Back To Module Home

Architecture of Scalable Applications

0% completed

Different Tiers in Software Architecture

Monolith and Microservices

- What is Monolithic Architecture?
- When should you pick a Monolithic Architecture?
- What is Microservice Architecture?
- When should you pick Microservices Architecture?
- Monolith and Microservices– Understanding the Trade-Offs – Part 1
- Monolith and Microservices– Understanding the Trade-Offs – Part 2

Conclusion

Mark Module as Completed

Monolith and Microservices– Understanding the Trade-Offs – Part 2

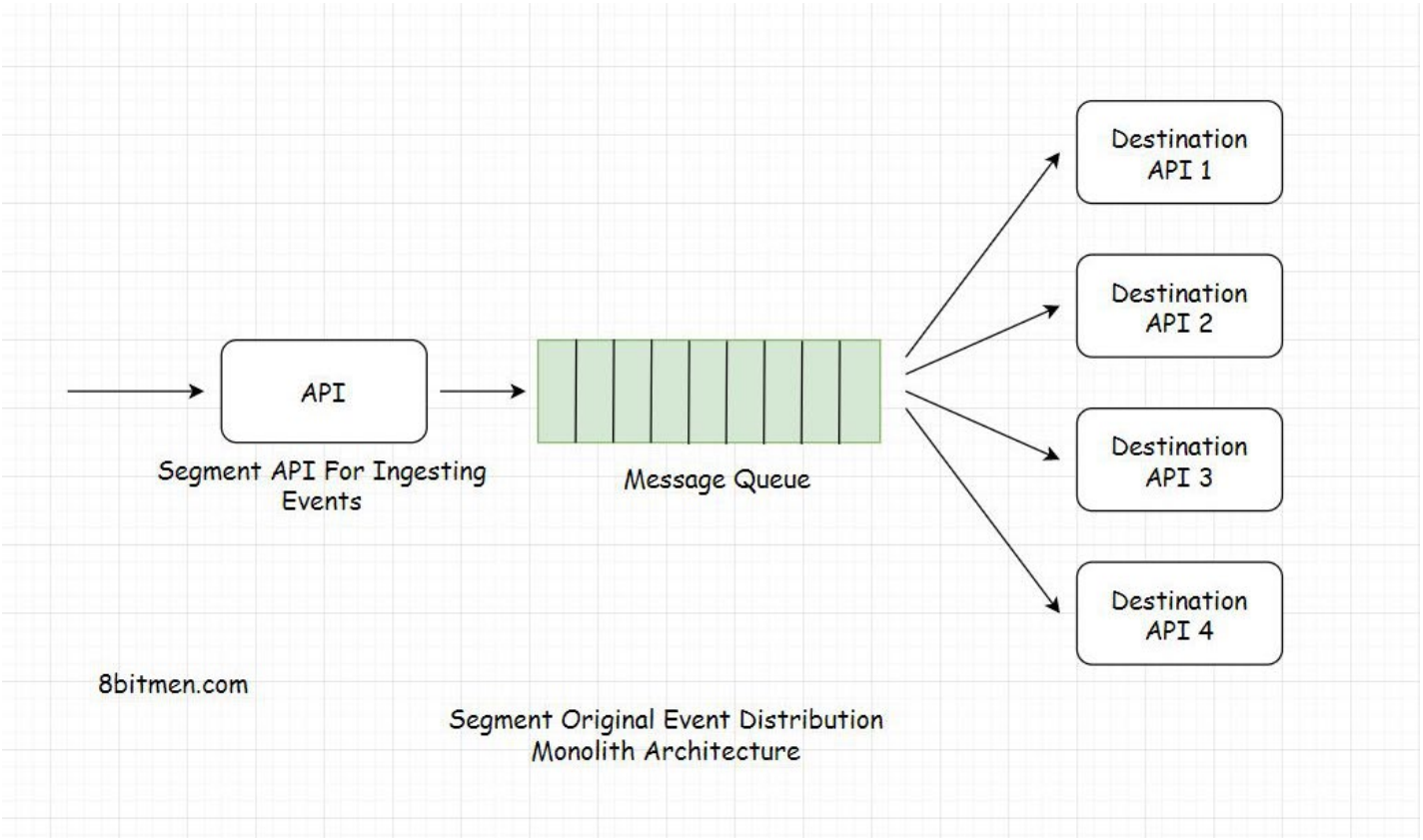
This lesson continues the discussion of the trade-offs of choosing between a monolith and microservices architecture.

