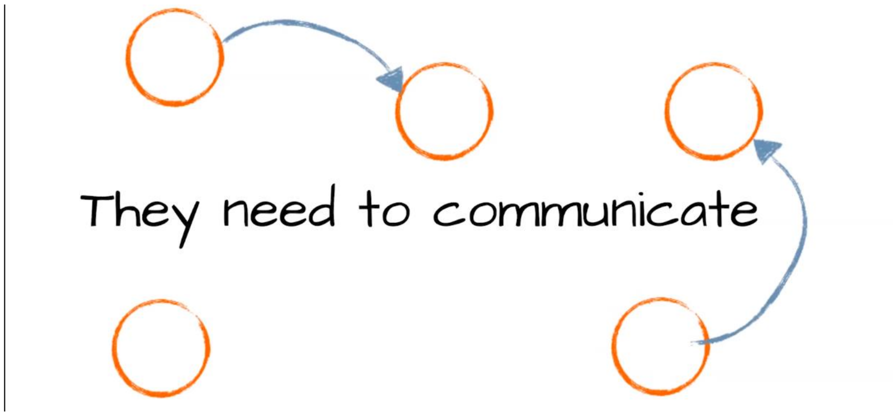
Beyond Microservices: Streams, State and Scalability

Gwen Shapira, Engineering Manager
@gwenshap

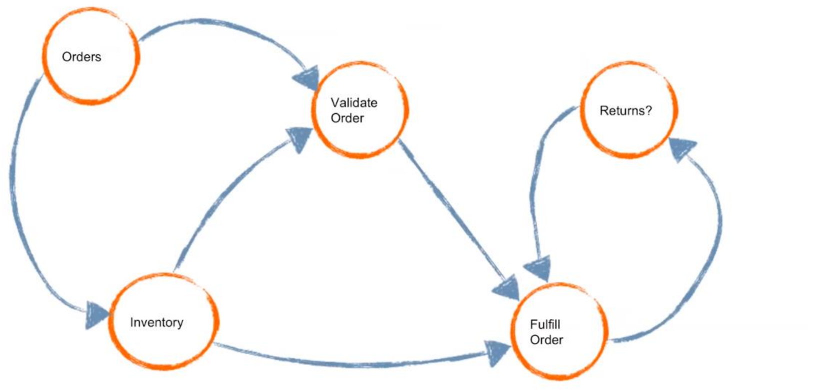
Microservices have been a popular architecture choice for at least 5 years by now. Over these years we've adopted microservices architectures to ever growing set of use-cases and different development and deployment strategies. Lessons were learned and our ability to design, develop, deploy and operate microservices has improved.

This presentation will give an opinionated view of how microservices evolved in the last few years, based on experience gained while working with companies using Apache Kafka to update their application architecture. We'll discuss the rise of API gateways, service mesh, state management and serverless architectures - what works well, and in which cases. We'll show real-world examples of how applications become more resilient and scalable when new patterns are introduced, and make sure to include caveats - because patterns...

In the beginning...



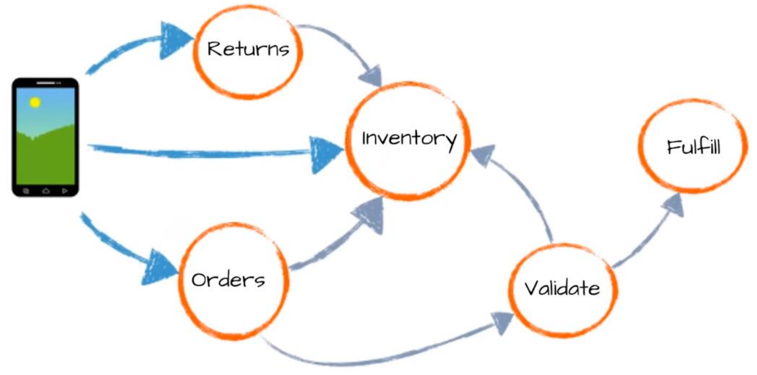
I know! I'll use REST APIs



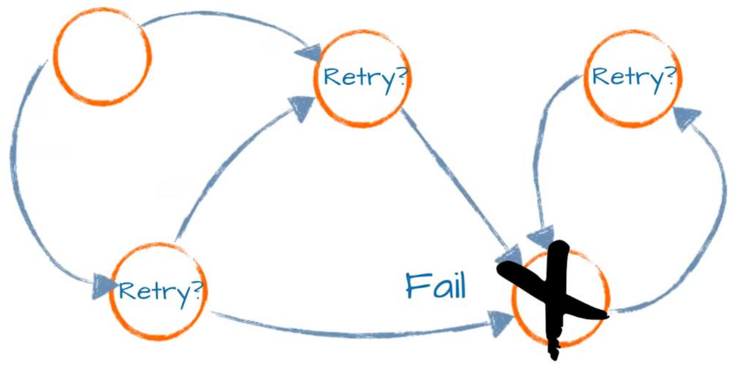
You now have distributed monoliths

Synchronous request-response communication
Leads to
Tight point-to-point coupling

