EVENT DRIVEN MICROSERVICES WITH DAPR

Anthony Sneed
Chief Technology Architect
Hilti F&P Solutions

North Dallas Developer Group
October 5, 2022

ABOUT ME

Anthony Sneed
Chief Technology Architect
Hilti F&P Solutions, Plano Texas

Email: anthony.sneed@hilti.com
Twitter: @tonysneed
Blog: https://blog.tonysneed.com
GitHub: https://github.com/tonysneed
NuGet: https://www.nuget.org/profiles/tonysneed

Fun Facts

- Married with three children
  Ages 16, 13, 11
- Lived in Europe 8 years
  Semi-fluent in Slovak
- Originally from California
  Worked for Disney, DevelopMentor, Wintellect
  Developer since 1994
- Moved to Texas in 2008
  Working for Hilti since 2018
- Active in Open Source
  Extensions for EF Core, Event Driven .NET
- Blog Visits: 1.4 million
  NuGet Downloads: 2 million

AGENDA

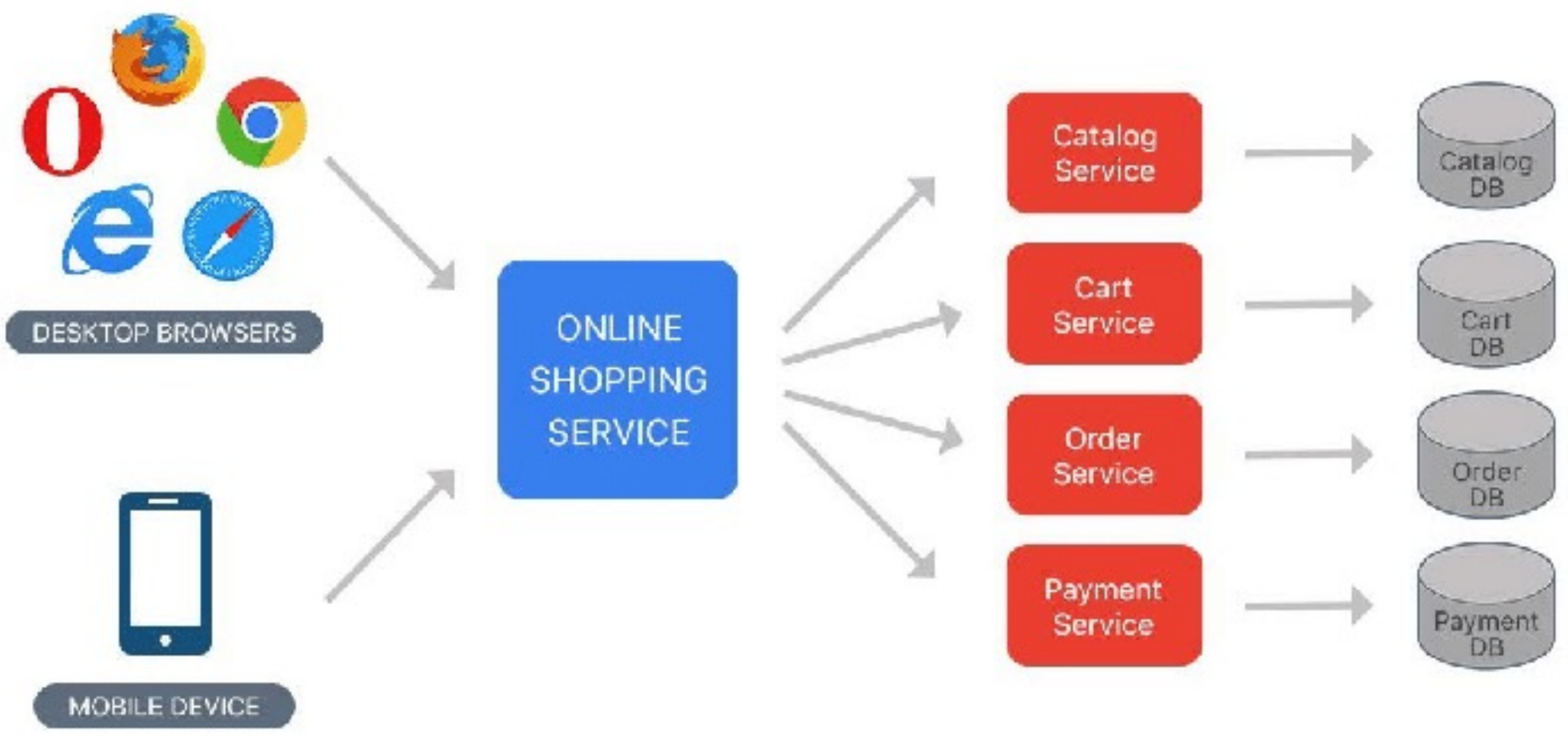
1. **Promise** of Microservices
2. **Peril** of Microservices
3. **Events** to the Rescue!
4. **Dapr -** Distributed App Runtime
   - https://dapr.io
5. **Event Driven .NET**
   - https://github.com/event-driven-dotnet

THE PROMISE OF MICROSERVICES

WHAT ARE MICROSERVICES?

Microservices is an architectural style that structures an application as a collection of services that are:*

- **Loosely coupled** and **independently deployable**
- Organized around **business capabilities**
- Highly **maintainable** and **testable**
- Owned by a **small team**

