Microservices

Nick Escalona @ Revature 5/20/2020

REVATURE

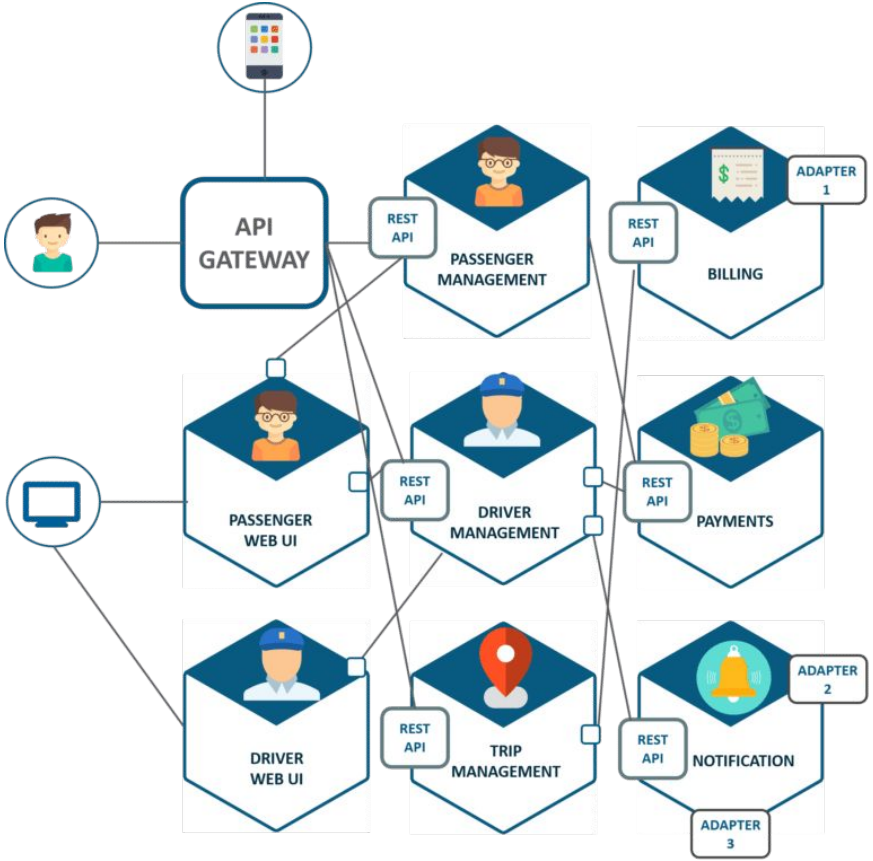
What problem are we solving?

Applications are often built as a “monolith” - all the code, except maybe for the DB and the UI, is compiled together and deployed together. What’s the problem?

- One small change = build and deploy the whole thing
- Hard to keep the code well organized with strong abstractions
- If one part of the app is a bottleneck, we have to scale the whole app

Principles of MSA

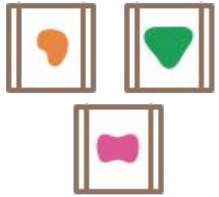
- Have many services - each service implements one business capability
- Services...
  - Are developed independently
  - Are deployed independently
  - Are scaled independently
  - Control their own logic and their own data
- Products not projects
- Smart endpoints and dumb pipes
- CI/CD


API
GATEWAY
REST
API
PASSENGER
MANAGEMENT
REST
API
BILLING
ADAPTER
1
PASSENGER
WEB UI
REST
API
DRIVER
MANAGEMENT
REST
API
PAYMENTS
DRIVER
WEB UI
REST
API
TRIP
MANAGEMENT
REST
API
NOTIFICATION
ADAPTER
2
ADAPTER
3