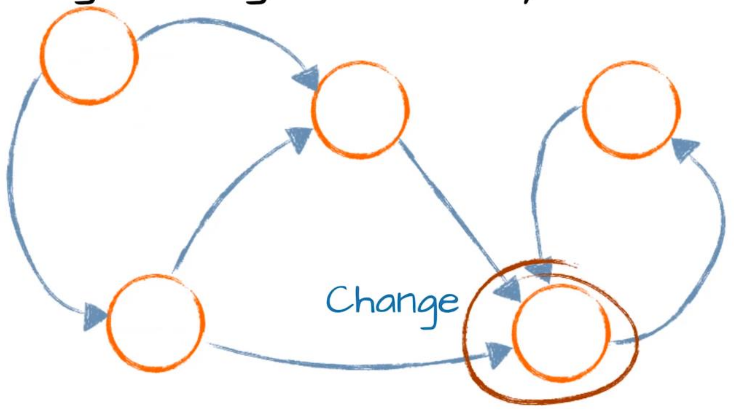
Clients know too much



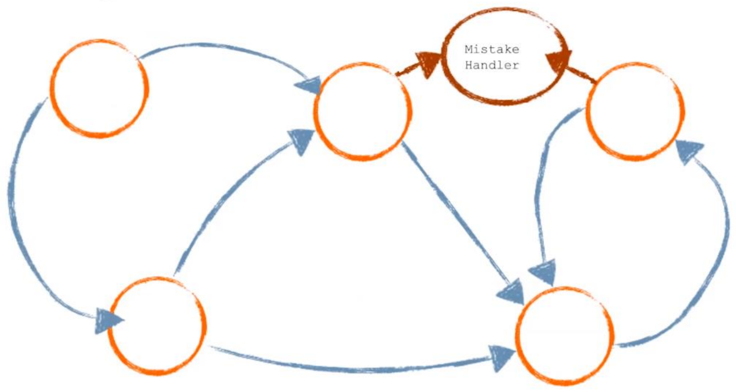
shifts in responsibility, redundancy



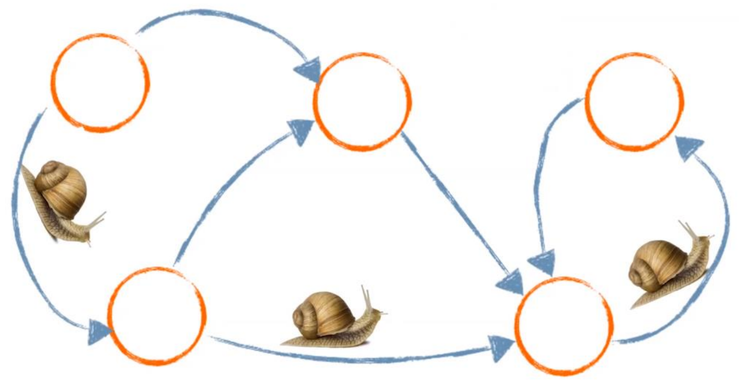
Making Changes is Risky



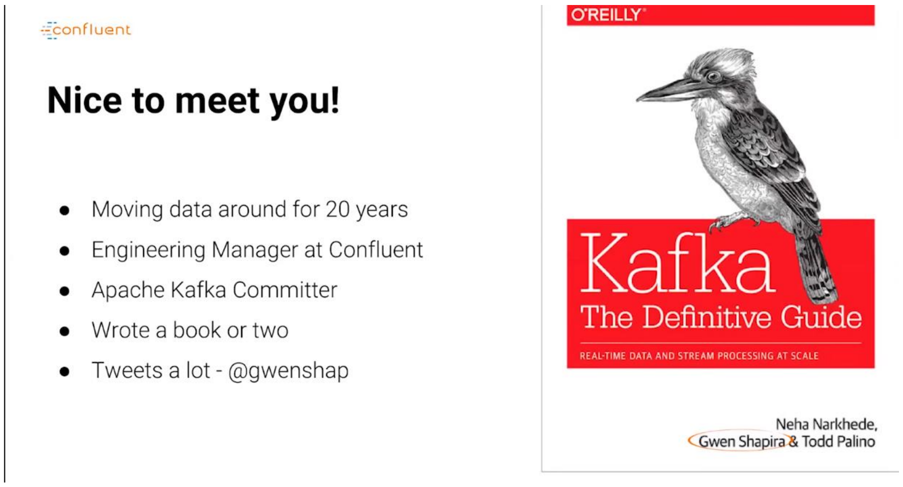
Adding Services Requires Explicit Calls



REST = HTTP + JSON = SLOW



We can do better.



confluent

Nice to meet you!

- Moving data around for 20 years
- Engineering Manager at Confluent
- Apache Kafka Committer
- Wrote a book or two
- Tweets a lot - @gwenshap

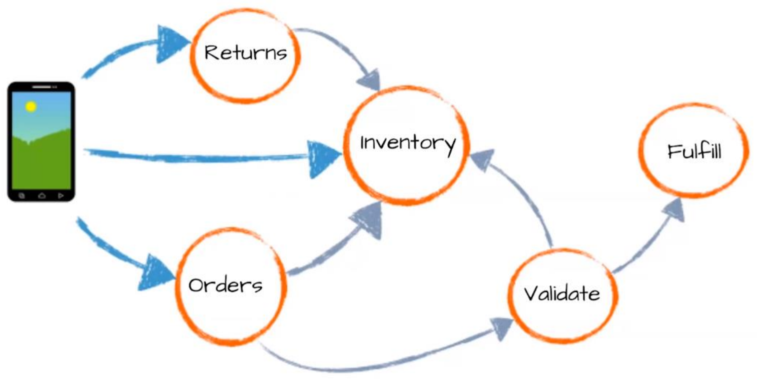
O'REILLY

Kafka
The Definitive Guide
REAL-TIME DATA AND STREAM PROCESSING AT SCALE

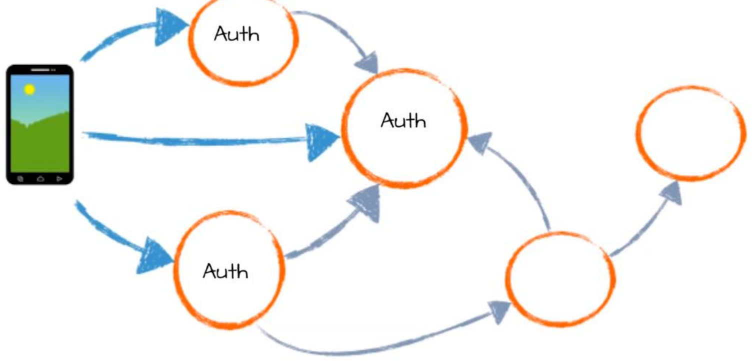
Neha Narkhede,
Gwen Shapira & Todd Palino

