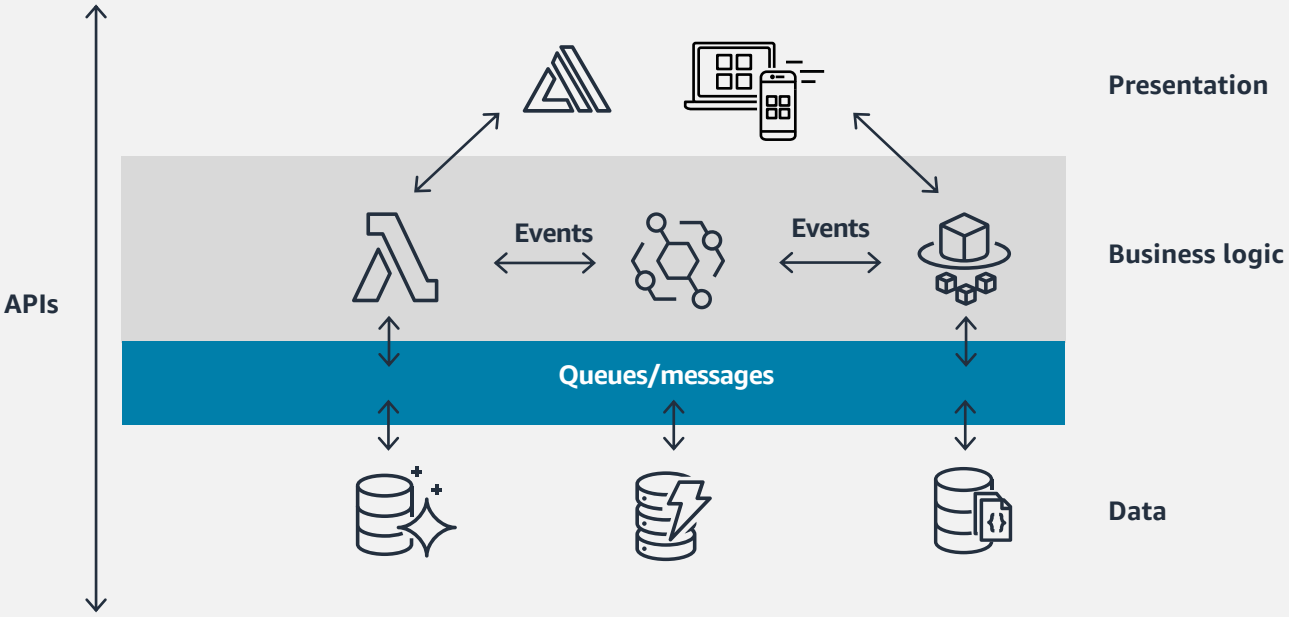
It's been mentioned already, but it can't be overstated that services need to be composed of other services. To gain speed and agility, the organization needs to focus its talent pool on building on top of what is already available to them. Don't try and build another service to replicate existing functionality. Doing so does not add business value to the organization.

**Building blocks that make up services**



## Decoupled architecture

The diagram below illustrates a modern architecture. Yes, it does indeed look a lot like a three-tier architecture. There are a lot of the same elements here, like data, logic, and presentation layers. But there are also a few key differences.




1. **What you are looking at here is a single microservice**—this service performs a specific business function and many AWS customers run dozens, hundreds, or even thousands of these kind of services to improve scalability and resilience.
2. **Integration and communication**—this architecture leverages a combination of events, messages, and queues that enable communication within the microservice, as well as APIs to communicate between services.
3. **A purpose-built data strategy**—rather than a single database, three are illustrated here, the point being that you can choose the data store that is the best fit for your specific need.
4. **Versioning**—what you can't see in this diagram is how you store state, an important consideration if you're running functions or containers.

# Prioritize an Everything-as-a-Service approach

Start a small project or new app as a cloud-native project, serverless as a compute option, and use existing building blocks likes API gateway. For EaaS to be successful, the products upon which the services are offered must be architected for EaaS.

## Build as a product, offer as a service

In a build-as-product mindset, the architect and designer must follow a strict product management discipline and design approach to build a configurable product that can satisfy a changing environment, incorporating the need for continuous improvement.

## Build where you must, acquire where you can.

By adopting a controlled EaaS practice, organizations can choose the best of both worlds. How? By providing services around acquired products that are not only cost-effective, efficient, and flexible but also built with design principles that offer resiliency and continuous improvement.

Today's enterprise revolves around a unique catalogue of services, so why should its IT services be any different? The IT department takes on the responsibility of providing a best-of-breed portfolio of services: Managed and optimized for service consumption, security, agility, reliability, responsiveness, service levels, and cost across multiple providers—some internal and others external to the organization. IT departments can move away from managing technology and costs to managing services and outcomes, allowing it to dictate a unique 'services' roadmap rather than being dictated to by technology limitations or tool vendors.

For EaaS to be successful, the products upon which the services are offered must be architected for EaaS. Furthermore, the organization's own digitized demands will require services to be built on products modelled on needs unique to the business and its first-level customer base, such as the need for availability, performance, and security.

## Benefits of using EaaS

- **Faster time to delivery**
- **Scalability**
- **Cost savings on infrastructure and licensing**
- **Better utilization of resources**
- **Customized for the needs of the organization**
- **Knowledge sharing between teams**

# Nielsen Marketing Cloud: High scale, low operations

One of the important benefits of AWS Lambda is that it responds to data in near-real time, and as customers create more data streams, a serverless approach to processing data is attractive.

Nielsen Marketing Cloud is doing this on an incredibly high scale. It's using AWS Lambda to process 250 billion events per day—all while maintaining quality, performance, and cost using a fully automated serverless pipeline. On a peak day, Nielsen receives 55 TB of data with 30 million Lambda invocations and their system manages it with no problem. Recently, they had up to 3,000 AWS Lambdas running concurrently.