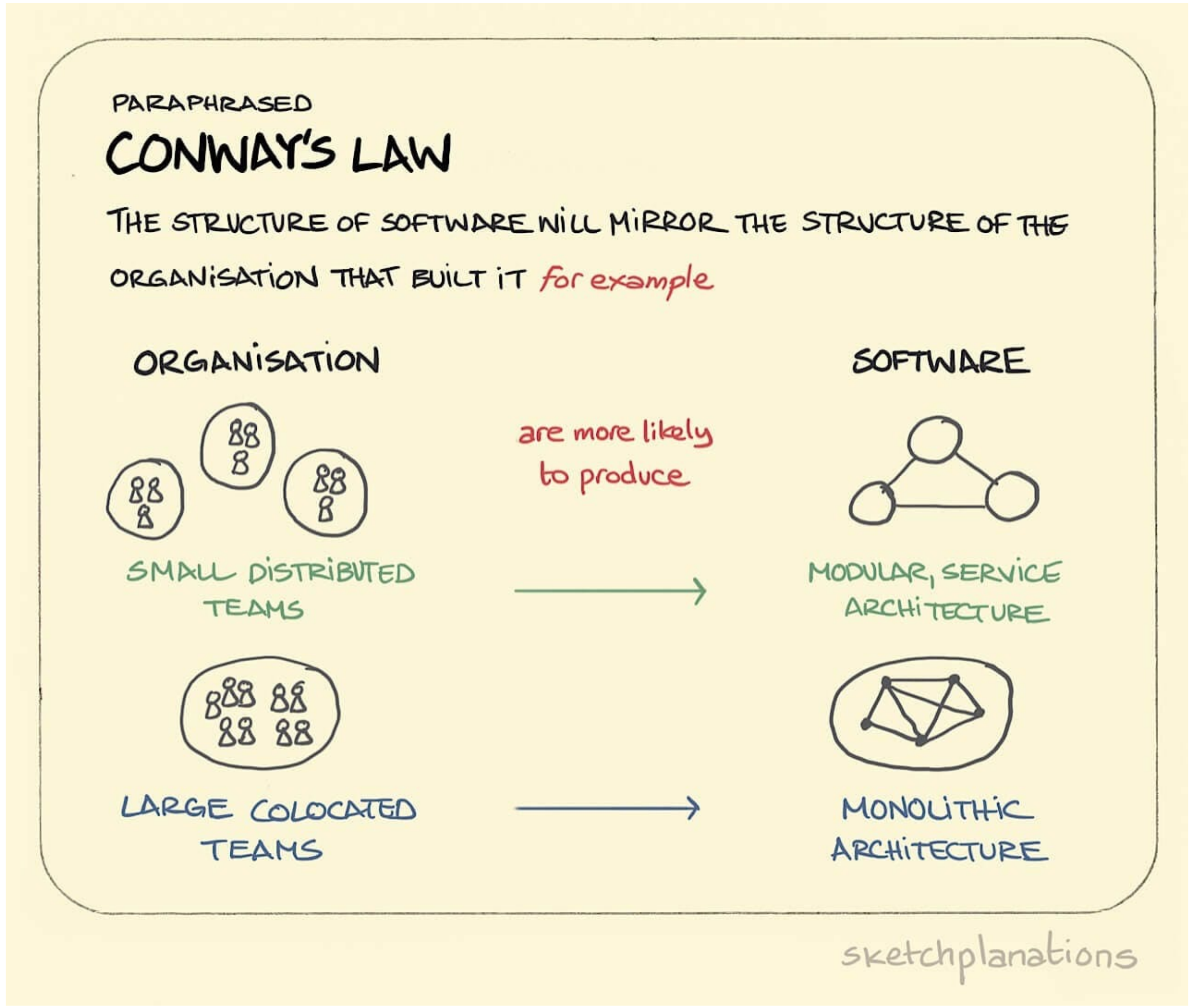


* https://microservices.io/

MICROSERVICE BENEFITS: SMALL TEAMS

Conway's Law

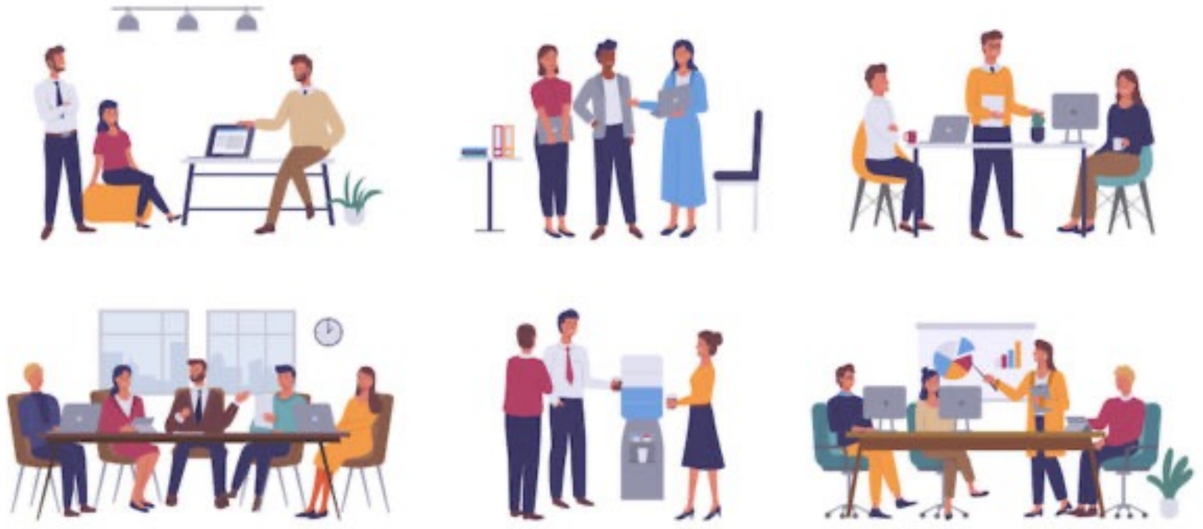
Any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure.


PARAPHRASED
CONWAY'S LAW
THE STRUCTURE OF SOFTWARE WILL MIRROR THE STRUCTURE OF THE ORGANISATION THAT BUILT IT for example
ORGANISATION
SOFTWARE
are more likely to produce
SMALL DISTRIBUTED TEAMS
MODULAR, SERVICE ARCHITECTURE
LARGE COLOCATED TEAMS
MONOLITHIC ARCHITECTURE
sketchplanations

MICROSERVICE BENEFITS: PARALLEL DEVELOPMENT

Parallel Team Development

- Different services **owned** by *different teams*
- Reduced **dependencies** among teams
- Different teams can develop in **parallel**
- Number of development teams can **scale** as the size of an application expands

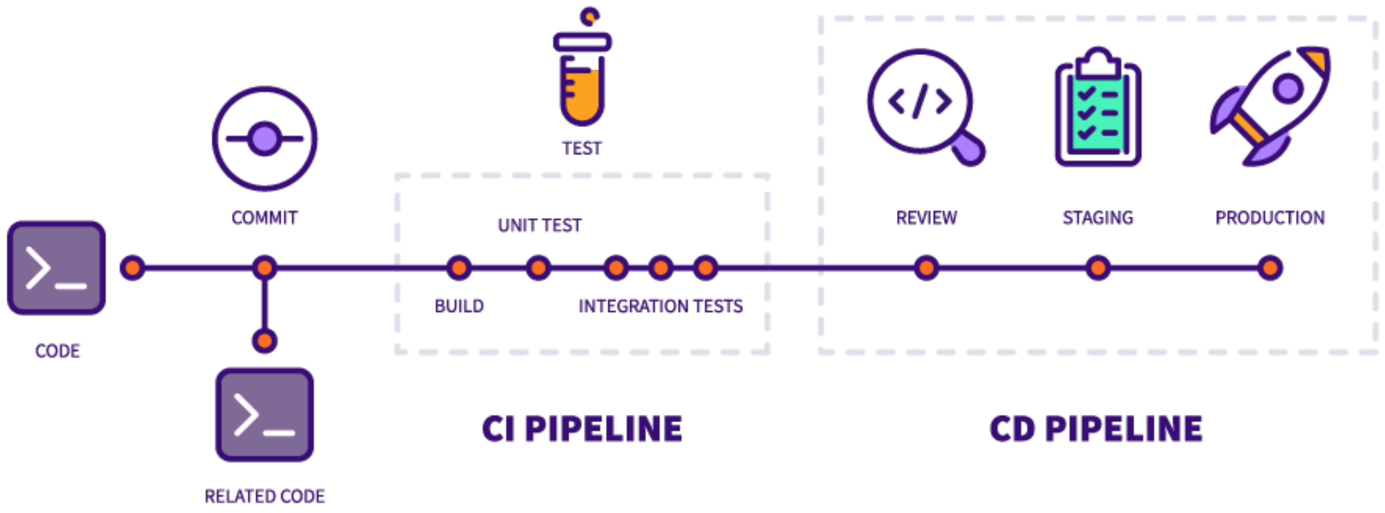


MICROSERVICE BENEFITS: DEPLOYMENT

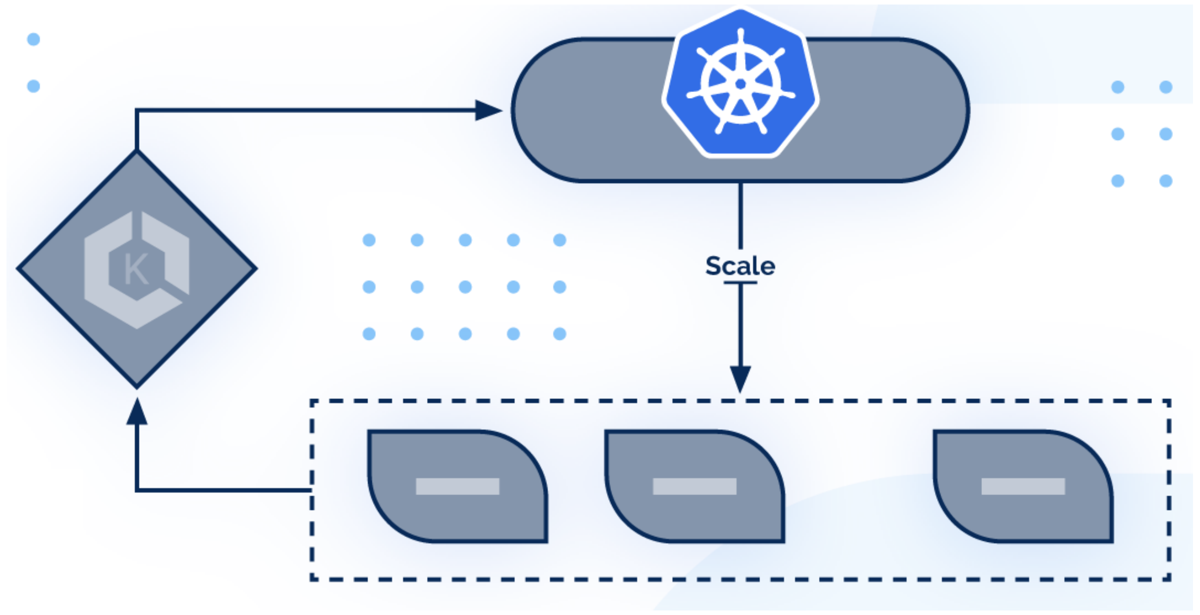
Independent Service Deployments

- No need to deploy the **entire** application all at once
- Each service can be deployed **independently**

- Services are **versioned** to avoid breaking clients
- Release **cadence** can increase
- Services can independently **scale**



MICROSERVICE BENEFITS: SCALING



Independent Service Scaling

- No need to scale entire application
- Each service can be **scaled independently**