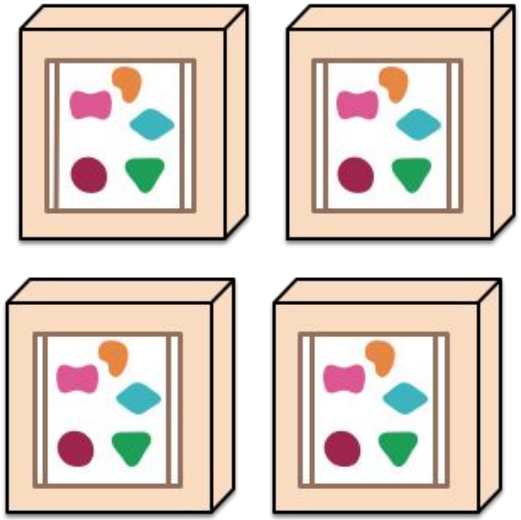
REVATURE

A monolithic application puts all its functionality into a single process...

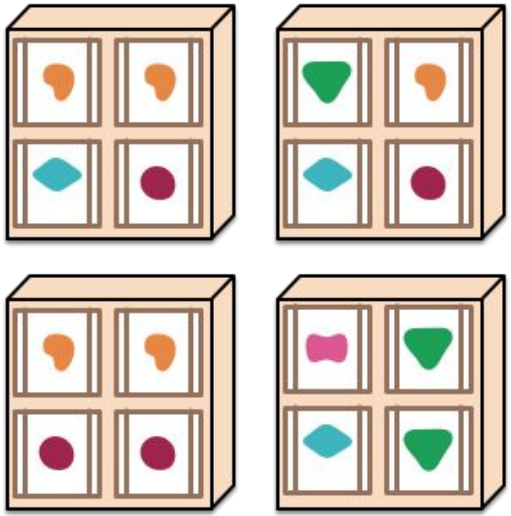
A microservices architecture puts each element of functionality into a separate service...



... and scales by replicating the monolith on multiple servers



... and scales by distributing these services across servers, replicating as needed.



