**What is Monolithic Architecture?**

In this lesson, we will discuss monolithic architecture.

**We'll cover the following**

* + [What is monolithic architecture?](https://www.educative.io/module/lesson/web-application-architecture-101/N8qv26ABVVm#What-is-monolithic-architecture?)

**What is monolithic architecture?**[#](https://www.educative.io/module/lesson/web-application-architecture-101/N8qv26ABVVm#What-is-monolithic-architecture?)

An application has a *monolithic architecture* if it contains the entire application code in a single codebase.

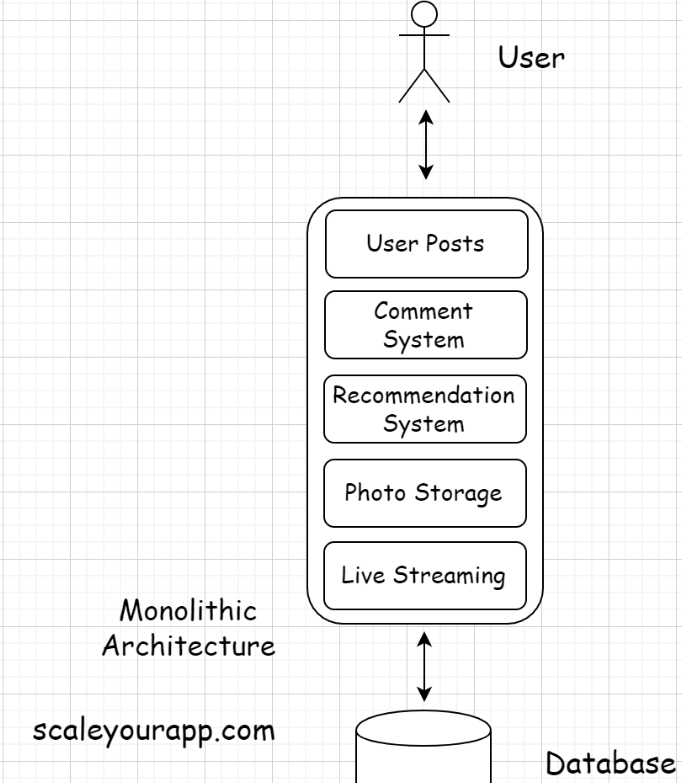
A *monolithic application* is a self-contained, tightly coupled software application. This is unlike the *microservices* architecture, where every distinct feature of an application may have one or more dedicated microservices powering it.

Taking example of a social networking site like *Facebook*. The application contains various features such as:

* User posts
* Comment system
* Groups
* Marketplace
* Portal Ads
* Photo storage
* Live streaming
* Recommendation system for recommending the latest and contextual content on the platform to the users and so on.

In a *monolithic architecture*, all the modules will be coded in a single codebase tightly coupled with each other as opposed to having one or more dedicated *microservice* for running respective features.

The diagram below represents a monolithic architecture.



Monolithic apps are simple to build, test, and deploy in comparison to a microservices architecture.

Often during the initial stages of a business, teams choose to move forward with a monolithic architecture, intending to branch out into a distributed microservices architecture later.

Well, this decision is a trade-off. We need to bear in mind refactoring and re-writing code has significant costs associated. Dismantling features from a tightly coupled architecture and re-implementing them into separate microservices demands a lot of time and resources.

There have been instances in the past where the dev teams decided to start with a monolithic architecture and later scaled out to a distributed microservices architecture.

This is what [*LinkedIn* did](https://engineering.linkedin.com/architecture/brief-history-scaling-linkedin). Though I would like to state here that *LinkedIn* started at a time when *cloud computing* and *microservices architecture* weren’t the norm.

In the present computing landscape, applications are built and deployed on the cloud. Also, businesses have to move fast. A wise decision is to pick the loosely coupled stateless microservices architecture from the start if we have multiple distinct features in our application and expect things to grow at a rapid pace in the future.

On the flip side, if our requirements are simple, monolithic architecture would suit best. Implementing a microservices architecture in this use case would be overkill. After all, managing numerous modules running in conjunction in a distributed environment isn’t a walk in the park.

In the next lesson, let’s go through some of the pros and cons of monolithic architecture.

# When should you pick a Monolithic Architecture?

In this lesson, you will learn about the pros and cons of monolithic architecture and when to choose it for your project.