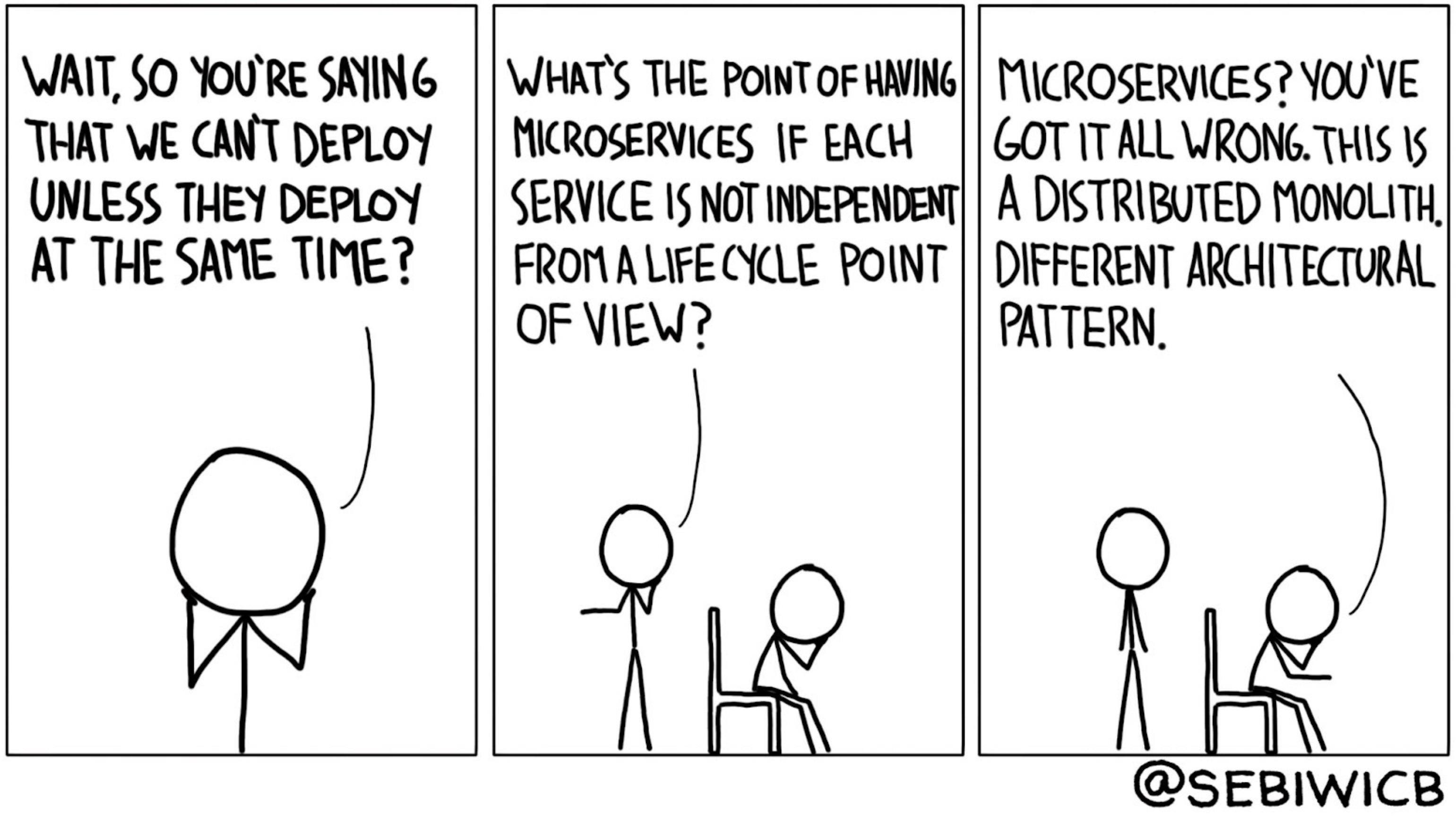
- Leverage EKS auto scaling:
  - pods increase with peak demand
  - pods decrease with reduced demand
- More **cost effective** than static scaling

THE PERIL OF MICROSERVICES

MICROSERVICE DANGERS

Distributed Monolith

Same problems as a monolith, but none of the benefits.


WAIT, SO YOU'RE SAYING THAT WE CAN'T DEPLOY UNLESS THEY DEPLOY AT THE SAME TIME?
WHAT'S THE POINT OF HAVING MICROSERVICES IF EACH SERVICE IS NOT INDEPENDENT FROM A LIFE CYCLE POINT OF VIEW?
MICROSERVICES? YOU'VE GOT IT ALL WRONG. THIS IS A DISTRIBUTED MONOLITH. DIFFERENT ARCHITECTURAL PATTERN.
@SEBIWICB

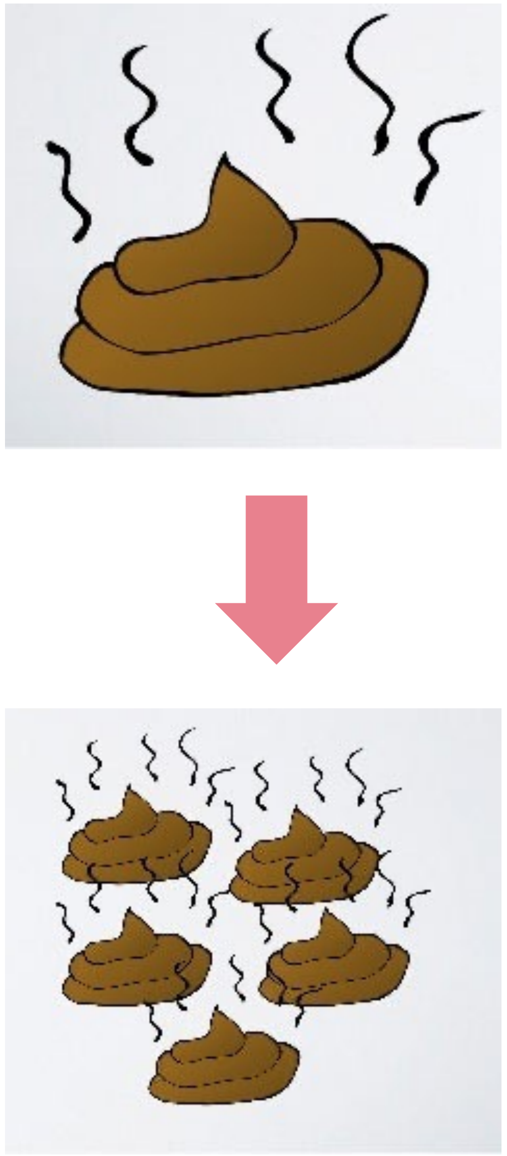
ARCHITECTURE SMELLS

Improperly Scoped Services

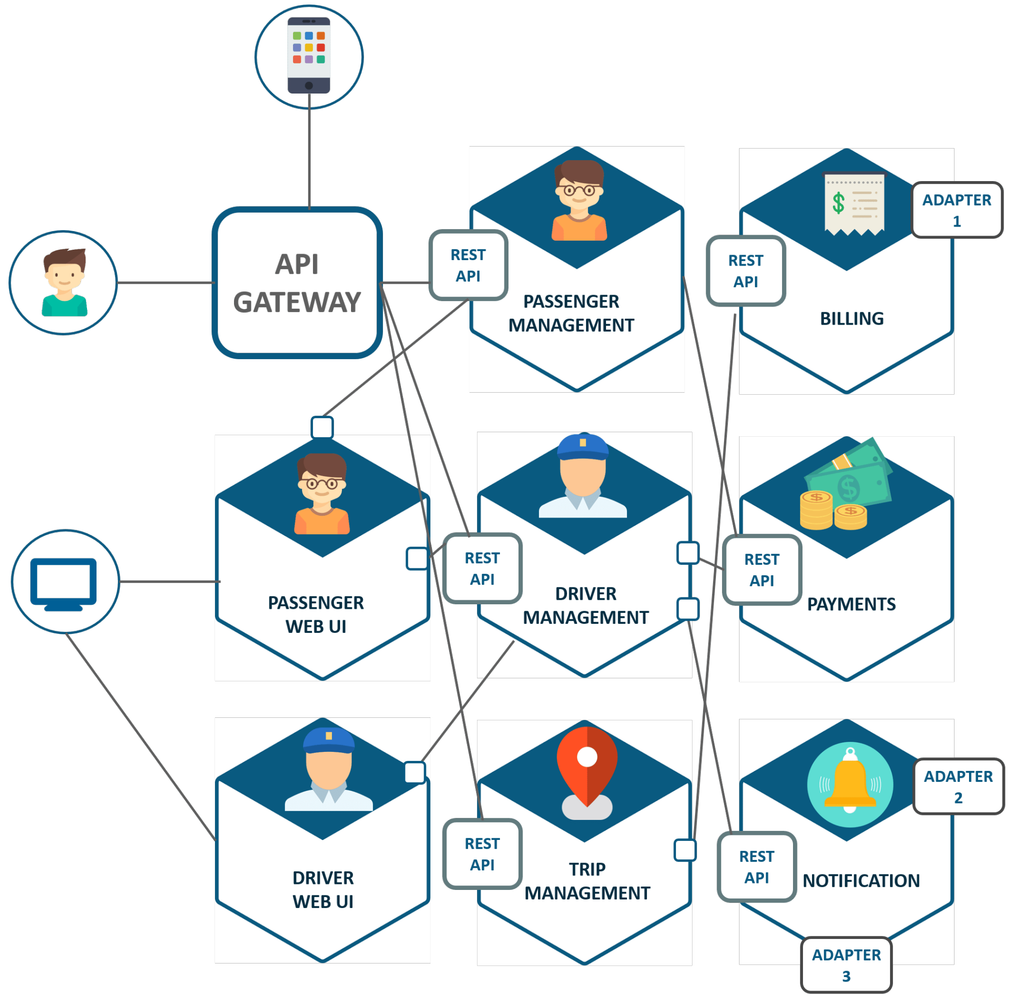
- Too granular or not granular enough
- Services cross **bounded contexts**
- Services incorporate multiple aggregate roots