We'll cover the following

Segment high-level architecture#

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.



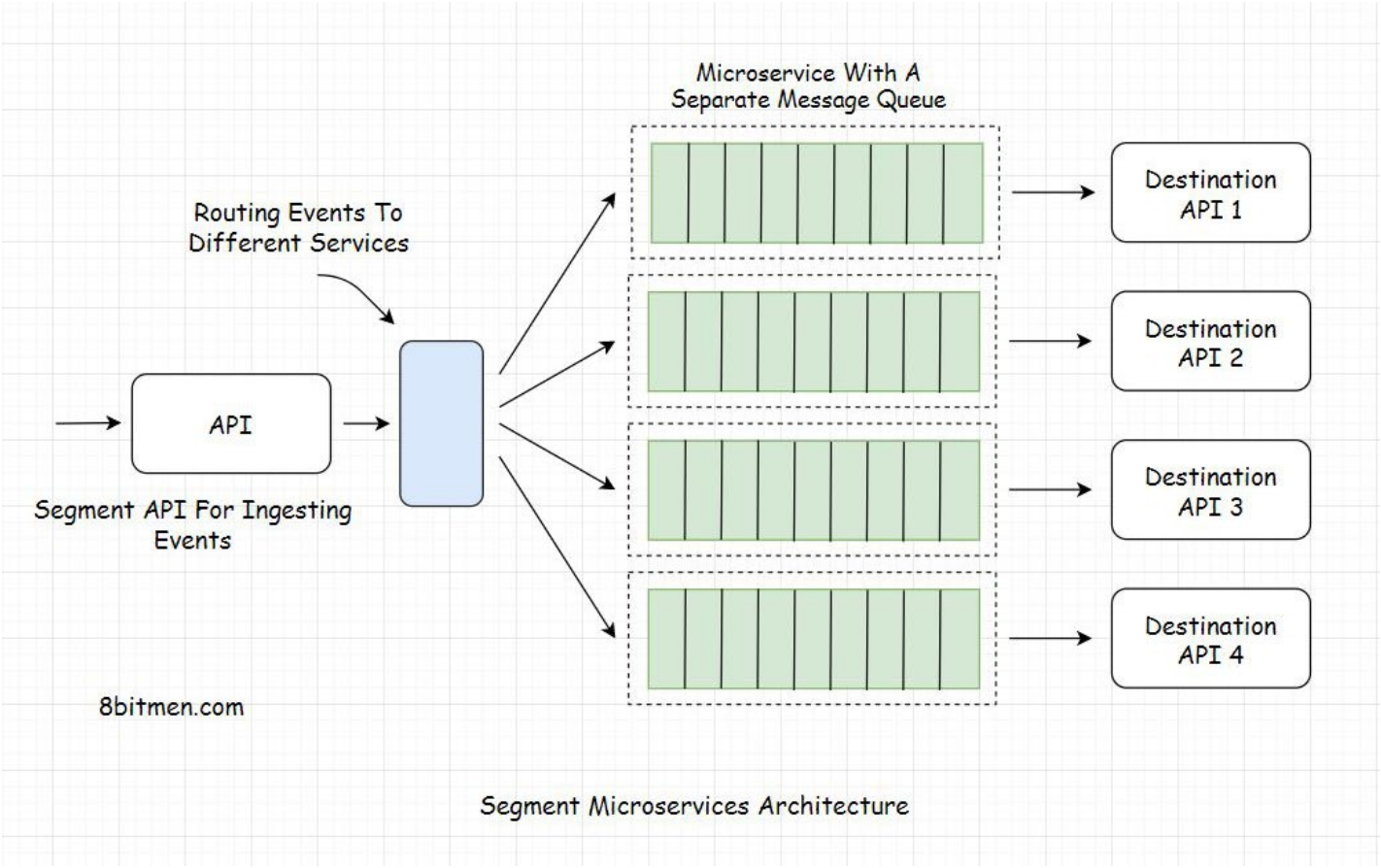
> If you aren't aware of what a message queue, webhook, and data ingestion are. No worries, we will discuss these in detail in the latter part of this course. The example I am currently discussing is pretty straightforward, nothing complicated. So, we can focus on this right now and delve into the rest 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: From 100s Of Problem Children To 1 Superstar

Centrifuge: A Reliable System 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#

Istio 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.

Monolith and Microservices– Underst…

Next

Monolith and Microservices Quiz

Mark as Completed

Report an Issue