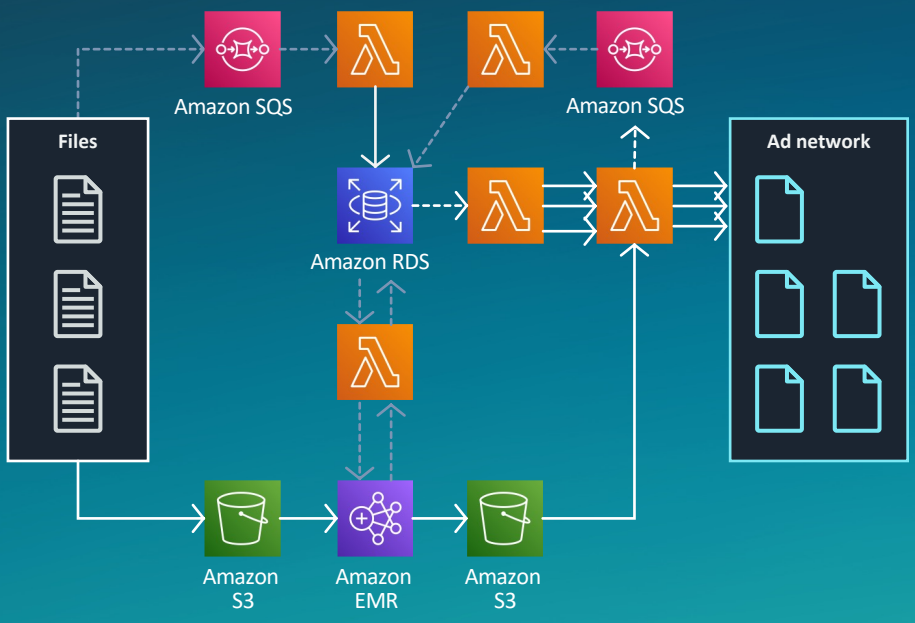
As you can see in the diagram below, they leveraged many "as a service" capabilities such as:

- **AWS Lambda**
- **Amazon SQS for queues**
- **Amazon Relational Database Service (Amazon RDS)—a fully managed relational database service**
- **Amazon Simple Storage Service (Amazon S3)**
- **Amazon EMR for its cloud big data platform (EMR now also has a serverless option)**

Leveraging these tools allows Nielsen to focus their efforts on building business logic rather than building and maintaining services.

nielsen

- Processes up to **55 TB of data per day**
- 250 billion events per day
- Up to 3,000 concurrent Lambda functions
- Consistent performance at any scale



# Key takeaways for Everything-as-a-Service

Here are few important concepts to take away from what has been covered so far:

### What Everything-as-a-Service is

and why you should adopt it as your app development approach

### How cloud-native development resembles a car assembly line

where pre-configured modules are bolted together to build and deliver cars quickly

### Common services like IaaS, PaaS, and SaaS;

and a few not-so-common services such as DevOps as a service, Data as a Service, ML as a Service, and more

### How to best organize your team

when building internal services to be offered as a service to the rest of the organization