Sharing a Database Among Services

- Services coupled at the **data layer**
- **Schema changes** propagate to multiple services



ARCHITECTURE SMELLS



Service-Level Coupling

- Service-to-service calls over HTTP / REST
- Both service **must be up** at the same time
- Retry logic *replicated* among services
- Serialization slows **performance**

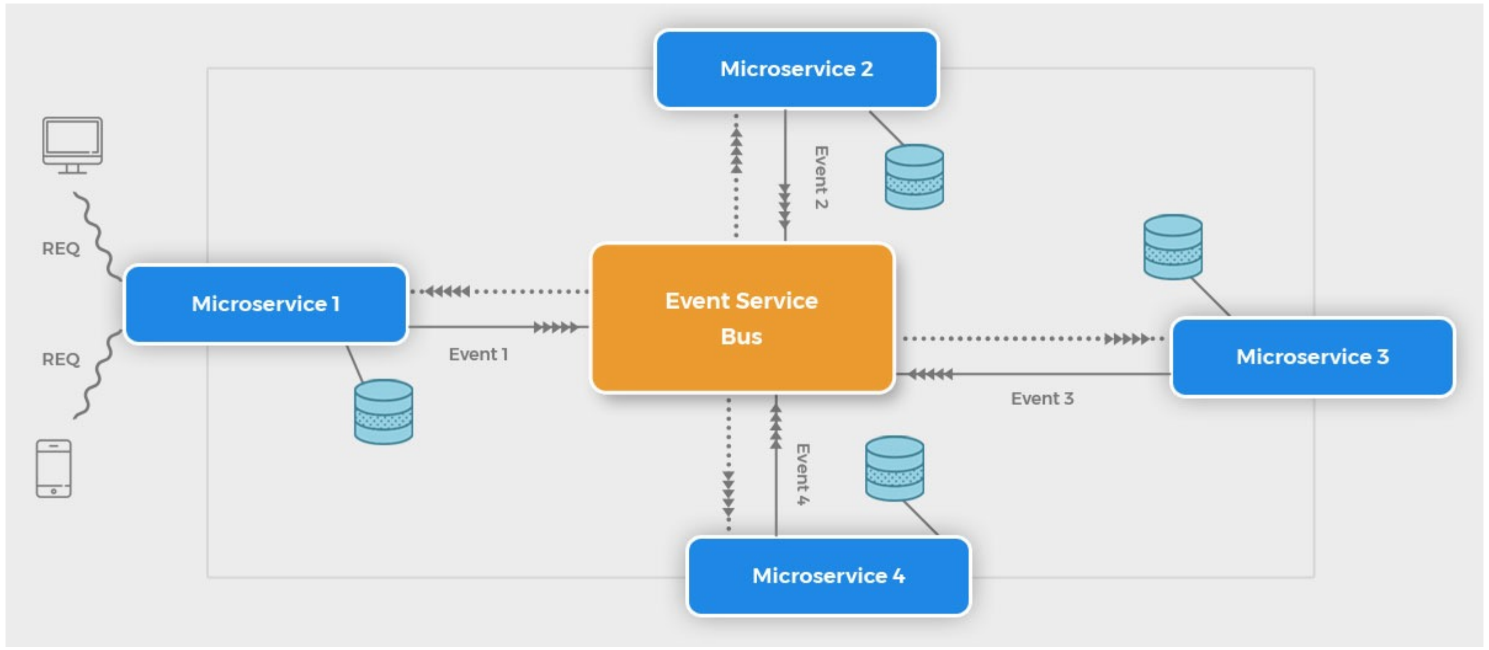
Schema-Level Coupling

- Schema changes can **break** message consumers
- Schemas as not **validated** for backward compatibility
- Schemas are not **versioned**

EVENTS TO THE RESCUE!

EVENT-DRIVEN MICROSERVICES

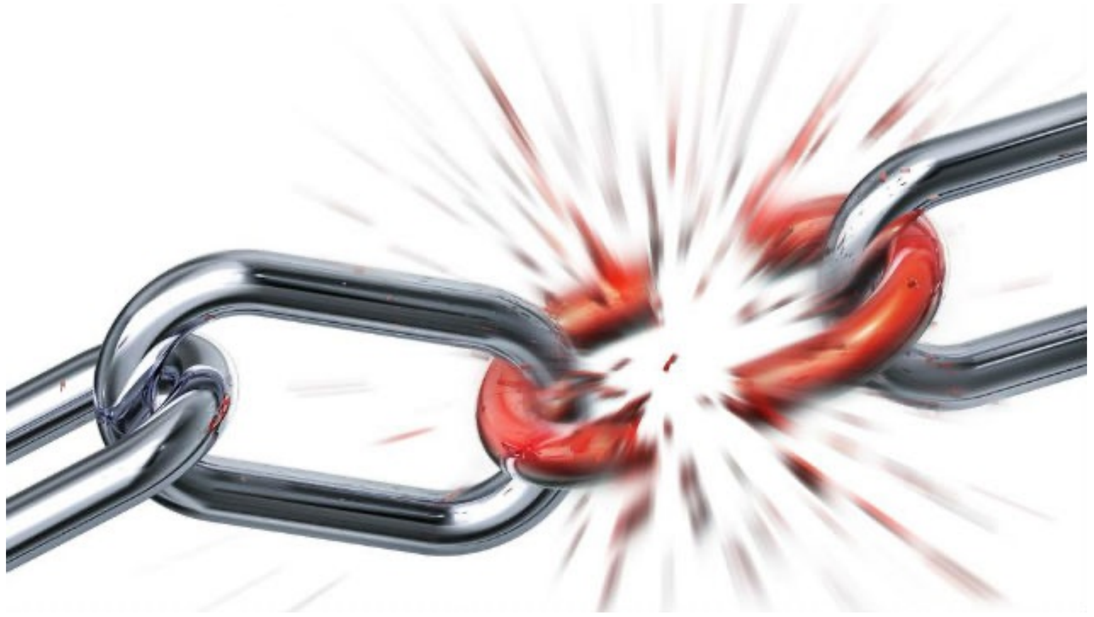
- Services **publish events** in response to changes in state
- Each service maintains its own **private data store**
- Queues can **buffer events** so that services don't have to be online at the same time
- Subscribers can **respond** to events by updating their own data store



