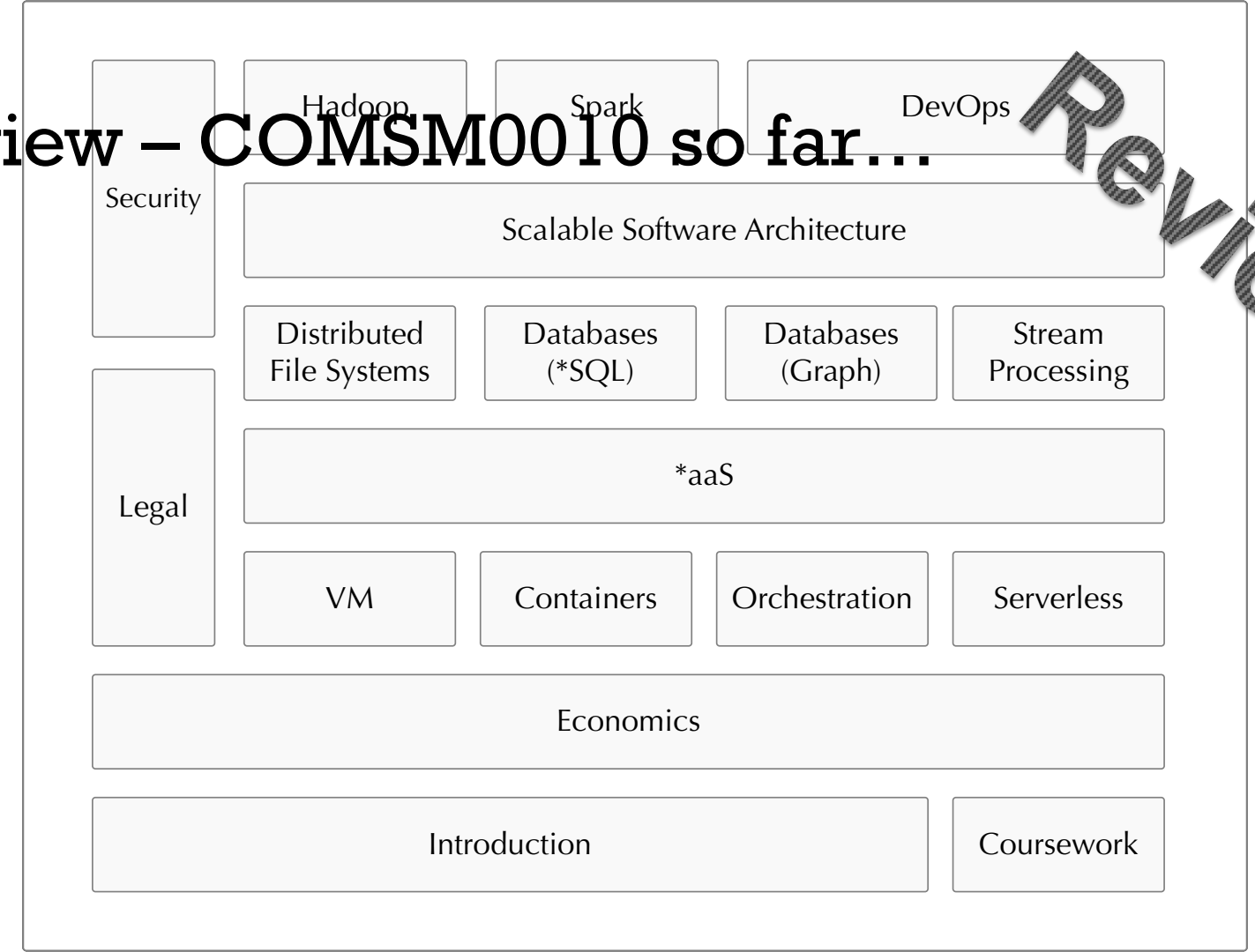
University of BRISTOL

Microservices and the CNCF

Dr Dan Schien – COMSM0010

Lecture 12

Review – COMSM0010 so far…



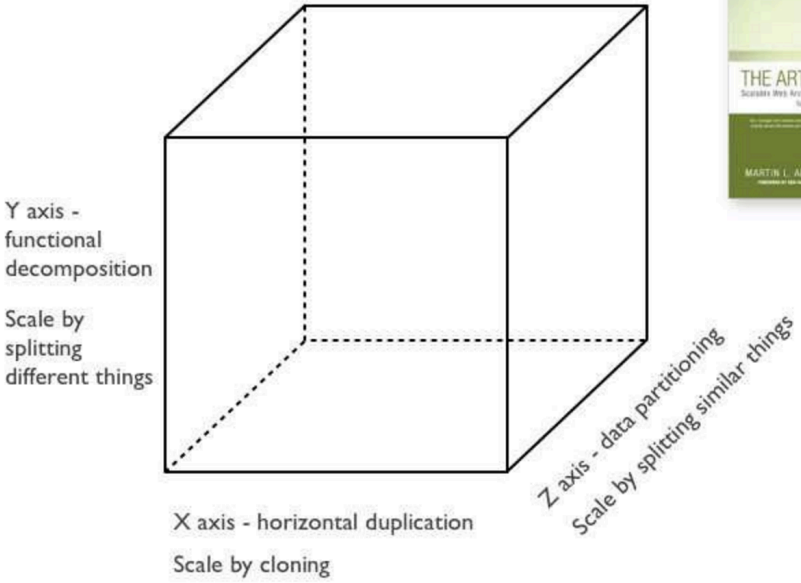
Review

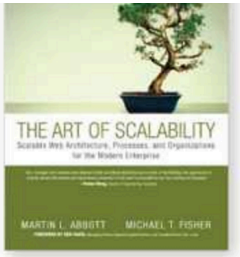
Learning outcomes

- Explain the architectural needs that Microservices address
- Describe the organizational goals that Microservices meet
- Understand some of the challenges that Microservices bring
- Recognize the role of the tools from the CNCF landscape
- Describe how observability of Microservices can be addressed