### How to express all services as APIs

and decoupled architectures

### Benefits of the EaaS model

such as resource optimization, cost savings, and time to market

aws marketplace

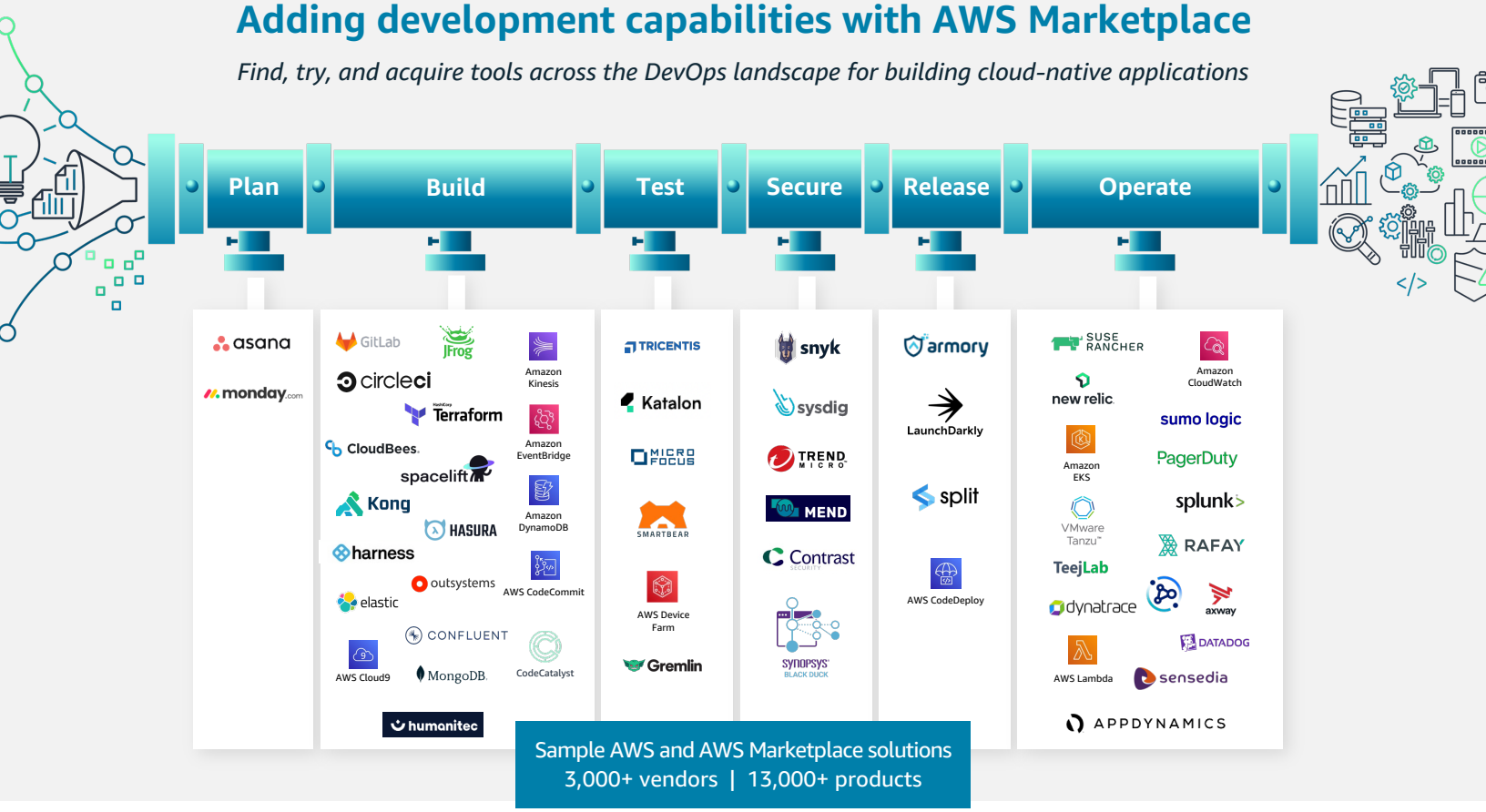
# Tools for building cloud-native

In this section, we'll look at some of the best-fit tools you could use to achieve the tenets discussed in the previous section. At AWS, we've long been believers in enabling builders to use the right tool for the job—and when you build with AWS, you're provided with choice. You can build using the native services AWS provides or use AWS Marketplace to acquire third-party software offered by AWS Partners to take away the heavy lifting and allow your development teams to focus on delivering value to customers.

Let's take a deeper look at three key components at this stage of your cloud-native journey: infrastructure designed to connect all the applications, systems, and data layers; an API layer to front your application and expose it as a service to other applications and systems; and a feature-rich platform to build on and from which to quickly launch the service to end customers

## Adding development capabilities with AWS Marketplace

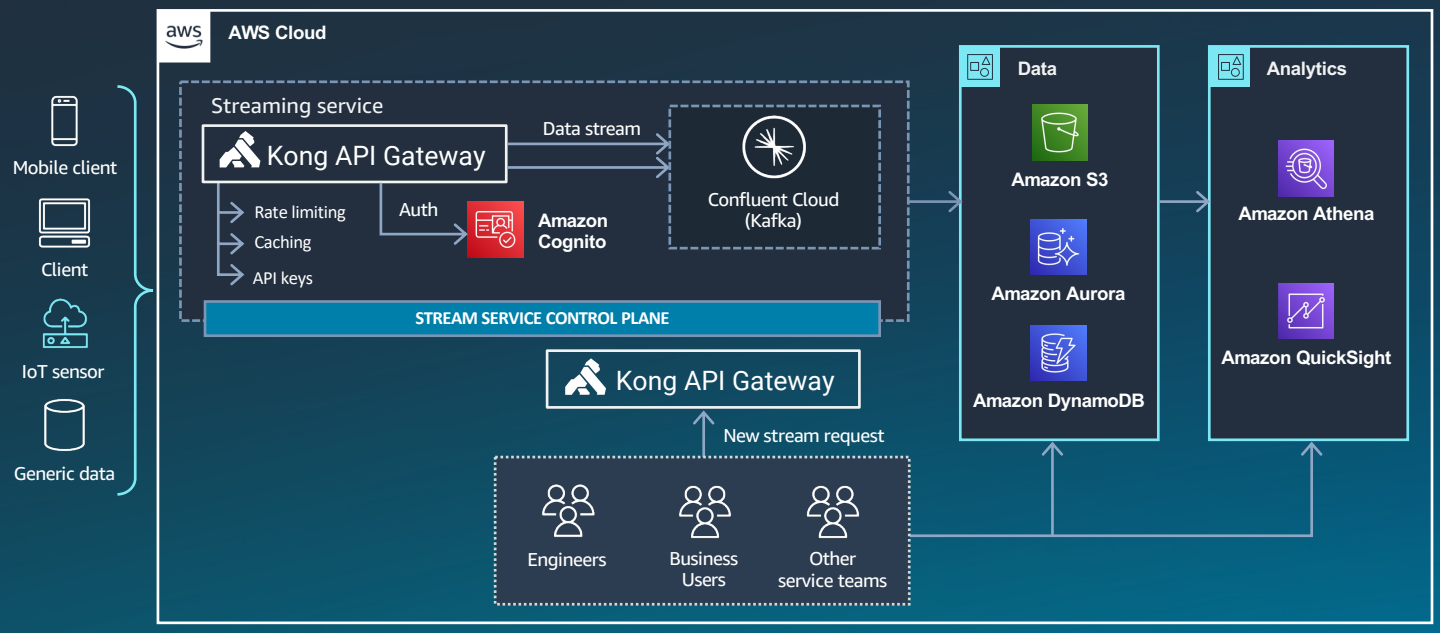
*Find, try, and acquire tools across the DevOps landscape for building cloud-native applications*




AWS Marketplace is a cloud marketplace that makes it easy to find, try, and acquire the tools you need to build cloud-native. More than 13,000 products from over 3,000 Independent Software Vendors are listed in AWS Marketplace—many of which you can try for free and, if you decide to use, will be billed through your AWS account.

## Example streaming service

The below diagram illustrates an example of a streaming service that aligns with all the best practices we've discussed so far.




This example shows how you can decouple streaming from other services, allowing teams to move faster as streaming data becomes a service they can consume versus trying to build.

The example is built to handle different client devices that connect into Kong's API gateway. Kong not only authenticates the connection but also applies rules and policies to wrap service-level agreements (SLAs) around the different service connections. Once Kong applies these rules, the connection is sent off to the Confluent Cloud, which is running a managed Kafka service that processes the payload data and connects that data to data stores such as Amazon S3, Amazon Aurora, and Amazon DynamoDB, which can be further consumed by analytic services such as Amazon Athena and Amazon QuickSight.

Staying true to the Everything-as-a-Service theme, the entire platform can be self-serviced through an internal facing Kong API that allows developers and other teams to request additional streams and publishing to other data destinations.

# Confluent and Kong

Before diving any deeper into the example architecture, let's look at the two components mentioned earlier.

