NOBODY TOLD ME ABOUT THAT

CHRONICLES ABOUT IMPLEMENTING MICROSERVICE ARCHITECTURE

I T.A.K.E.
Unconference

Khaled Souf
twitter: @khaledsouf

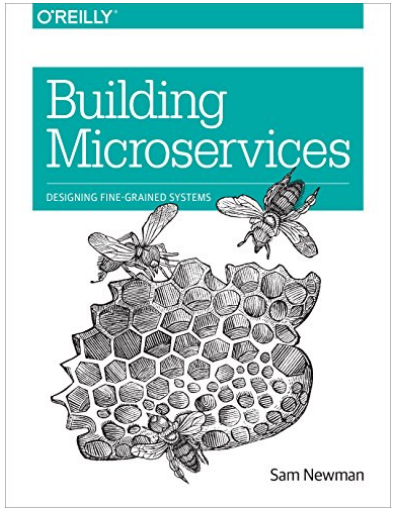
ABOUT ME

- Developer/Crafting Software Coach/ DDD Expert/Trainer
- Co-organizer of Software Crafters Montréal meetup
- Co-organizer of SOCRATES Canada unconference
- more details on : https://ksouf.com


DON'T START WITH
MICROSERVICE ARCHTECTURE
imgflip.com

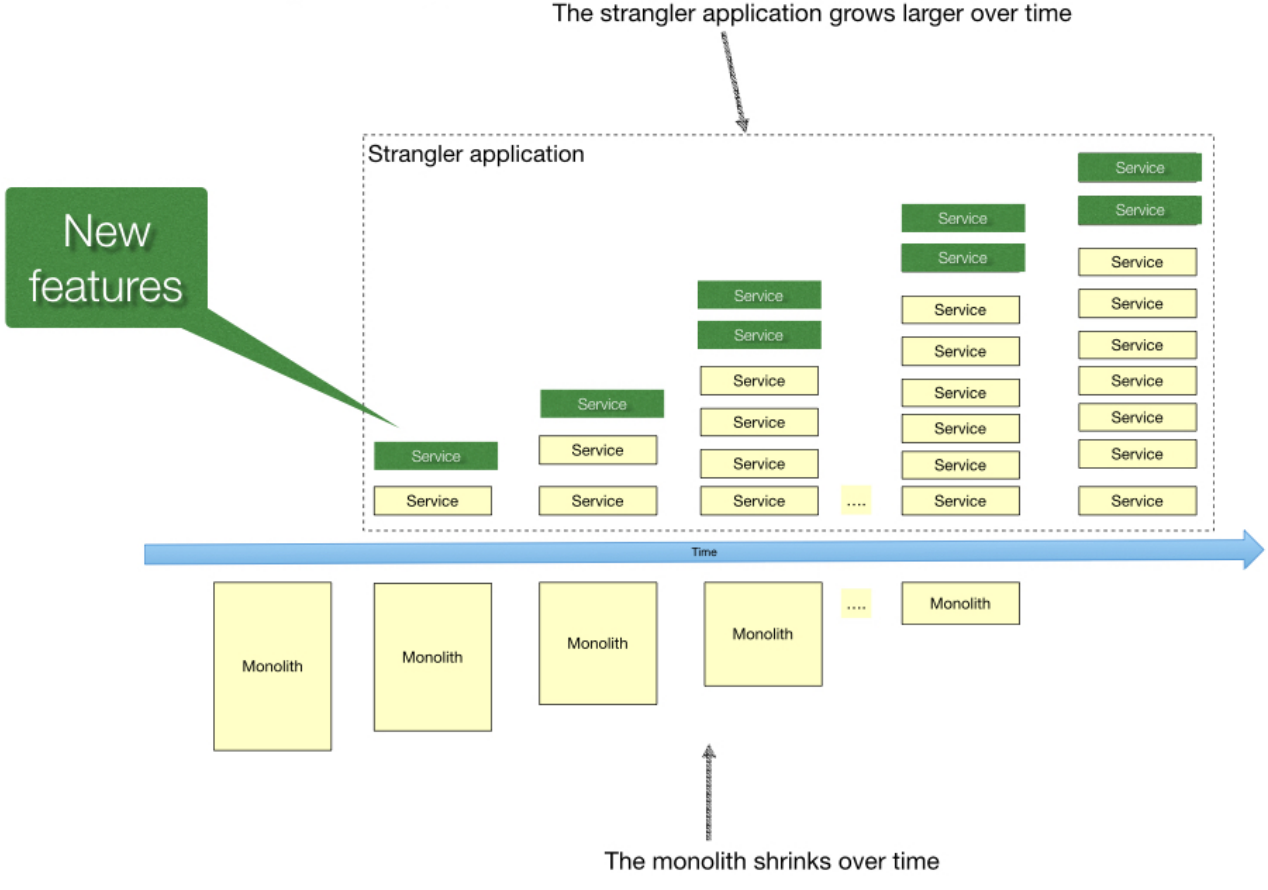

How do we start?
Monolith Architecture maybe?