API Gateway

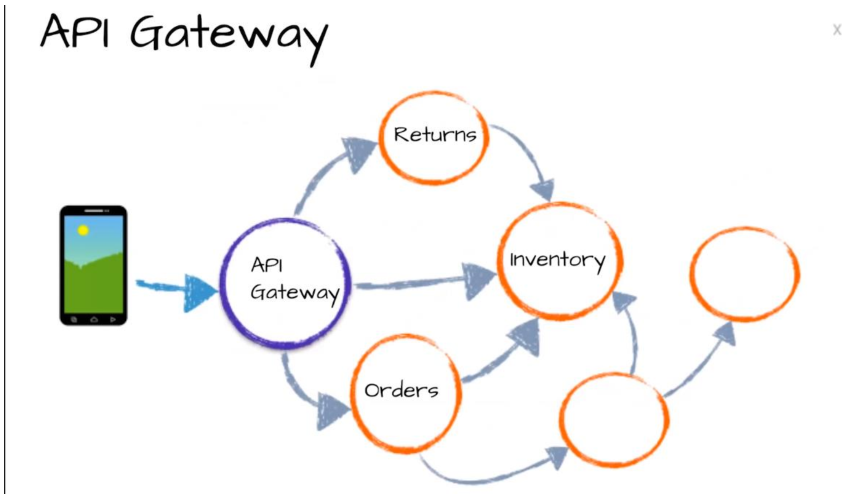
Clients know too much

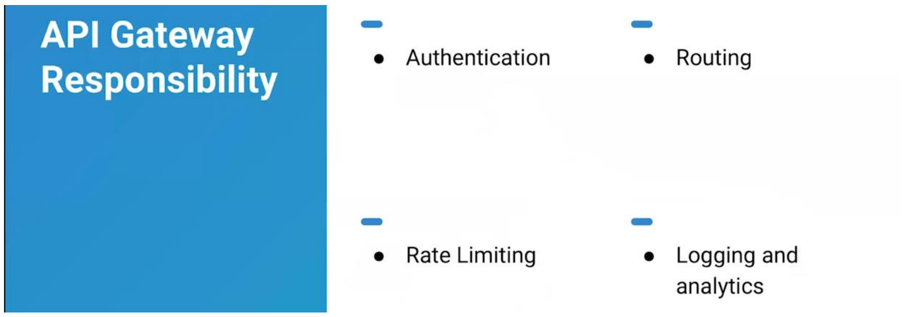


Shift in responsibility, redundancy



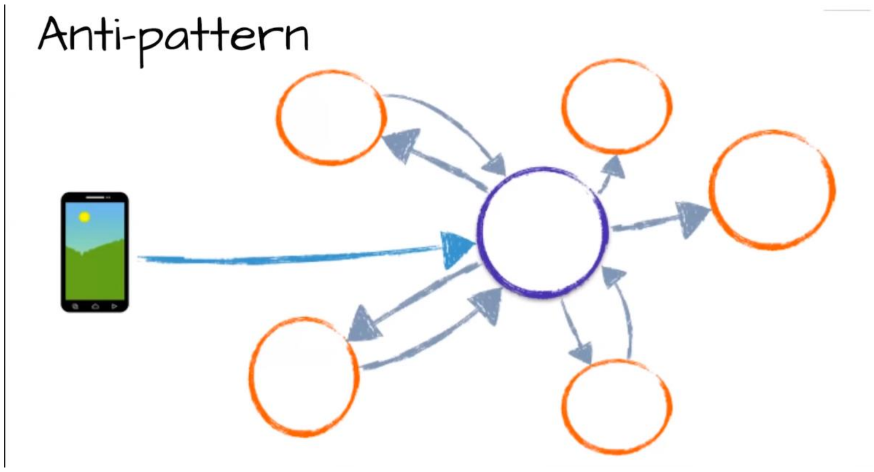
We don’t need all microservices having their own authentication logic





API Gateway Responsibility

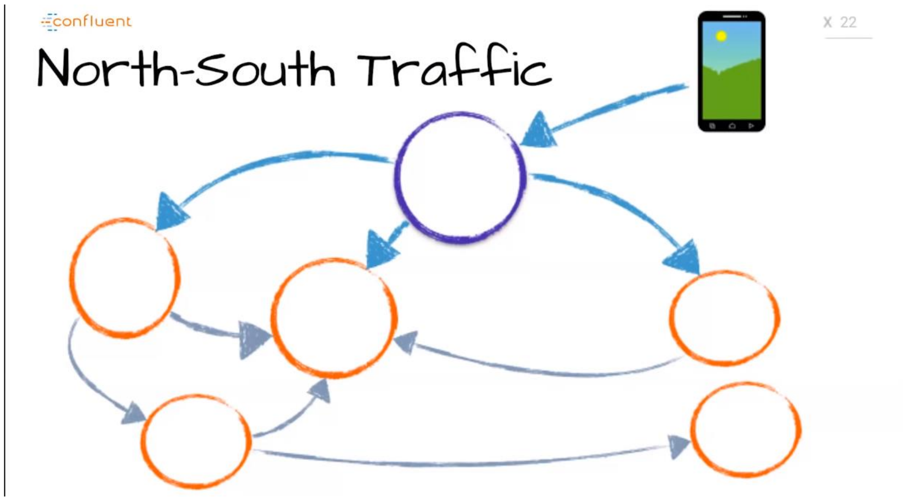
- Authentication
- Routing
- Rate Limiting
- Logging and analytics



The risk is that every microservice wants to use the API Gateway and this becomes a bottleneck and dependency