MICROSERVICE DECOUPLING

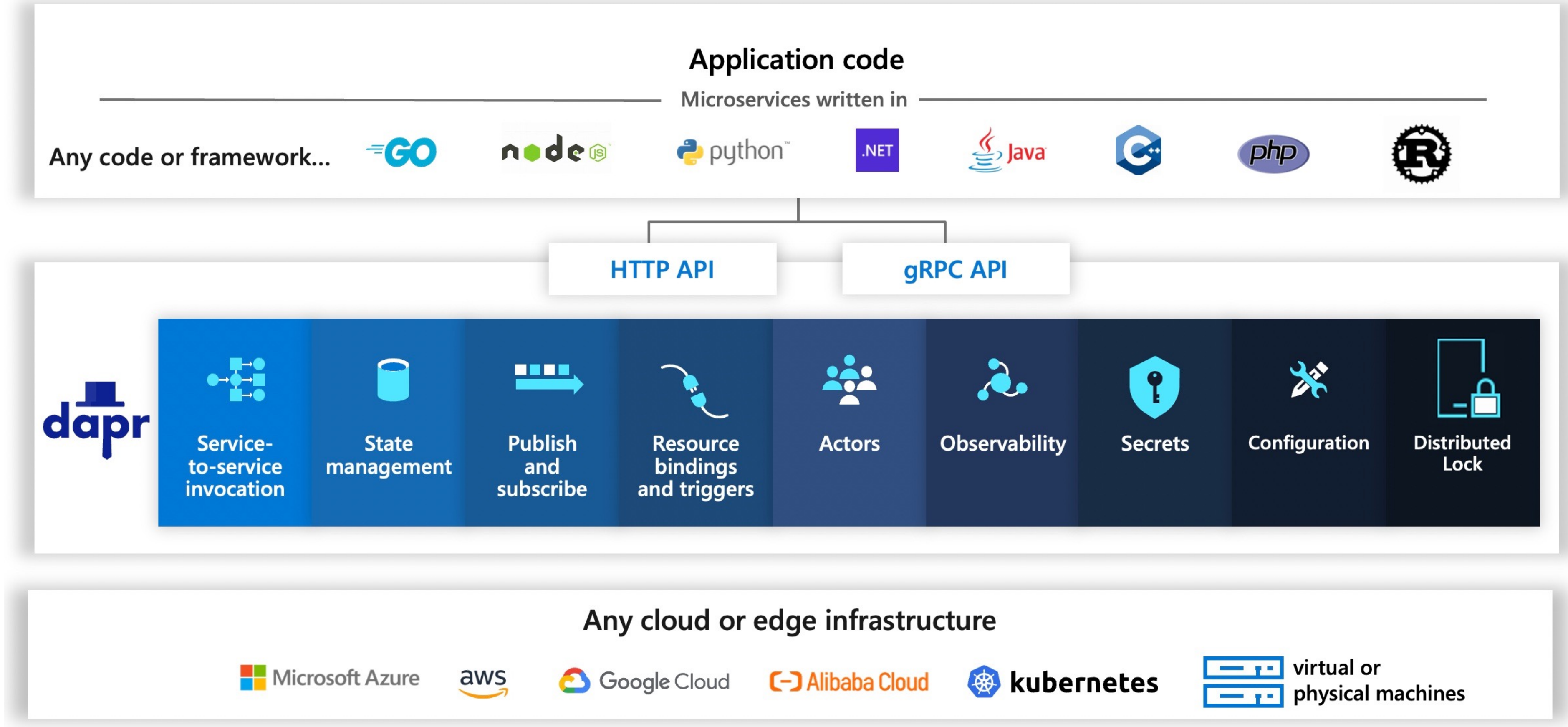
Decoupling Microservices with Events

- Services do not have to be **available simultaneously**
- Reduces need for **retry logic**
- Service scaling tied to queue length
- Events can be *stored and re-played*
- Event streams can be processed for **real-time analytics**



