Event driven architecture patterns

- boiling it down

A helpful old pattern

- Lets talk about the observer pattern

We have to talk about Queues

- Though you can have event driven systems that don't use queues it is often implied as the mode of communication

So what is a queue?

- A First in First out data structure
- communicates via some network protocol
- AMQP is an example

What's Kafka? Is it a queue?

- Kafka uses the term "topic" to describe an event that gets persisted to disk
- This lets us add multiple consumers to a single topic
- We can also have things execute different logic based on the same event
- A kafka event will only get dispatched once to a each "consumer group"
- we can have multiple consumers in different "consumer groups" and they will each get a copy of the message

Simply Sending Messages

- low coupling
- can be hard to debug

Carrying state

- low latency
- copies of data passed around everywhere

CQRS

- When CRUD doesn't cut it
- Used for complex domains
- Great when there are lots of reads and a few writes

CQRS

Things that we gain

- complex domains are easier to tackle
- handling high performance applications
- when you need to scale reads and writes independently

What to watch out for

- this is complex
- it may slow down productivity
- it's a congnitive leap

When should I use it?

- Martin Fowler in a blog post recommends only using CQRS in a single "Bounded context"
- Use DDD to organize aggregates and work with product to create a "Ubiquitous Language"
- when you need more performance and CRUD is too complex

Event sourcing

- changes are recorded as events
- think version control
- Link

Things we gain

- A log of all the events
- Could also do this by just keeping an audit list
- But we architect things so its easy to add new features
    - do a complete rebuild
    - we can do temporal queries and search for states of applications at any given time
    - we can change and replay events that are wrong

Things to watch out for

- if events are fired off to external systems they don't know what is a "replay"
    - this can add lots of complexity

When should I use it?

- when we need an audit log for the system
- when you are looking for something that is very scalable
- new applications can be added easily and can tap into the event streams

Breaking down the monolith

- may want one or all of these patterns
- consider a strictly consistent system becoming eventually consistent
- have a conversation with your product and create an ubiquitous language with your system

Rethinking your system in events is easy!

- There are two options orchestration and choreography

  "In orchestration, we rely on a central brain to guide and drive the process, much like the conductor in an orchestra.
  With choreography, we inform each part of the system of its job, and let it work out the details, like dancers all
  finding their way and reacting to others around them in a ballet"
  - Building Microservices by Sam Newman

#Orchestration

- Often have to create specific endpoints to handle requests
- These endpoints are sometimes coupled to other systems
- Whose responsibility is it to orchestrate everything?

Choreography

- A single event can be dispatched
- Systems can listen to the events and do their own thing
- no need for complex orchestration
- Stack overflow

The pros

- super low latency
- high decoupling of the system
- adaptable
- independently deployable apps
- logging, monitoring and auditing
- easy to make async

The cons

- can be more complex
- can take longer to implement new features

Event based architectures move quickly

- Its depricated
- The patterns are moving fast
- Applications built recently are becoming out of date

Enter the Saga pattern

- Problem: the system is no longer acid complient
- A pattern to fix it

Organizing aggregates in DDD

- an aggregate is a cluster of objects that can be treated as a single unit
- an example is an order and its line items
- once we understand these aggregates we can use events to communication between them
- helps with SOLID principles and seperation of concerns

Setting up spring with kafka is easy!

- See git repo here
- Short code walk through incoming

Sources

Martin Fowler on Events

Martin Fowler on CQRS

[Martin Fowler on Event Sourcing][https://martinfowler.com/eaaDev/EventSourcing.html]

Chris Richardson on Sagas