The Scale Cube (of Scaling Out – horizontally only)

Review

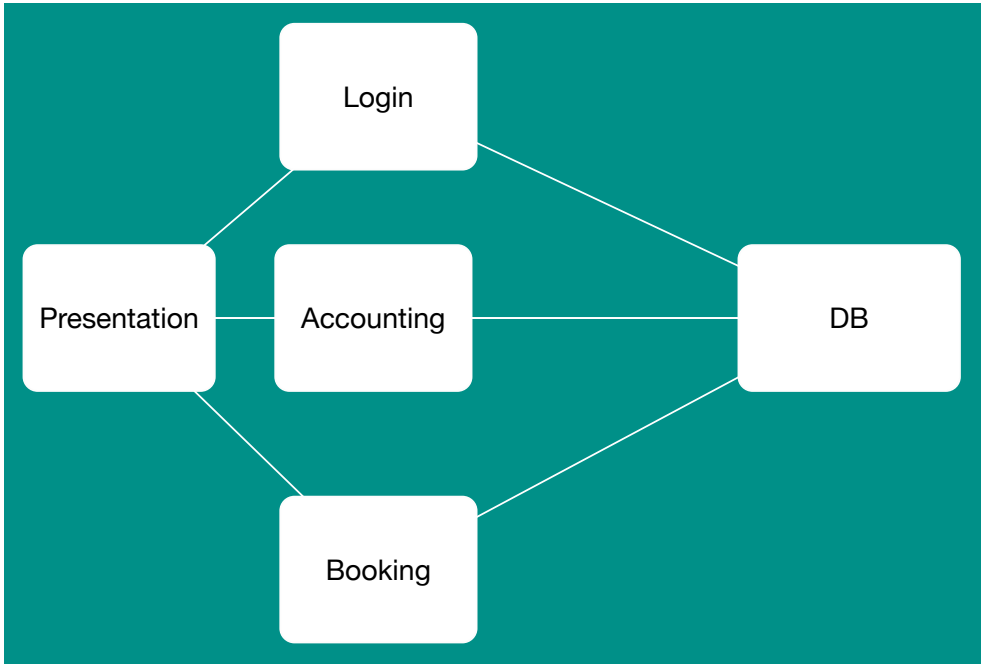




bristc

Y-Axis: decompose by function

Review



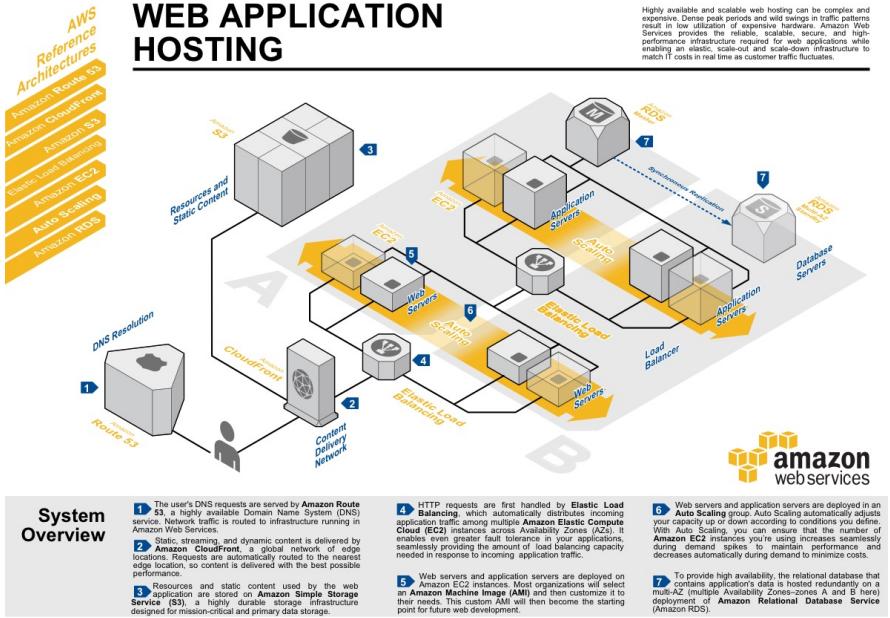
Decoupling of Components

Review

- Enables independent scalability
- Fault Tolerance and High Availability
- Four important mechanisms to **decouple**
  - Load Balancers
  - Message Queues
  - Message Topics
  - Service Registries