**We'll cover the following**

* + [Pros of monolithic architecture](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Pros-of-monolithic-architecture)
    - [Simplicity](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Simplicity)
  + [Cons of monolithic architecture](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Cons-of-monolithic-architecture)
    - [Continuous deployment](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Continuous-deployment)
    - [Regression testing](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Regression-testing)
    - [Single points of failure](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Single-points-of-failure)
    - [Scalability issues](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Scalability-issues)
    - [Cannot leverage heterogeneous technologies](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Cannot-leverage-heterogeneous-technologies)
    - [Not cloud-ready, hold state](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Not-cloud-ready,-hold-state)
  + [When should you pick a monolithic architecture?](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#When-should-you-pick-a-monolithic-architecture?)

## Pros of monolithic architecture[#](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Pros-of-monolithic-architecture)

### Simplicity[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Simplicity)

Monolithic applications are simple to develop, test, deploy, monitor and manage since everything resides in one repository.

Things are relatively simple when dealing with one repository averting the complexity associated when handling and monitoring multiple components deployed separately.

## Cons of monolithic architecture[#](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Cons-of-monolithic-architecture)

### Continuous deployment[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Continuous-deployment)

Continuous deployment is a pain in monolithic applications as even a minor code change in a certain application layer or a feature necessitates a re-deployment of the entire application.

### Regression testing[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Regression-testing)

The downside of re-deployment of the entire application is that we need to perform a thorough regression testing of the whole application after the deployment is done. A minor code change in one feature can potentially impact the functionality of other features significantly since all the features are tightly coupled with each other.

### Single points of failure[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Single-points-of-failure)

Monolithic applications have a single point of failure. A bug in any of the application features can bring down the entire application.

### Scalability issues[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Scalability-issues)

Maintenance and scalability are a challenge in monolith apps as all the components are so tightly coupled with each other. As the code size increases, things get trickier to manage.

### Cannot leverage heterogeneous technologies[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Cannot-leverage-heterogeneous-technologies)

Building complex applications with a monolithic architecture is tricky since multiple technologies and programming languages need to be leveraged to implement the respective features of an application.

Using multiple programming languages in a single codebase becomes a mess also often times not possible. Heterogeneous technologies have compatibility issues and microservices architecture suits best to leverage them.

It is tricky to use Java and NodeJS together in a single codebase, and when I say tricky, I am being optimistic. I am not sure if it is even possible.

### Not cloud-ready, hold state[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#Not-cloud-ready,-hold-state)

Generally, monolithic applications are not cloud-ready as they may hold state in the static variables. Though not all monolith apps are designed that way, it’s a general observation that legacy apps use static variables to quite an extent. An application to be cloud-native, to have a consistent behavior on the cloud, has to be stateless.

## When should you pick a monolithic architecture?[#](https://www.educative.io/module/lesson/web-application-architecture-101/3YPjDrzPYW9#When-should-you-pick-a-monolithic-architecture?)

Monolithic applications fit best for use cases where the app requirements are pretty simple; when the application is not that complex. Examples of this are a to-do list app, a sports news app, an organization’s internal tax calculation app, a similar open public tool, etc.

These are the use cases where the business is certain that the application will have limited features and complexity and will not require any serious expansion in the near future.

In the next lesson, we will learn about microservice architecture.

