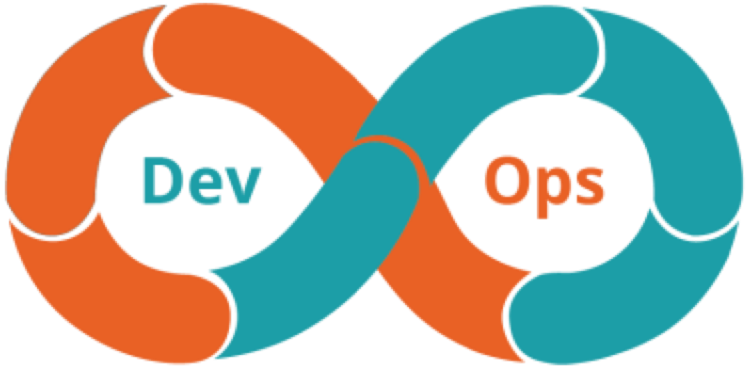
Microservices Engineering Boot Camp



Welcome!

Logistics (breaks, facilities, lunch, etc.)

Rules of Engagement

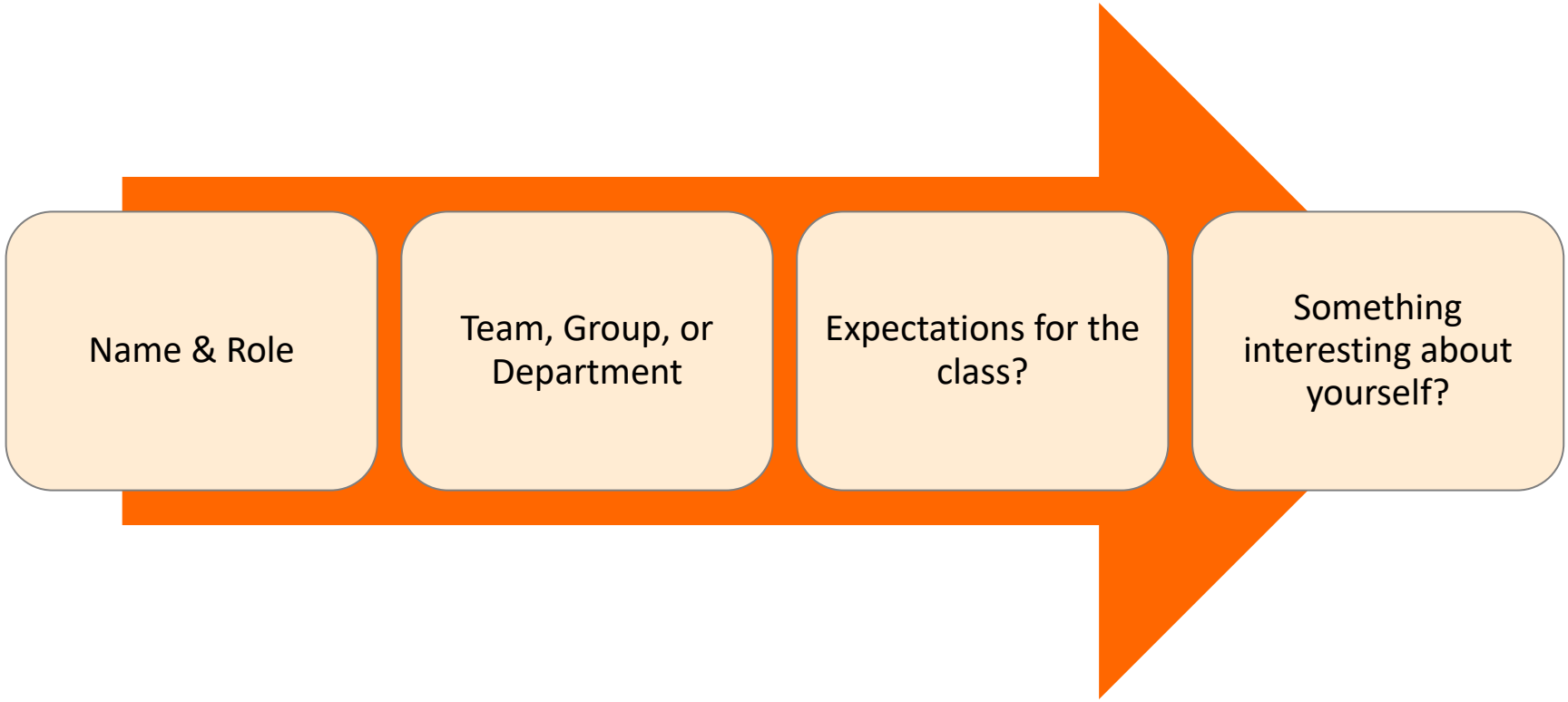
Introductions

Let’s Get Started!

Who is your instructor?

A little about me…

Introductions



Core Objectives

- Understand how to adopt, plan or improve your transition to Microservices
- Navigate different tools for enabling Microservices and how to use them
- Map technical practices to the business strategy behind Microservices
- Get hands-on practice with tools which enable core Microservices architecture
- Build more DevOps practices through Microservices adoption
- Apply Microservices use cases to continuous integration, delivery and testing
- Understand how to refactor monolithic systems into more modular, component-based systems

What to Expect from this Class

Flexibility

Conversations

Literacy and awareness on many of the principles, tools, and practices surrounding Microservices culture, architecture and implementation.

An effort to focus on your own situations and challenges so you can act on what you learn.

Introduction to Microservices

Part 1

Microservices is defined as a loosely-coupled, service-oriented architecture with bounded context.

The Journey is a “Team of Teams”

“First I needed to shift my focus from moving pieces on the board to shaping the ecosystem.”

– General S. McChrystal, *Team of Teams: New Rules of Engagement for a Complex World*

