Microservices + Software Development Life Cycle

Team 3
Edgar Lopez-Garcia
Ashot Chobanyan

Overview

- Microservices In-Depth
- Implementation Techniques
- Case Studies

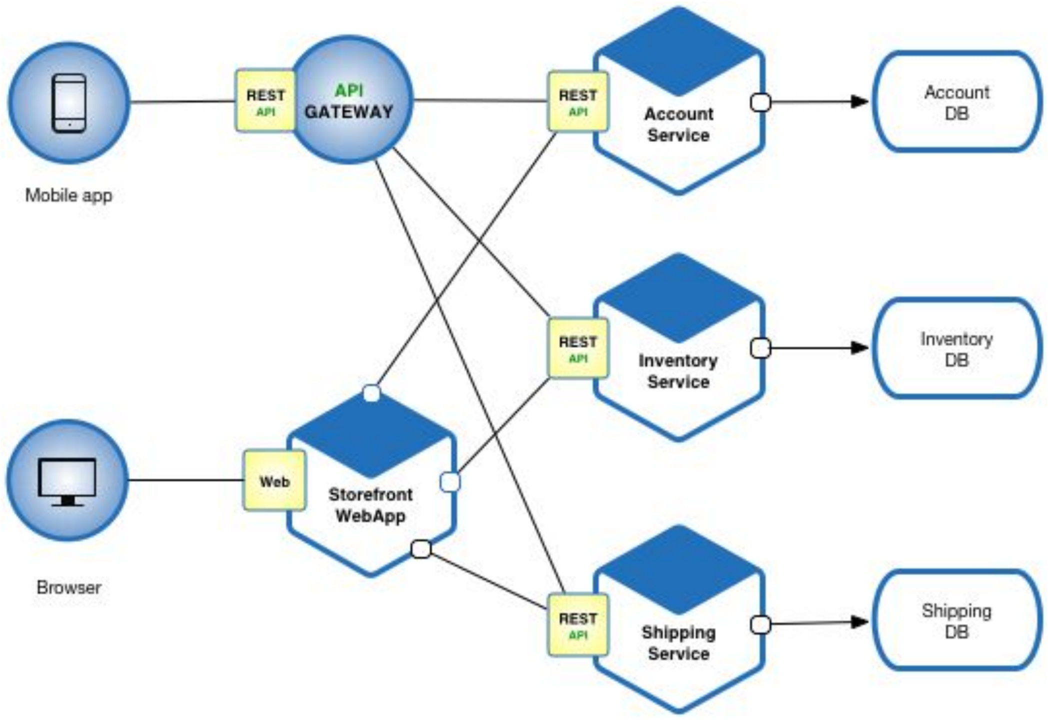
Microservices In-Depth

Recap

- Break down complex system into manageable services
- Benefits
    - Single responsibility
    - Separation of concerns
    - Modularity
- Physically, microservices can be in
    - Own separate server
    - A virtual machine/container
    - Serverless/FaaS (lambda, google/firebase cloud functions, Azure functions)

Microservices In-Depth

Recap



Microservices In-Depth

Inter-service communication considerations

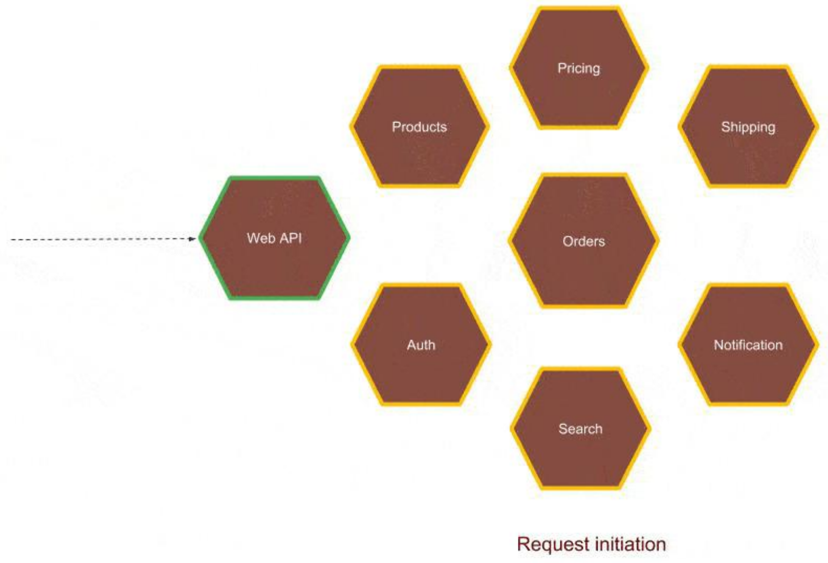
- How will services in system communicate?
- Communication interface? REST? SOAP?
- Synchronous vs asynchronous?
- Centralized vs Decentralized?

All of these questions must be considered because they affect the system architecture

Microservices In-Depth

A Decentralized and Synchronous approach

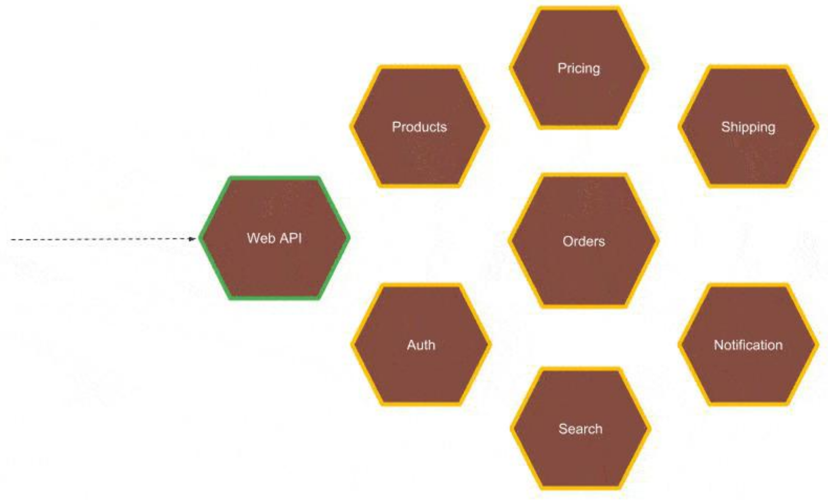
- Each request trickles down to all required services
- Service response trickles up to all services that invoked request
- Response sent to requestor once Web API gets response



Microservices In-Depth

Decentralized and Synchronous

1. Request initiation
2. Web API acts as the inceptor

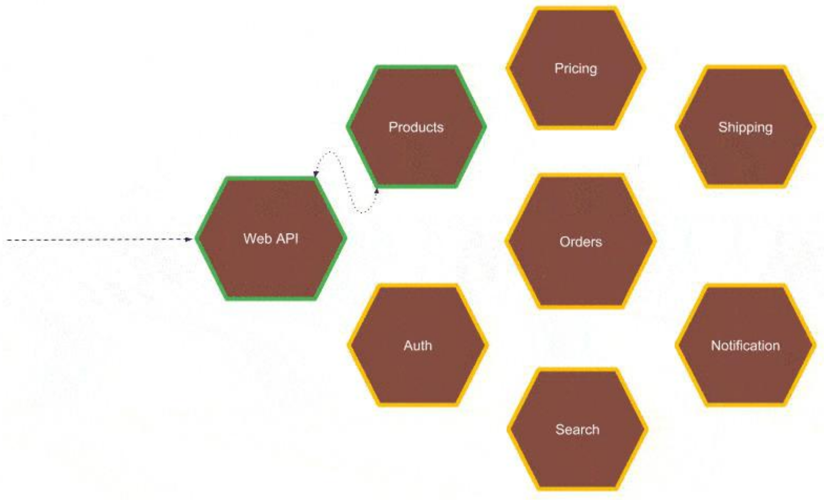


Request initiation

Microservices In-Depth

Decentralized and Synchronous

3. Web API makes sync call to Products service
4. Web API waits till Products service returns

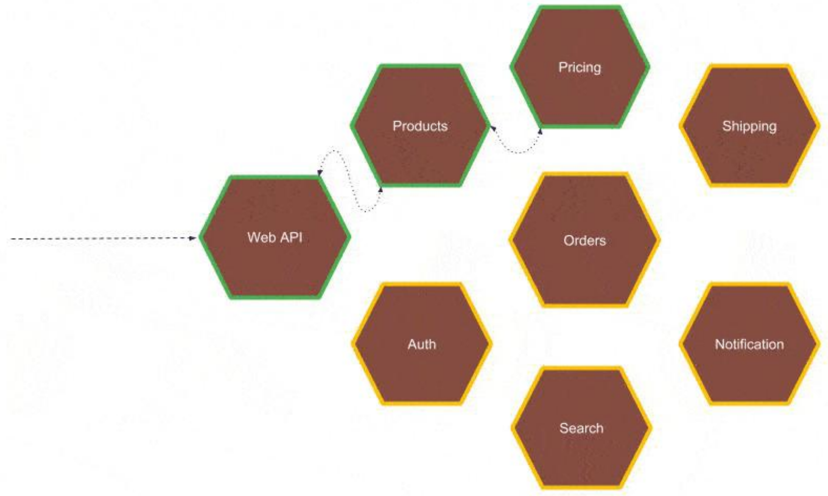


Retrieve product details

Microservices In-Depth

Decentralized and Synchronous

5. Products services makes sync call to Pricing services
6. Products services waits till Pricing service returns
7. Web API is also waiting for Products