# What is Microservice Architecture?

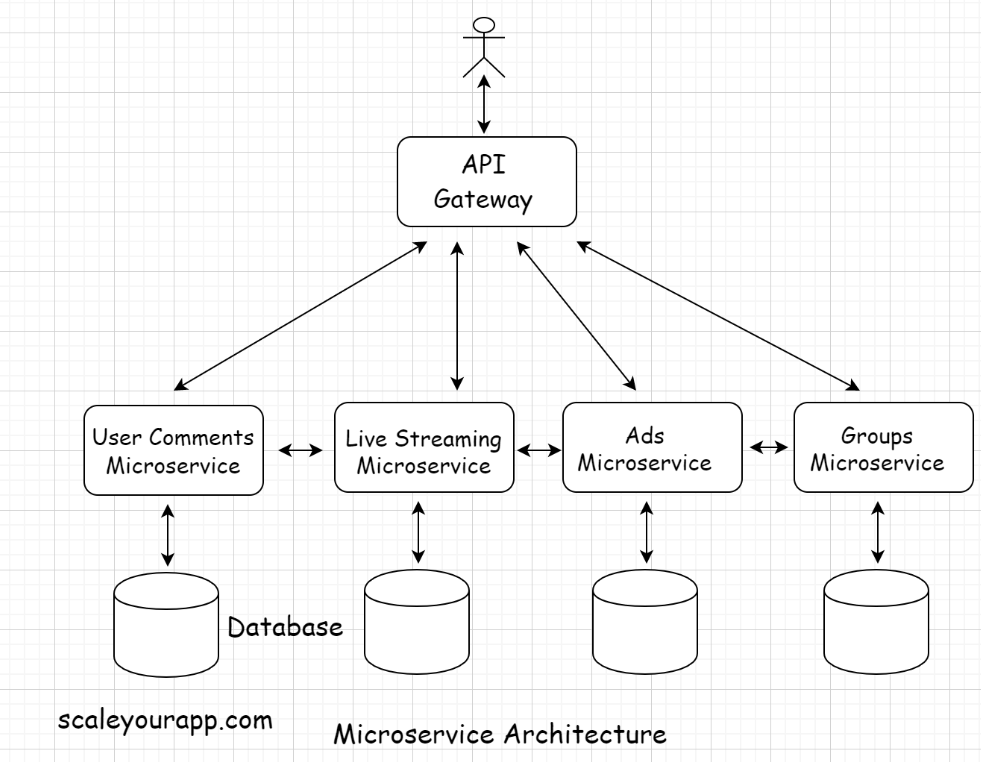
In this lesson, you will learn about microservice architecture.

**We'll cover the following**

* + [What is microservices architecture?](https://www.educative.io/module/lesson/web-application-architecture-101/R1GBlvMKG70#What-is-microservices-architecture?)

## What is microservices architecture?[#](https://www.educative.io/module/lesson/web-application-architecture-101/R1GBlvMKG70#What-is-microservices-architecture?)

In a microservices architecture, different features of an extensive service like Facebook are deployed separately as smaller loosely coupled services called microservices. These microservices work in conjunction to form a large distributed online service as a whole.



Remember the *single responsibility* and the *separation of concerns* principles? Both principles come into effect in a microservices architecture.

Every service has a single responsibility of running a specific feature and is separated from other services facilitating a loosely coupled architecture.

This particular architecture facilitates easier, cleaner app maintenance, feature development, testing, and deployment of individual modules in contrast to a monolithic architecture.

Imagine accommodating every feature in a single repository. How complex would things get? It would be a maintenance nightmare.

Also, when the project is large, it is managed by several different teams. When application modules/features are separate, they can be assigned to dedicated teams with minimal fuss, smoothing out the development process.

With microservices, scalability becomes easy too. The architecture is inherently designed to scale. Services that need scaling can be scaled independently without affecting other services.

Also, every microservice ideally has a separate database. This eliminates single points of failure and system bottlenecks.

In the next lesson, let’s go through some of the pros and cons of using a microservices architecture.

# When should you pick Microservices Architecture?

In this lesson, you will learn about the pros and cons of microservice architecture and when to pick it for your project.

**We'll cover the following**

* + [Pros of microservice architecture](https://www.educative.io/module/lesson/web-application-architecture-101/qA88zQPVZ6r#Pros-of-microservice-architecture)
    - [No Single Points of failure](https://www.educative.io/module/lesson/web-application-architecture-101/qA88zQPVZ6r#No-Single-Points-of-failure)
    - [Leverage the heterogeneous technologies](https://www.educative.io/module/lesson/web-application-architecture-101/qA88zQPVZ6r#Leverage-the-heterogeneous-technologies)
    - [Independent and continuous deployments](https://www.educative.io/module/lesson/web-application-architecture-101/qA88zQPVZ6r#Independent-and-continuous-deployments)
  + [Cons of microservices architecture](https://www.educative.io/module/lesson/web-application-architecture-101/qA88zQPVZ6r#Cons-of-microservices-architecture)
    - [Management complexity](https://www.educative.io/module/lesson/web-application-architecture-101/qA88zQPVZ6r#Management-complexity)
    - [Strong consistency](https://www.educative.io/module/lesson/web-application-architecture-101/qA88zQPVZ6r#Strong-consistency)
  + [When should you pick a microservices architecture?](https://www.educative.io/module/lesson/web-application-architecture-101/qA88zQPVZ6r#When-should-you-pick-a-microservices-architecture?)