McChrystal (US Army General) understood that the only way to scale the agility and speed of a small team to a scope that could be employed in a hierarchical organization of many thousands of people was to employ radical transparency and independent decision-making power that was decoupled from the approval of the command structure. This “team of teams” approach reduced silos and flattened the organization, allowing much greater speed and adaptability.

Situation: Enterprise Team Building an Enterprise App

Lets imagine a medium sized enterprise app that has a team with roughly

- **12 engineers**
- **2 project managers**
- **2 scrum masters**
- **3 database engineers**
- **8 UI engineers**
- **2 engineering managers**
- **1 UI engineering manager**
- **1 QA manager**
- **1 database engineering manager**
- **2 directors**
- **4 QA**

36 individuals total, with 1/3 partially allocated to this and other projects

DevOps Foundation
- Toyota Production System

8 Wasters

Situation: Enterprise Team Building an Enterprise App

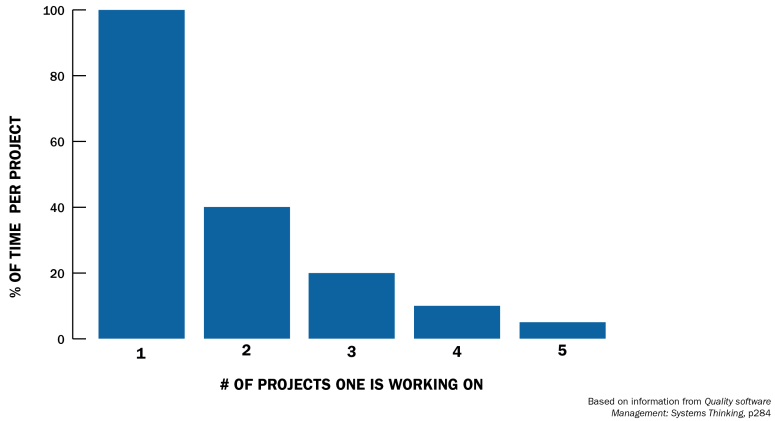
Lets imagine a medium sized enterprise app that has a team with roughly

- **12 engineers**
- **2 project managers**
- **2 scrum masters**
- **3 database engineers**
- **8 UI engineers**
- **2 engineering managers**
- **1 UI engineering manager**
- **1 QA manager**
- **1 database engineering manager**
- **2 directors**
- **4 QA**