imgflip.com

I remain convinced that it is much easier to partition an existing, "brownfield" system than to do so up front with a new, greenfield system. -- Sam Newman (Author of building microservices)



If you can't build a well-structured monolith, what makes you think you can build a well-structured set of microservices? -- Simon Brown (Author of Software Architecture for Developers)

Strangling the monolith



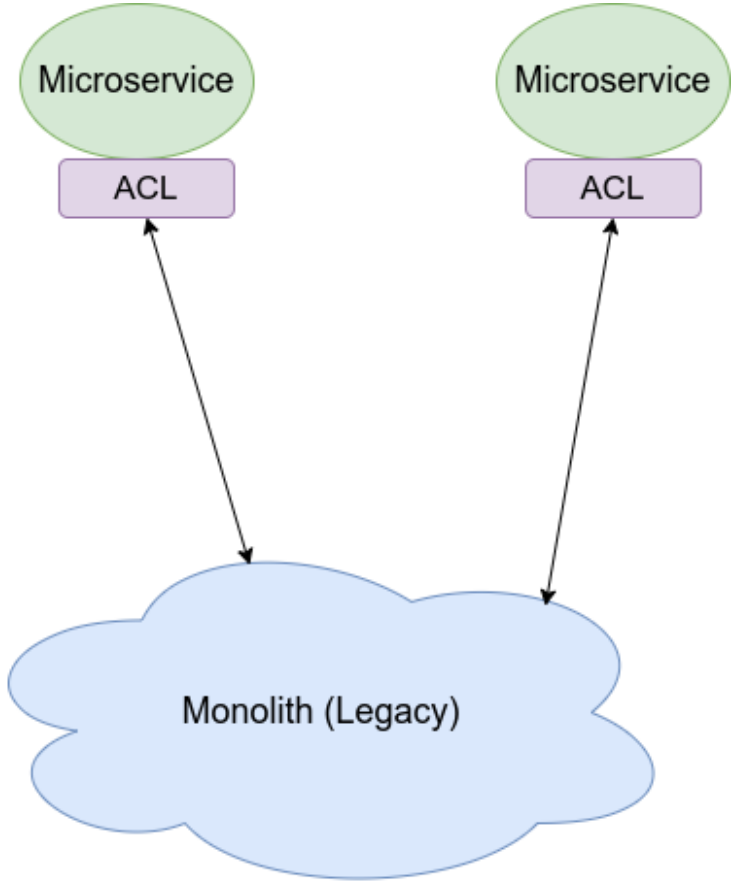
source microservices.io -- Chris Richardson (Author of microservices Patterns)