Architectural Patterns

Review



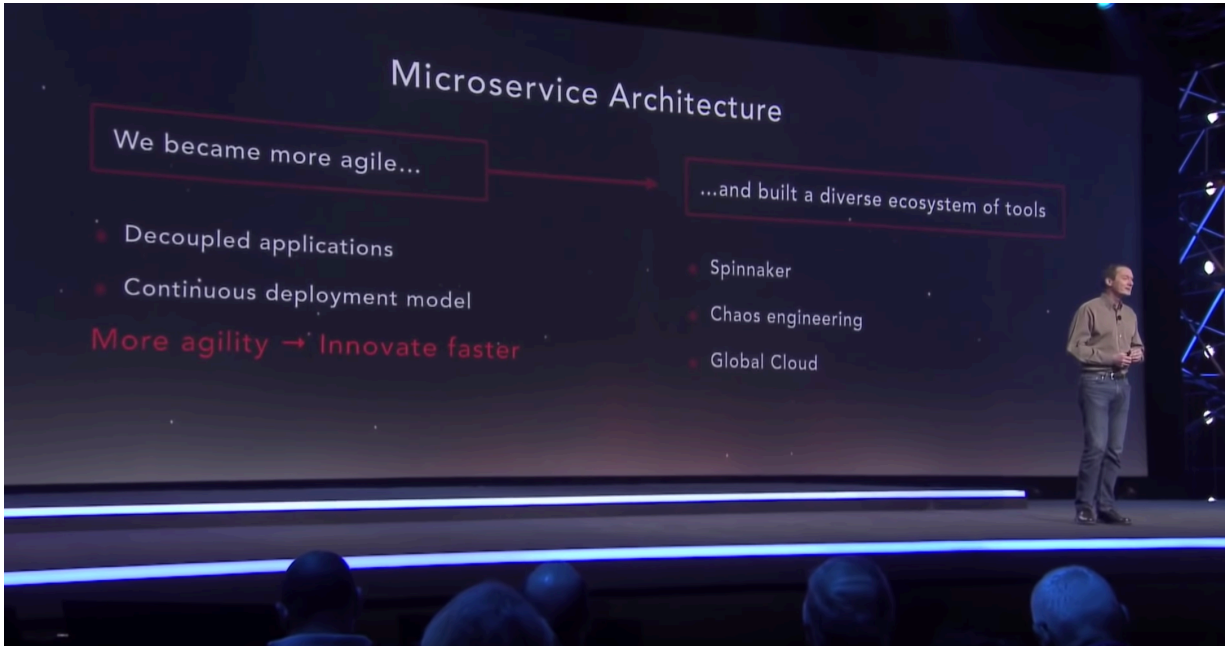
https://aws.amazon.com/architecture

What is Cloud Native?

- Container based
  - Capabilities
- Elastic
- Microservices
  - Polyglot
- DevOps

AWS Biggest Customer?

- Netflix
- >100.000 instances
- +500 microservices
- 1 application change per second

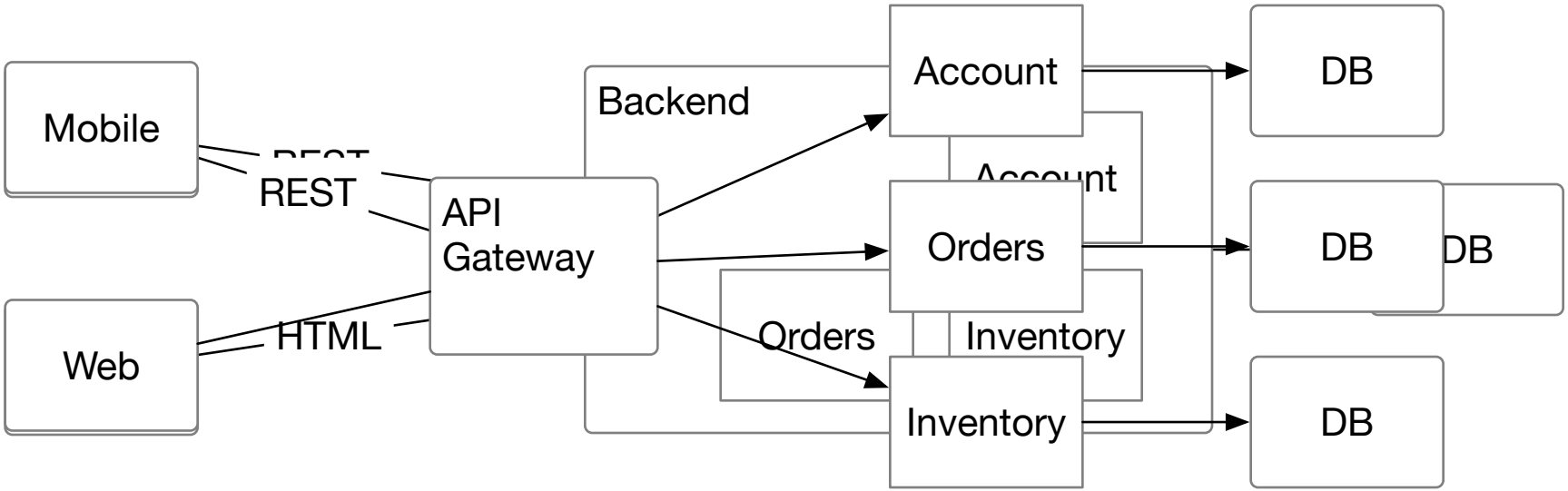


What Are Microservices?

- Martin Fowler first used the term 2014
- Opposite “monolithic” applications
- Organisational agility
  - Rolling releases
  - Hot swapping
  - No down time