## Pros of microservice architecture[#](https://www.educative.io/module/lesson/web-application-architecture-101/qA88zQPVZ6r#Pros-of-microservice-architecture)

### No Single Points of failure[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/qA88zQPVZ6r#No-Single-Points-of-failure)

Since microservices is a loosely coupled architecture, there is no single point of failure. Even if a few services go down, the application as a whole would still be up.

### Leverage the heterogeneous technologies[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/qA88zQPVZ6r#Leverage-the-heterogeneous-technologies)

Every microservice interacts with each other via a REST API gateway interface. A system with microservices can leverage the polyglot persistence architecture and other heterogeneous technologies like Java, Python, Ruby, NodeJS, etc.

Polyglot persistence uses multiple database types, like SQL and NoSQL, together in the architecture. We will discuss this in detail in the database lesson.

### Independent and continuous deployments[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/qA88zQPVZ6r#Independent-and-continuous-deployments)

The deployments can be independent and continuous. We can have dedicated teams for every microservice, and they can be scaled independently without impacting other services.

## Cons of microservices architecture[#](https://www.educative.io/module/lesson/web-application-architecture-101/qA88zQPVZ6r#Cons-of-microservices-architecture)

### Management complexity[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/qA88zQPVZ6r#Management-complexity)

Microservices is a distributed environment with several services powered by clusters of servers. This makes system management and monitoring complex.

We need to set up additional components to manage microservices, such as a node manager like Apache Zookeeper, a distributed tracing service for monitoring the nodes, etc.

We need skilled resources and even have to set up a dedicated team just to manage these services.

### Strong consistency[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/qA88zQPVZ6r#Strong-consistency)

Sometimes, strong consistency is hard to guarantee in a distributed environment, especially when trying to achieve a single consistent state across several microservices.

Things are eventually consistent across the nodes, and this limitation is due to the distributed design.

In the database chapter, we will discuss both strong and eventual consistency in detail.

## When should you pick a microservices architecture?[#](https://www.educative.io/module/lesson/web-application-architecture-101/qA88zQPVZ6r#When-should-you-pick-a-microservices-architecture?)

The microservice architecture fits best for complex use cases; for apps that need to expand quick from adding new features standpoint. A social network application is a good example of a complex use case.

A typical social networking application has various components such as messaging, real-time chat, LIVE video streaming, photo uploads, post like and share features, etc.

This use case fits best for a microservices architecture. Also, the microservices architecture enables a business move fast. A business can separately develop, test, deploy an application feature without affecting the current features. There is not much need for regression testing and so on.

Writing every feature in a single codebase would take no time to become a mess.

So, by now, we have seen three ways to proceed with the design of our application:

* Picking a monolithic architecture
* Picking a microservice architecture
* Starting with a monolithic architecture and later scaling out into a microservice architecture.

Picking a monolithic or a microservice architecture largely depends on our use case. I suggest keeping things simple and thoroughly understanding the requirements before deciding on an architecture.

In the next lesson, let’s understand the trade-offs involved when choosing monolith and microservices architecture.

# Monolith and Microservices– Understanding the Trade-Offs – Part 1

This lesson discusses the trade-offs of choosing between a monolith and microservices architecture.

**We'll cover the following**

* + [Fault isolation](https://www.educative.io/module/lesson/web-application-architecture-101/xlrAgL4D3r3#Fault-isolation)
  + [Development team autonomy](https://www.educative.io/module/lesson/web-application-architecture-101/xlrAgL4D3r3#Development-team-autonomy)
  + [Segment – From monolith to microservices and back again to the monolith](https://www.educative.io/module/lesson/web-application-architecture-101/xlrAgL4D3r3#Segment-%E2%80%93-From-monolith-to-microservices-and-back-again-to-the-monolith)

So far, in this chapter, we have discussed what monolith is, what a microservice is, their pros and cons, and when to pick which. This lesson continues our discussion on the trade-offs of choosing between a monolith and microservices architecture to design our application.