DAPR: DISTRIBUTED APPLICATION RUNTIME

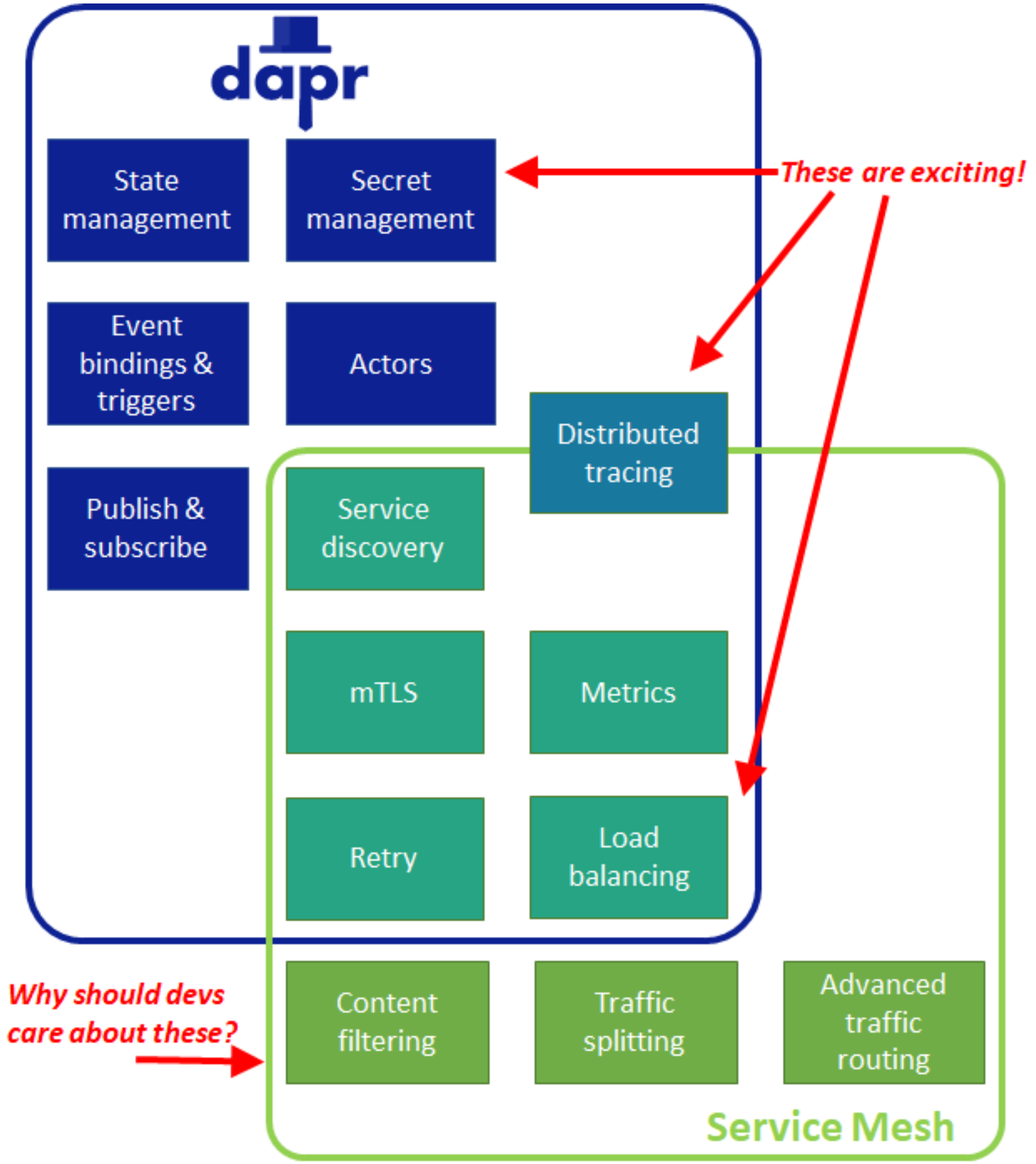
DAPR ARCHITECTURE



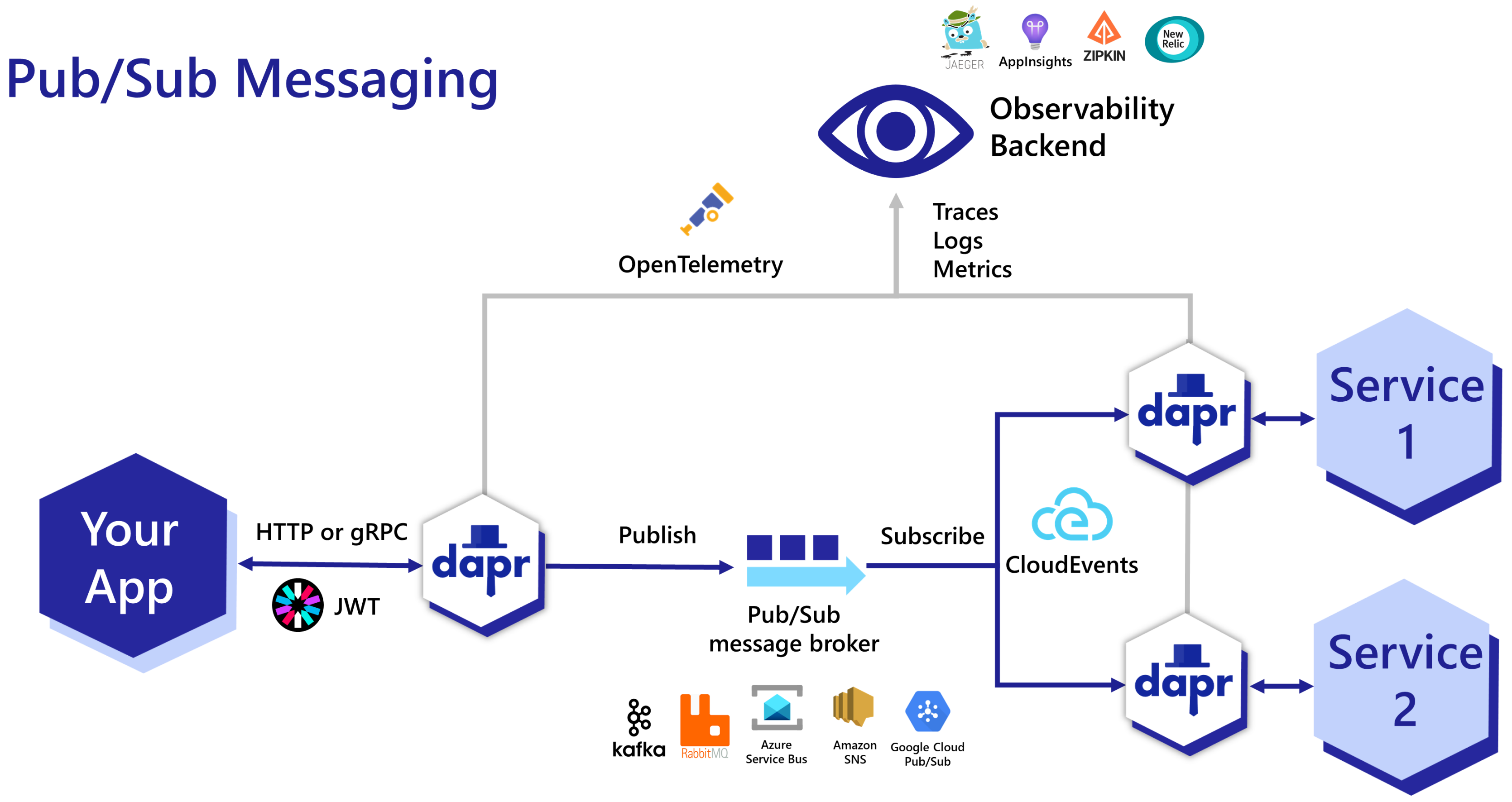
DAPR: SERVICE MESH

Dapr: Application-Level Service Mesh

- Language agnostic **side-car** pattern
- Some **service mesh** capabilities, but at the application level
- Can plug in **Istio** for additional capabilities
- Includes *service discovery, transport security*
- Configurable auto-retry function
- Supports distributed **observability** with providers such as Zipkin or Data Dog



DAPR: PUB-SUB ABSTRACTION



INTRODUCING EVENT DRIVEN .NET

EVENT DRIVEN .NET

Layered Approach

- Teams can apply one layer at a time

Opt-In

- Select which layers to apply based on organization, process and technical maturity

Community Ownership

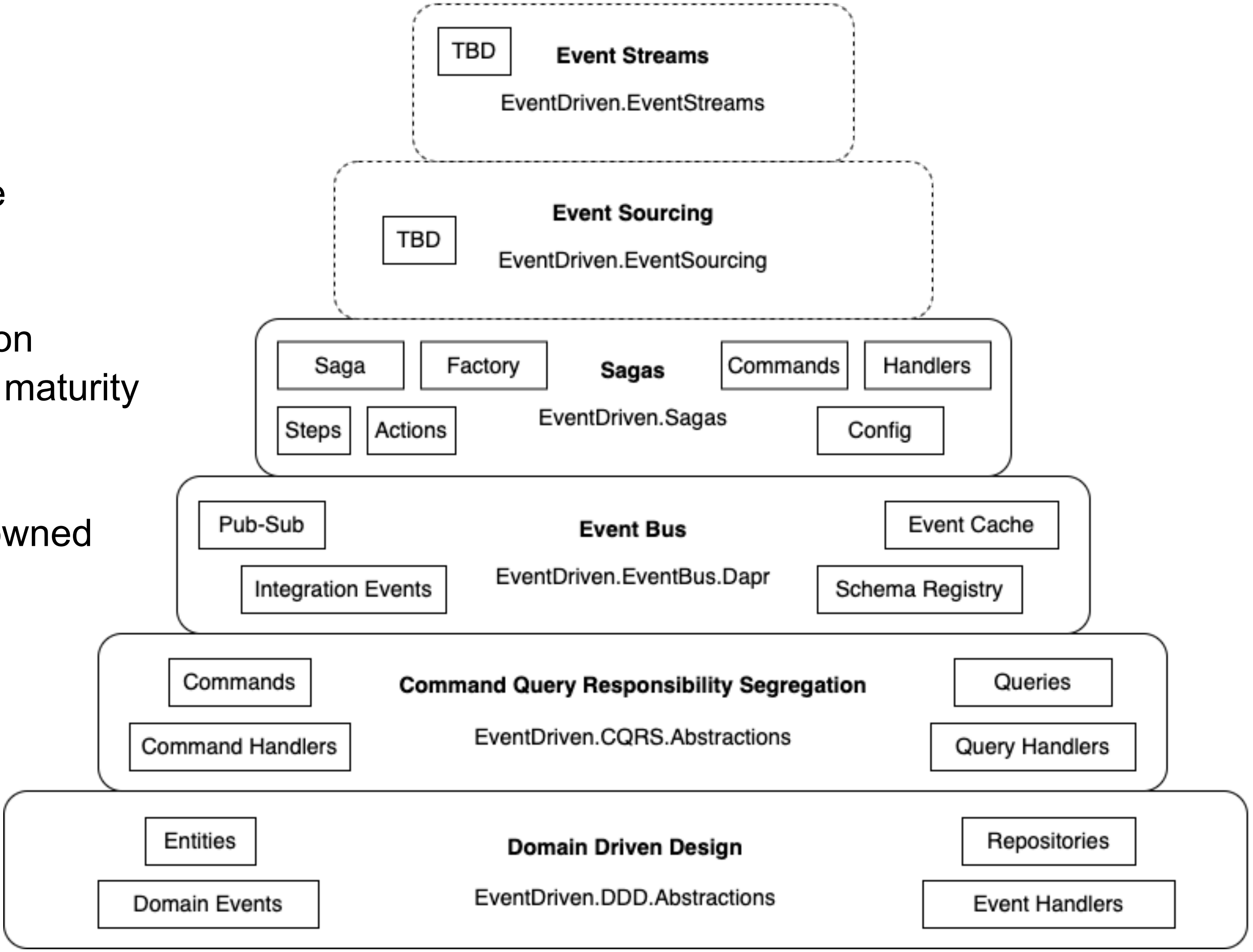
- General purpose elements can be owned by the community.