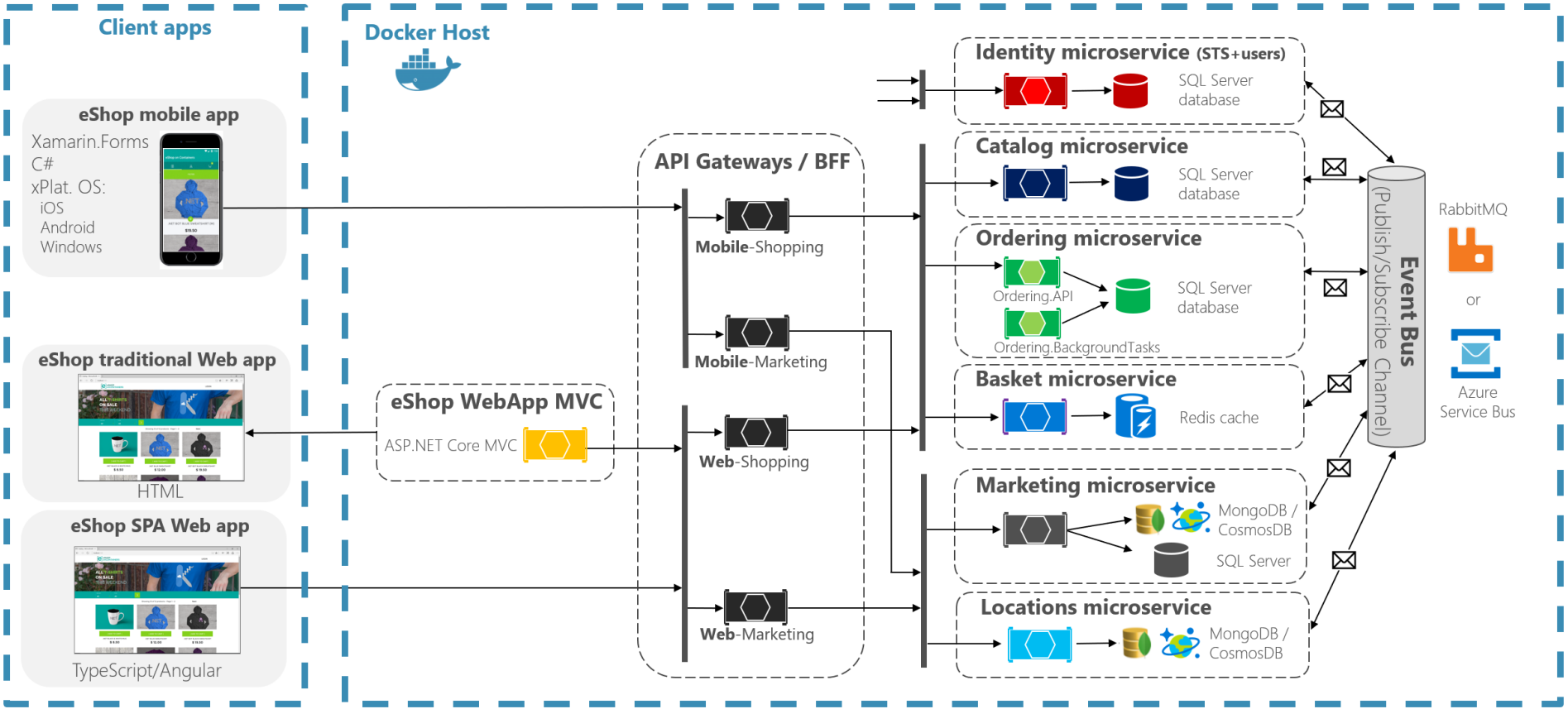
- Through
  - Horizontal Auto-Scaling
  - Design for Failure
    - Replication, Health Monitoring
  - Modularity
    - Decoupling
    - Containerised
  - API contracts
    - Discovery
  - Automation
    - DevOps
    - Observability

Breaking up the monolith



eShopOnContainers reference application

(Development environment architecture)



bristol.ac.uk

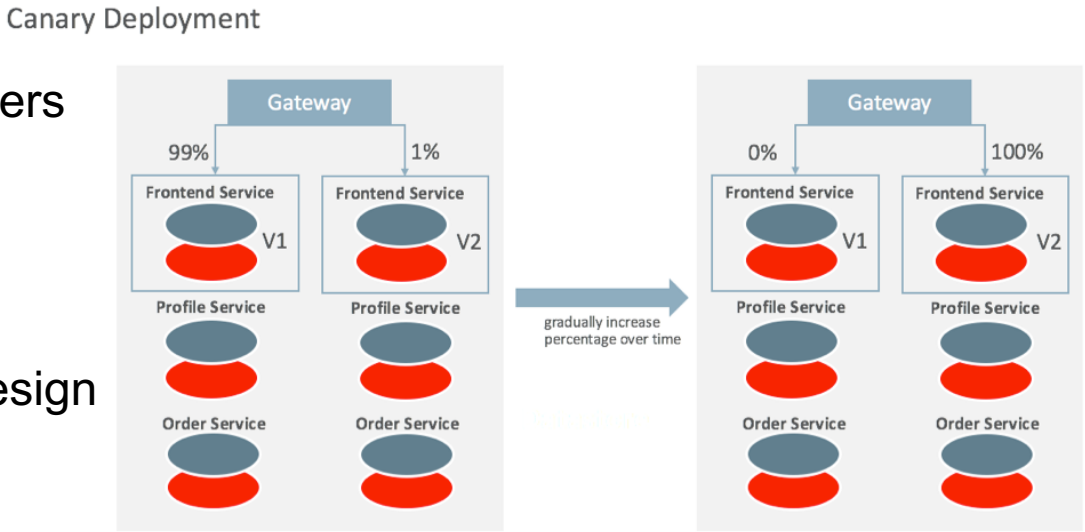
https://docs.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-micros

Upsides

- Strong module boundaries — great for large teams
- Independent deployment
  - Simple services are easier to deploy, and since they are autonomous, are less likely to cause system failures when they go wrong.
- Technology Diversity:
  - With microservices you can mix multiple languages, development frameworks and data-storage technologies.
- Re-use services