CONFLUENT

Try it with AWS Marketplace ›

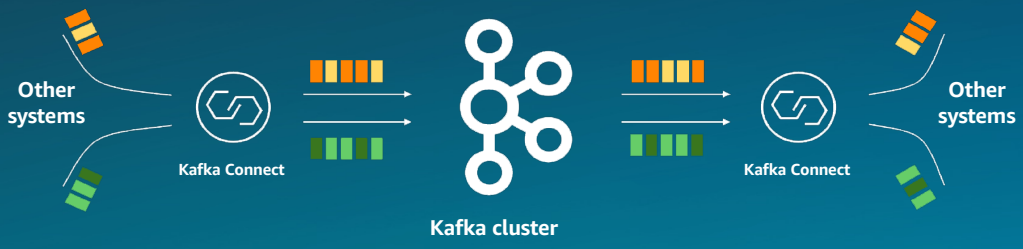
Watch a demo ›

Start a hands-on lab ›

**Confluent** improves on Apache Kafka's event-streaming platform by providing additional community and commercial features designed to enhance the streaming experience of both operators and developers in production, at massive scale. Confluent's cloud-native, complete, and fully managed service goes above and beyond Kafka, so your best people can focus on what they do best, **delivering value to your business**.

**Kafka's core features are:**

- Publish and subscribe to a stream of events
- Store your event streams
- Process and analyze your event streams




Kong

Try it with AWS Marketplace ›

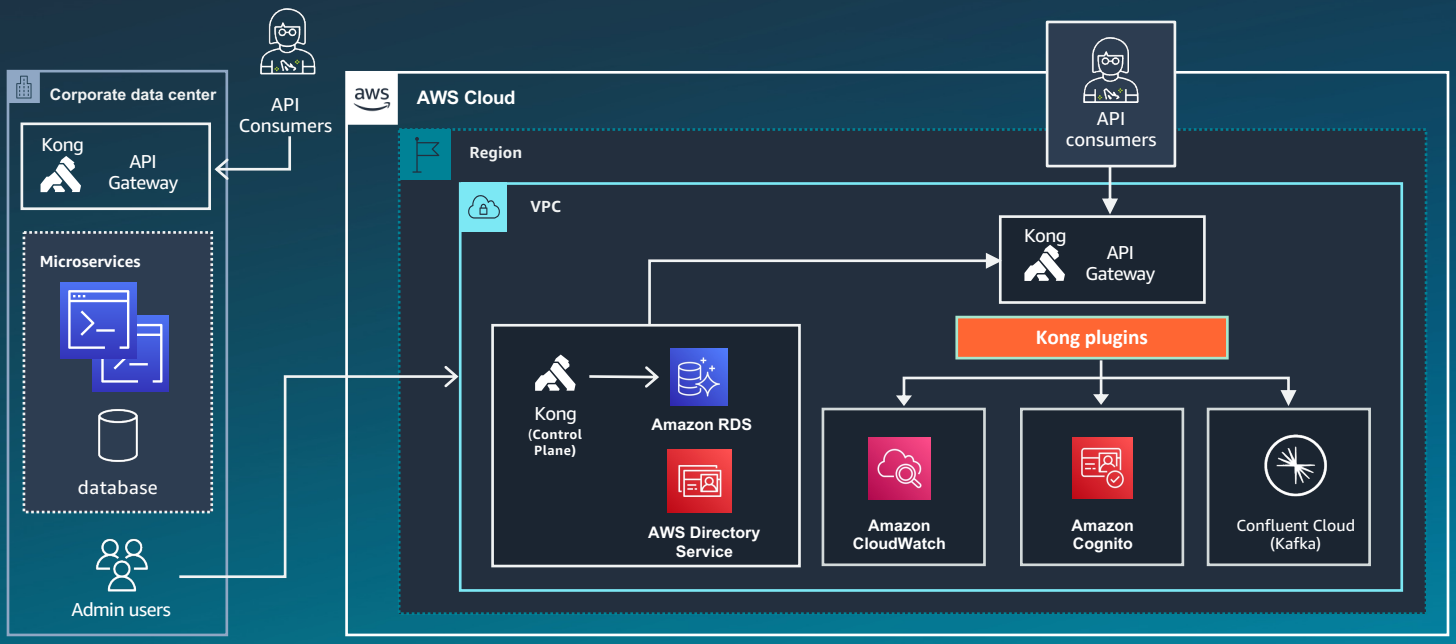
**Kong** offers open-source platforms and cloud services to manage, monitor and scale APIs and microservices. The main products offered are:

- **Kong Gateway:** An open-source API gateway
- **Kong Enterprise:** An API platform built on top of Kong Gateway
- **Kong Konnect:** A service connectivity platform
- **Kong Mesh:** An enterprise-grade service mesh built on top of open-source Kuma
- **Kong Insomnia:** An open-source API design and testing tool

**Kuma:** Kuma is a single and multi-zone connectivity tool that provides support for modern Kubernetes environments and virtual machine workloads in the same cluster

# Hybrid cloud API gateway architecture

Going back to our example streaming services:  The Kong solution has two components—Control Plane and Data Plane. Kong's Control Plane is deployed in AWS Cloud in its own subnet and uses Amazon RDS as a backend database to orchestrate the Data Planes. The Kong Data Plane is deployed on both the AWS Cloud and an on-premises environment. The Data plane acts as a frontend to the microservices and is exposed to the end users as shown above. This solution can deliver independently to cloud-native customers as well as to on-premises customers.