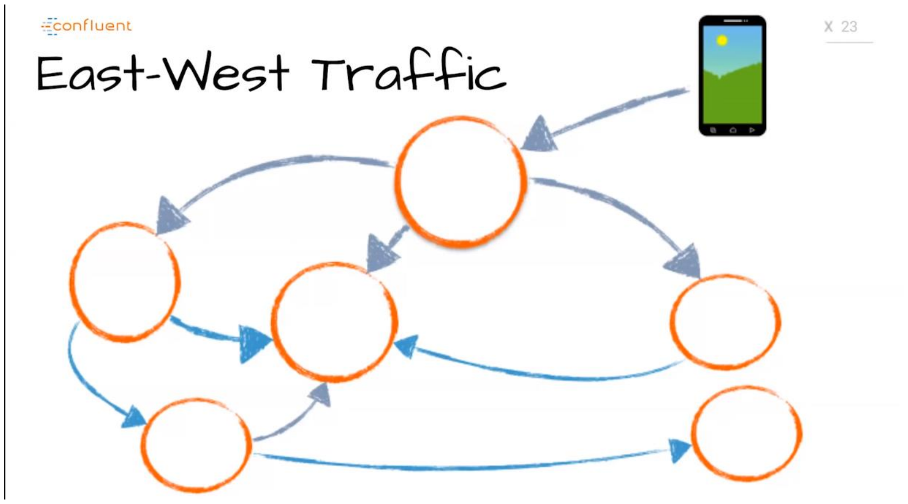
Service Mesh

Service Mesh is your distributed internal API Gateway



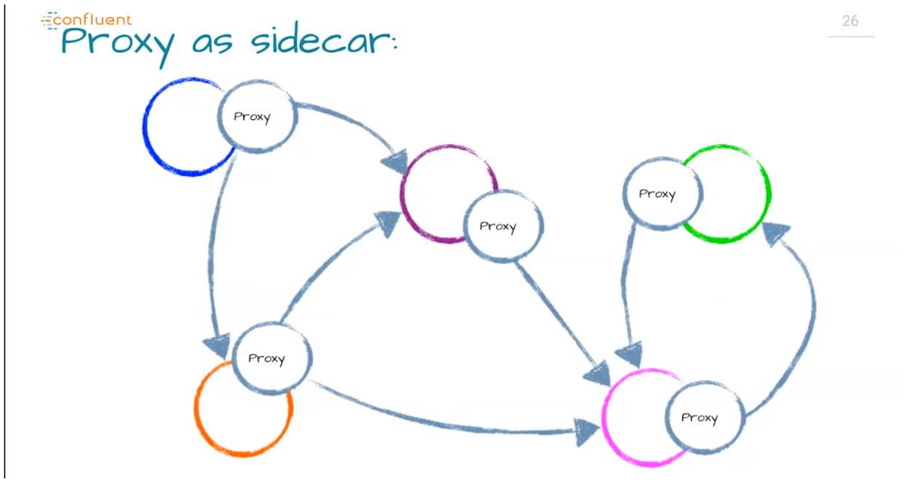
North-South Traffic

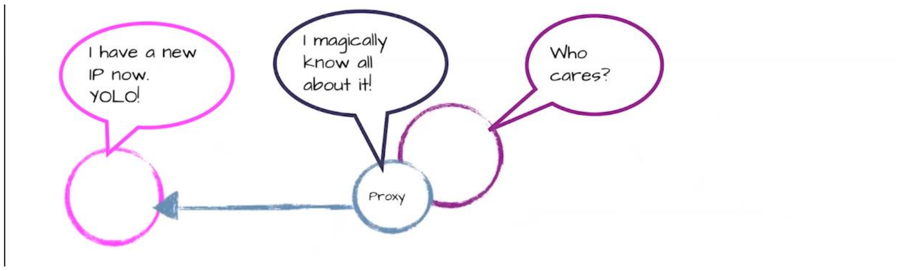
We want to take this north-south traffic and turn it into an east-west traffic using the Service Mesh



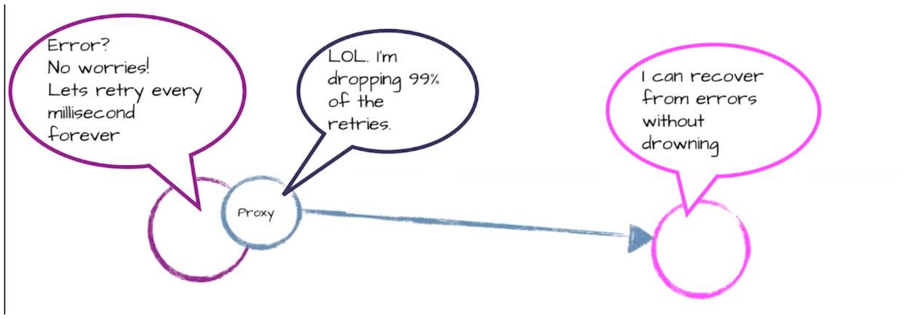
East-West Traffic

The side-cars is ran next to every microservice in a separate container to implement the distributed API Gateway



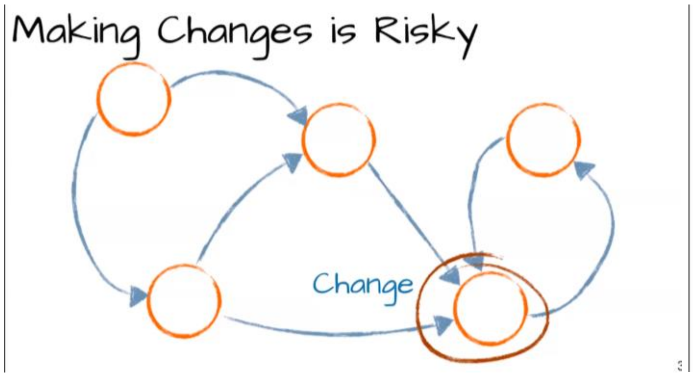


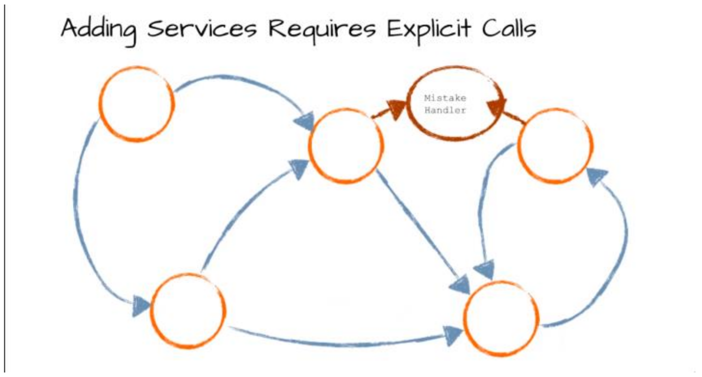
The sidecar automatically knows about the new IP via the Service Mesh's discovery/control plane/layer called Istio, the microservice does not care at all.


Error? No worries! Lets retry every millisecond forever
LOL. I'm dropping 99% of the retries.
I can recover from errors without drowning
Proxy