## Fault isolation[#](https://www.educative.io/module/lesson/web-application-architecture-101/xlrAgL4D3r3#Fault-isolation)

When we have a microservices architecture, it becomes easy for us to isolate faults and debug them. When a glitch occurs in a certain service, we just have to fix the issue in that particular service without the need to scan the entire codebase to locate and fix the issue. This is known as fault isolation.

## Development team autonomy[#](https://www.educative.io/module/lesson/web-application-architecture-101/xlrAgL4D3r3#Development-team-autonomy)

In the case of a monolith architecture, if the number of developers and the teams working on a single codebase grows beyond a certain number, it may impede the productivity and the velocity of the teams.

In this scenario, things become a little tricky to manage. As the size of the codebase increases, the compile-time and tests runtime increase too. This is because, in a monolith architecture, the entire codebase has to be compiled after a code change as opposed to just compiling the module we work on.

A code change made, in the codebase, by any other team has a direct impact on the features we develop. It may even break the functionality of our feature. Due to this, thorough regression testing is required every time anyone pushes new code or an update to production.

Also, as the code is pushed to production, we need all the teams to stop working on the codebase until the change is pushed to production.

The code pushed by a certain team may also require approval from other teams in the organization working on the same codebase. This process is a bottleneck in the system.

On the contrary, in the case of microservices, dedicated teams have complete ownership of their codebases. They have full development and deployment autonomy over their modules with separate deployment pipelines. Code management becomes easier. It becomes easier to scale individual services based on their traffic load patterns.

So, if you need to move fast, quickly launch a lot of features to the market and scale. Moving forward with microservices architecture is a good bet.

Having a microservices architecture sounds delightful, but we cannot ignore the increase in the complexity in the architecture due to this. Adopting microservices has its costs.

With the microservices architecture comes the need to set up distributed logging, monitoring, inter-service communication, service discovery, alerts, tracing, dedicated build and release pipelines, health checks, and so on. You may even have to write a lot of custom tooling from scratch for yourself.

I believe you get the idea. There are always trade-offs involved, and there is no perfect solution. We need to be crystal clear on our use case and see what architecture suits our needs best.

Let’s understand this further with the help of the real-world example of a company called Segment that started with a monolith architecture, moved to microservices and then moved back again to the monolith architecture.

## Segment – From monolith to microservices and back again to the monolith[#](https://www.educative.io/module/lesson/web-application-architecture-101/xlrAgL4D3r3#Segment-%E2%80%93-From-monolith-to-microservices-and-back-again-to-the-monolith)

[Segment](https://segment.com/) is a customer data platform that initially started with a monolith and later split it into microservices. As their business gained traction, they again decided to revert to the monolith architecture.

Why did they do that?

Let’s take a look.

The segment engineering team split their monolith into microservices for fault isolation and easy debugging of issues in the system.

Fault isolation with microservices helped them minimize the damage a fault caused in the system. When a fault occurred, it was confined to a particular service as opposed to impacting, even bringing down the entire system as a whole.

Given the original monolith architecture had low management overhead, there were single points of failure. A glitch in a certain functionality could impact the entire system.

Segment integrates data from many different data providers into their systems. As the business gained traction, they integrated more data providers into their system, creating a separate microservice for every data provider. The increase in the number of microservices led to a significant increase in the complexity of their architecture, subsequently taking a toll on their productivity.

The defects with regards to microservices started increasing significantly. They had three engineers solely dedicated to getting rid of these defects to keep the system online. This operational overhead became resource-intensive and slowed down the organization immensely.

To tackle the issue, they made the decision to move back to the monolith, giving up on fault isolation and other nice things that the microservices architecture brought along.

They ended up with an architecture with a single code repository called Centrifuge that handled billions of messages per day delivered to multiple APIs.

Let’s dive deeper into their architecture in the next lesson.

# Monolith and Microservices– Understanding The Trade-Offs – Part 2

This lesson continues the discussion of the trade-offs of choosing between a monolith and microservices architecture.