Service deployability

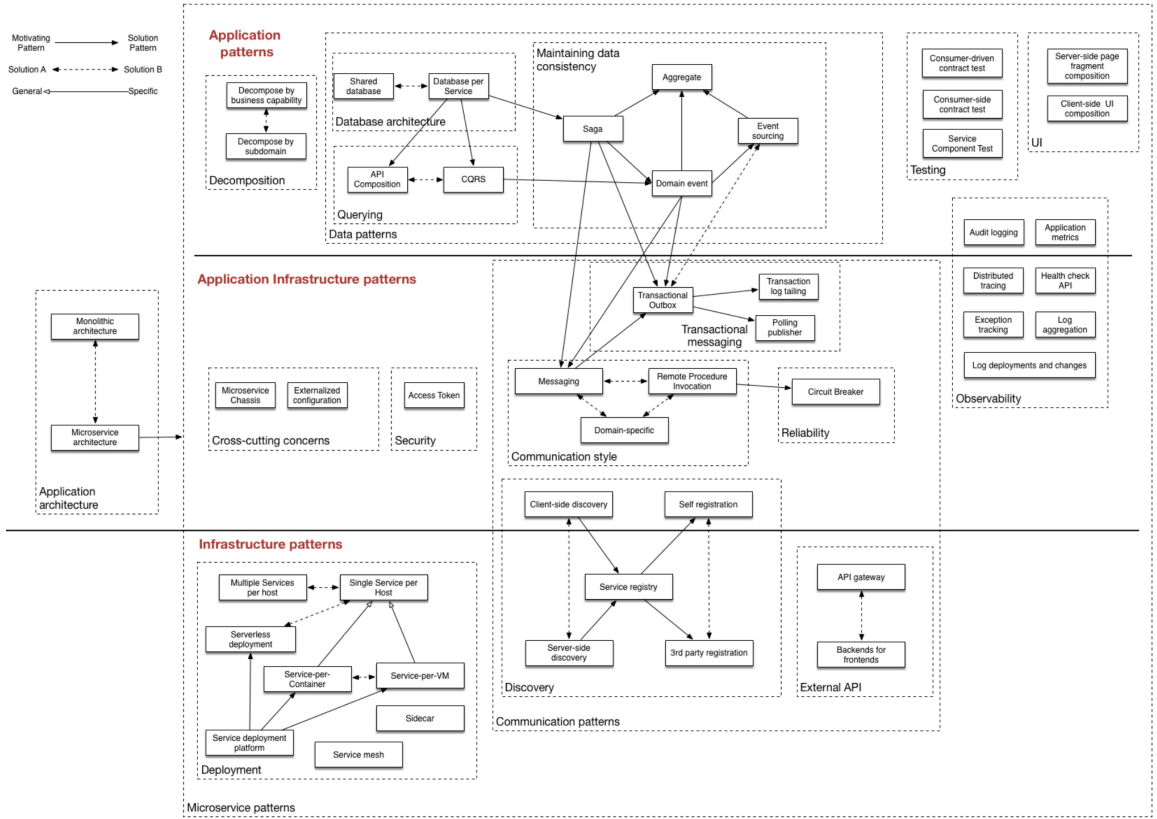
- Canary
  - a small percentage of users
- Blue/Green
  - Standby prod env
- A/B
  - Empirically evaluate a design



https://blogs.oracle.com/developers/getting-started-with-microservices-part-four

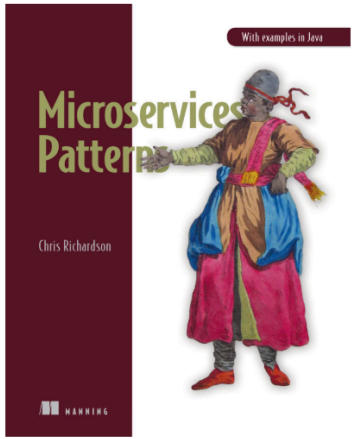
Downsides

- Distribution:
  - Distributed systems are harder to program, since remote calls are slow and are always at risk of failure.
- Eventual Consistency:
  - Maintaining strong consistency is extremely difficult for a distributed system, which means everyone has to manage eventual consistency.
- Operational Complexity:
  - You need a mature operations team to manage lots of services, which are being redeployed regularly.