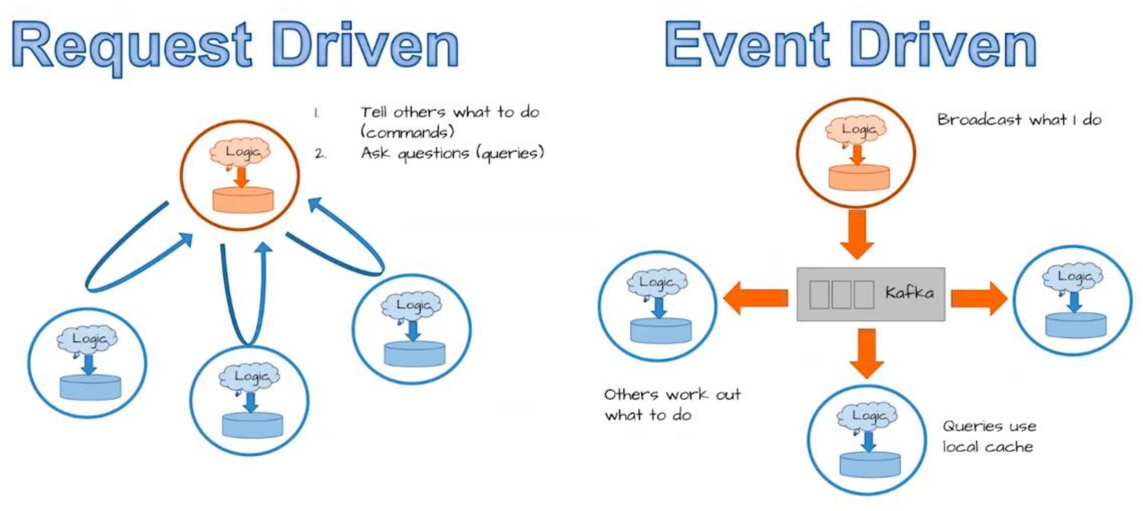
The sidecar has the logic to limit retries automatically using the configured setting from the control plane Istio

Event Driven





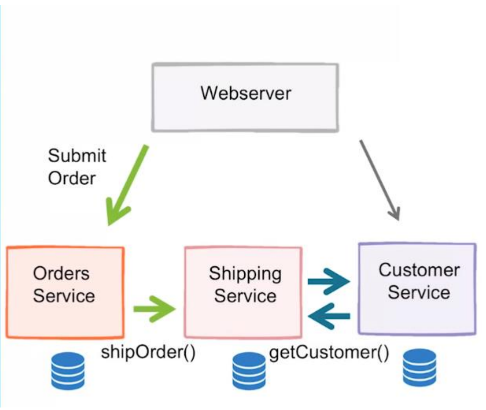
We can solve the 2 problems above by using the event-driven pattern to just tell what it known (events) below


Request Driven
1. Tell others what to do (commands)
2. Ask questions (queries)
Logic
Event Driven
Broadcast what I do
Kafka
Others work out what to do
Queries use local cache

Events are both facts and triggers

Buying an iPad (with REST)

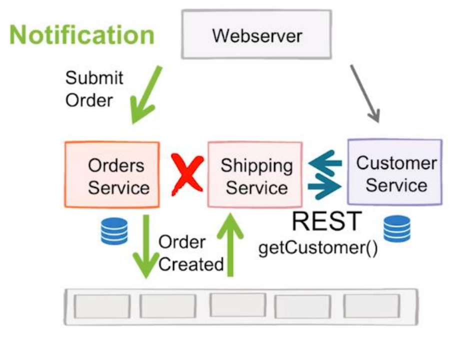
- Orders Service calls Shipping Service to tell it to ship item.
- Shipping service looks up address to ship to (from Customer Service)



What if the Shipping Service is down? Does the Orders Service have to keep retrying?

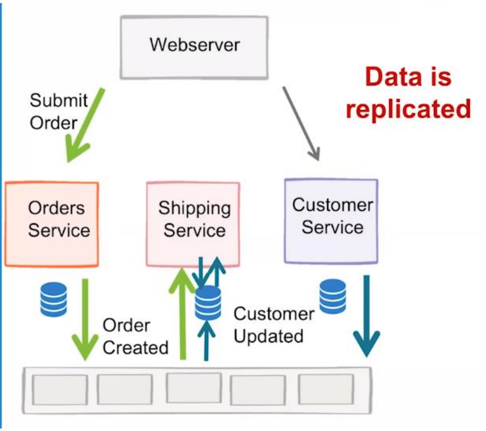
Using events for Notification

- Orders Service no longer knows about the Shipping service (or any other service). Events are fire and forget.



Using events to share facts

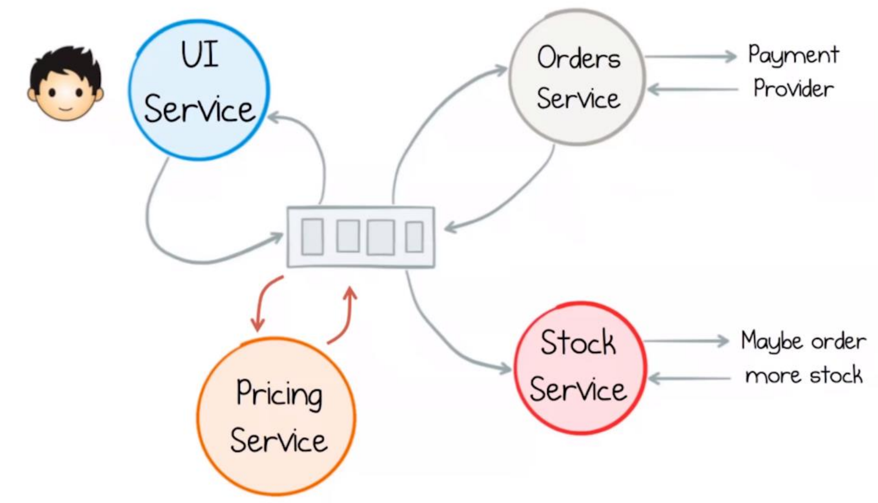
- Call to Customer service is gone.
- Instead data in replicated, as events, into the shipping service, where it is queried locally. .



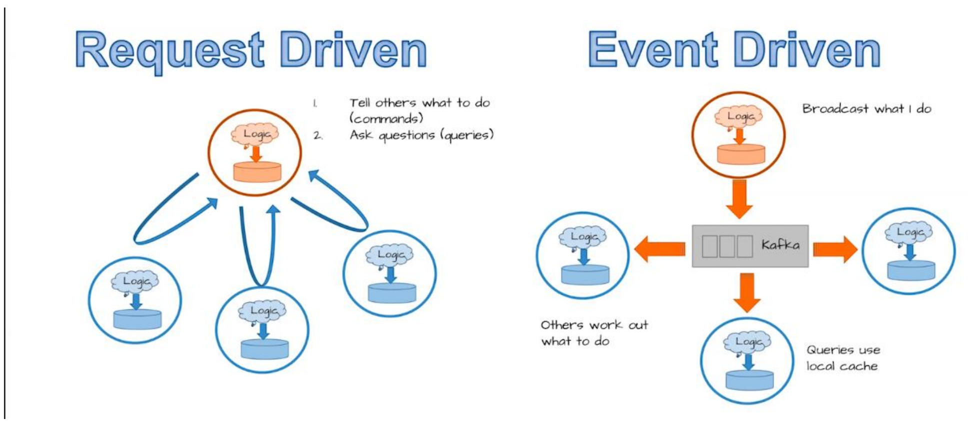
DB for Each Microservice?