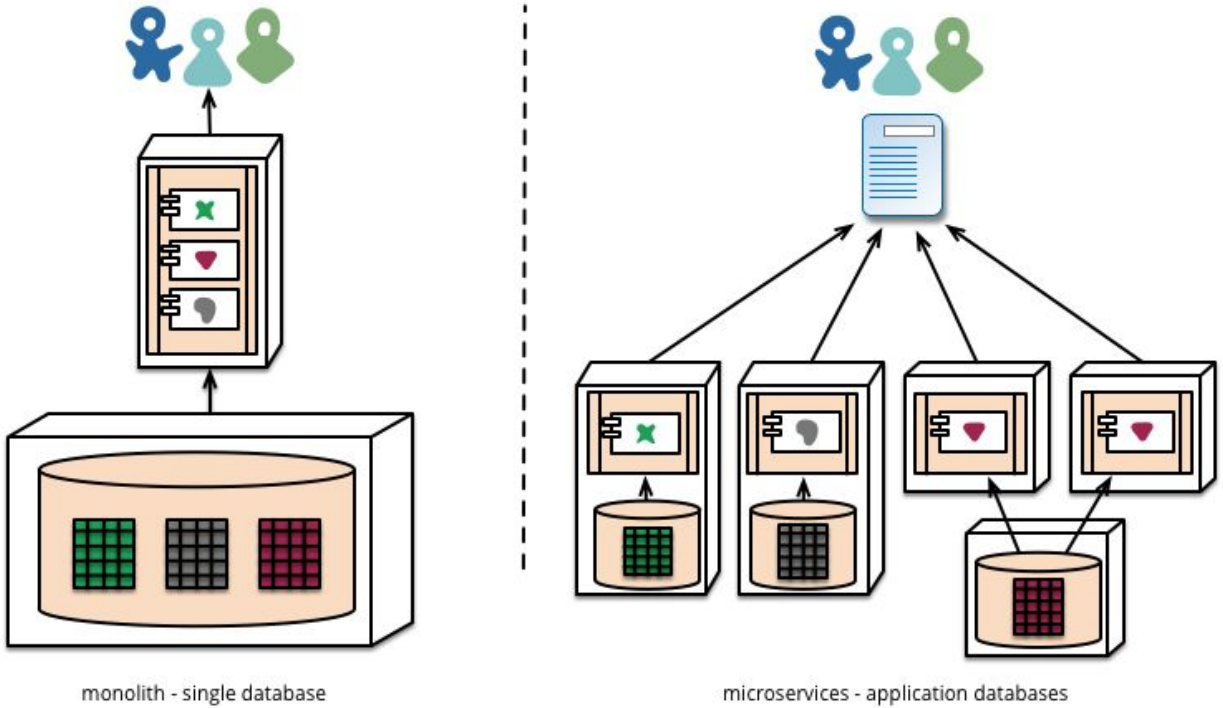
Advantages

- Long-term flexibility
- Scaling of actual function that needs more resources = cost savings
- Works well with Agile
- Loose coupling is more enforced by the architecture

Why not one database?

- Centralized database is a performance bottleneck
- Bloated tables with dozens or 100+ columns
- Would you use the same map for a hike, for a road trip, and for learning geography?

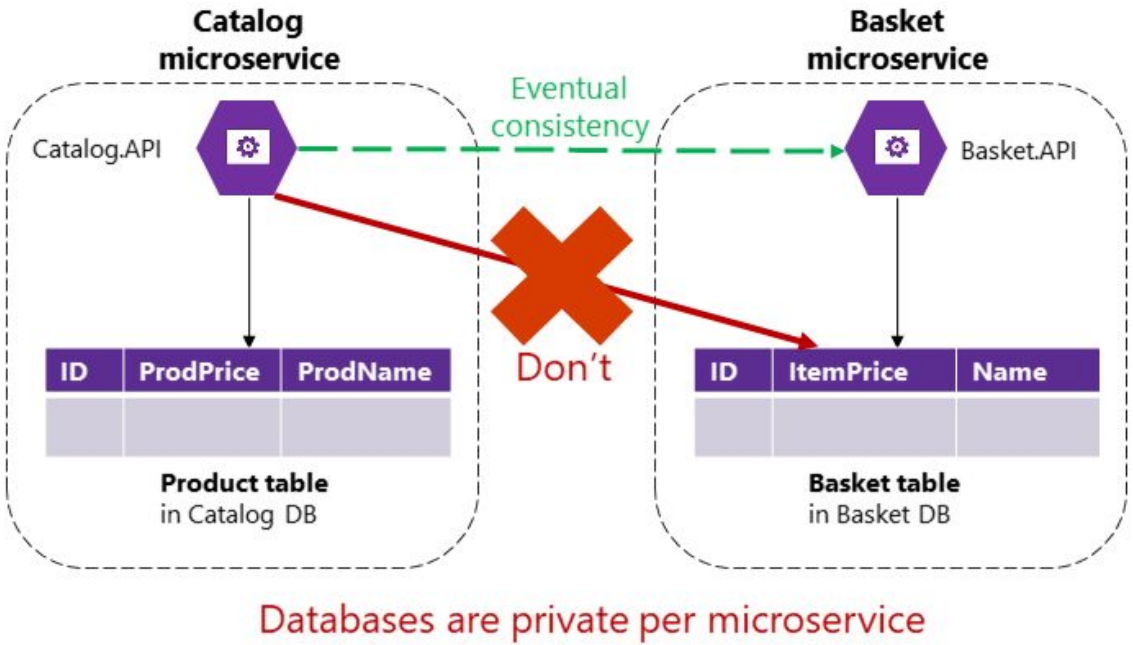
- Each service stores only the data it needs
- Free to use the data tech suiting its needs
- Free to format the data as it needs
- Data sovereignty


monolith - single database
microservices - application databases

Downsides of data sovereignty

- Lose full power of SQL
- Lose easy ACID transactions across services
  - We can't lock down a service until transaction is finished: too slow
  - We don't want to in the first place: services are independent