DOMAIN DRIVEN DESIGN TO THE RESCUE

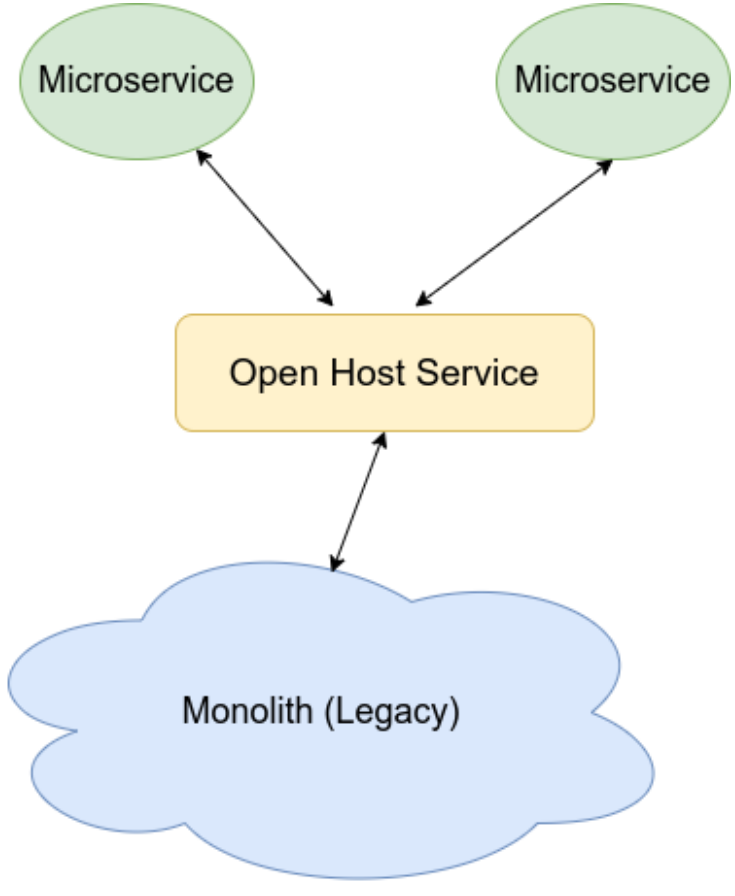
STRATEGIC DESIGN TO BE PRECISE

- Extract your Microservice from the monolith based on business capabilities
- Create New Feature on new Bounded Context (Bubble Context)
- Use some Context Mapping Patterns for communication

ANTI-CORRUPTION LAYER (ACL)



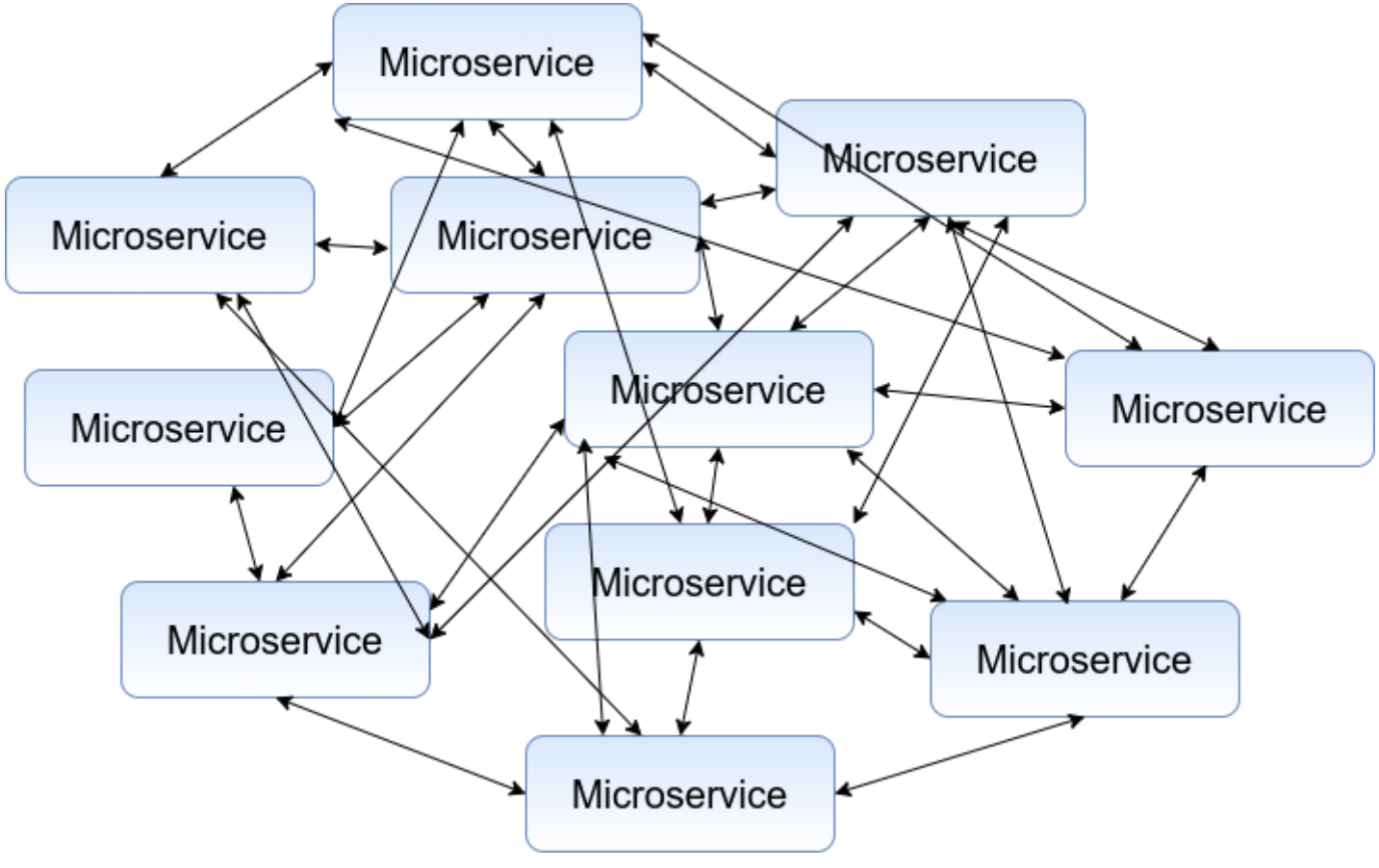
OPEN HOST SERVICE (OHS)