Not Overly Prescriptive

- Leave room for flexibility

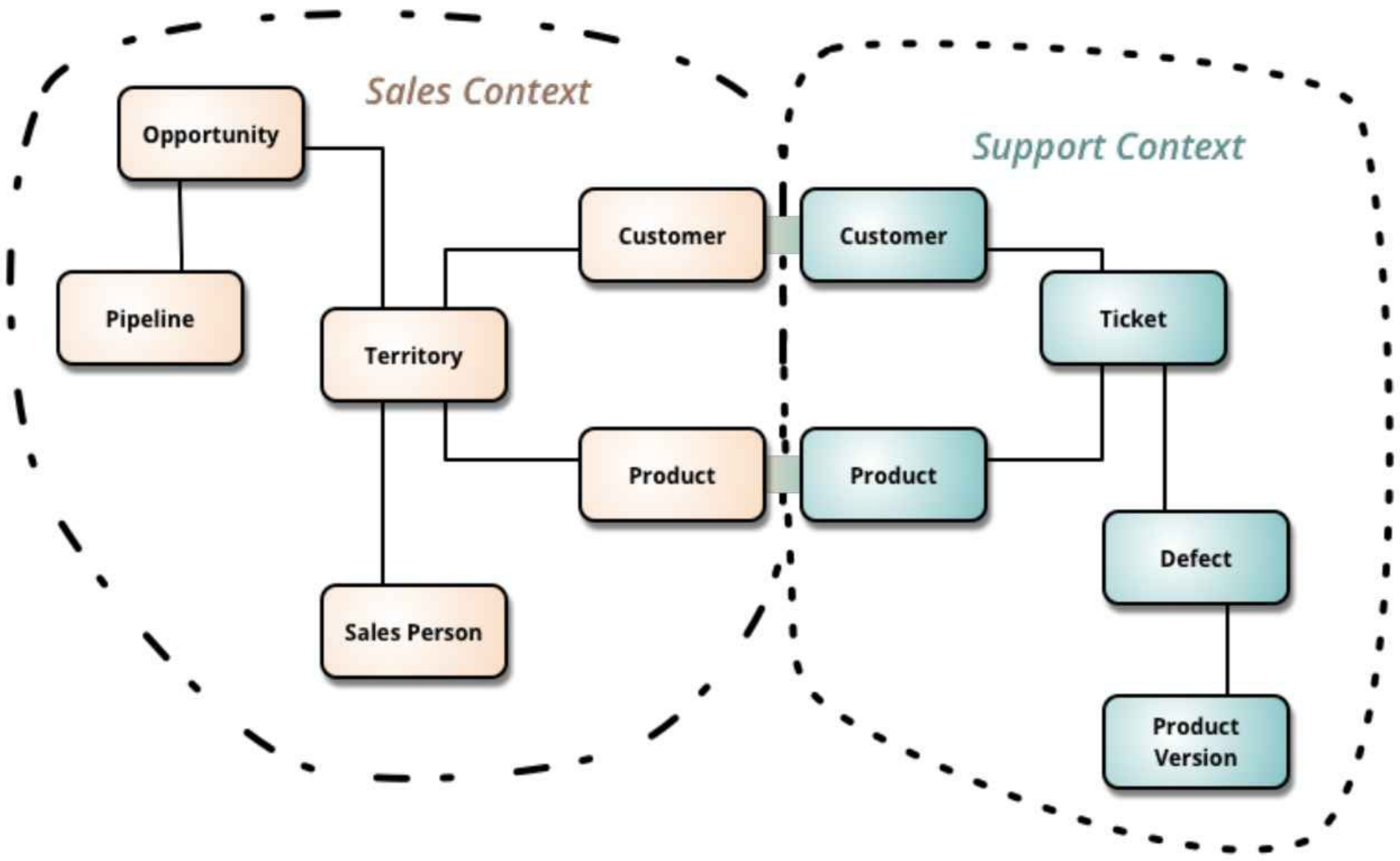
Examples

- There should be plenty of examples and use cases



DDD: SCOPING MICROSERVICES

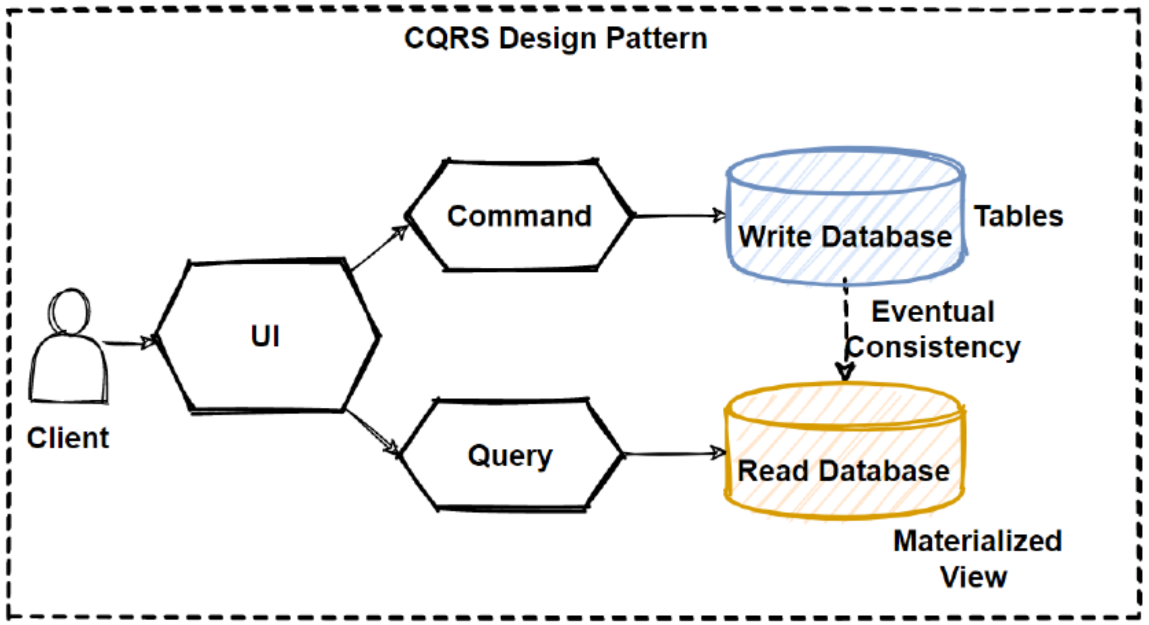
- Use DDD to define a business **domain model**.
- **Event storming** brings devs and domain experts together.
- Ubiquitous language within a **bounded context** helps communication.
- **Service per aggregate root** is a common pattern for scoping microservices.



DDD deals with large models by dividing them into different Bounded Contexts and being explicit about their interrelationships.

http://martinfowler.com/bliki/BoundedContext.html

CQRS: COMMAND QUERY RESPONSIBILITY SEGREGATION

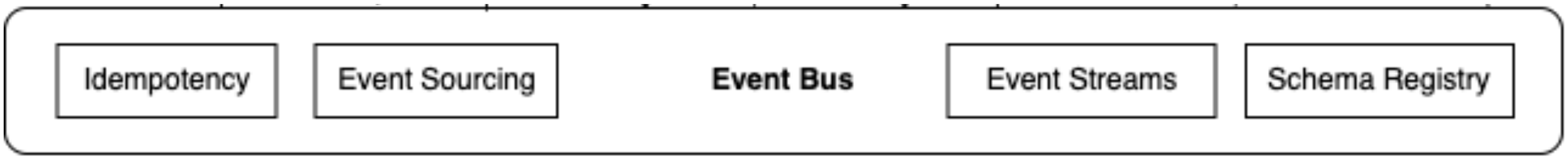


- CQRS **separates** command and query responsibilities.
- Read and write operations can exist in **separate services** and **separate databases**.
- Different **models** can be used which are optimized for read and write operations.
- Different **database technologies** can be used.
- A **message bus** can be used to replicate updates from write to read database.
- **Replication lag** can be mitigated on UI side by updating model locally.
- **Sharing DB** among services may be necessary when large dataset are involved – but watch out for service **coupling**.