- Eventual consistency


Catalog microservice
Basket microservice
Eventual consistency
Catalog.API
Basket.API
Don't
ID
ProdPrice
ProdName
Product table in Catalog DB
ID
ItemPrice
Name
Basket table in Basket DB
Databases are private per microservice

Special Concerns in MSA

- Where to draw service boundaries?
  - Different language for the “same thing” is a clue
    - E.g. account, user, profile, customer, buyer
  - Domain-driven design (DDD) fits well
- How to combine data spread out across all these services?
  - API gateways
    - Danger - coupling services, single point of failure
  - Too much of this can be a sign of bad service boundaries

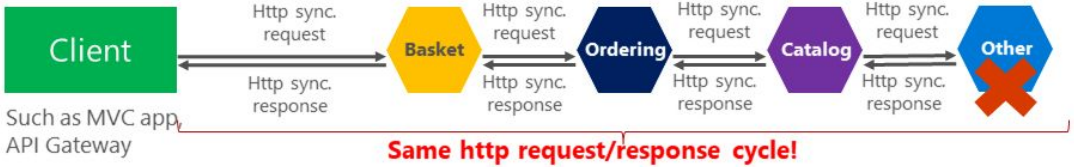
Special Concerns in MSA

- Eventual consistency
  - Services should be independent, autonomous
  - Lightweight message queue / event bus instead of HTTP
  - One service publishes event, many can subscribe

Synchronous vs. async communication across microservices

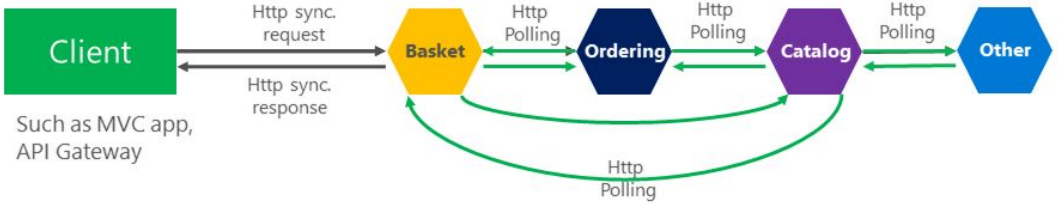
Anti-pattern

Synchronous
all request/response cycle



Asynchronous
Comm. across internal microservices
(EventBus: like **AMQP**)

"Asynchronous"
Comm. across internal microservices
(Polling: **Http**)



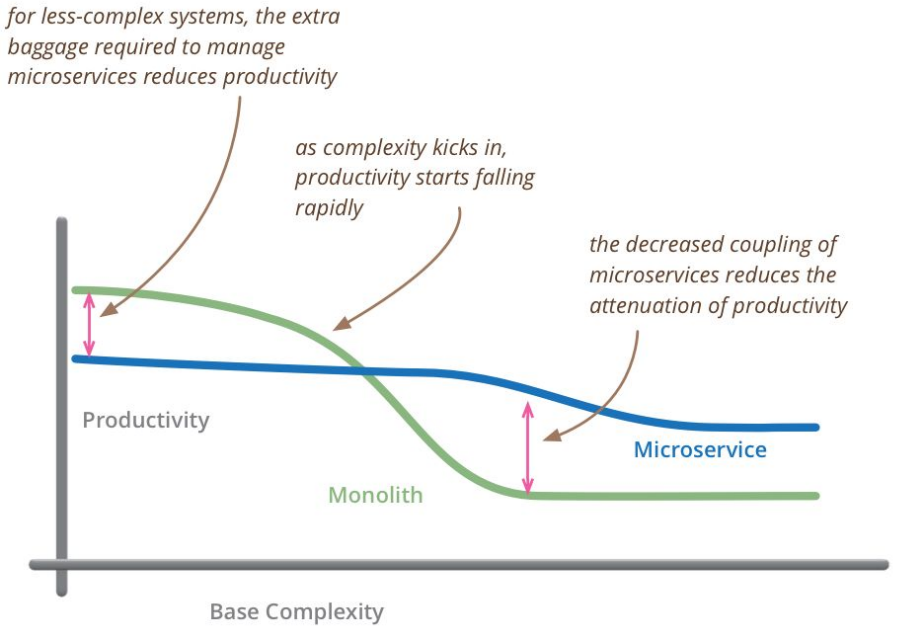
REVATURE

Special Concerns in MSA

- Service failure resiliency
- Monitoring of service health in production
- Scalability of services and their resources
- Security in auth and service communication