More details on: Applied Domain Driven Design On Legacy Code

Don't Make Microservices communicate with each other (Directly)!!

UNLESS YOU WANT THIS!!!

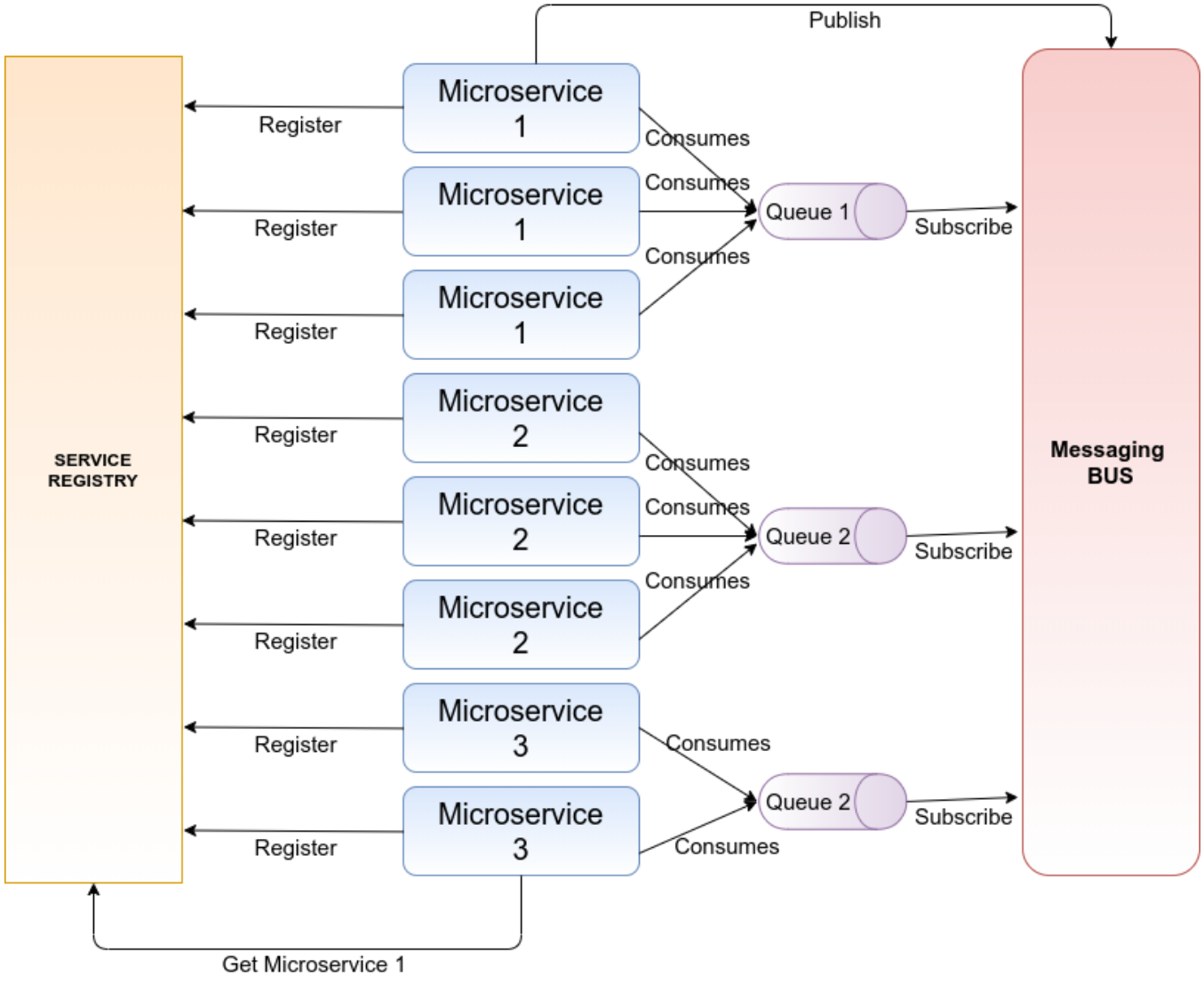



IT WAS A MONOLITH
38
BEFORE
imgflip.com

HOW WE DO AVOID IT?

- Service registry: every instance of service is registered and can be found only by the registry
- Microservices publish messages to the bus to chain actions
- Microservices consumes from queues that subscribed to specific event of the messaging bus
- Messages are used for asynchronous commands
- Registry is used to find API's (REST for the most) for querying