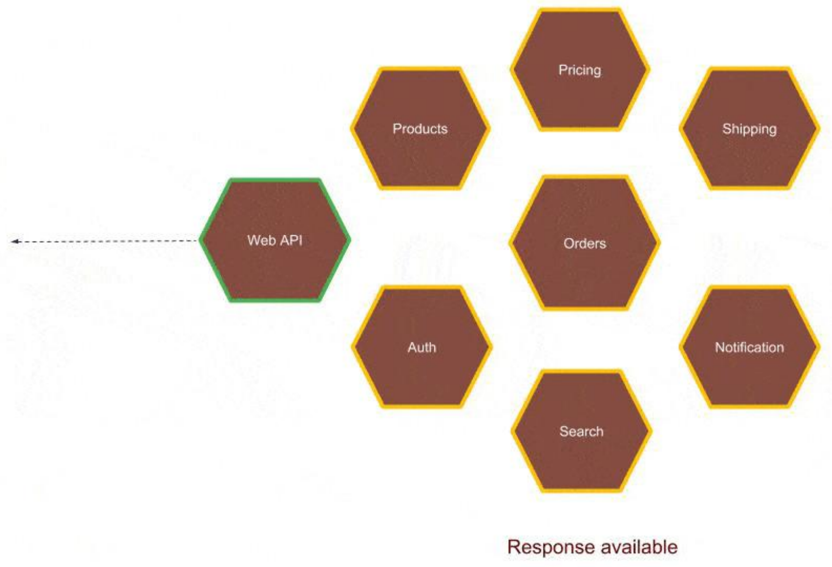


Product service internally fetches pricing info

Microservices In-Depth

Decentralized and Synchronous

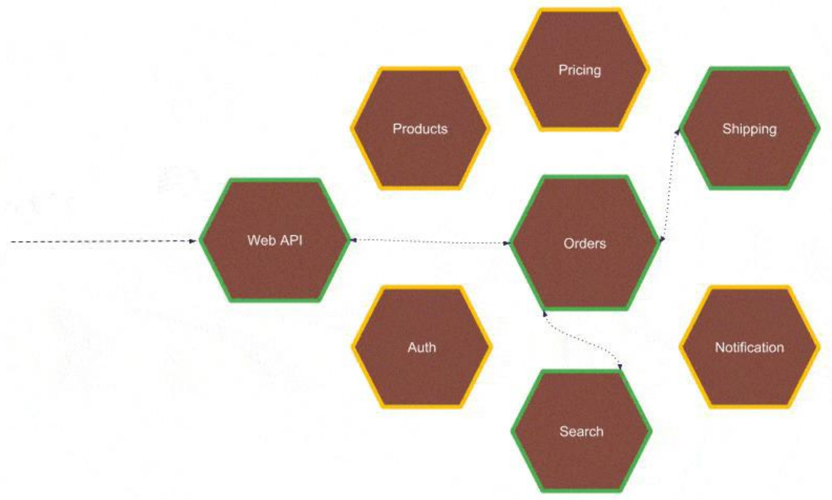
8. After all services return, the response is available
9. Web API service sends response to requester



Microservices In-Depth

Decentralized and Synchronous

- Services can do requests to multiple services
- Orders service depends on Shipping and Search services
- Orders service must wait for Search and Shipping services to return



Order system notifies shipping and indexes self

Microservices In-Depth

Decentralized and Synchronous

Pros

- Straightforward implementation
- Communication details explicit in system

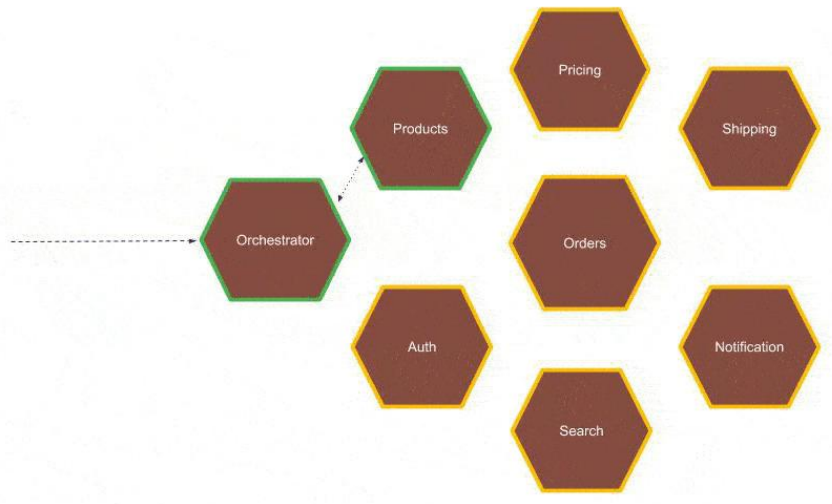
Cons

- Tight coupling between components and execution flow
- No flexibility; goes against changing requirements
- Blocking; bad for high read/write operations
- Deeply nested service calls are costly

Microservices In-Depth

An orchestrated, synchronous, and sequential approach

- Orchestrator service makes all required calls to services sequentially
- Once orchestrator gets response, makes request to next service (if applicable)
- Orchestrator returns response once all services return

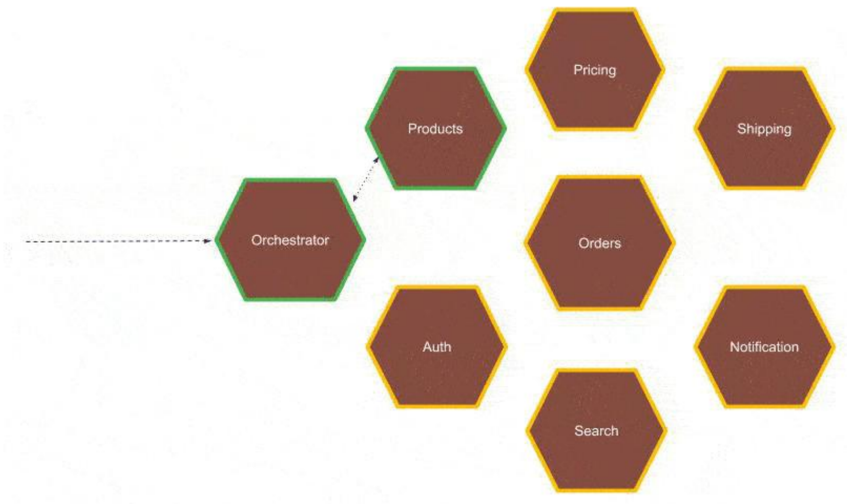


Retrieve product details

Microservices In-Depth

An orchestrated, synchronous, and sequential approach

1. Orchestrator receives request
2. Sends request to Products service and waits for response

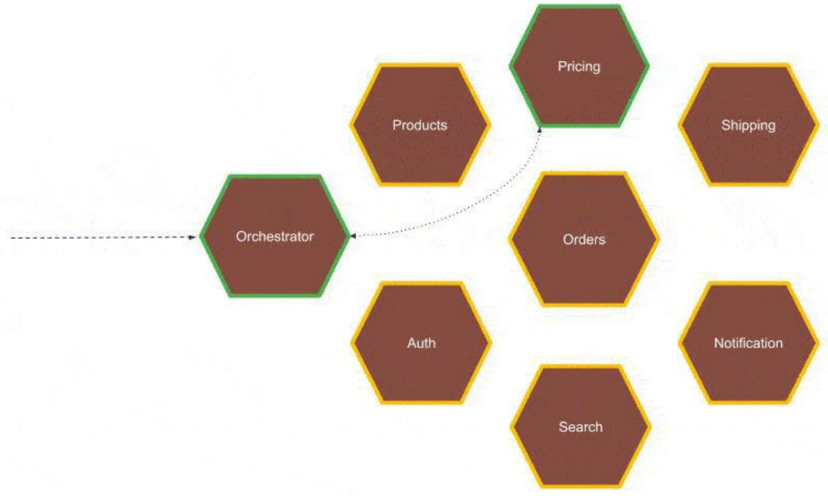


Retrieve product details

Microservices In-Depth

Orchestrated, synchronous, and sequential

3. Products services returns response
4. Orchestrator sends request to Pricing service
5. Orchestrator waits for response

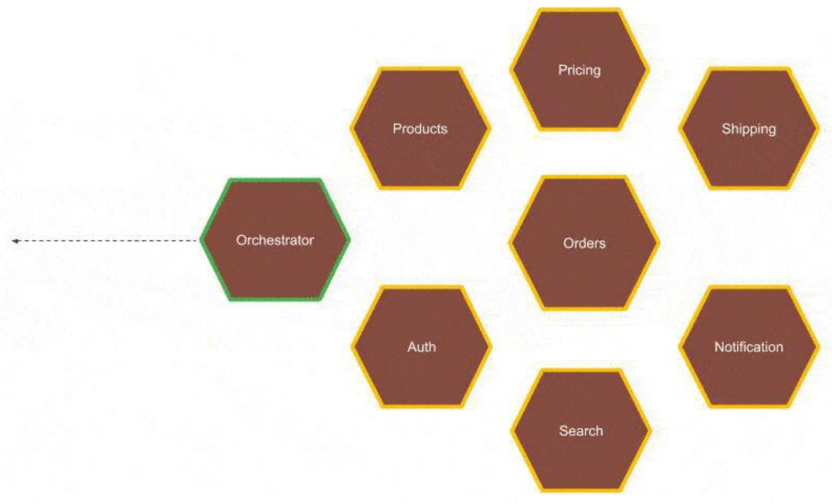


Populate product pricing

Microservices In-Depth

Orchestrated, synchronous, and sequential

6. Pricing service returns response
7. Orchestrator has all information required to fulfill request
8. Orchestrator returns response



Return response to the caller

Microservices In-Depth

Orchestrated, synchronous, and sequential

Pros

- Flexible; communication details in single place
- Less inter-service dependencies
- Sync calls - no need for mediating component

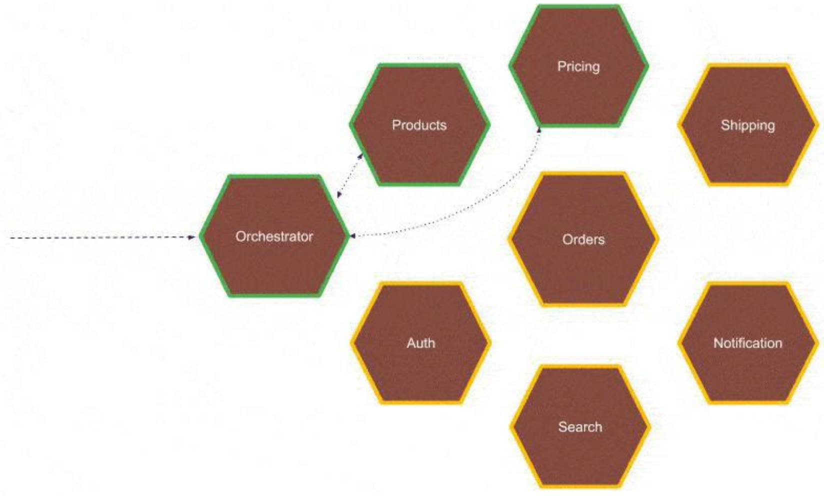
Cons

- Orchestrator holds all active requests
- Single point of failure

Microservices In-Depth

An orchestrated, synchronous, and parallel approach

- Orchestrator service makes all required calls to services in parallel
- Orchestrator doesn't have to wait for service response to make next call
- Orchestrator returns response once all services return