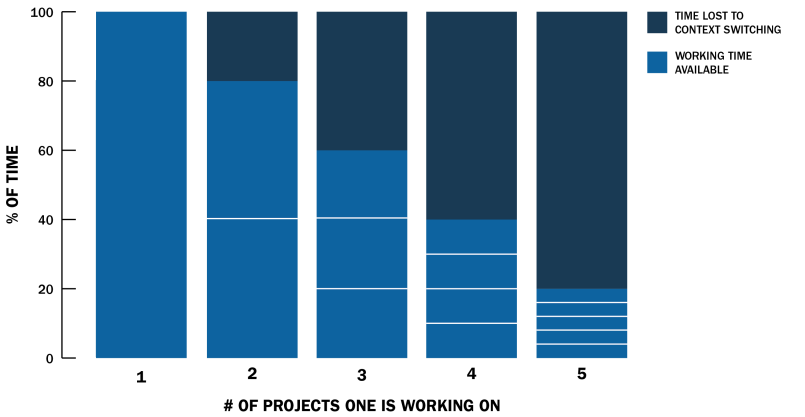
36 individuals total, with 1/3 partially allocated to this and **other projects**

Complication: Context Switching is Expensive

- According to the Software Engineering Institute at Carnegie Mellon University, **context switching** is **extremely expensive.**
- Switching projects results in rapidly diminishing results
- By implication, engineers manually tracking dependencies and switching to each manual task required for a specific service is similar in its impact on productivity
- Given that dependencies assignments are clustered with dev leads handling 3-7 each, productivity can be heavily impacted



Based on information from *Quality software Management: Systems Thinking*, p284



Based on information from *Quality software Management: Systems Thinking*, p284

- At 2 different tasks or projects, an engineer is losing 20% of their time to overhead. At 3 overhead jumps to 40%.
- Any value over 3 results in losing the majority of a resource's time to overhead and context switching
- Frequently, staff is assigned to multiple projects and jumps between meeting, reporting and development activities

Complication: Context Switching is Expensive

DevOps practices can help guard against some of the pitfalls of context switching, as well as alert the team when context switching is impacting product quality and team productivity. By leveraging continuous integration, any build failure will alert the team members when their contributions are impeding application or feature development. Likewise, automating the assignment of code reviews can insure code committed meets proper style and security standards.

According to Lewis and Fowler, a good way to overcome this hurdle – which resides among microservices and with external APIs – is through **domain-driven design**. DDD divides and re-bundles complex domains into multiple bounded contexts, which then maps out the relationship between them.

Before diving into Microservices, let's go through few aspects of DevOps.

DevOps – First Way – Optimize Flow

DevOps – Second Way – Amplify Feedback

DevOps – Third Way – Continual Learning

Your DevOps Journey: Optimize Flow – First Way

- Infrastructure As Code
- Deployment Pipeline
- Automated Testing
- Continuous Integration
- Containerization/Microservices
- Architect for Low-Risk Releases

Your DevOps Journey: Amplify Feedback – Second Way

- Telemetry: Metrics, Monitoring and Alerting
- Use Telemetry to Anticipate Problems
- Feedback for Safe Deployment of Code
- Hypothesis-Driven Development
- Change Review and Coordination

Your DevOps Journey: Continual Learning – Third Way

- Learning culture
  - Blameless Postmortems
- Innovation culture
  - Rehearsing Failures
  - Knowledge Sharing
  - Reserve Time for Organizational Learning

Situation: Executive leadership wants a roadmap, time to plan!

- Fly everyone to one location
- Lock the team in a meeting room for 4-5 days
- Size and plan user stories for the next 10, 15, or 20 weeks using scaled agile or another agile methodology
- Deliver roadmap to leadership!

