Monolith & Microservices – Understanding The Trade-Offs – Part 2

This lesson continues the discussion on the trade-offs involved when choosing between the monolith and the microservices architecture

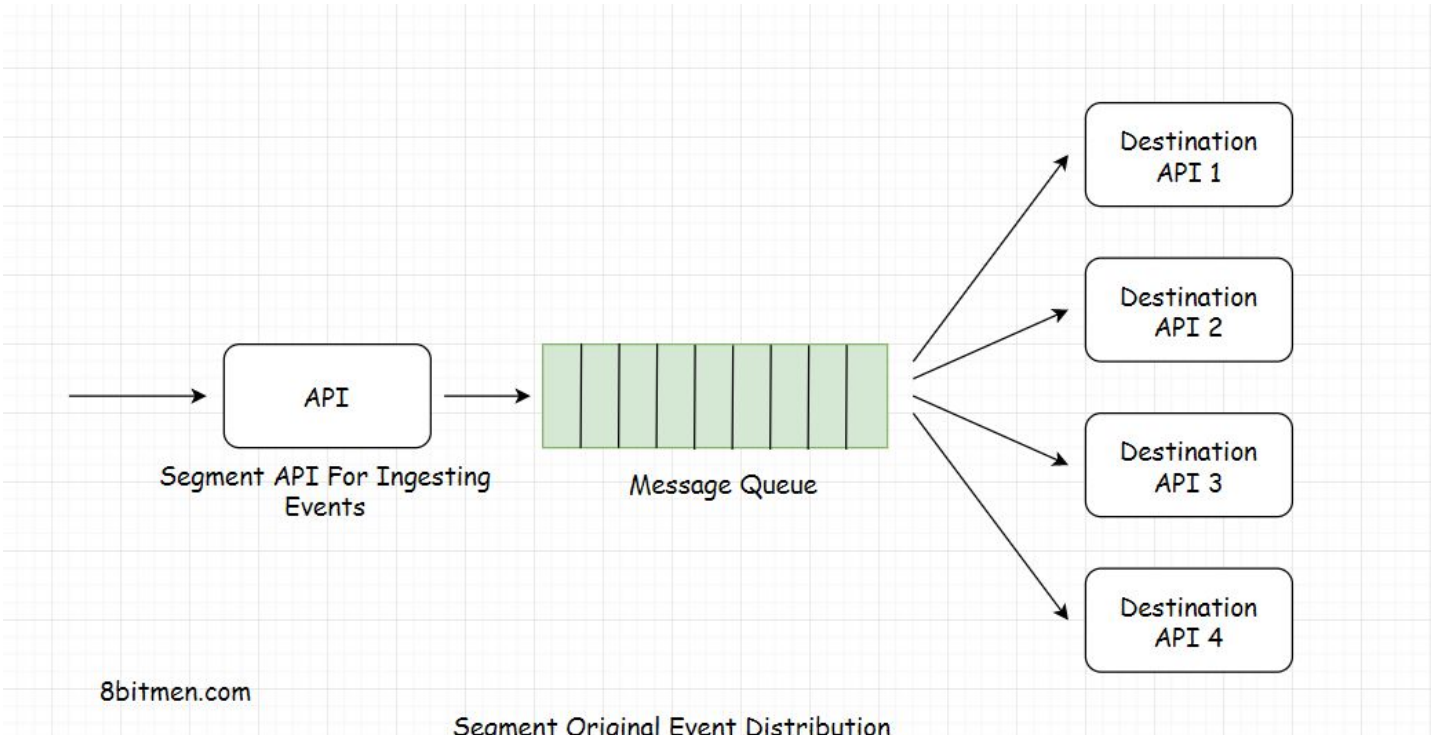
We'll cover the following

- Segment High-Level Architecture
- Istio – The Move From Microservices To A Monolith

Segment High-Level Architecture

Segment's data infrastructure ingests hundreds of thousands of events per second. These events are then directed to different *APIs* & webhooks via a message queue. These *APIs* are also called as server-side destinations & 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 those events were then forwarded to a distributed message queue. The queue based on configuration and settings further moved the event payload to different destination APIs.