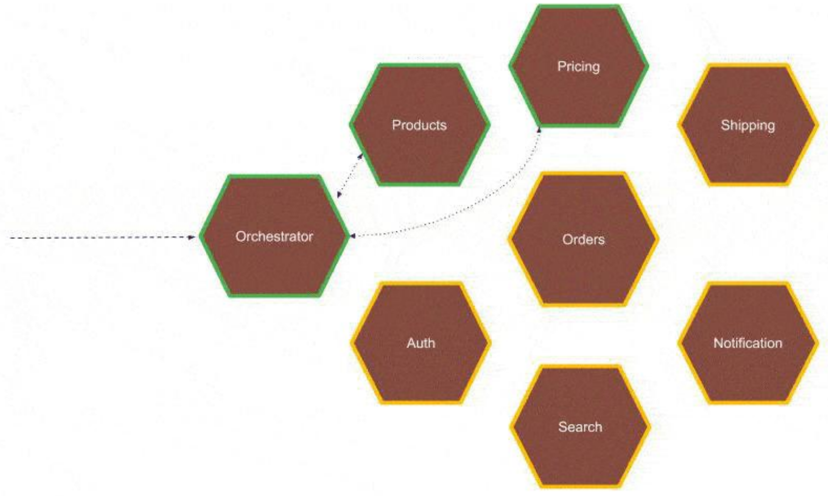


Product details and pricing info retrieval

Microservices In-Depth

An orchestrated, synchronous, and parallel approach

1. Orchestrator receives request
2. Sends request to Products and Pricing services

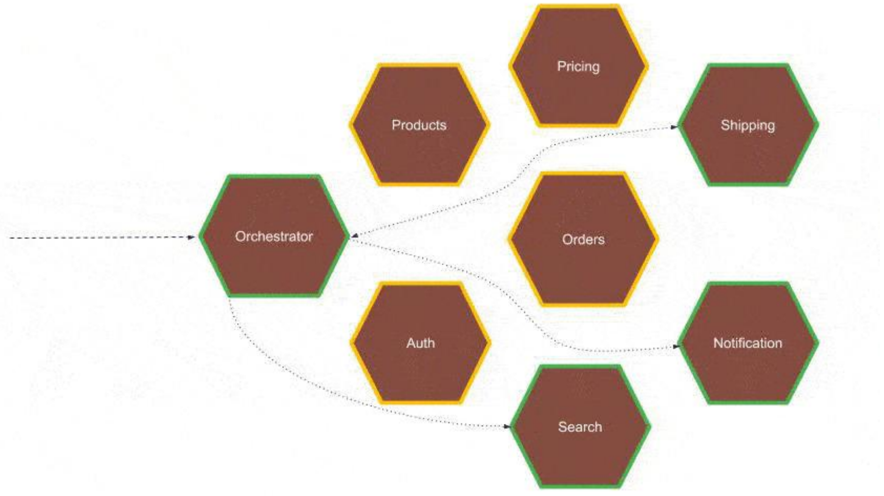


Product details and pricing info retrieval

Microservices In-Depth

Orchestrated, synchronous, and parallel

3. Orchestrator sends requests in parallel to all other services
4. Orchestrator collects responses of each service as they return
5. Once all requests are fulfilled, orchestrator returns response



Index order for search, notify shipping and send notifications

Microservices In-Depth

Orchestrated, synchronous, and parallel

Pros

- Independant requests
- Higher efficiency and performance
- Shorter response time = higher throughput

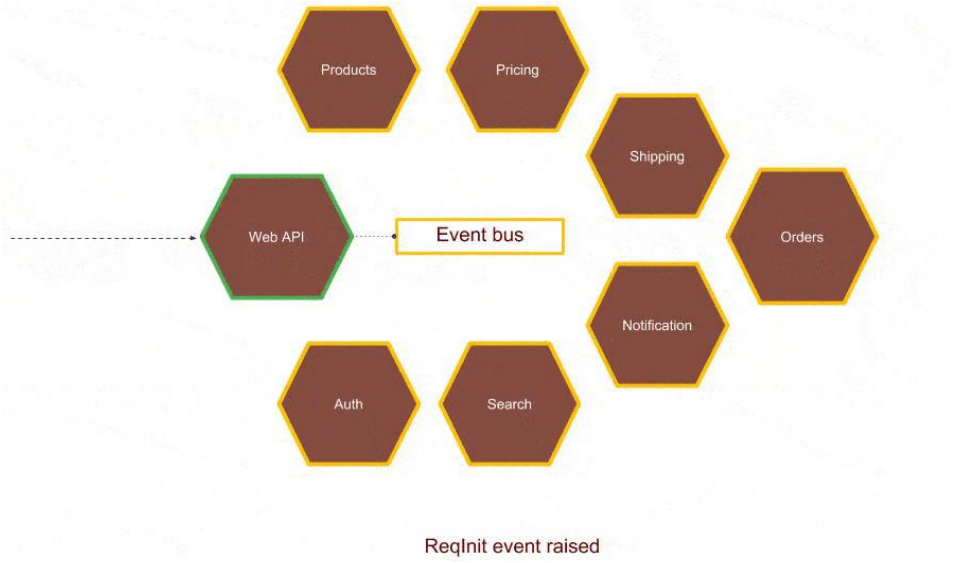
Cons

- Adds complexity to implementation and orchestration

Microservices In-Depth

Choreographed asynchronous events approach

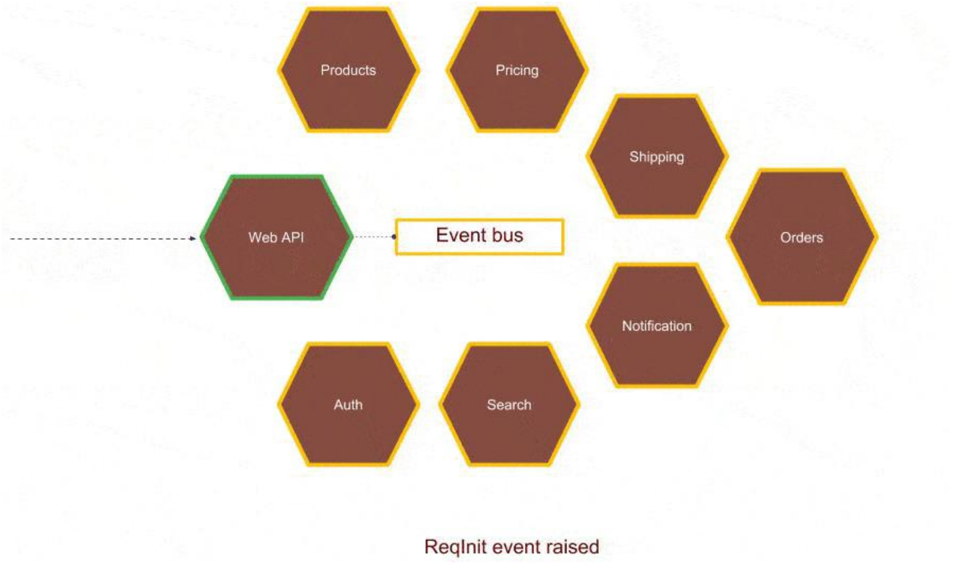
- All requests sent to event bus
- Each service responsible of querying and processing events
- Once request fulfilled, a Web API response event is sent
- Web API processes response event and sends response to requestor



Microservices In-Depth

Choreographed asynchronous events approach

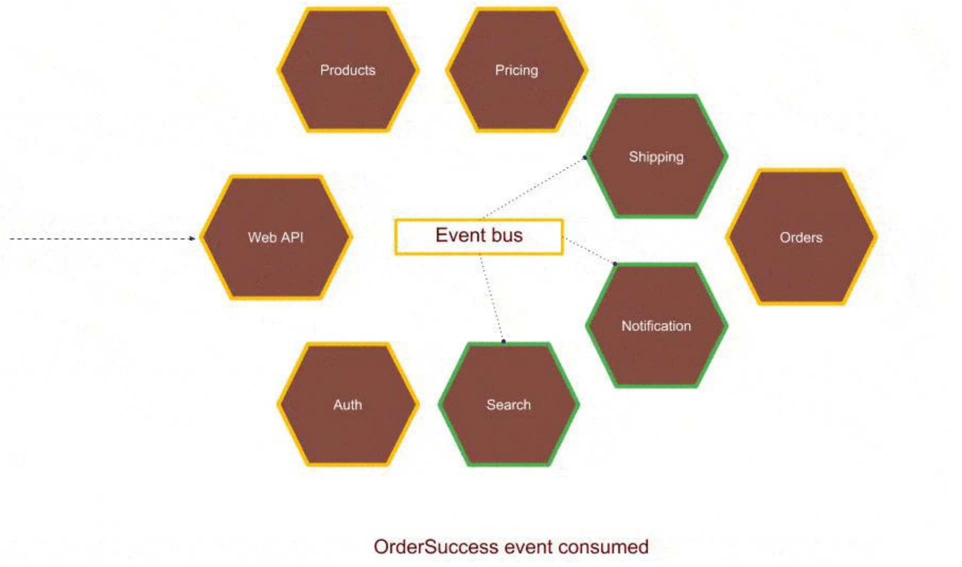
1. Web API service receives request
2. Request converted to event and sent to event bus