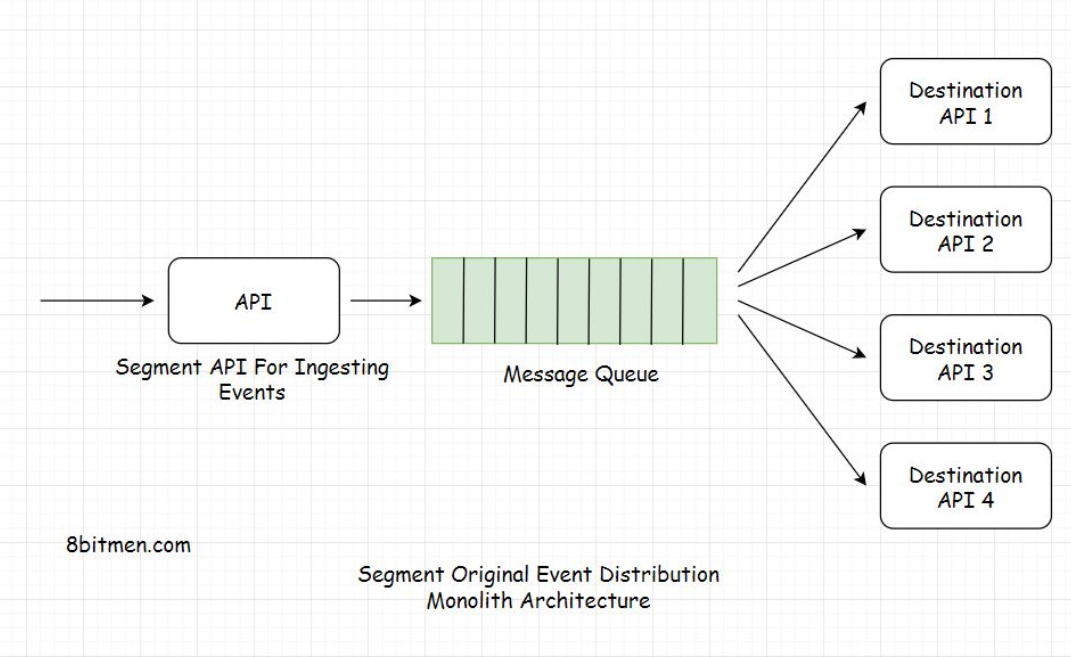
**We'll cover the following**

* + [Segment high-level architecture](https://www.educative.io/module/lesson/web-application-architecture-101/qVmNYymjqW7#Segment-high-level-architecture)
  + [Istio – The move from microservices to a monolith](https://www.educative.io/module/lesson/web-application-architecture-101/qVmNYymjqW7#Istio-%E2%80%93-The-move-from-microservices-to-a-monolith)

## Segment high-level architecture[#](https://www.educative.io/module/lesson/web-application-architecture-101/qVmNYymjqW7#Segment-high-level-architecture)

Segment’s data infrastructure ingests hundreds of thousands of events per second. These events are then directed to different APIs and webhooks via a message queue. These APIs are also called server-side destinations, and there are over a hundred of these destinations at Segment.

When they started with a monolith architecture, they had an API that ingested events from different sources, and the events were then forwarded to a distributed message queue. The queue moved the event payload further to other destination APIs according to configuration and settings.