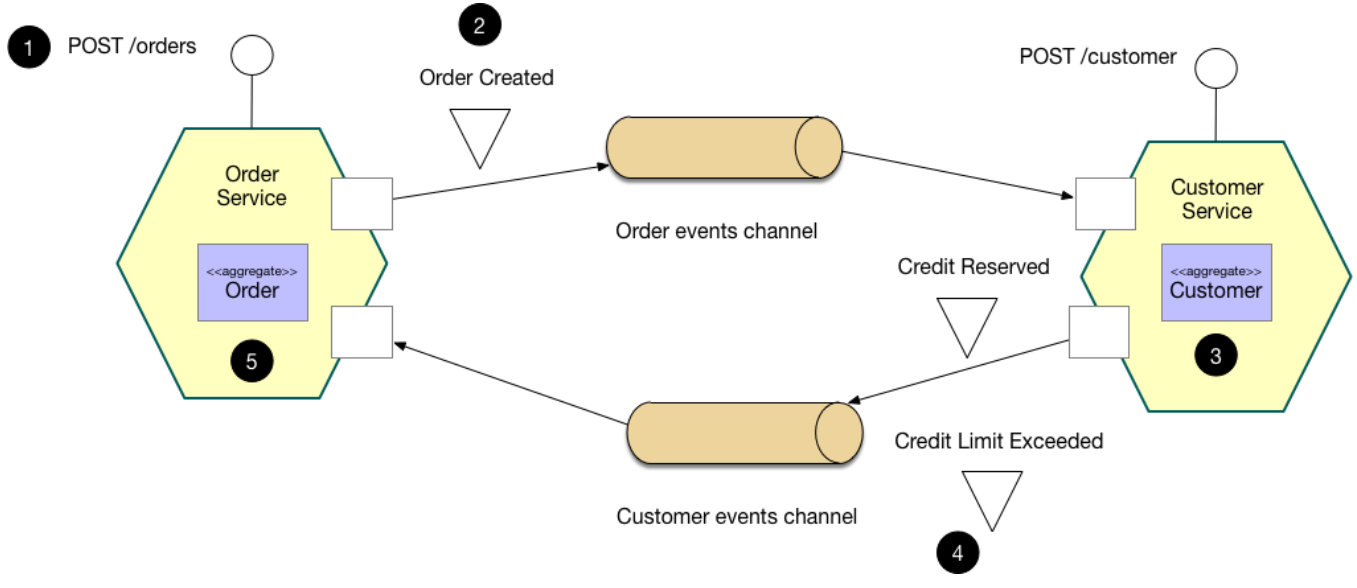
MICROSERVICE ARCHITECTURE (WITH REGISTRY & MESSAGING BUS)


Publish
SERVICE REGISTRY
Register
Microservice 1
Microservice 1
Microservice 1
Microservice 2
Microservice 2
Microservice 2
Microservice 3
Microservice 3
Consumes
Queue 1
Queue 2
Queue 2
Subscribe
Messaging BUS
Get Microservice 1

WHAT ABOUT DATABASES?

- Use choreography-based saga pattern (aka eventual consistency in Domain Driven Design).
- A Microservice manage his own data (shared database is Anti-Pattern).
- Keep two copies for starting (one in monolith/ one in microservice).