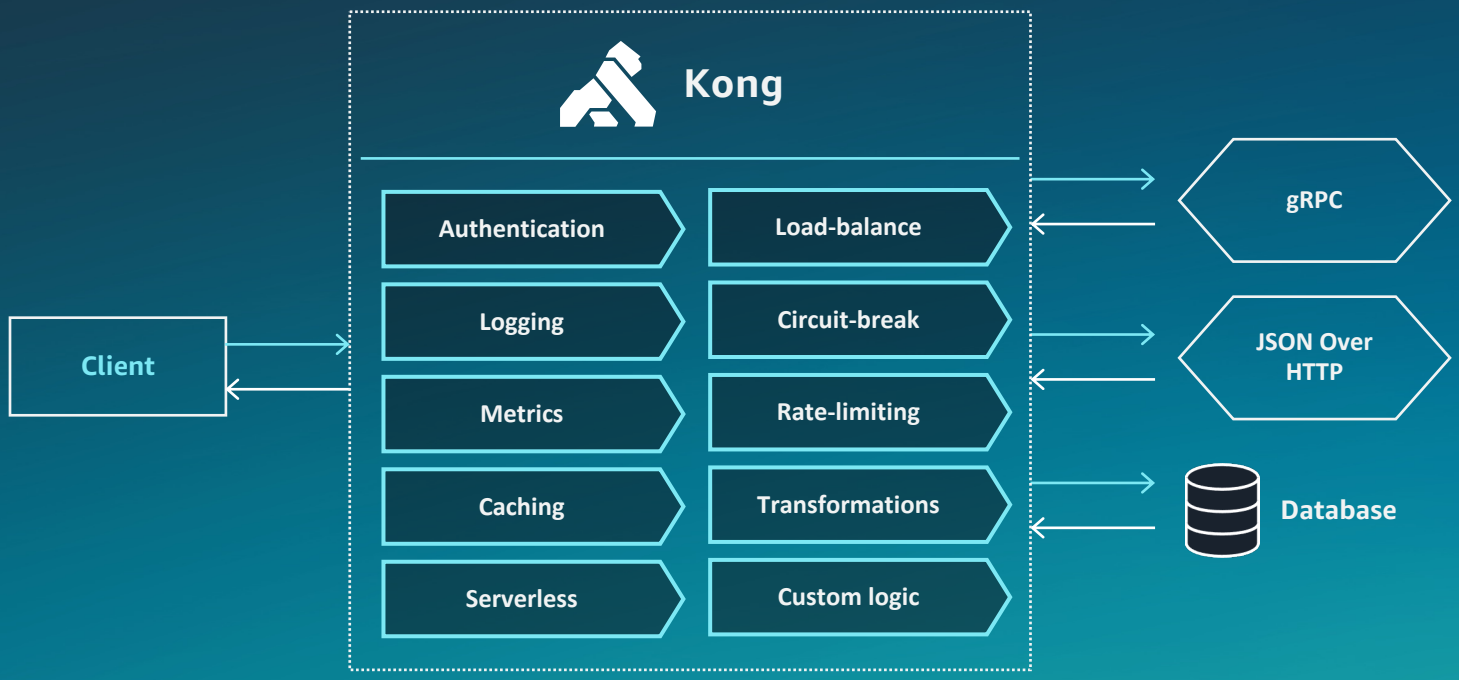



In the example above, an AWS Direct Connect is used to connect an on-premises environment to a Kong Control-Plane on AWS. The Data Planes are distributed to multiple locations to provide scalability and availability. Kong takes care of managing and maintaining the infrastructure so you can focus on building the business logic and application.

A typical installation of Kong Enterprise on Amazon EKS involves installing the Kong Control Plane and Data Plane on one or more clusters across multiple Availability Zones or AWS Regions. Plugins provide advanced functionality to extend the use of Kong Gateway. In the example, Kong's plugins send logs to Amazon CloudWatch, authenticates users via Amazon Cognito, and provides access for streaming data to the Confluent Cloud.