- **It is safe:** They are all derived from same stream of events
- **Custom projection** just the data each service needs.
- Reduced dependencies
- Low latency

Event Driven Microservices are Stateful

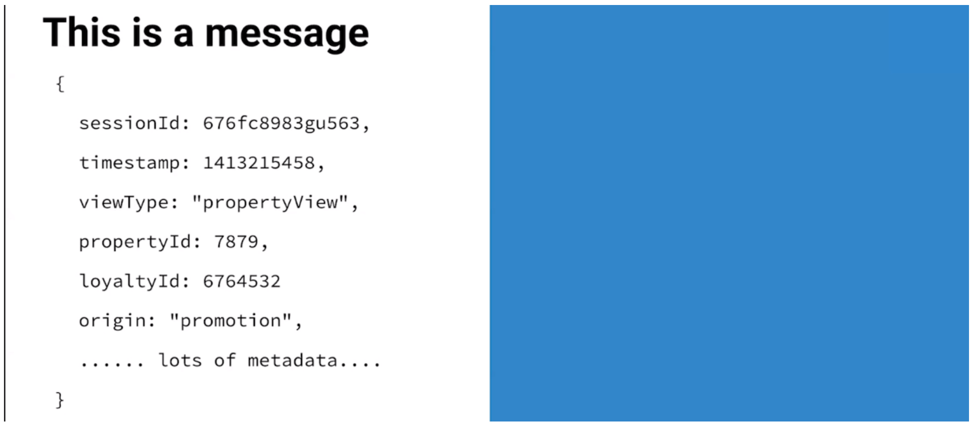


Schema

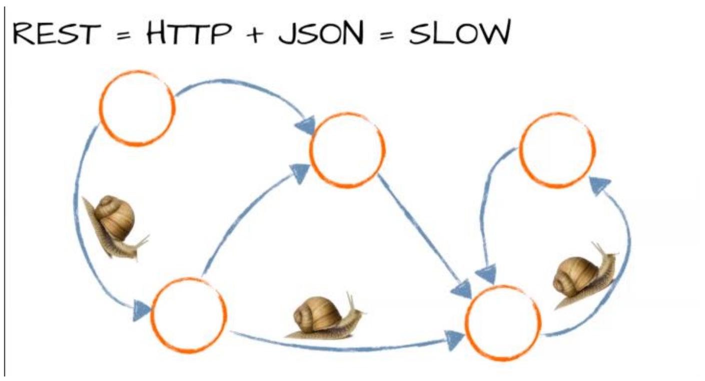

Request Driven
1. Tell others what to do (commands)
2. Ask questions (queries)
Logic
Event Driven
Broadcast what I do
Kafka
Others work out what to do
Queries use local cache

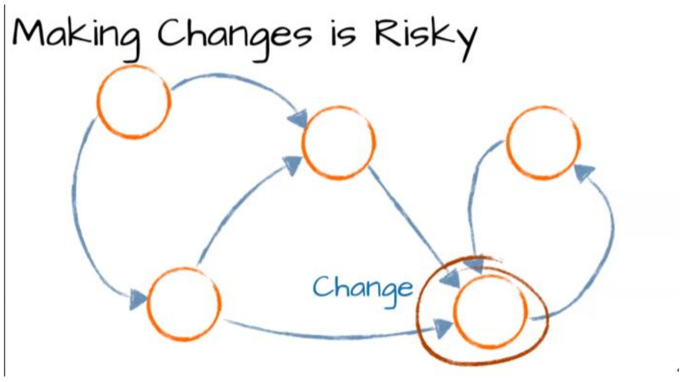
Event driven world runs on events with messages

The medium is not the message.


This is a message

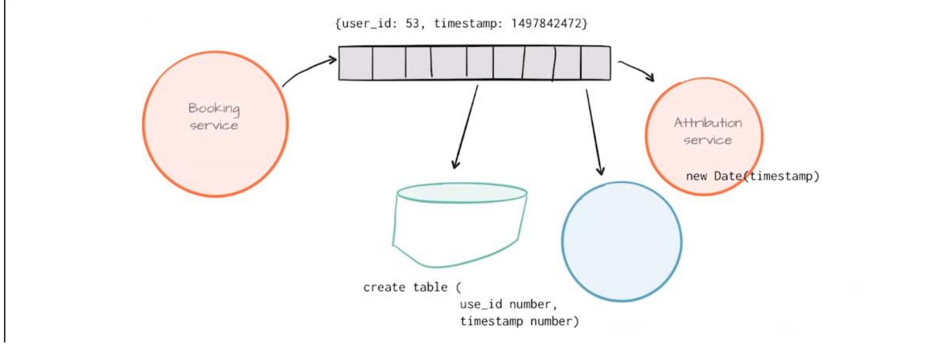
```
{
  sessionId: 676fc8983gu563,
  timestamp: 1413215458,
  viewType: "propertyView",
  propertyId: 7879,
  loyaltyId: 6764532
  origin: "promotion",
  ...... lots of metadata....
}
```