Complication: Dependencies

Typical app in a good case will have approx. 15 dependencies on other development groups

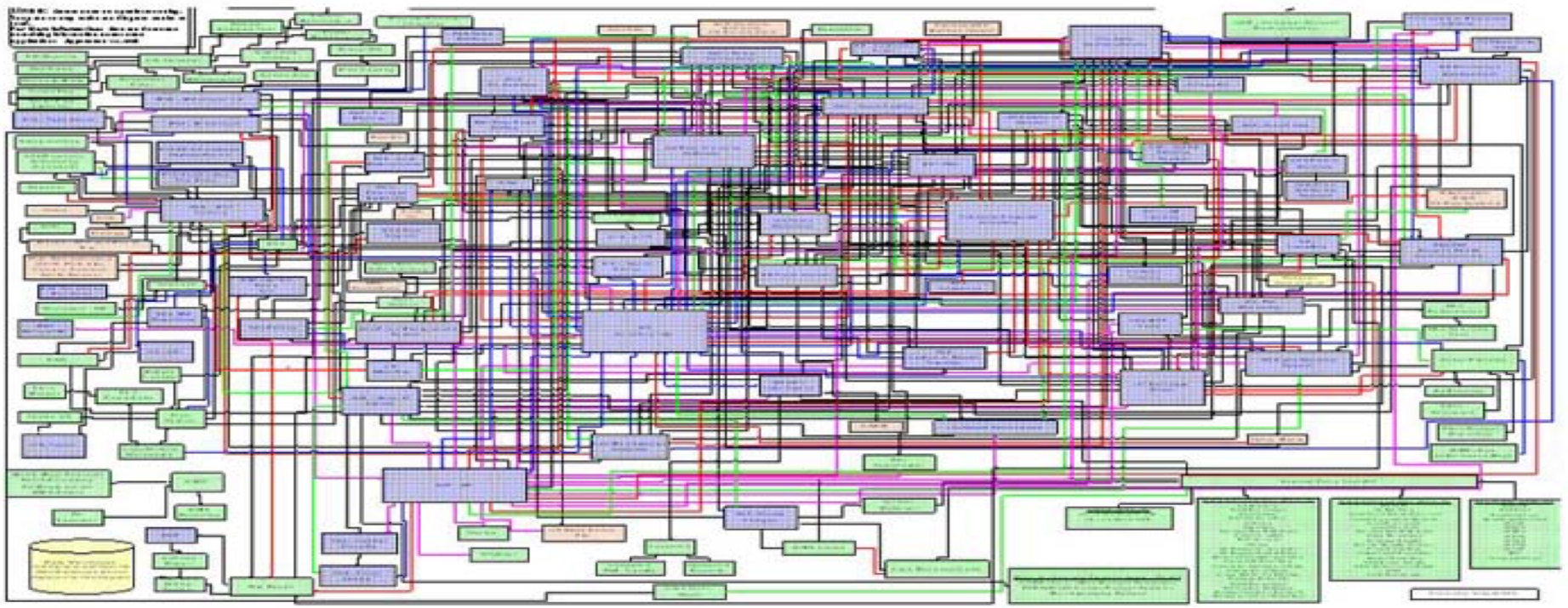
- **Manual handoffs of data in file share drives**
- **Sharing of data tables**
- **Requisition of data center resources**
- **Approval from internal security groups**
- **Future creation of API's**
  - **Often private and undocumented**

Complication

With no mechanism to reduce complexity, the application grows more complex

- **Add dependencies**
- **Add features**
- **Add Security**
- **Etc.**

The Hairball



Complex App

- Deployment ??
- Scaling ??
- Updates ??

Complication: Deploying the Complex App is Hard

- Updates are less frequent
- Velocity slows
- Long QA cycles
- Multiple development teams must coordinate deployment and releases
- FEAR of new features

Complication: Scaling the Complex App is Hard

- We must scale the monolith, but it is highly coupled
- The best option is often to place it in a larger server (more CPU, memory)
- Typically very expensive to scale vertically
- Application becomes more vulnerable to outage and downtime

Complication: Updating the Complex App is Hard

- Long term commitment to technology choices
- Slow application startup times can drag down developer velocity as well
- Cost increases
- Perhaps fall behind newer competitors implementing on new technologies
  - In 1965, the average tenure of companies on the S&P 500 was 33 years. By 1990, it was 20 years. It's forecast to shrink to 14 years by 2026.
  - About 50 percent of the S&P 500 will be replaced over the next 10 years

Resolution: Microservice Architecture

Resolution: Create a Loosely Coupled API Service Platform, aka "Microservices"

- Decentralize planning and eliminate interdependencies

This will make it possible for a team of developers to conduct their planning and development in isolation, vastly improving time to market and cutting cost.