Microservices In-Depth

Choreographed asynchronous events

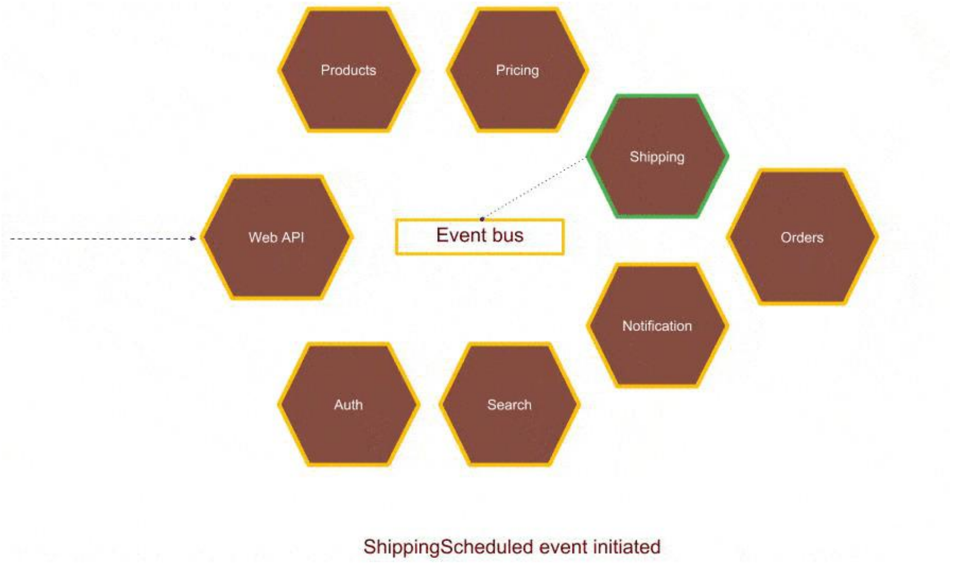
3. All required services (Shipping, Search, Notification) consume event from event bus



Microservices In-Depth

Choreographed asynchronous events

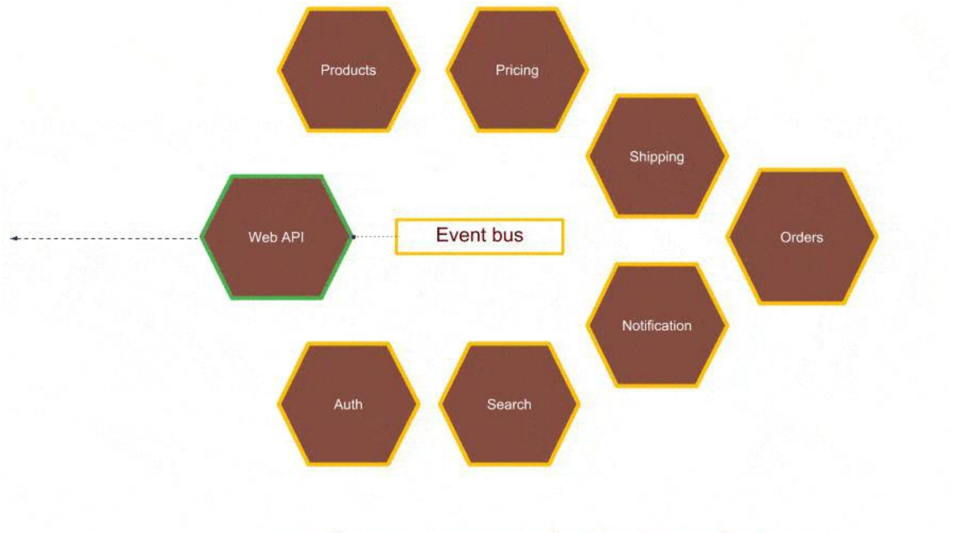
4. Shipping service is ready for response
5. Response event created and sent to event bus



Microservices In-Depth

Choreographed asynchronous events

6. Web API service consumes event and processes it
7. Web API returns response



ShippingScheduled event consumed, response returned

Microservices In-Depth

Choreographed asynchronous events

Pros

- Decoupled microservices
- Decentralized

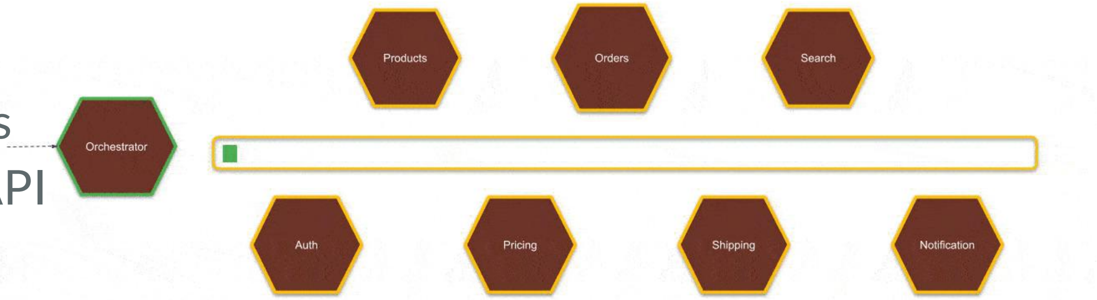
Cons

- Each microservice responsible for triggering downstream events
- Synchronous execution flows are awkward

Microservices In-Depth

Orchestrated, asynchronous, and sequential approach

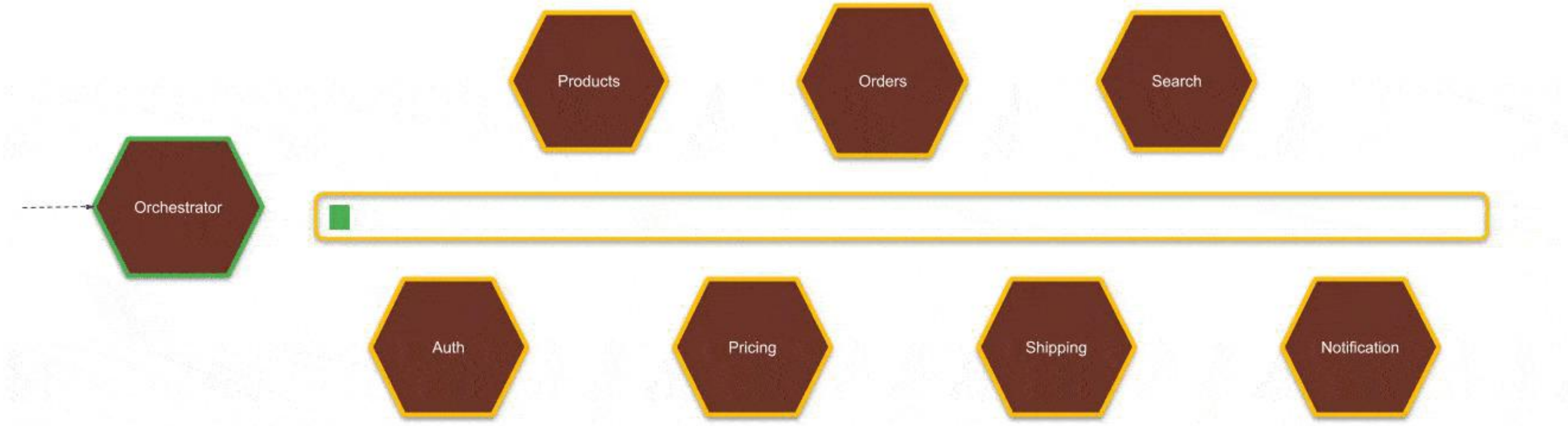
- All requests sent to event bus
- Each service responsible of querying and processing events
- Once request fulfilled, a Web API response event is sent
- Web API processes response event and sends response to requestor



Microservices In-Depth

Orchestrated, asynchronous, and sequential approach (retrieve product details)

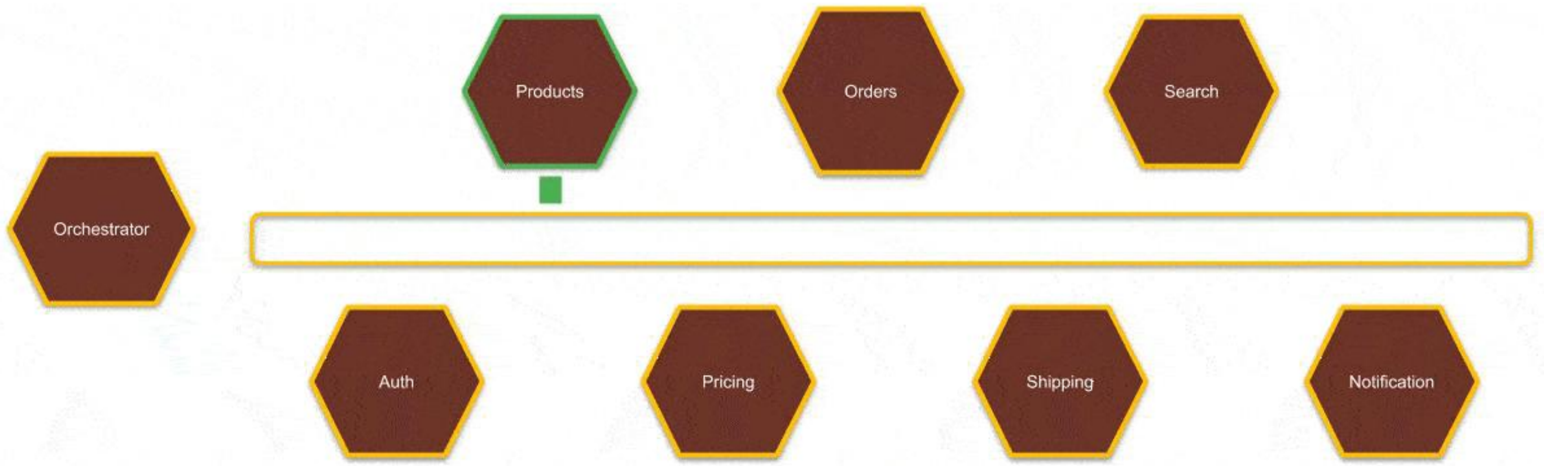
1. Orchestrator receives request



Microservices In-Depth

Orchestrated, asynchronous, and sequential approach (retrieve product details)