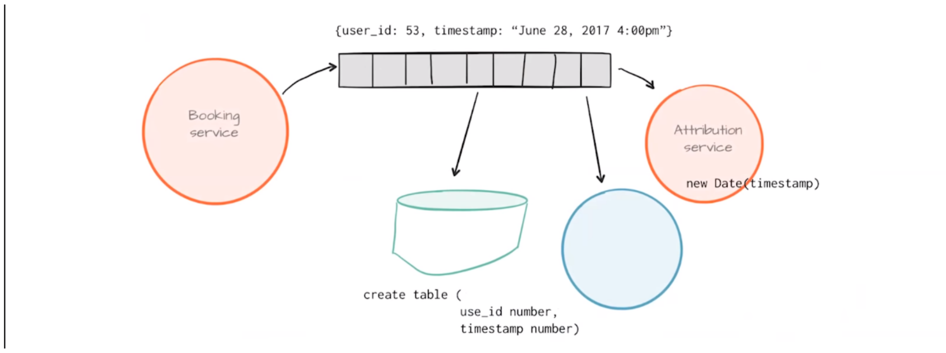




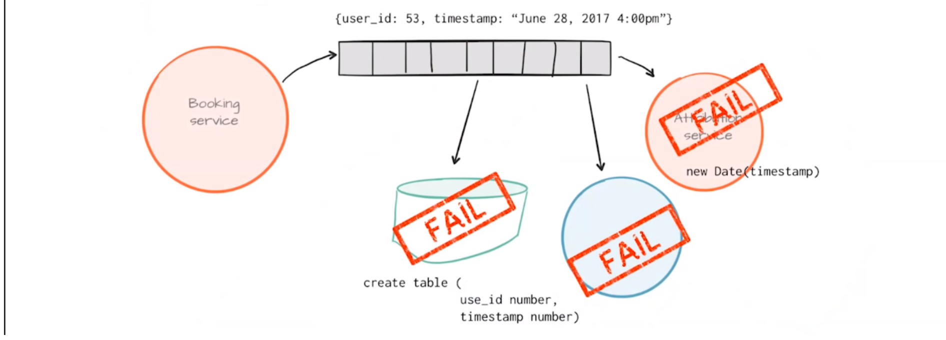
JSON is slow and has very little validation and events can be written with new, unvalidated data fields that can break our microservices. We need to have another data format for our messages or add JSON validations using schemas.

There are lots of dependencies

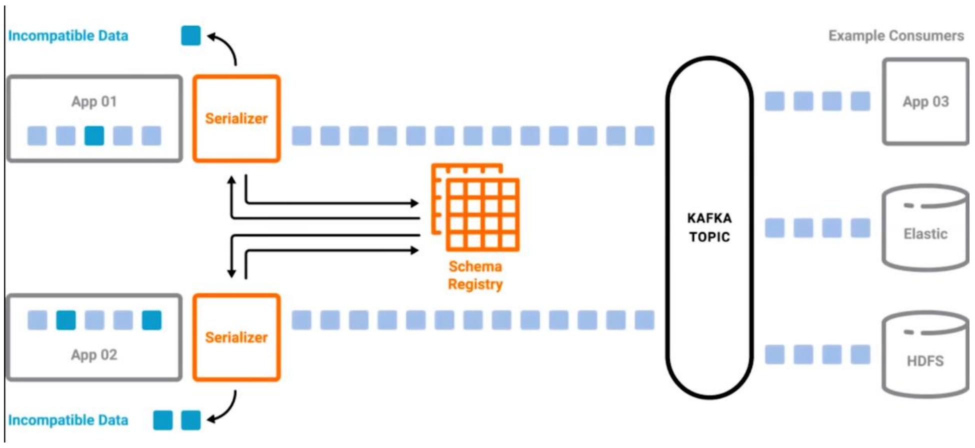




Moving fast and breaking things

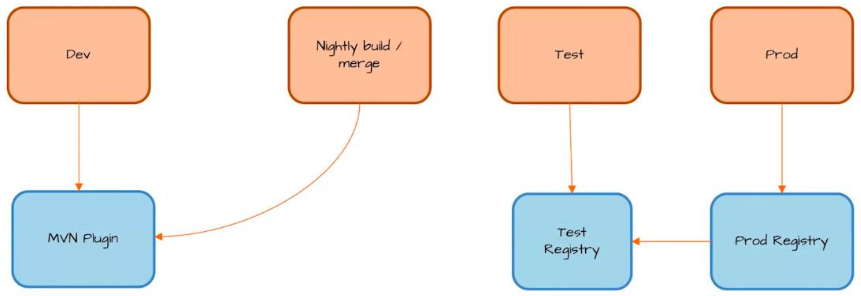


APIs between services are **Contracts**
In **Event Driven** World – **Event Schemas** ARE the API



The Confluent Schema Registry now supports writing schemas in Avro, JSON, and ProtoBuf.

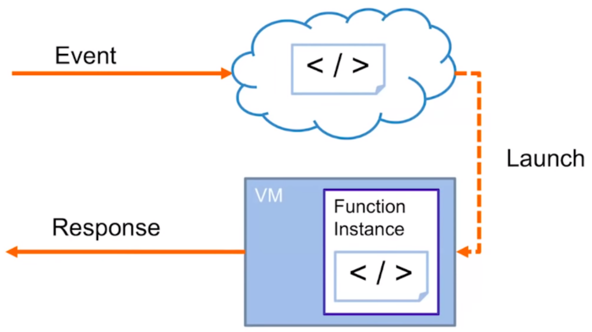
So the flow is…



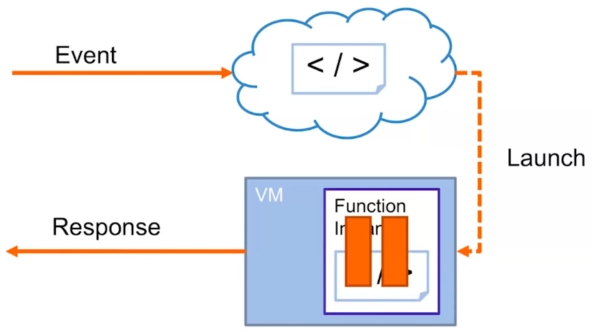
Every schema change can now be validated for compatibility before adding it to the schema registry