EVENT BUS: DAPR PUB-SUB ABSTRACTION

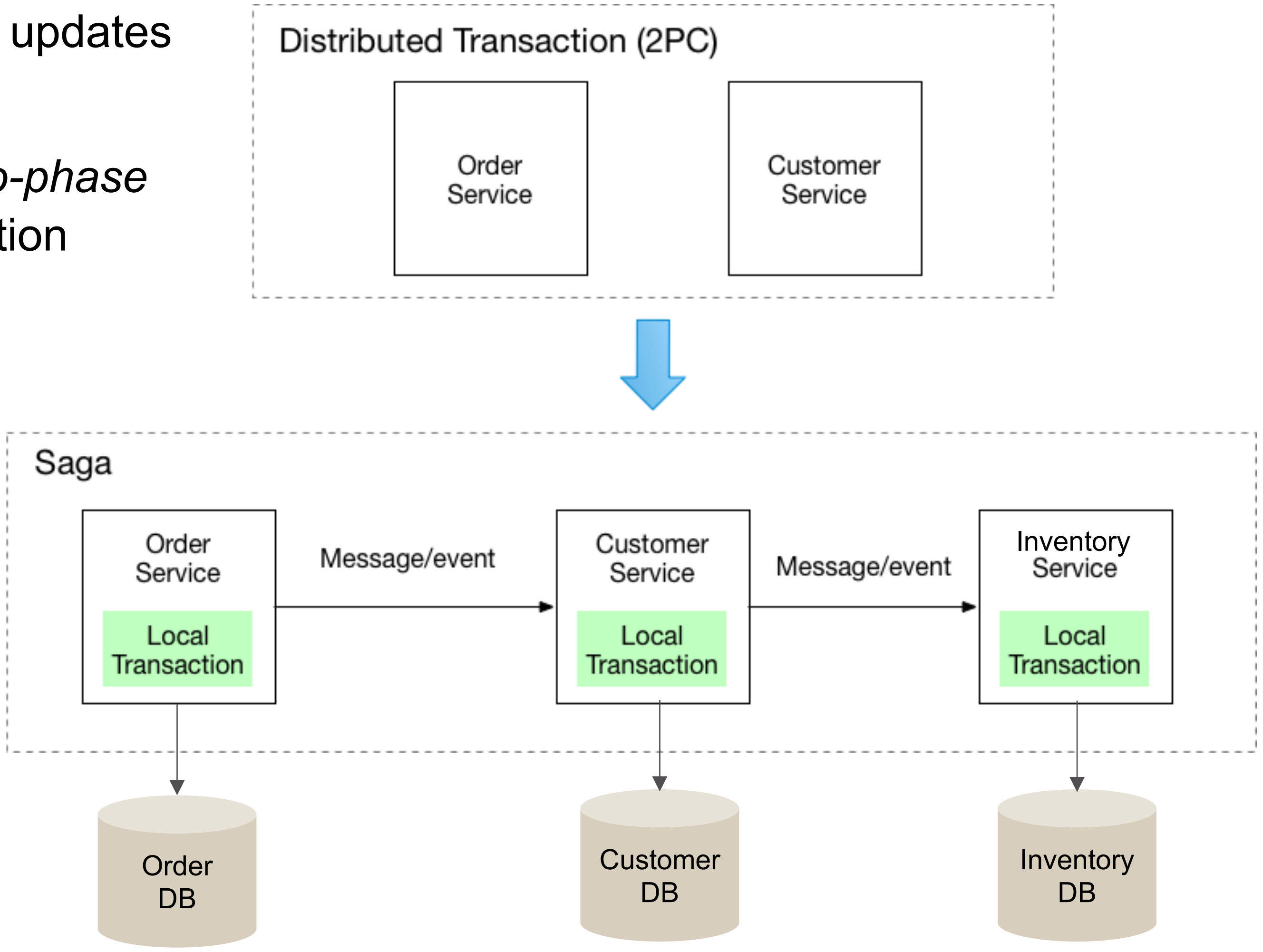
Dapr Event Bus with Event Cache and Schema Registry

- Includes **abstraction** over Dapr pub-sub
- Provides **registration** of producers and consumers
- Uses durable event cache for **idempotency** (ignoring duplicate messages)
- Supports **schema registry** at the event bus level (versus at the message broker level)
- **Event sourcing** can be added to treat persistence and publish operations atomically
- Other message brokers can be plugged in to support **event stream** processing



SAGAS: EVENTUALLY CONSISTENT TRANSACTIONS

- Sometimes you need atomic operations in which updates must **span multiple services**.
- Traditional distributed ACID transactions with *two-phase commit* are **not practical** in a distributed application architecture.
- Sagas provide a way for updates to roll back with **compensating** actions.
- With **choreography**-based sagas, services communicate directly with one another.
- With **orchestration**-based sagas, an orchestrator coordinates updates across services via a message bus by means of a state machine.

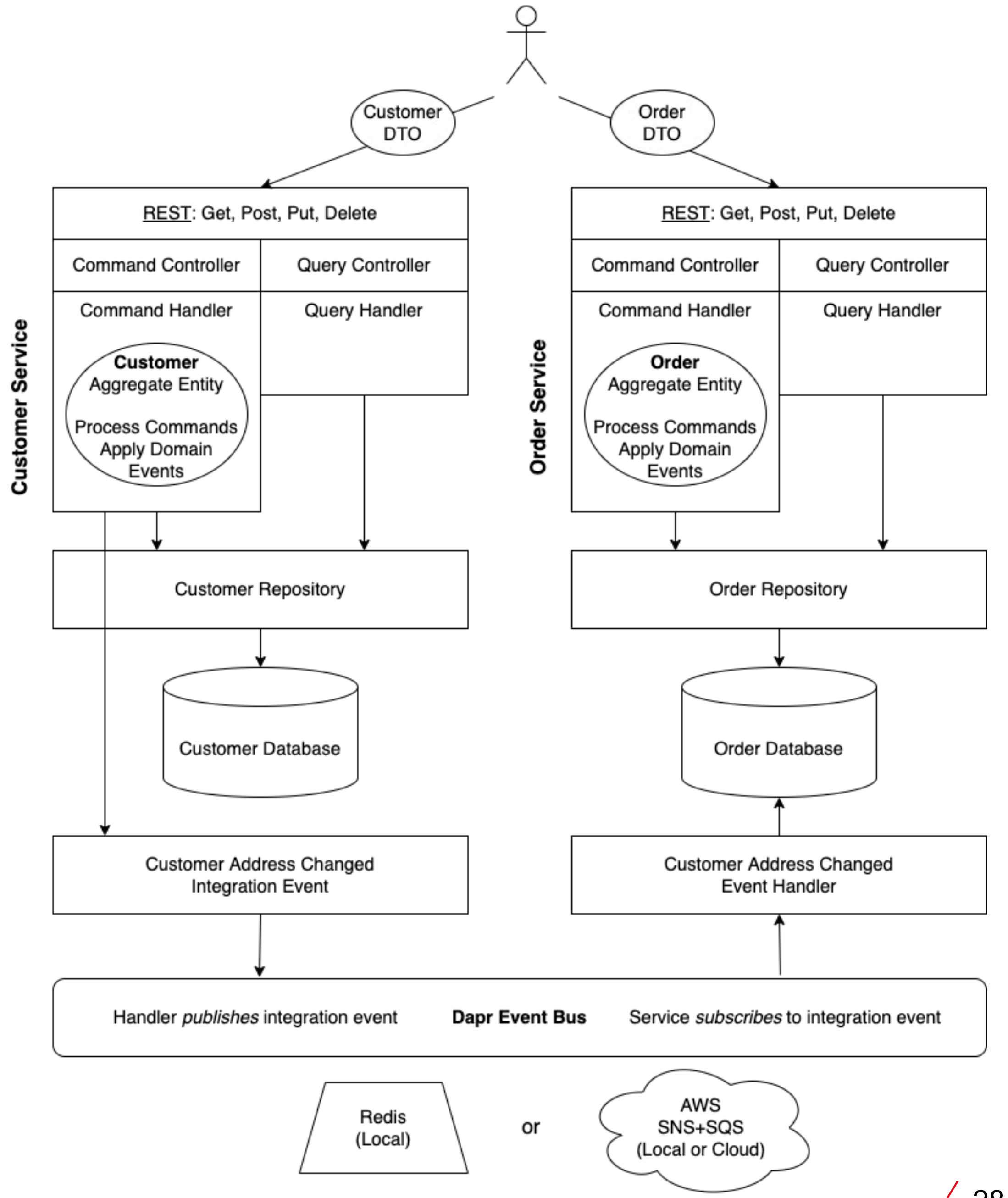


EXAMPLES

nsure Asset Uptime
Improve Processes
Other
Smart Mining
Drives
SIPAPER – Drive Train Monitoring
DCS
DCS Lifecycle Portal
IC Monitors and Advisors
Instrumentation
Smart Asset Management
Fleet Monitor
PAT in the Cloud
PlantSight Digital Reliability
Valves & Pumps
Anomaly Detection for ESP
Pump Monitoring
Asset Model
Valve Monitoring
Drives & Turbines
Dow Solution for Sinamics VFDs
Drive Train Analytics
Equipment Predictive Analytics
SIDRIVE IQ
Control Loops
PID
Control Performance Analytics
Process Event Operation
Anomaly Detection for Oil and Gas
Process Event Analytics
Optimization & Analytics
SiCue
Industry Specific
Disruptive Process Control for Pharma
eBR Review
SIPAPER – Drive Performance Analytics
Operation Intelligence
SIWA Sewer
Unmanned Pressure Control
SIWA Optim
XHQ
XHQ Operation Intelligence
Pumps
Smart Pumping
Water Distribution Pumping

REFERENCE ARCHITECTURE

https://github.com/event-driven-dotnet/EventDriven.ReferenceArchitecture



Customer DTO
Order DTO
Customer Service
REST: Get, Post, Put, Delete
Command Controller
Query Controller
Command Handler
Query Handler
Customer Aggregate Entity
Process Commands Apply Domain Events
Customer Repository
Customer Database
Customer Address Changed Integration Event
Order Service
REST: Get, Post, Put, Delete
Command Controller
Query Controller
Command Handler
Query Handler
Order Aggregate Entity
Process Commands Apply Domain Events
Order Repository
Order Database
Customer Address Changed Event Handler
Handler publishes integration event
Dapr Event Bus
Service subscribes to integration event
Redis (Local)
or
AWS SNS+SQS (Local or Cloud)