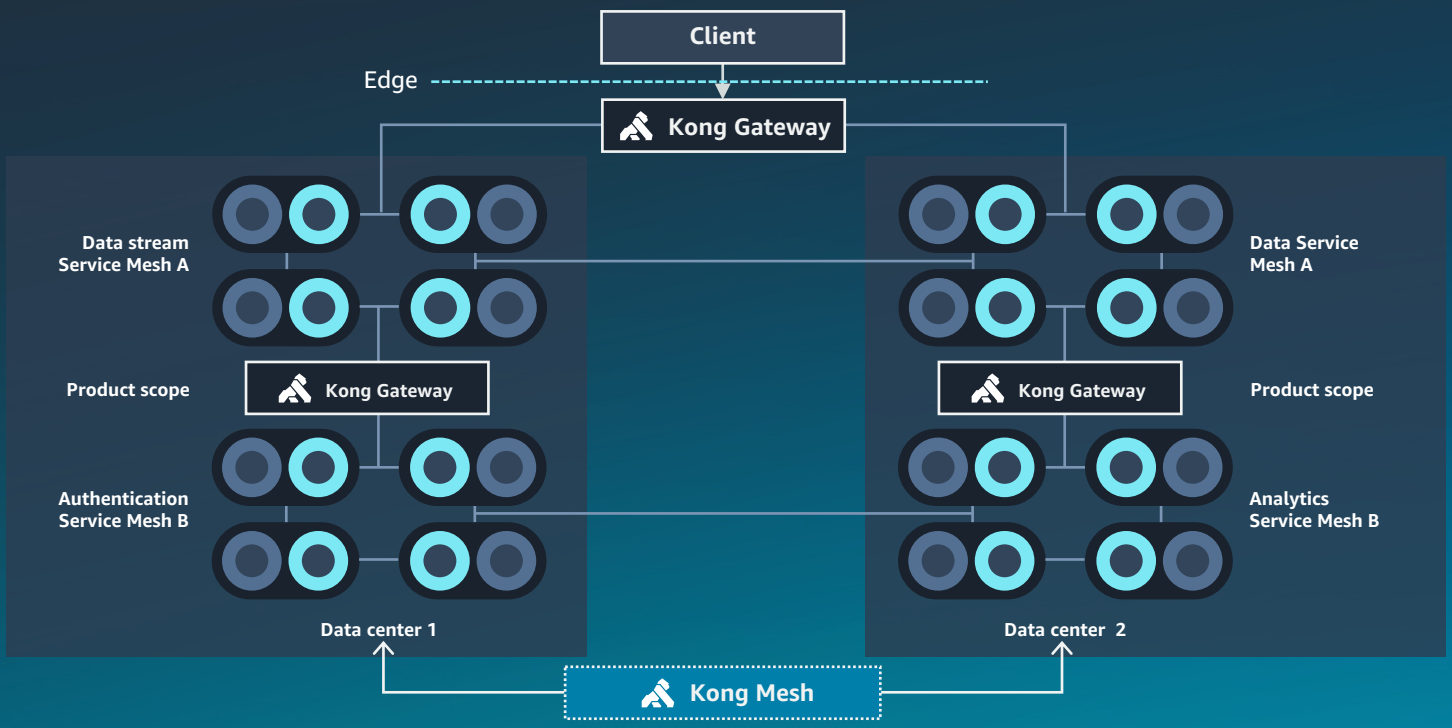
## Kong's API Gateway

The value of Kong's API gateway is the ability to easily add authorization, load balancing, rate limiting, caching, circuit-breaking, and data transformations without having to write this logic into code. With Kong API Gateway, you get this functionality and many more through a rich set of plugins.



## Kong Mesh for connectivity

As services expand, they'll need to communicate with each other. That's where a service mesh comes into play. Kong Mesh enables services that make up Everything-as-a-service within an organization to seamlessly communicate.




For the example solution, Kong Gateway and Kong Mesh give us:

- **Out-of-the-box service connectivity and discovery**
- **Zero-trust security**—each service is independent, and both the data stream service and analytics service are both consuming authentication
- **Traffic reliability**—the mesh ensures that traffic is routed to an available service whether that is in the same Availability Zone, different Availability Zone, or even a different Region
- **Global Observability across all traffic**, including cross-cluster deployments and hybrid deployments of on-premises and cloud

# Confluent and Apache Kafka

**Let's transition now to Confluent.**

What is Kafka? Kafka is a distributed data store optimized for ingesting and processing streaming data in real-time. Streaming data is data that is continuously generated by thousands of data sources, which typically send the data records simultaneously. A streaming platform needs to handle this constant influx of data and process the data sequentially and incrementally. Kafka provides three main functions to its users:

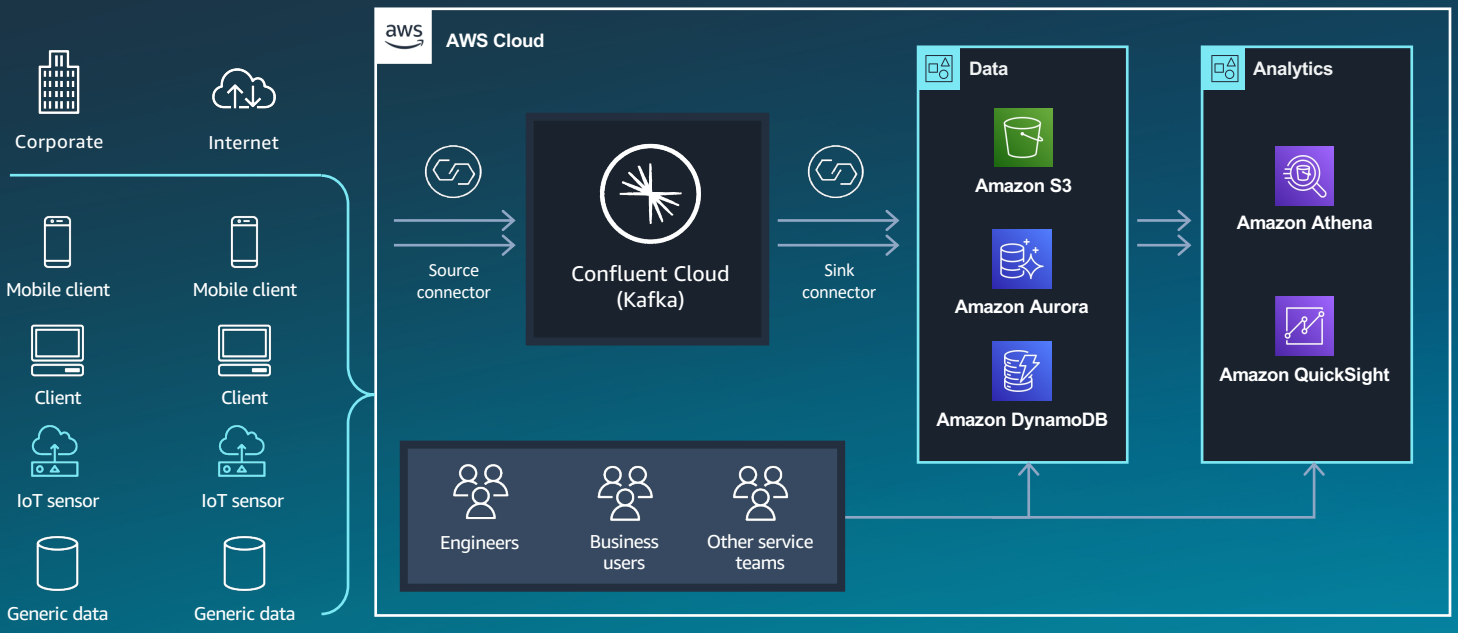
1. **Publish and subscribe to streams of records**
2. **Effectively store streams of records in the order in which records were generated**
3. **Process streams of records in real time**

Kafka is primarily used to build real-time streaming data pipelines and applications that adapt to the data streams. It combines messaging, storage, and stream processing to allow storage and analysis of both historical and real-time data.

## Confluent Cloud

In the example solution on the next page, Confluent Cloud is a fully managed Apache Kafka on AWS, which helps organizations focus on building apps and not managing clusters with a scalable, resilient, and secure service. With Confluent Cloud, you can connect to your existing data services to build real-time, event-driven applications with managed connectors to Amazon S3, Amazon Aurora, Amazon DynamoDB, and more.

Confluent Cloud is a hybrid solution connecting applications across both on-premises and AWS Cloud.




In the above example, Confluent takes data streamed in from mobile clients, servers, desktops, IoT sensors, and other generic data and streams those into Kafka as source connectors. Sink connectors are used to connect that data to data storage targets such as Amazon S3, Amazon Aurora, and Amazon DynamoDB.