Patterns and Strategies

- Chris Richardson
- microservices.io

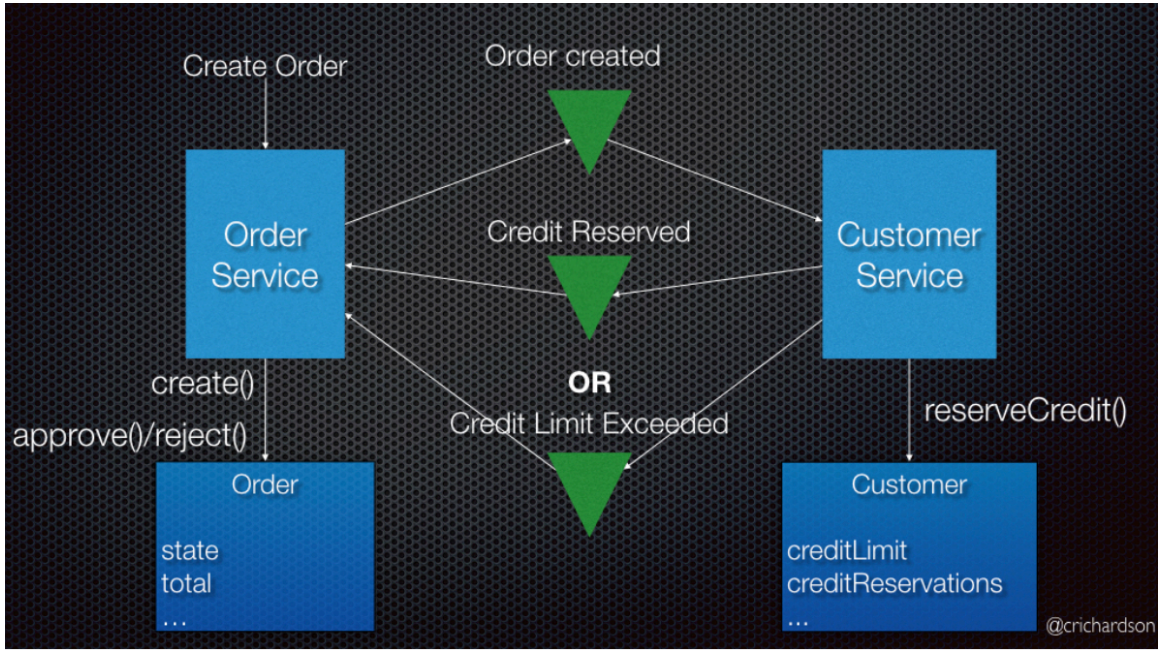


Data locality

- Each micro service has own DB - Limit access to peer service data to API only
- Org level: Allows each service to manage data in the way that makes the most sense for that service
- Performance benefit: caching data locally in microservices enables faster response
- Challenges:
  - invariants that span multiple services; query data that is owned by multiple services
  - Implementing business transactions that span multiple services is not straightforward
  - Avoided Distributed transactions (-> CAP theorem lecture)

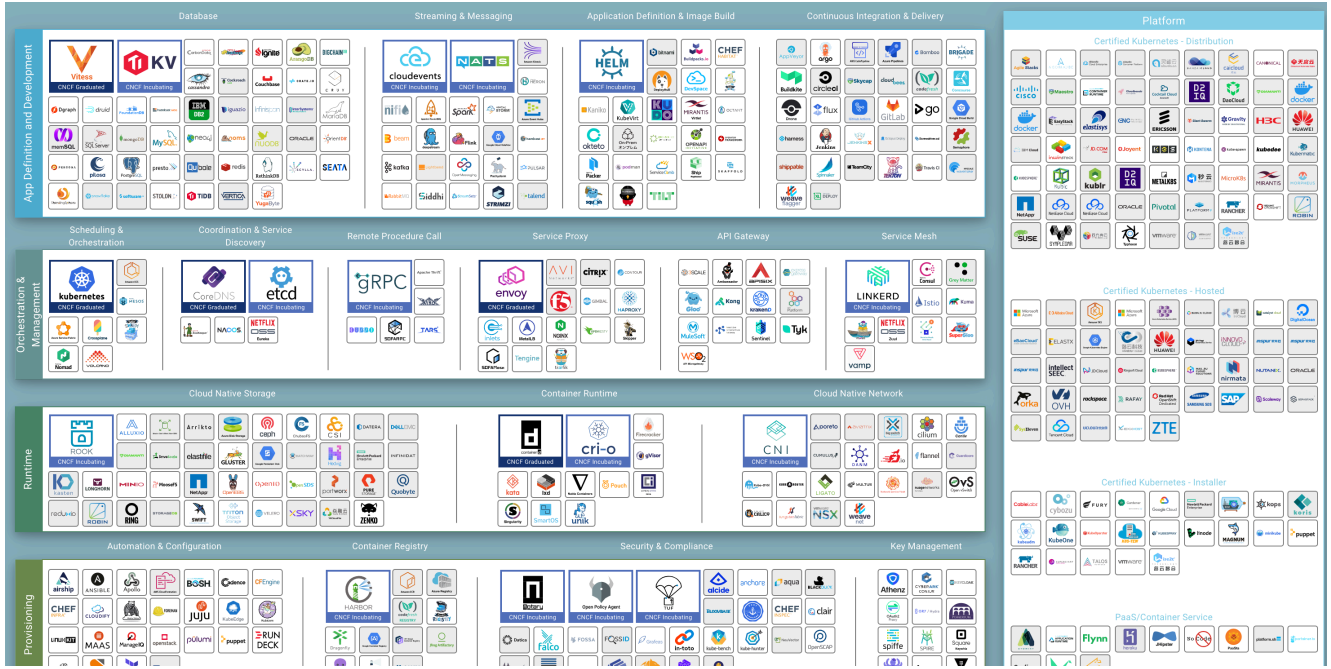
Eventual consistency

- “no shared database” —> share state through HTTP
- No guarantee that those updates occur immediately
- Guarantee that they occur eventually



Saga Pattern -- https://microservices.io/patterns/data/saga.html

CNCF Landscape



bristol.ac.uk

CNCF Project Clusters

- Database
- Streaming & Messaging
- Observability and Analysis
  - Monitoring
  - Logging
  - Tracing
  - Chaos Engineering
- Continuous Integration & Delivery
- Scheduling & Orchestration
- Coordination & Service Discovery
- Remote Procedure Call
- API Gateway
- Service Mesh
- Cloud Native Storage
- Container Runtime
- Cloud Native Network
- Automation & Configuration
- Container Registry
- Security & Compliance
- Key Management
- Service Proxy

CNCF – Graduated projects

- Kubernetes
- Prometheus (9.8.2018)
  - monitoring system and time series database
- Envoy
  - Service Network Proxy
- CoreDNS
  - DNS server and Service Discovery
- Containerd (28.2.2019)
  - Container Runtime
- Fluentd (11.4.2019)
  - Logging
- Jaeger (4.11.2019)
  - Monitoring and tracing
- Vitess (6.11.2019)
  - Database
  - Sharded MySQL

Observability

- What is going, was and will happen
- *"Observability is a measure of how well internal states of a system can be inferred from knowledge of its external outputs"*

- Logging
- Monitoring & metrics
- Diagnostics
- Health checks
- Debugging
- Fault injection