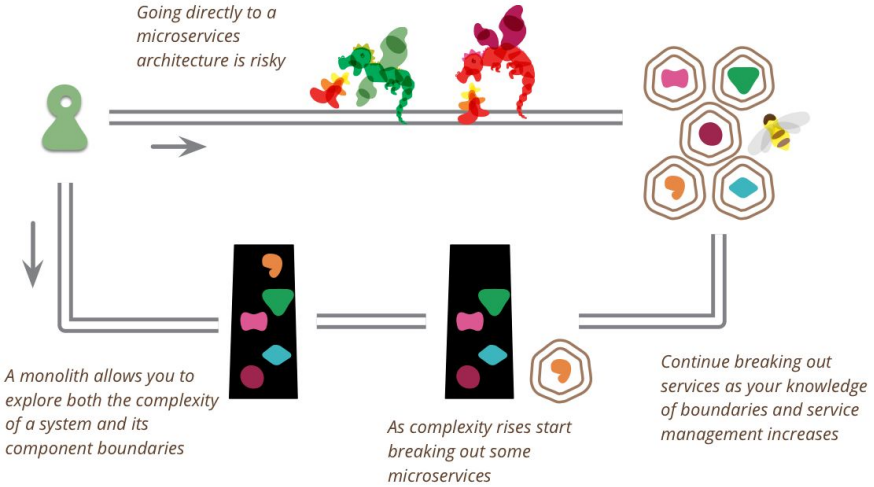
Container orchestration can help with those!

When not to use MSA



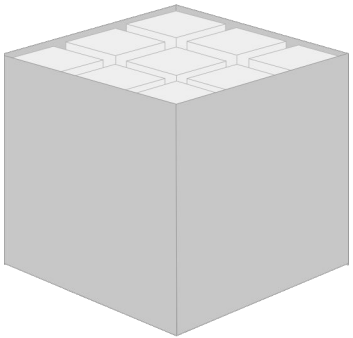
but remember the skill of the team will outweigh any monolith/microservice choice

Monolith + MSA?



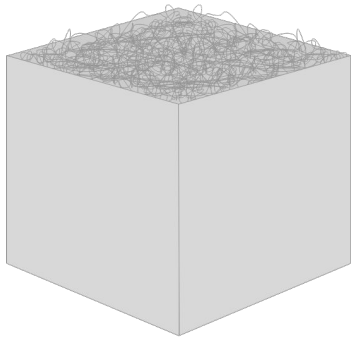
- Some say, start with a monolith and evolve it to MSA if and when needed
- Pros:
  - That's what most MSA success stories did
  - Do we *really* know where to draw all the service boundaries before we have an MVP?

Monolith + MSA?



Hope

vs.



Reality

- Some say, start with a monolith and evolve it to MSA if and when needed
- Cons:
  - The monolith's parts will inevitably be more tightly coupled to each other
  - Good module separation in a monolith might not be the same as good service boundaries

Things that go well with MSA

- Agile
- DevOps, CI/CD
- Containers/Docker
- Orchestration/Kubernetes
- Automated testing
- REST

MSA pioneers

- Netflix
  - Open source tools
  - Simian Army
  - High value on async communication between services

- Amazon
  - High value on devs owning product through production; devs = ops

Resources

- https://martinfowler.com/microservices/
- https://docs.microsoft.com/en-us/dotnet/architecture/microservices/