In the example solution, the data storage service is decoupled from the data service. Implementing the streaming service in this fashion allows us to focus on the needs of the organization for streaming services and expand those capabilities to support current and future company products and service.

The methodology of separating these services also allows teams to compose products from different services instead of trying to reinvent the wheel and duplicating efforts associated with building a stream service.

## Three modalities of stream processing with Confluent

1. **Through Kafka Clients:** The Confluent platform includes client libraries for multiple languages to provide both low-level and higher-level access. Some of those include Java, C++, Go, .NET, Python, and Nodejs.
2. **Through Kafka Streams:** Kafka Streams is a client library for building applications and microservices, where the input and output data are stored in an Apache Kafka cluster.
3. **KsqlDB:** ksqlDB is a database purpose-built to help developers create stream processing applications on top of Apache Kafka.

## AWS Data and Analytics services

So far, we've talked about Kong for the API Gateway and Mesh, and Confluent for event stream processing with Kafka. Here are a few AWS services to note:

### Amazon S3

**Amazon S3** is a general object store and also a destination target for the data stream solution. Amazon S3 is used in this solution as it is a great long-term storage service that can be used directly by Amazon Athena and Amazon QuickSights for analytics processing and report generation.

### Amazon DynamoDB

**Amazon DynamoDB** is a NoSQL service that is fully managed with single-digit millisecond performance at any scale. In the event stream service, as streams of information are coming into Confluent Cloud, updates to a user's profile can be made in near-real time to DynamoDB. As an example event stream: A shipping company could be receiving real-time updates of a package location and part of the stream processing jobs could update the DynamoDB table on where the package is currently at based on this stream of information.

### Amazon Aurora

**Amazon Aurora** is a relational database management system (RDBMS) service that provides SQL access to the data tables of the event stream. Aurora complements the solution as there are use cases that require quick access to data for analytical processing. Aurora is highly scalable, highly durable, and fully managed.

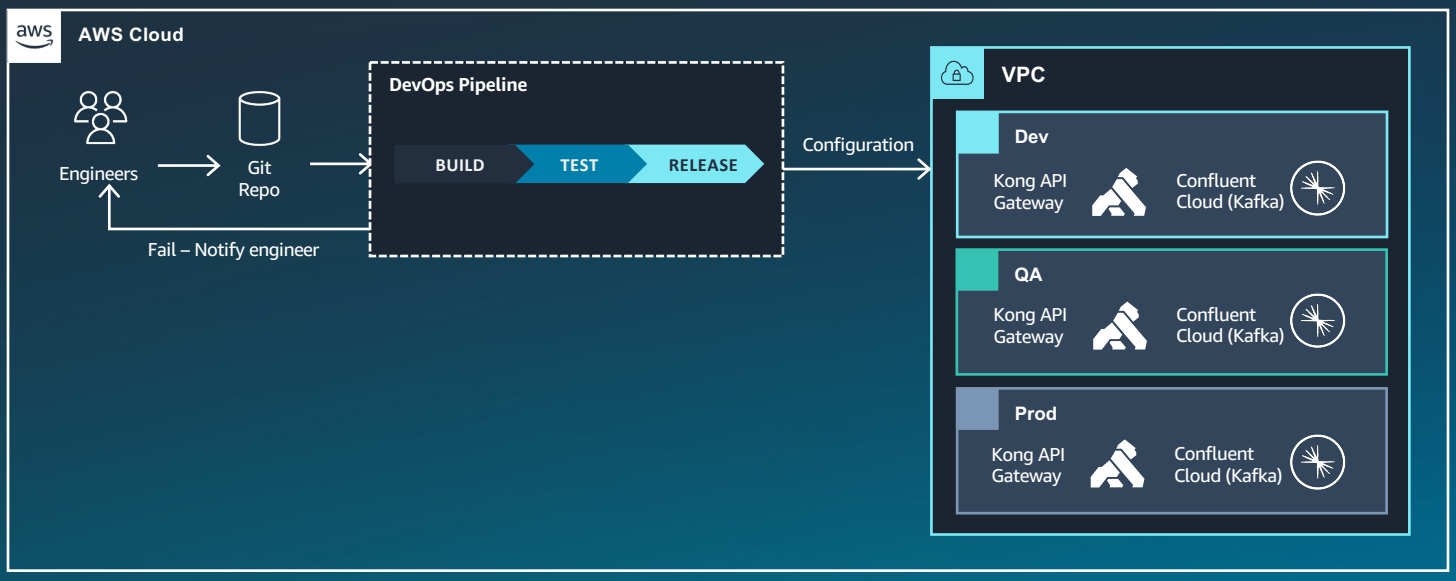
### Amazon Athena

### Amazon QuickSight

Both **Amazon Athena** and **Amazon QuickSight** are part of the analytics service in the example solution. Amazon Athena provides a SQL interface to data stored on Amazon S3 and Amazon QuickSight is a dashboard, reporting, and visualization tool that leverages data from Amazon S3, Amazon Aurora, and Amazon DynamoDB.

## CI/CD for streaming service

All of the services mentioned—including Kong and Confluent Cloud—all support automated build, test, deployment, and management through a CI/CD pipeline. Having a robust CI/CD pipeline for a service unlocks a team's speed and agility. Having fast feedback for changes of the service is important and allows developers to spend more time developing features and functionality versus shepherding releases to production.




## Why Kong and Confluent for the solution?

AWS understands the complexities that enterprises have to deal with and therefore looked for services that could support these different facets. Services often straddle many different environments, locations, and clouds. So the services were chosen to account for this complexity. There is also a need to make sure to stay true to the Everything-as-a-Service theme by providing platforms that companies could build upon while reducing the overhead of having to manage and maintain additional lower-level services.

Kong in particular provides the centralized control plane where the API can be managed and supports highly scaling not only the API connections but also the number of APIs and gateways that are supported. The Everything-as-a-Service philosophy will exponentially expand the number of APIs used and the idea is to make sure that doesn't become taxing on the teams that need to support them.