Logging

- Local log files
  - Grows potentially unlimited (log rotate)
  - Ephemeral storage
- Aggregated Logs
  - Syslog
  - Central server
    - ElasticSearch, Logstash, Kibana

- ELK in Kubernetes
- side car logstash container
- Application level or platform level
- Fluentd

Metrics

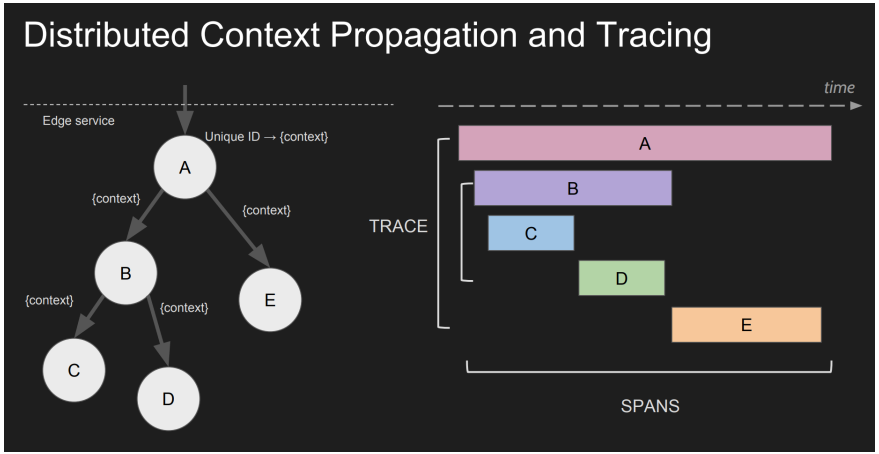
- More than logs
  - CPU, load average, interrupts
  - Free memory
  - Network packets received, transmitted, dropped
  - Disk space
- Identify bottlenecks
- Monitor the utilisation
- Identify hardware failures
- diagnosing

- TSDB
  - Efficient storage and indexing
  - Very compressable
- Prometheus
- Kubernetes metrics-server
  - Runs as a pod
  - Connects to the kubelet to retrieve pod and node cgroup information.
  - Exposes /metrics endpoint to be scraped by Prometheus

Jaeger

- Open sourced by Uber
- Logs contain messages from many requests -> hard to tell a story
- OpenTracing
- Trace single request across your microservices
- Context per request, propagate across call boundaries

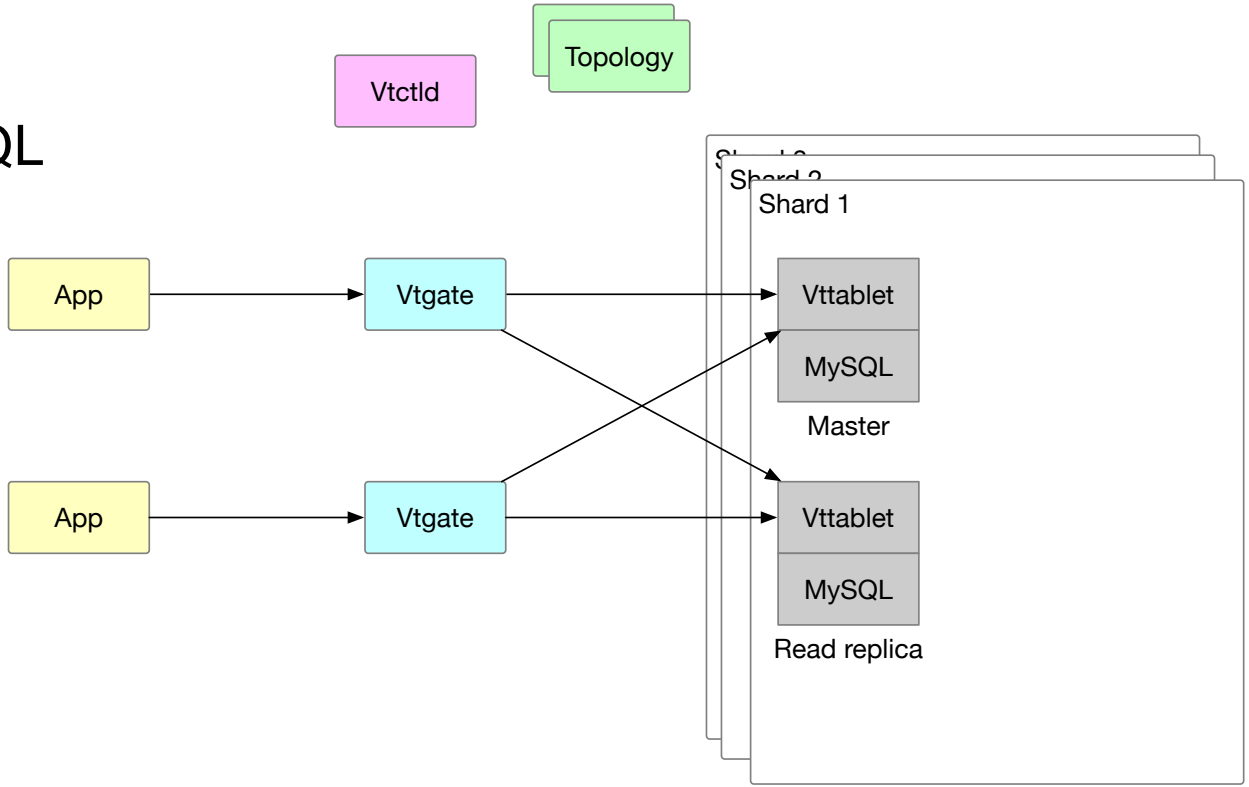
```
io.opentracing.Tracer tracer = ...;
...
Span span = tracer.buildSpan("someWork").start();
```