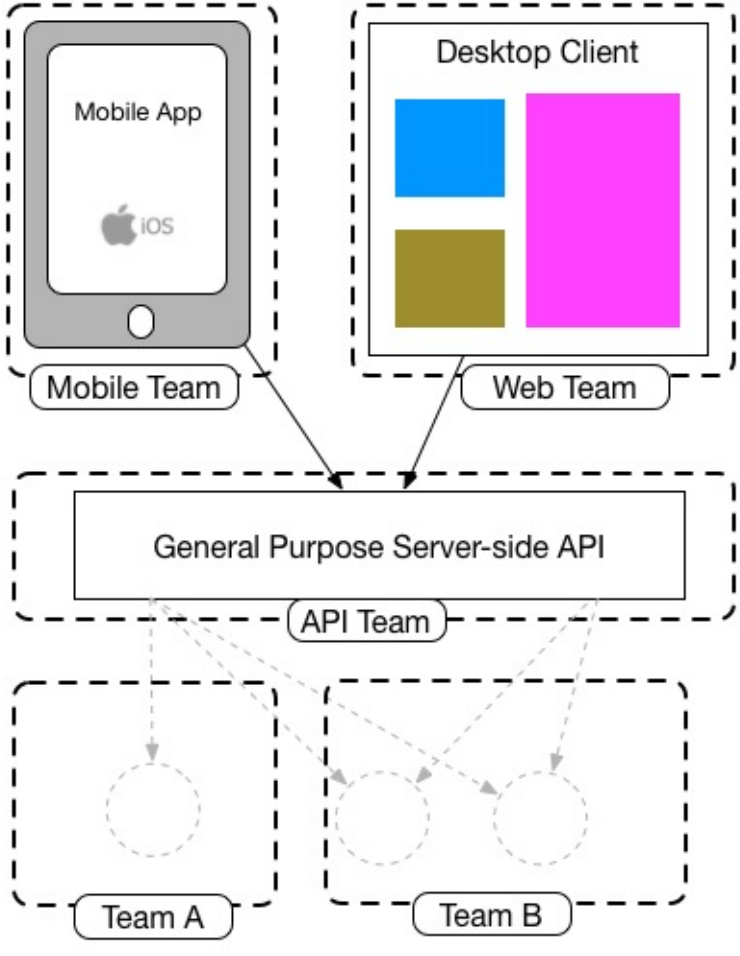
CHOREOGRAPHY-BASED SAGA PATTERN



BACK-END FOR FRONT-END (BFF)

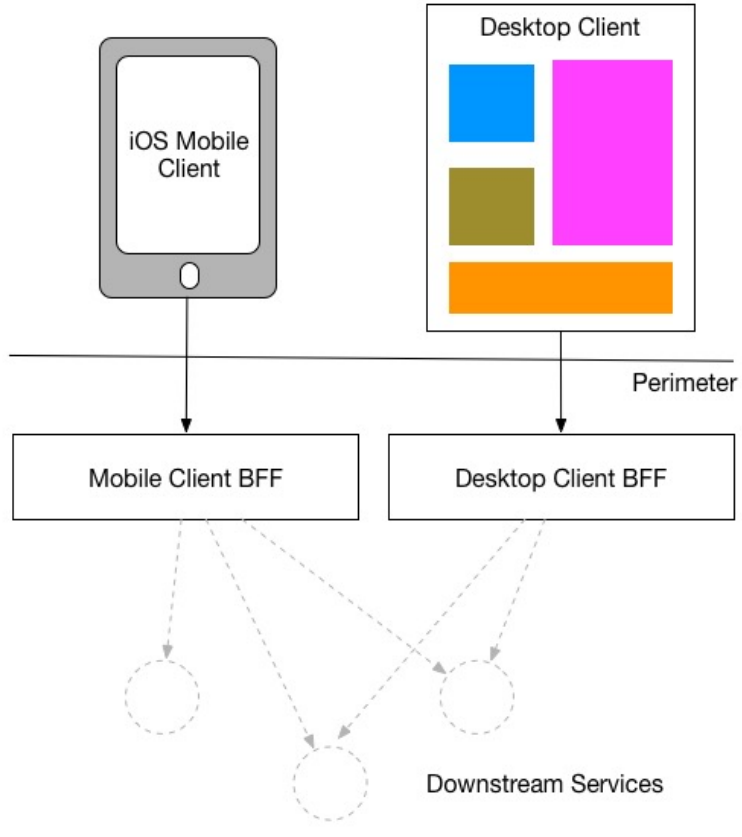
- Don't expose your Microservices directly
- Use BFF as view and information aggregator for general purpose or specific customer (depends on context)
- Use a gateway if you're exposing BFF to something external