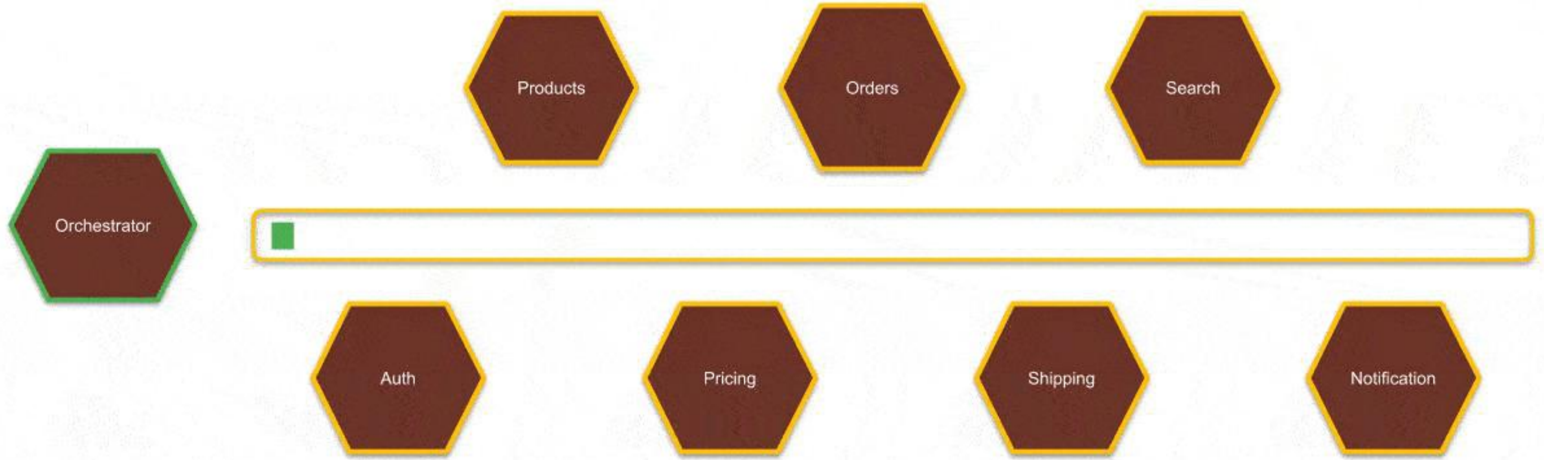
2. Orchestrator sends request message to Products service
3. Products service consumes message while Orchestrator waits



Microservices In-Depth

Orchestrated, asynchronous, and sequential approach (retrieve product details)

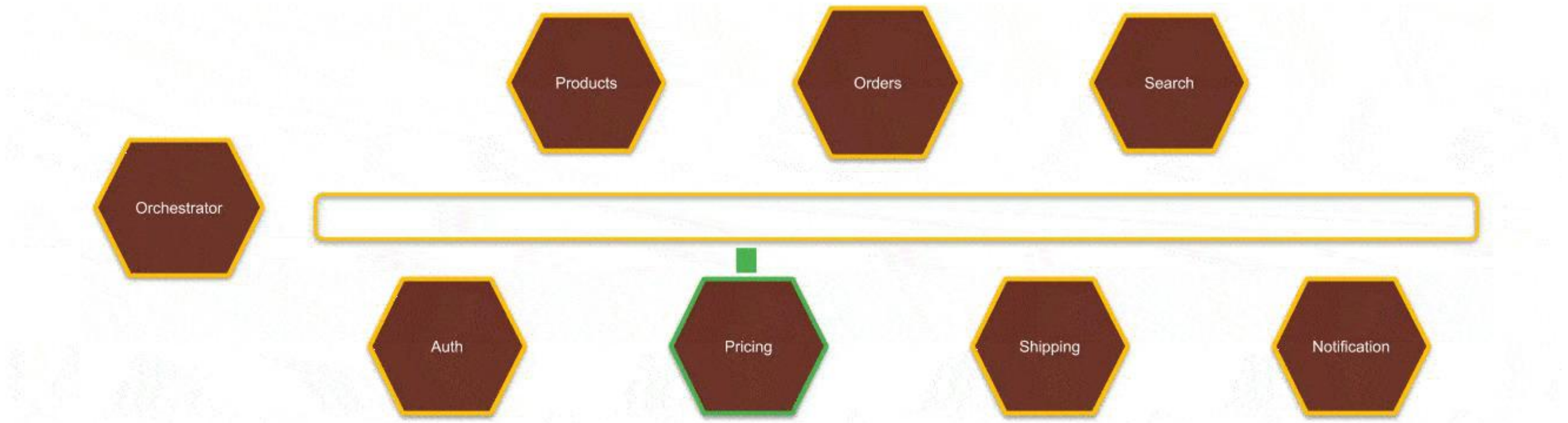
4. Products service sends response message
5. Orchestrator processes the response message



Microservices In-Depth

Orchestrated, asynchronous, and sequential approach (retrieve product details)

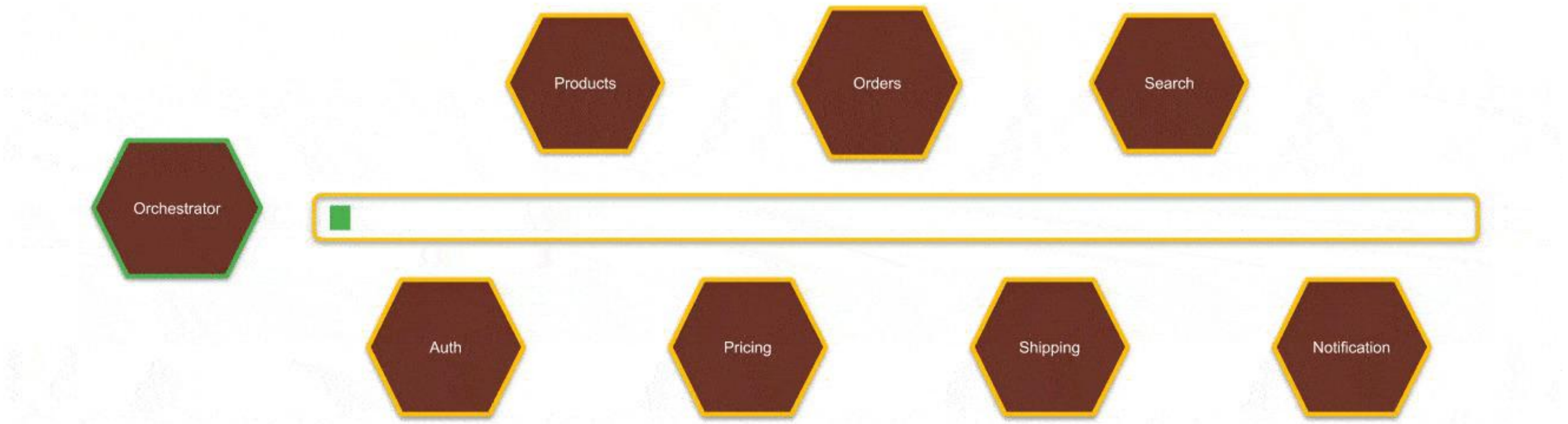
6. Orchestrator sends request message to Pricing service
7. Pricing service consumes message and processes request



Microservices In-Depth

Orchestrated, asynchronous, and sequential approach (retrieve product details)

8. Pricing service send response message to orchestrator
9. Orchestrator processes message and returns response to requestor



Microservices In-Depth

Orchestrated, asynchronous, and sequential

Pros

- Centralized execution flow
- More manageable choreographing
- Decoupled requests
- Requests never lost (queue messaging)

Cons

- Each microservice responsible for triggering downstream events

Microservices In-Depth

Synchronous vs Asynchronous

Synchronous Considerations

- Requires capacity service for traffic bursts
- Cascading failures
- Tight coupling - direct bindings to services

Asynchronous Considerations

- Handles temporary traffic bursts
- No request is lost due to message queue
- Async request can be hard to follow
- Each service must subscribe to async nature (consumer/producer)

Implementation Techniques

There exists no single implementation of microservices, they can:

- Be in any language
    - Lets the task dictate the language
- Be hosted on any platform
    - Or even across platforms
- Intermingle freely

Implementation Techniques

Microservices as a technique:

- Command Query Responsibility Segregation (CQRS)
  - Separate read and write interfaces
- Based on REST APIs
- CI/CD
  - Ex.. Jenkins
- Multiple containers
  - One for each microservice
- Fault tolerant site design
  - Exceptions for when microservices may fail
  - Circuit breakers to protect services

Implementation Techniques

Common features:

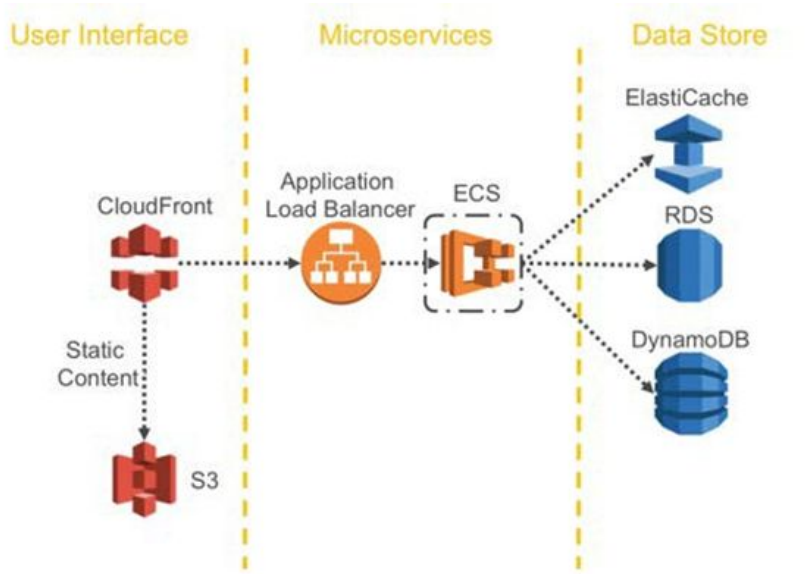
- Service providers/hosts
- API Gateways
- Service Discovery
- Orchestration
- Monitoring/logging
- Databases/storage

Implementation Techniques

Service providers by type:

- Container based
- Serverless

GCP, AWS, and Azure all offer both services



Implementation Techniques

Orchestration software will try to achieve a desired state

If done correctly this desired state can scale horizontally

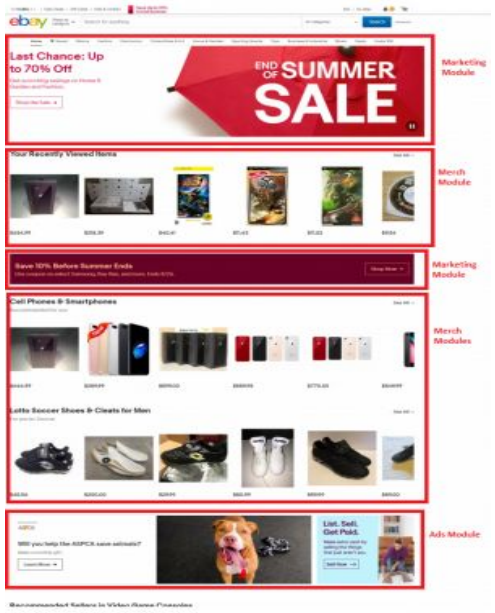
(ex Horizontal Pod Autoscaler in Kubernetes)