- Aggregated logs & profiling per request and process

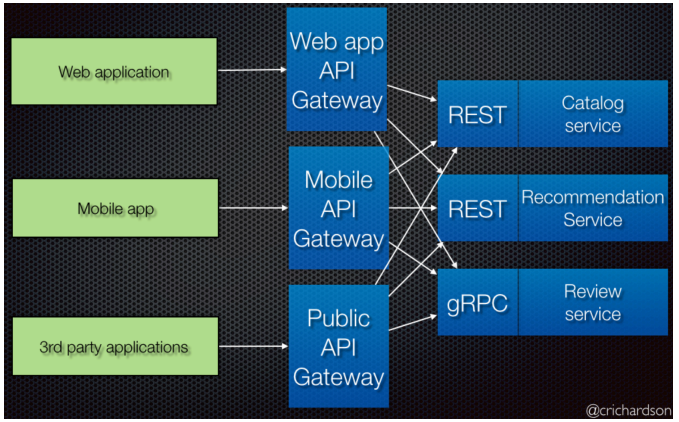
Vitesse

- Sharding for MySQL
- HA
- Pinterest, Slack, Github

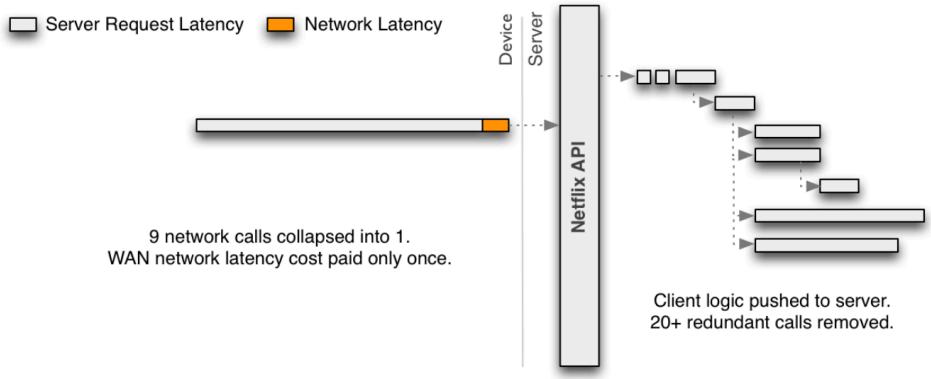


API Gateway

- Manage granularity of APIs
- Hide services from clients
- Hide diverse set of protocols
- Open API Spec
  - Integrated design, documentation, and implementation
- Manage network latency



https://microservices.io/patterns/apigateway.html



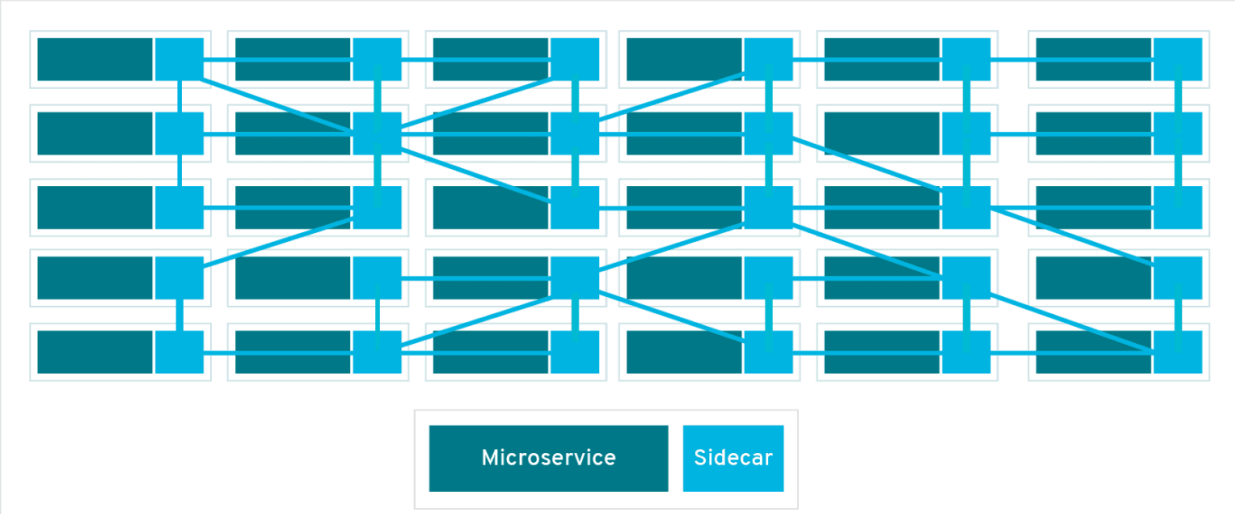
Chaos Monkey

- For three days in August 2008, Netflix couldn’t ship DVDs to customers because of a major database corruption.
  - Moved to a more resilient architeture
- Chaos Monkey randomly takes virtual machines
- Netflix closed all their DCs
- Netflix keeps backups of everything in Google Cloud Storage

Service Meshes

LINKERD

- Networking model
- requests are routed between microservices through proxies in their own infrastructure layer.
- Sidecar pattern (Burns et al.)
- Security and Identity
- Routing Management
- Proxy collects data
  - Performance
  - Locates problem pods
  - Applies patterns
    - circuit-breaking,
    - latency-aware load balancing,
    - eventually consistent (“advisory”)
    - service discovery,
    - retries, and deadlines



Thanks!