Segment Original Event Distribution

Monolith Architecture

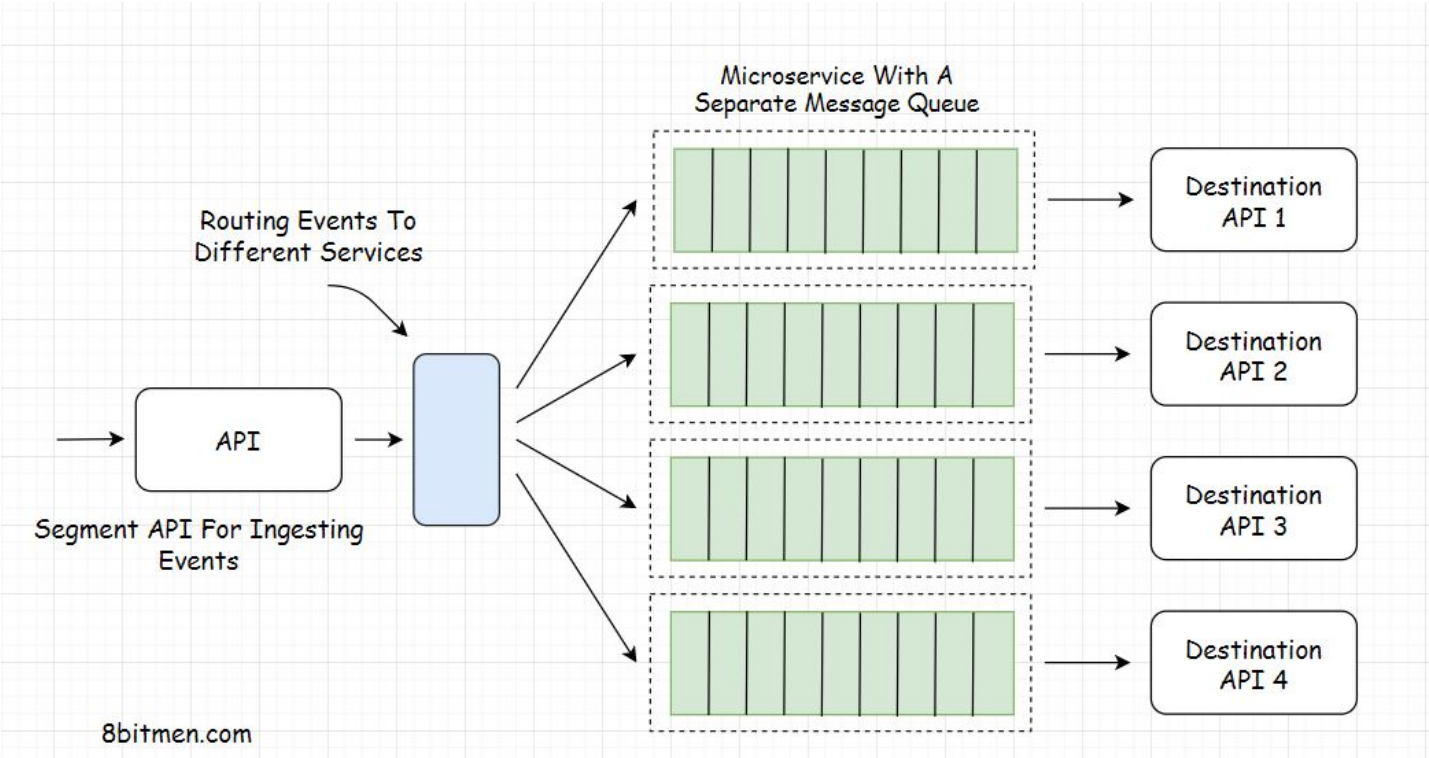
> *If you aren't aware of what a message queue, webhook & data ingestion is? No worries, I've discussed these in detail in the latter part of this course. This example that I am discussing right now is pretty straight forward nothing complicated, so we can focus on this right now & can delve in detail into the rest of the concepts 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. This eventually flooded the queue resulting in a delay of 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.

Every service contained its own individual distributed message queue. This helped cut down the load on a single queue & enabled the system to scale also increasing the throughput.



Segment Microservices Architecture

Segment Microservices Architecture

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* & *traffic load patterns*. A single scale policy couldn't be applied on all the queues commonly. Every service and the queue needed to be scaled differently based on its traffic load pattern. And this process had to be done manually.

Auto-scaling was implemented in the infrastructure but every service had distinct CPU & memory requirements. This needed manual tuning of the infrastructure. This meant - *more queues needed more resources for maintenance.*

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

The info that I have provided on Segment architecture in this lesson is very high-level. If you wish to go into more details, want to have a look at the Centrifuge architecture. Do 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 enables us to have control over 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.

Recommended Watch - Microservices - Are They Still Worth It?