BFF: GENERAL PURPOSE API



source samnewman.io

BFF: SPECIFIC PURPOSE API



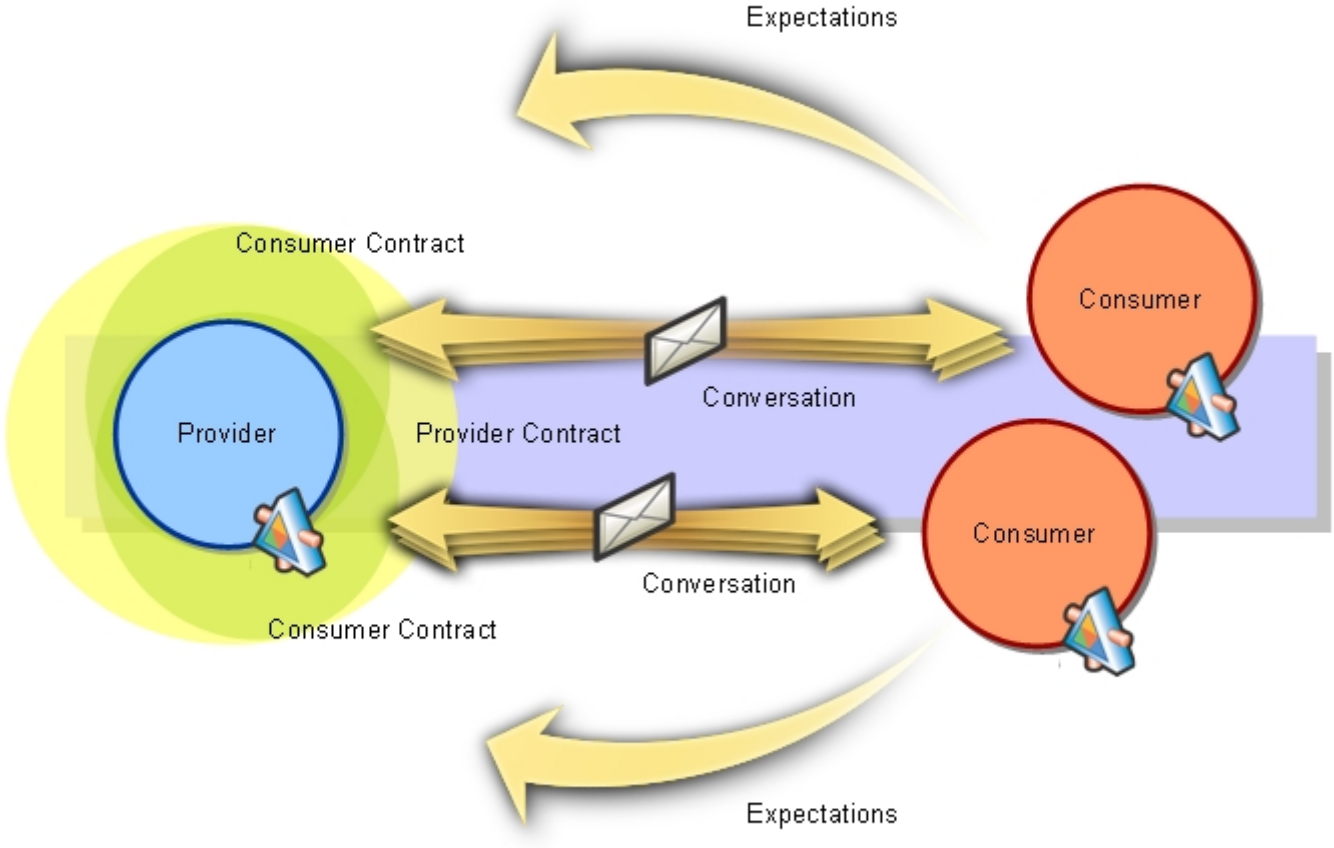
source samnewman.io

USE CONTRACTS TESTING FOR API

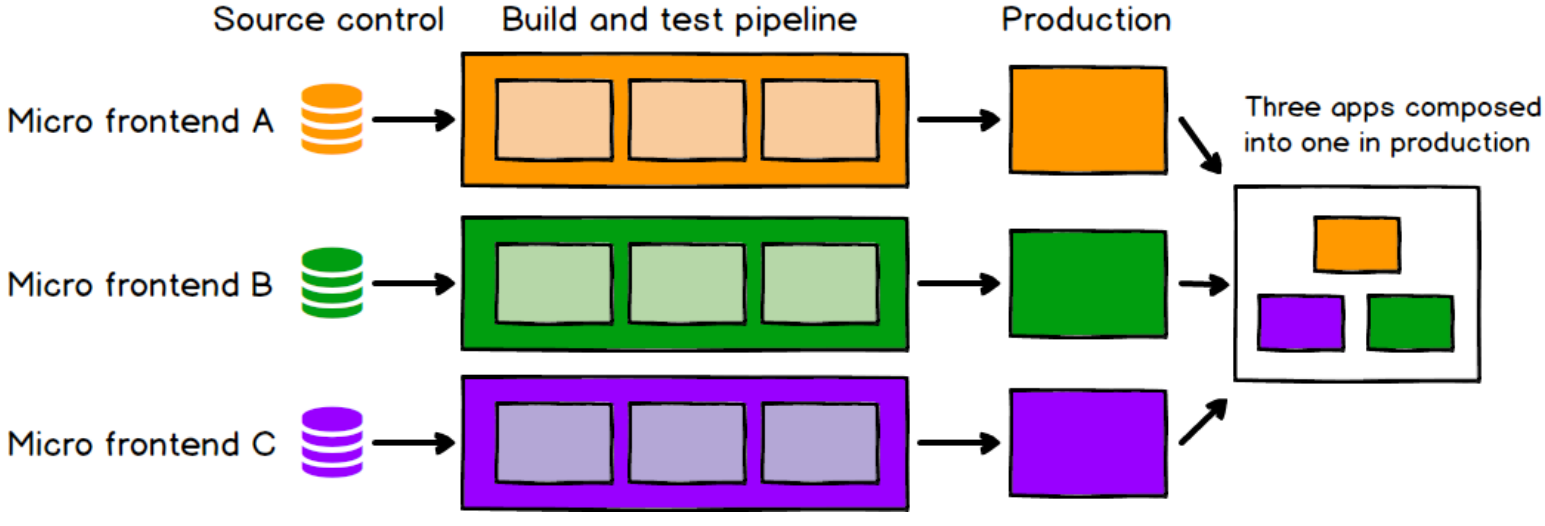
(INSTEAD OF END TO END TESTING)

- Let your consumer define the contract (Consumer Driven Contracts Testing)
- Use a contract broker to publish/verify your contracts

CONSUMER DRIVEN CONTRACTS TESTING



USE MICRO FRONTEND



INDUSTRIALIZE YOUR APPLICATIONS

- Create Template by Type of Project.
- Focus only Business Value when creating your Microservice.
- Define Template by used language and capabilities.
- Your Template should contain also you definition of build & deployment pipelines.

LOGGING

- Use Centralized logs solution
- Use structured logs
- Log your input/output of your Microservice
- Use Correlation Id to monitor cross Microservices requests

MONITORING

- Implement always a health check for your Microservice
- Centralize your metrics in one simple dashboard
- Use alerting system when any weird behavior

PROVISIONING/CONFIGURATION MANAGEMENT

- Use tools to manage configuration (Ansible, Saltstack, ...).
- build a git repository infrastructure as code to automate your provisioning.
- Make sure that the only possible modification of infrastructure is by modifying infra as code.

SECRETS MANAGEMENT

- Separate your Secrets from your configuration in a secrets server (Vault Server)
- Use key rotation technique to change keys periodically to prevent that a key is compromised

WARP-UP

- Don't start with microservices
- Use Strategic Design of Domain Driven Design and Strangler Pattern to split your monolith
- don't make microservices communicate directly (Service Registry for API Query, Messaging BUS for Command)
- Use Consumer Driven Contracts Technique instead of End to End Testing
- Centralize your cutting-edge concern (logging, monitoring, secrets, configuration, infrastructure)
- Automate as much as possible (infrastructure provisioning, config management, secrets management,build & deployment, ..)

BEFORE FINISHING: I DID NOT INVENTED ALL THESE TECHNIQUES!!!

- Domain Driven Design
- Microservices Patterns (The Book)
- Twelve Factor
- DevOps Practices

THANK YOU FOR ATTENDING

- Questions ?
- twitter: @khaledsouf