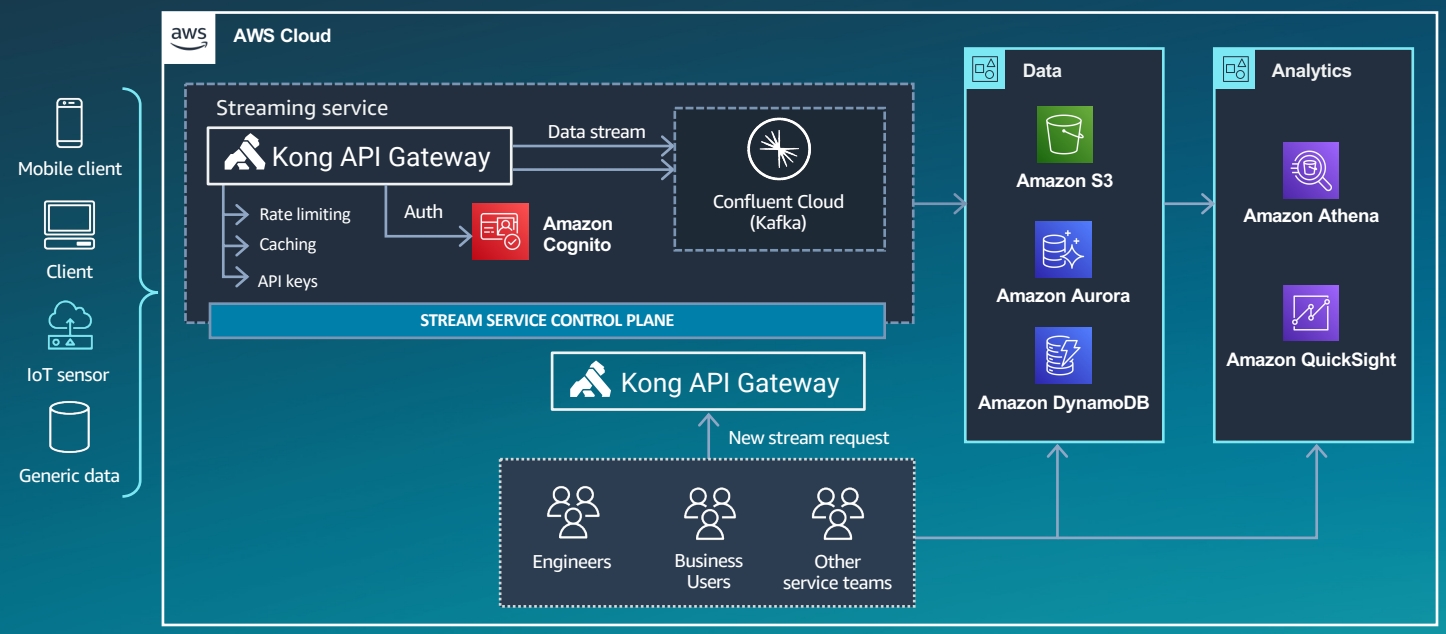
And Confluent was chosen because no one wants to get in the business of having to manage and maintain clusters of Kafka servers. It is a lot of work, having to spin up new clusters, backups, patching, and all the other things that go into maintaining a highly reliable and performant service. Let the heavy lifting of the lower-level components be done by someone else so the streaming service team can focus on providing products that the rest of the organization can consume. Confluent Cloud automatically scales and is managed for you.

# Example streaming service recap

Now that you've absorbed quite a bit of information, return to the example solution below and see how it operates with new eyes. To sum it up:

- You leveraged the Kong API management platform and Kong Mesh to expose the entire application as a service for both external producers and internal users.
- You leveraged Apache Kafka through a managed platform from Confluent Cloud to connect all the applications and data layers to other services that consume that data such as Amazon S3, Amazon Aurora, and Amazon DynamoDB. Downstream from the data service is an analytics service that consumes the data streamed to the data service.

**All of these services are SaaS offerings on AWS to abstract away the heaving lifting of managing and maintaining servers and infrastructure so you and your team can focus on adding business value.**



aws marketplace

# Continue your journey with AWS Marketplace

Confluent Cloud and Kong can be used together along with AWS to establish a well-engineered approach to EaaS. Gaining these capabilities can provide a strong foundation for continuing to advance through your cloud-native journey. You can explore these and other DevOps tools in AWS Marketplace.

To get started, visit: https://aws.amazon.com/marketplace/solutions/devops

## AWS Marketplace

Third-party research has found that customers using AWS Marketplace are experiencing an average time savings of 49 percent when needing to find, buy, and deploy a third-party solution. And some of the highest-rated benefits of using AWS Marketplace are identified as:

**Time to value**

**Cloud readiness of the solution**

**Return on Investment**

Part of the reason for this is that AWS Marketplace is supported by a team of solution architects, security experts, product specialists, and other experts to help you connect with the software and resources you need to succeed with your applications running on AWS.

Over 13,000 products from 3,000+ vendors:

CONFLUENT

Kong

APPDYNAMICS

circleci

HashiCorp

JFrog

Buy through AWS Billing using flexible purchasing options:

- Free trial
- Pay-as-you-go
- Hourly | Monthly | Annual | Multi-Year
- Bring your own license (BYOL)
- Seller private offers
- Channel Partner private offers

Deploy with multiple deployment options:

- AWS Control Tower
- AWS Service Catalog
- AWS CloudFormation (Infrastructure as Code)
- Software as a Service (SaaS)
- Amazon Machine Image (AMI)
- Amazon Elastic Container Service (ECS)
- Amazon Elastic Kubernetes Service (EKS)

aws marketplace

# Get started today

Visit aws.amazon.com/marketplace to find, try and buy software with flexible pricing and multiple deployment options to support your use case.

https://aws.amazon.com/marketplace/solutions/devops


## Authors:

**James Bland**
Global Tech Lead for DevOps, AWS

**Aditya Muppavarapu**
Global Segment Leader for DevOps, AWS