Serverless

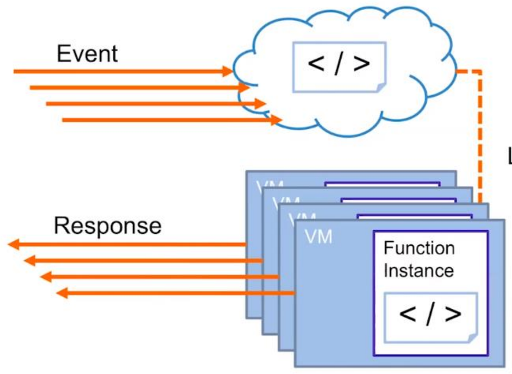
Function as a Service



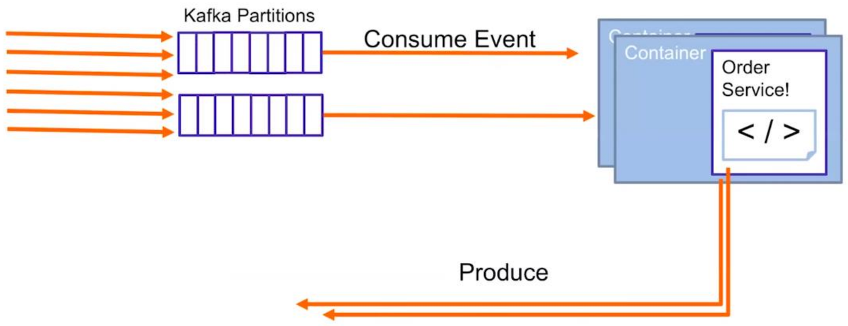
When nothing happens



At scale

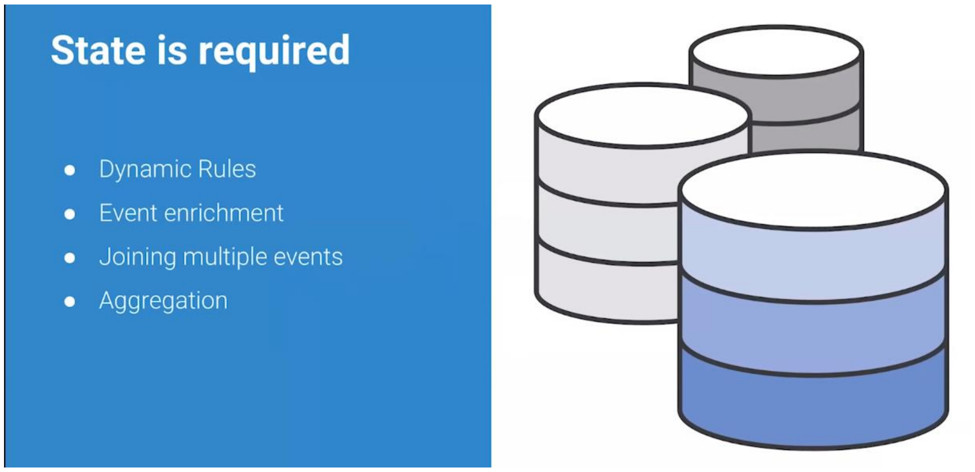


Wait, this is super familiar



Event-driven is all about events, the only missing thing is our state

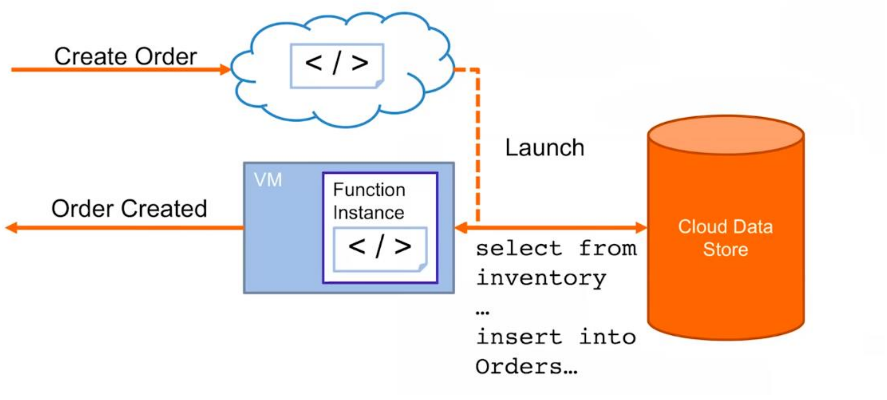
Up Next:
Stateful Serverless



State is required

- Dynamic Rules
- Event enrichment
- Joining multiple events
- Aggregation

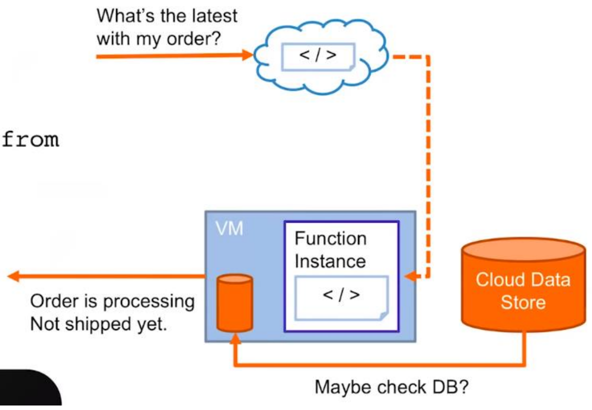
How You Probably Do State



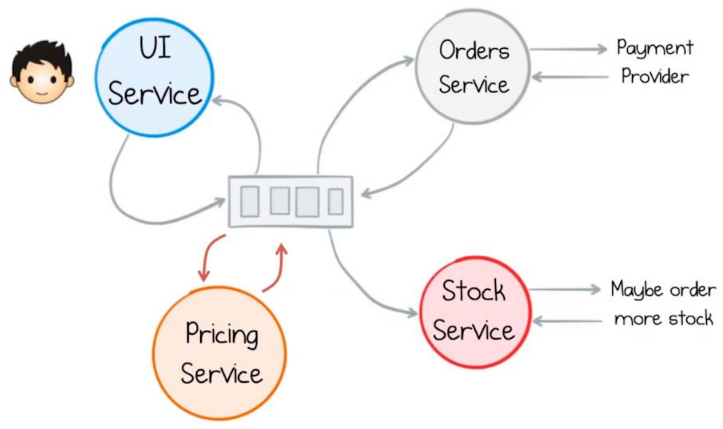
```
select from
inventory
…
insert into
Orders…
```

We can do a bit better

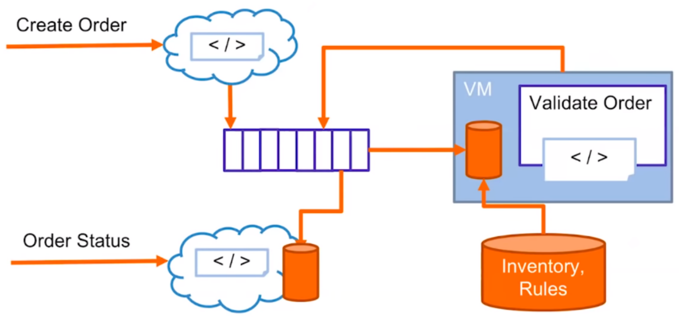
```
Select order_id,
customer_name, product,
quantity, price, state from
orders
where state != “CLOSED”
```



But I really want this back:



Stateful Serverless



What’s Still missing?

- Durable functions everywhere
- Triggers and data from data stores to functions
- Unified view of current state