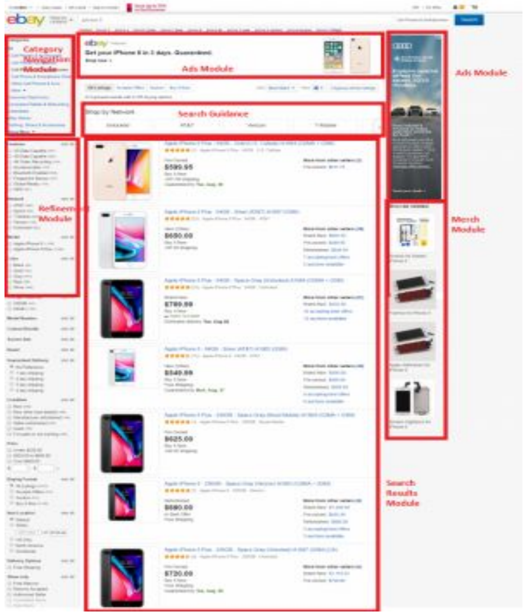
Implementation Techniques

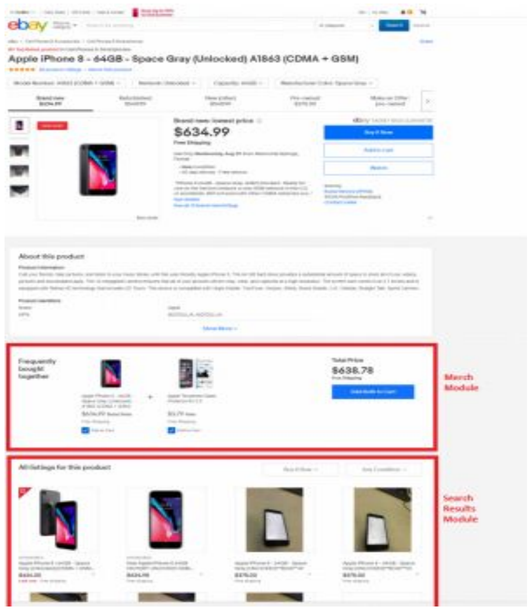
Do NOT allow services to call other services directly if avoidable:

- Use queues (ex Kafka) to avoid service overloading
- Can lead to services calling themselves through long chains
- Build in fault tolerance at every level

Case Studies: eBay


Last Chance: Up to 70% Off
END OF SUMMER SALE
Marketing Module
Your Recently Viewed Items
Merch Module
Save 10% Before Summer Ends
Marketing Module
Cell Phones & Smartphones
Merch Modules
Lotto Soccer Shoes & Cleats for Men
ASPCA
List. Sell. Get Paid.
Ads Module
Recommended Sellers in Video Game Consoles




Apple iPhone 8 - 64GB - Space Gray (Unlocked) A1863 (CDMA + GSM)
$634.99
About this product
Frequently bought together
$638.78
Merch Module
All listings for this product
Search Results Module

Case Studies: Walmart

Conversions were up by 20%

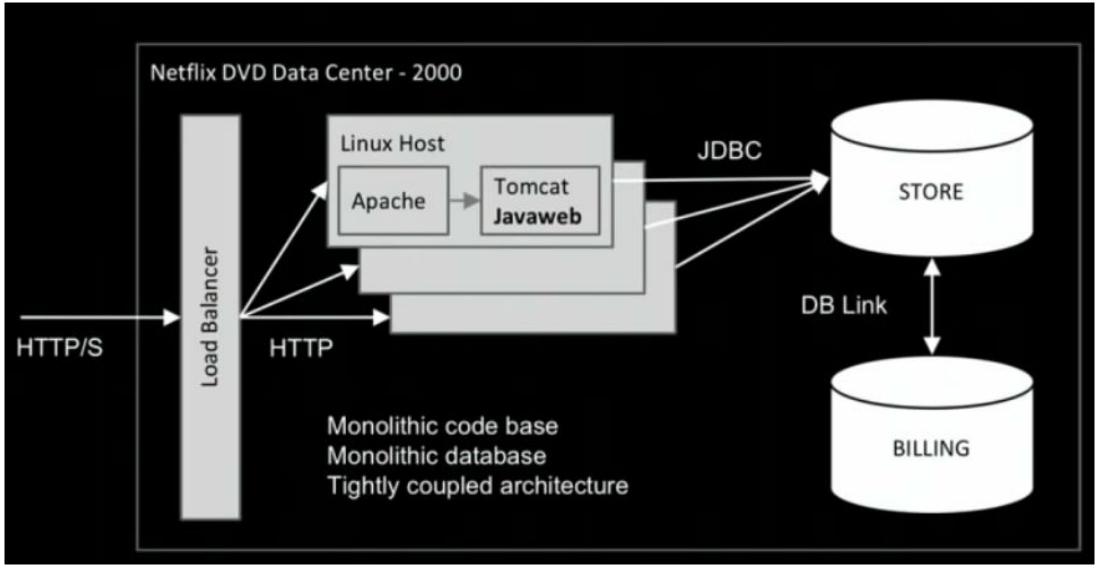
Mobile orders were up by 98%

No downtime on Black Friday or Boxing Day

They saved 40% of the computing power and experienced 20-50% cost savings

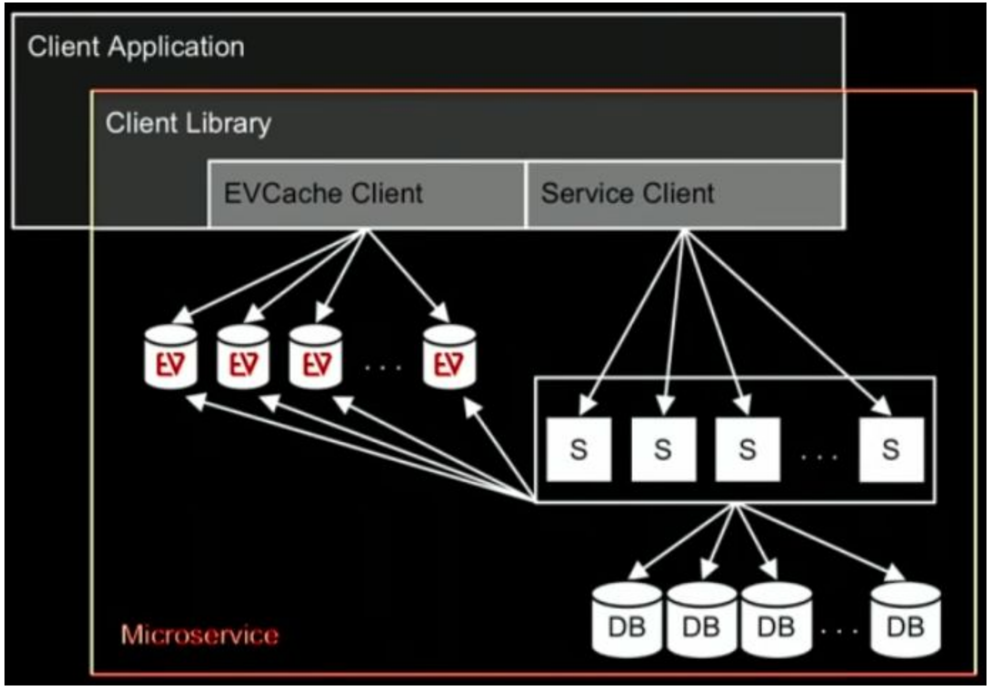
Case Studies: Netflix

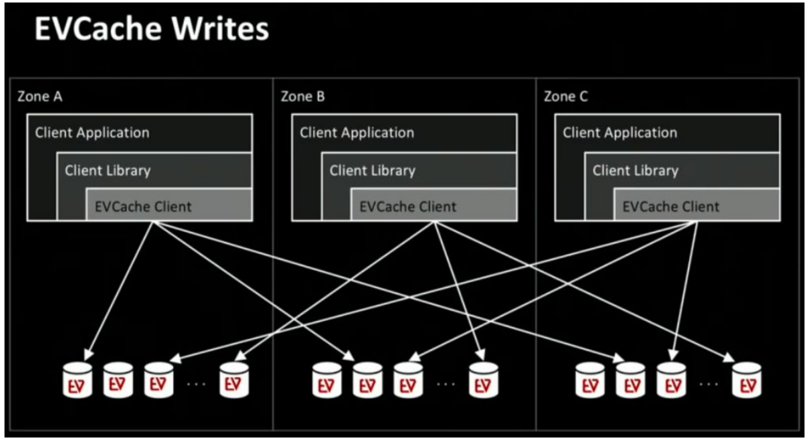
Transitioned from **monolith** to microservices:

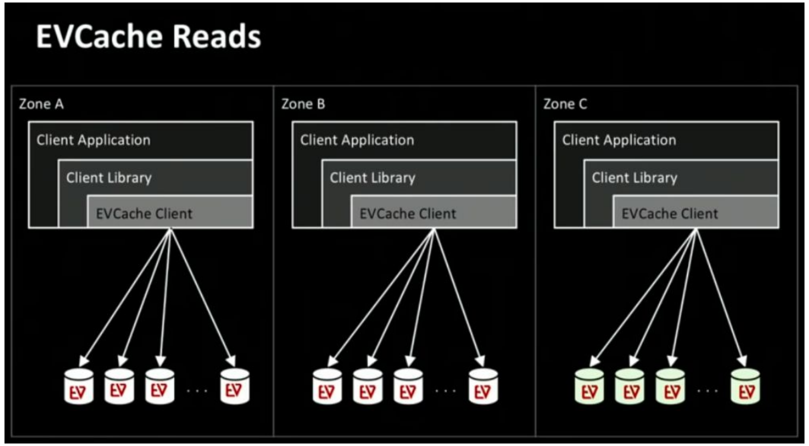


Case Studies: Netflix

Transitioned from monolith to **microservices**:







Case Studies: Netflix

Hystrix - Latency/Fault tolerance; prevent cascading failures

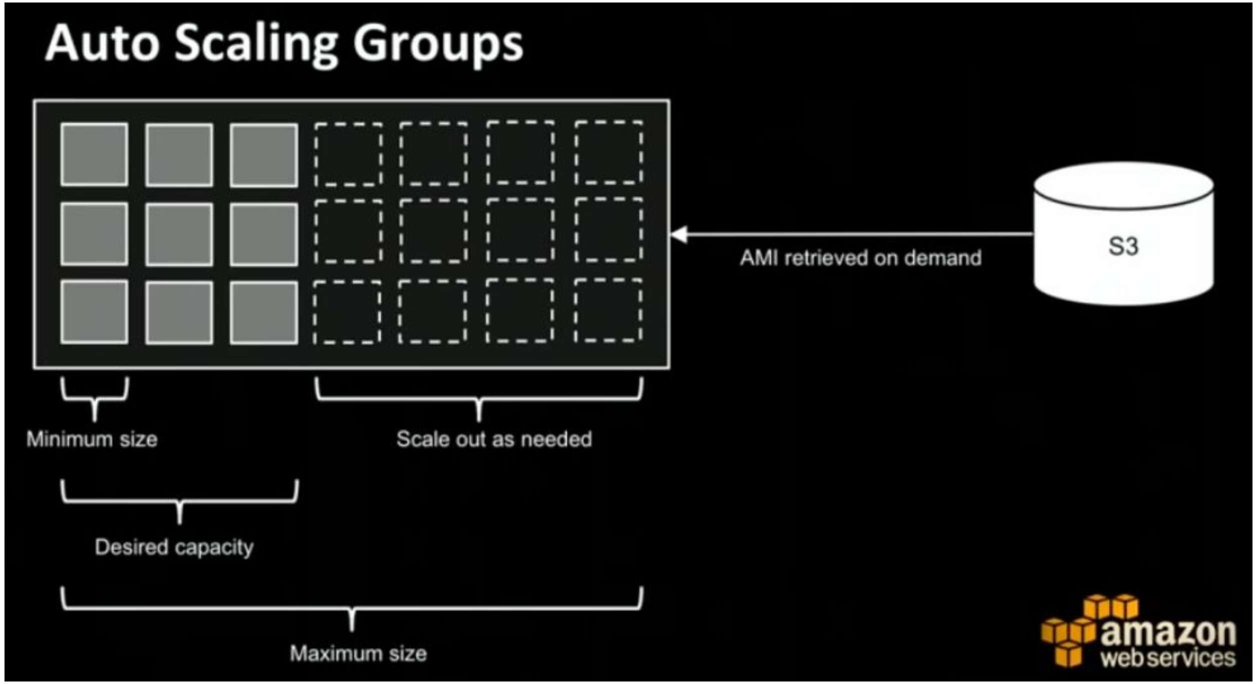
Apache Cassandra - Eventually consistent

-Critical Microservices

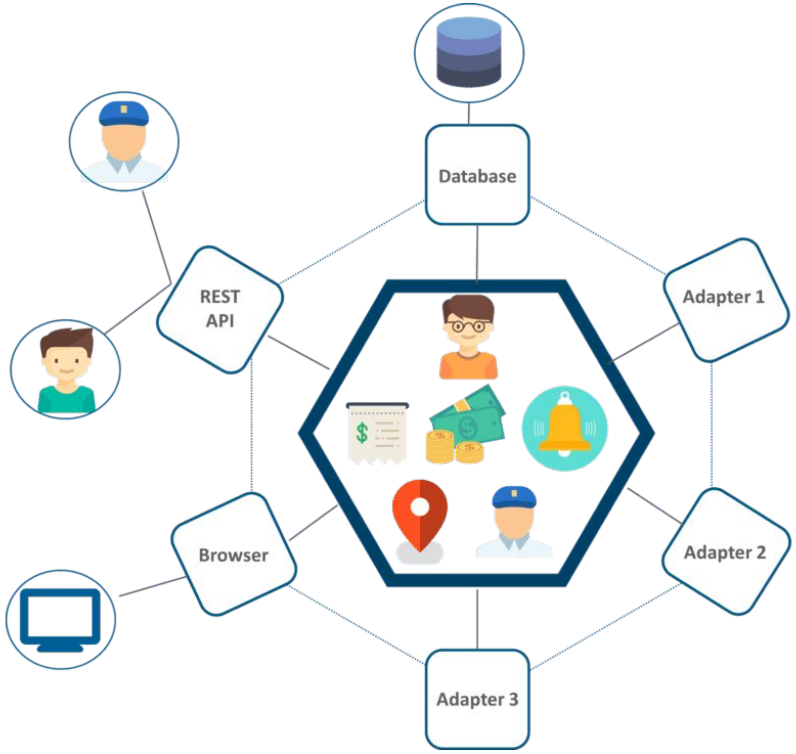
Use 3 AWS regions

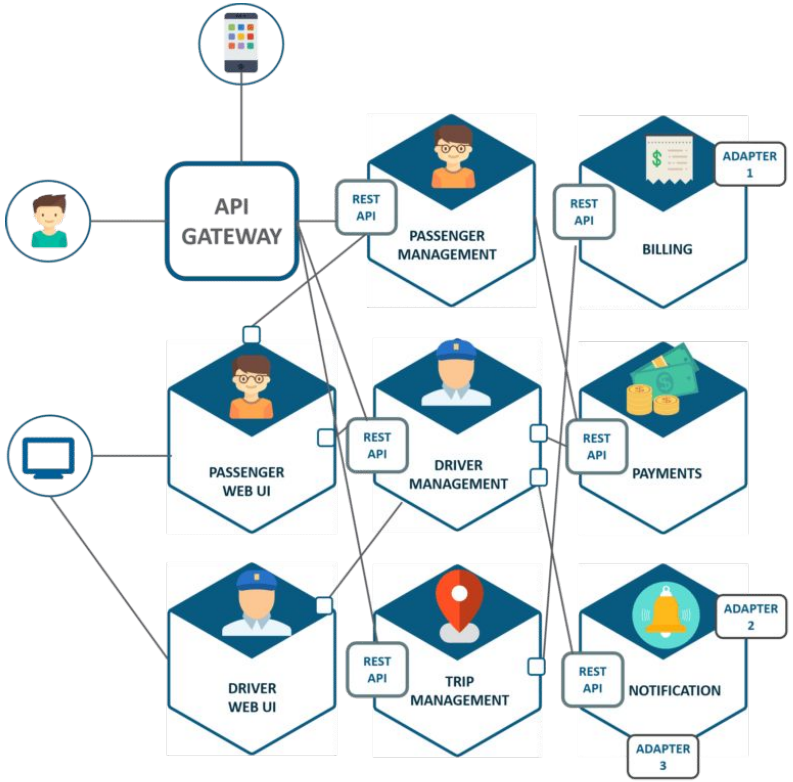
Case Studies: Netflix

For stateless services



Case Studies: Uber





Case Studies: SoundCloud

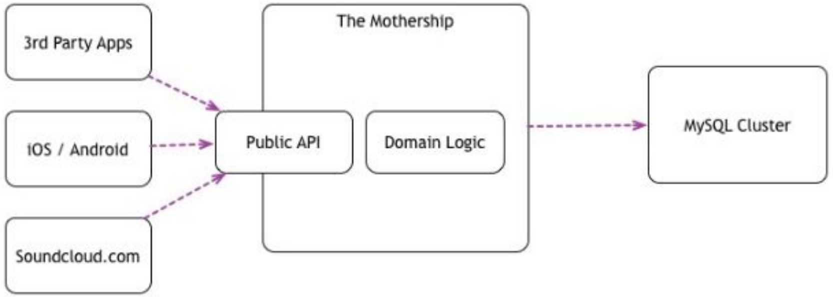
Began transitioning to microservices and SPA back in 2012

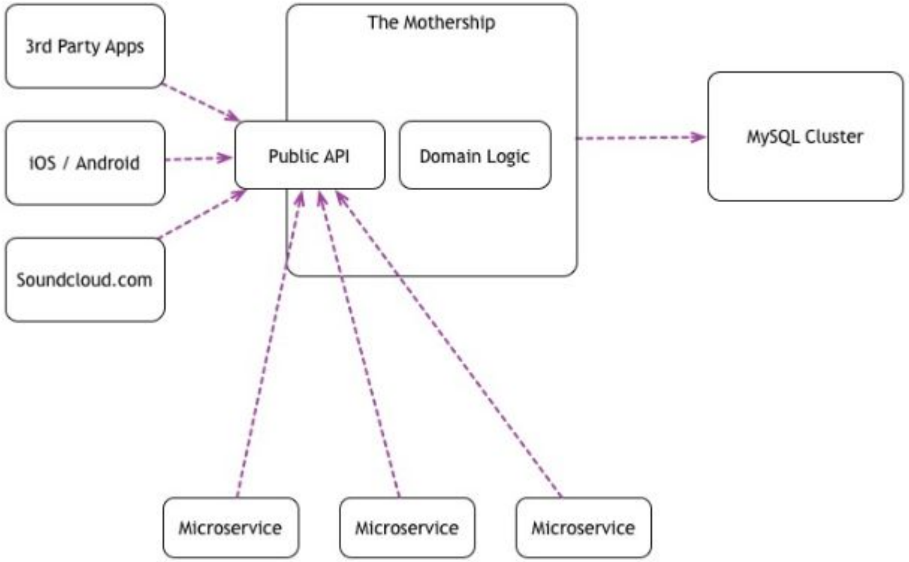
Use 100s of microservices currently

Now use Prometheus (monitoring), Docker, Kubernetes, and Finagle (RPC system)