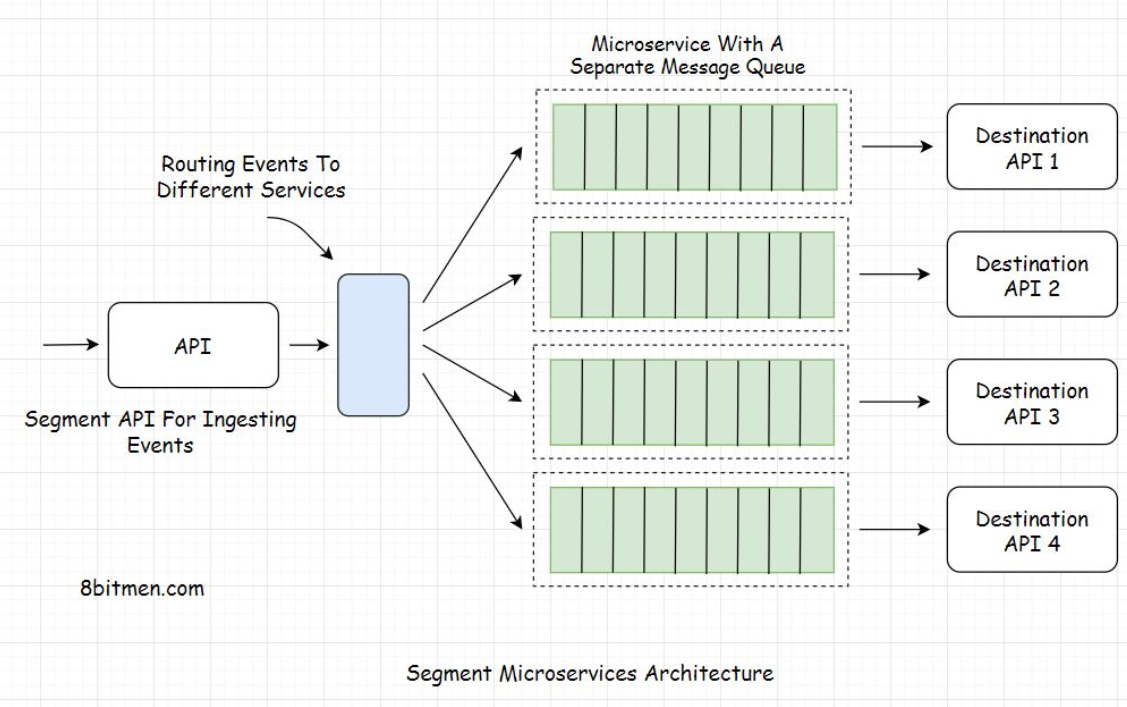
If you aren’t aware of what a message queue, webhook, and data ingestion are. No worries, we will discuss these in detail in the latter part of this course. The example I am currently discussing is pretty straightforward, nothing complicated. So, we can focus on this right now and delve into the rest of the concepts in detail later.

In the monolithic architecture, as all the events were moved into a single queue, some of the events often failed to deliver to the destinations and were retried by the queue after stipulated time intervals.

This made the queue contain both the new as well as the failed events waiting to be retried. As a result, the queue would be eventually flooded, resulting in delays in the delivery of events to the destinations.

To tackle the queue flooding issue, the engineering team at Segment split the monolith into microservices and created a separate microservice for every destination.

Now every service contained its own individual distributed message queue. This helped cut down the load on a single queue and enabled the system to scale, increasing the throughput.



In this scenario, even if a certain queue got flooded, it didn’t impact the event delivery of other services. This is how Segment leveraged fault isolation with the microservices architecture.

Over time as the business gained traction, additional destinations were added. Every destination had a separate microservice and a queue. The increase in the number of services led to an increase in the complexity of the architecture.

Separate services had separate event throughput and traffic load patterns. A single-scale policy couldn’t be applied to all the queues commonly. Every service and the queue needed to be uniquely scaled based on its traffic load pattern and this process had to be done manually.

Autoscaling was implemented in the infrastructure, but every service had different CPU & memory requirements. This required manual tuning of the infrastructure, meaning more queues needed more resources for maintenance.

To tackle this, Segment eventually reverted to monolith architecture, calling their architecture a Centrifuge that combined all the individual queues for different destinations into a single monolith service.

The info I have provided on Segment architecture in this lesson is very high-level. If you wish to go into more details and take a look at the Centrifuge architecture, go through these resources:

[Goodbye Microservices:](https://segment.com/blog/goodbye-microservices/) (<https://segment.com/blog/goodbye-microservices/> )From 100s Of Problem Children To 1 Superstar

[Centrifuge: A Reliable System](https://segment.com/blog/introducing-centrifuge/) (<https://segment.com/blog/introducing-centrifuge/>) For Delivering Billions Of Events Per Day

Below is another instance of a popular service that transitioned from microservices to a monolith architecture.

## Istio – The move from microservices to a monolith[#](https://www.educative.io/module/lesson/web-application-architecture-101/qVmNYymjqW7#Istio-%E2%80%93-The-move-from-microservices-to-a-monolith)

[Istio](https://istio.io/) (<https://istio.io/> )is an open-source service mesh that enables us to connect, secure, control, and observe microservices. It allows us to control how microservices share data with each other.

It recently transitioned from a microservices to a monolith architecture. According to the Istio team, having a monolith architecture enabled them to deliver value and achieve the goals they intended to.

**Recommended read** – Istio is an example of [when not to do microservices](https://blog.christianposta.com/microservices/istio-as-an-example-of-when-not-to-do-microservices/). (<https://blog.christianposta.com/microservices/istio-as-an-example-of-when-not-to-do-microservices/> )