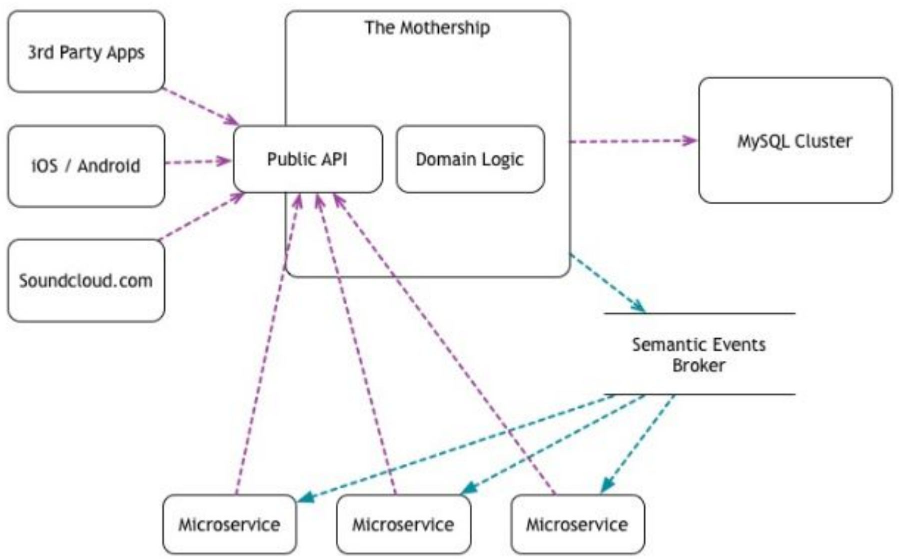
Case Studies: SoundCloud

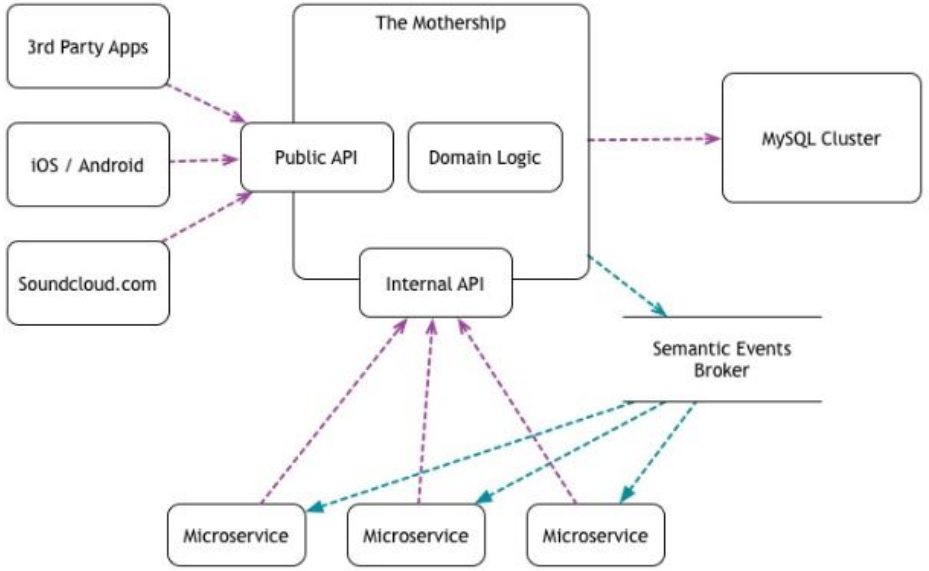
Start





Case Studies: SoundCloud

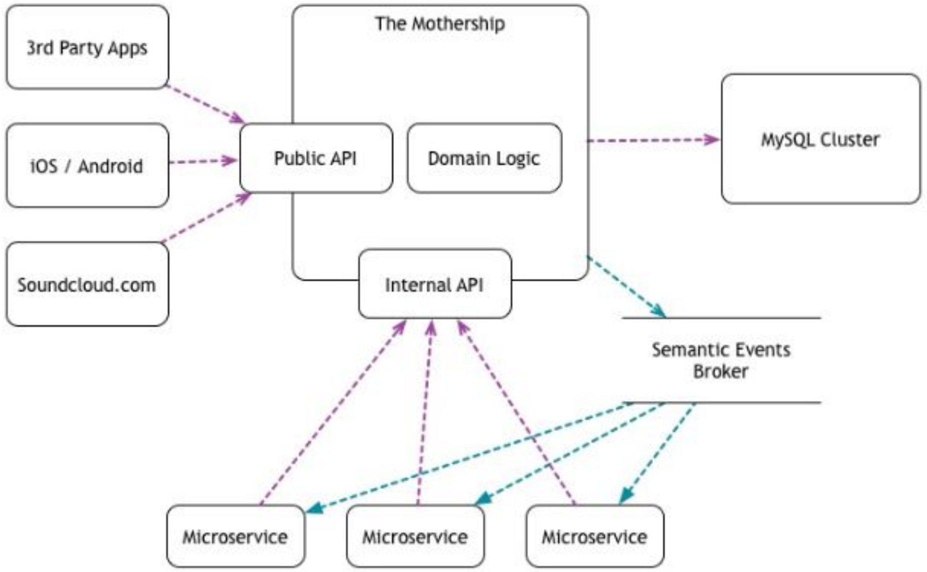




Allowed for Event Sourcing

to handle shared data

Case Studies: SoundCloud



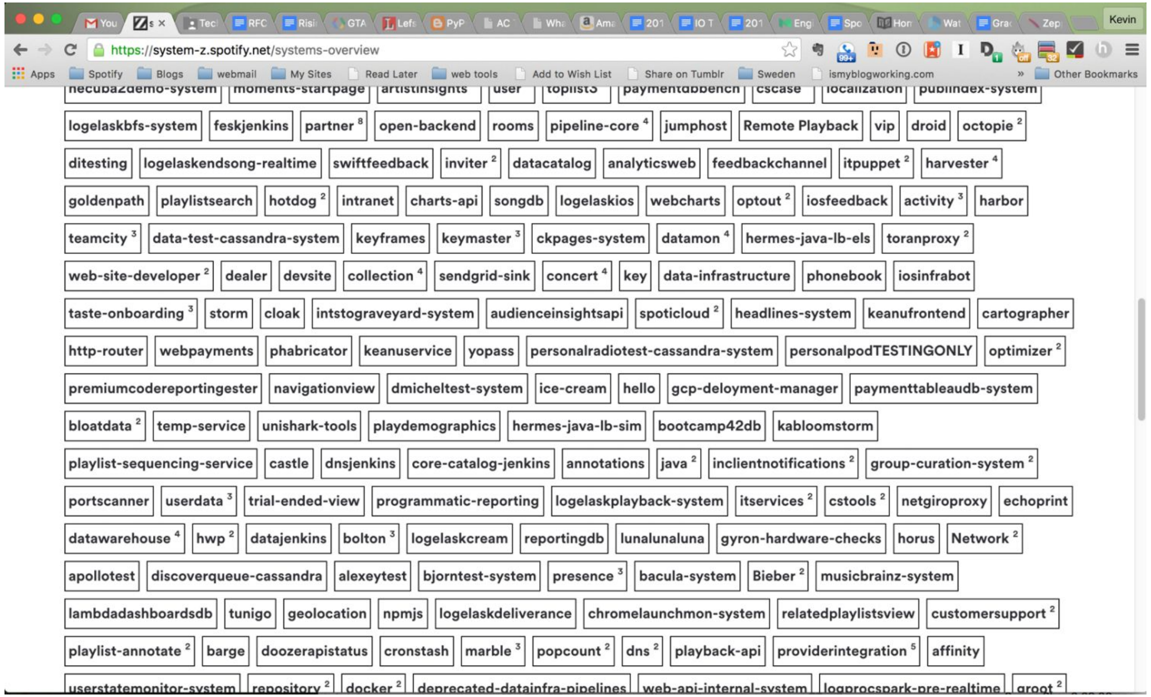
"When making significant changes to a feature or entity which exists inside the monolith, consider implementing these in a new service which is outside the monolith. We call these *extraction projects*."

-SoundCloud Blog Post by Matthias Käppler

Case Studies: SoundCloud

An aside:

A look at Spotify's microservice list



https://system-z.spotify.net/systems-overview

hecuba2demo-system | moments-startpage | artistinsights | user | toplist3 | paymentabbench | cscase | localization | publindex-system

logelaskbfs-system | feskjenkins | partner 8 | open-backend | rooms | pipeline-core 4 | jumphost | Remote Playback | vip | droid | octopie 2

ditesting | logelaskendsong-realtime | swiftfeedback | inviter 2 | datacatalog | analyticsweb | feedbackchannel | itpuppet 2 | harvester 4

goldenpath | playlistsearch | hotdog 2 | intranet | charts-api | songdb | logelaskios | webcharts | optout 2 | iosfeedback | activity 3 | harbor

teamcity 3 | data-test-cassandra-system | keyframes | keymaster 3 | ckpages-system | datamon 4 | hermes-java-lb-els | toranproxy 2

web-site-developer 2 | dealer | devsite | collection 4 | sendgrid-sink | concert 4 | key | data-infrastructure | phonebook | iosinfrabot

taste-onboarding 3 | storm | cloak | intstograveyard-system | audienceinsightsapi | spoticloud 2 | headlines-system | keanufrontend | cartographer

http-router | webpayments | phabricator | keanuservice | yopass | personalradiotest-cassandra-system | personalpodTESTINGONLY | optimizer 2

premiumcodereportingester | navigationview | dmicheltest-system | ice-cream | hello | gcp-deloyment-manager | paymenttableaudb-system

bloatdata 2 | temp-service | unishark-tools | playdemographics | hermes-java-lb-sim | bootcamp42db | kabloomstorm

playlist-sequencing-service | castle | dnsjenkins | core-catalog-jenkins | annotations | java 2 | inclientnotifications 2 | group-curation-system 2

portscanner | userdata 3 | trial-ended-view | programmatic-reporting | logelaskplayback-system | itservices 2 | cstools 2 | netgiroproxy | echoprint

datawarehouse 4 | hwp 2 | datajenkins | bolton 3 | logelaskcream | reportingdb | lunalunaluna | gyron-hardware-checks | horus | Network 2

apollotest | discoverqueue-cassandra | alexeytest | bjorntest-system | presence 3 | bacula-system | Bieber 2 | musicbrainz-system

lambdadashboardsdb | tunigo | geolocation | npmjs | logelaskdeliverance | chromelaunchmon-system | relatedplaylistsview | customersupport 2

playlist-annotate 2 | barge | doozerapistatus | cronstash | marble 3 | popcount 2 | dns 2 | playback-api | providerintegration 5 | affinity

userstatemonitor-system | repository 2 | docker 2 | deprecated-datainfra-pipelines | web-api-internal-system | logprocspark-pre-realtime | groot 2

Questions?