- A team of 7 planning without dependencies, with all the personnel required can reasonably plan for 10 weeks in a 1-2 days which would **result in agile planning event savings alone of 50%**

Skeptical? Consider a new team developing on Public Cloud (a Loosely Coupled API Service Platform)

(+) For example, a new development team on Cloud can submit API calls to create a VM, configure security routes, setup a firewall and create a database within minutes.

(-) The same operations within a common large enterprise often requires a complicated set of tickets and sophisticated connections to manage the hand-offs taking weeks to get started

Resolution: Microservices

Microservices enable Platforms

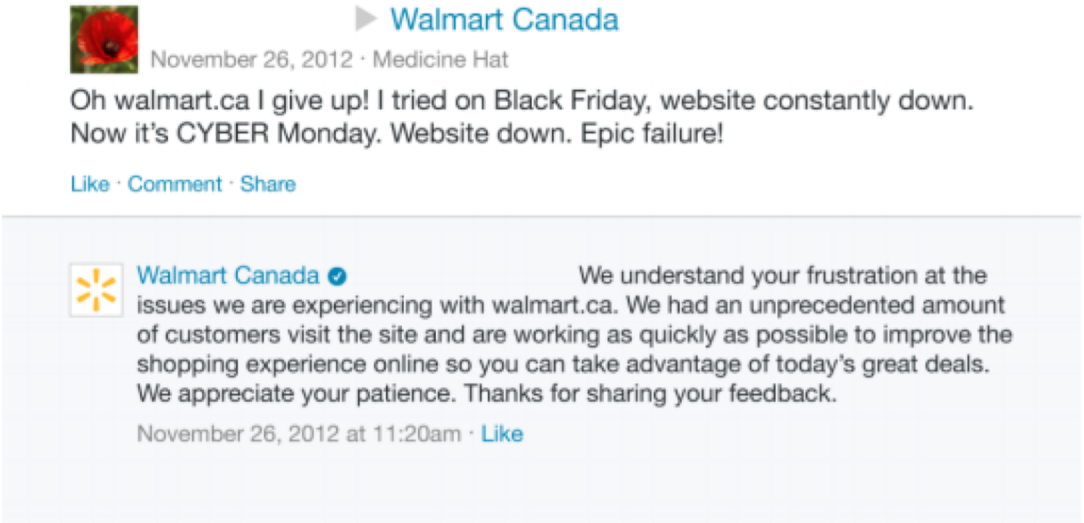
- A platform is an ecosystem of APIs that third parties may build functionality on top of.
- Notable Microservice Platform examples: Hootsuite, Gilt Group, SoundCloud, Amazon Web Services, Netflix, Spotify, Google Cloud, Box, and more!

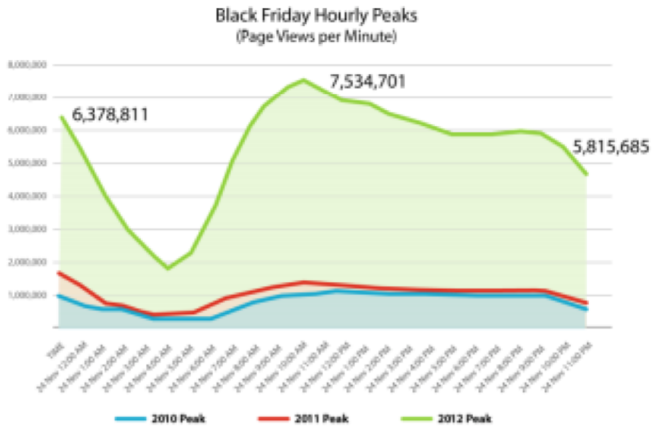
Walmart Canada Case Study

Walmart Canada: Situation

Designed in 2012, Walmart Canada's website failed on Black Friday for two years in a row.

- Traffic was increasing every year, keeping the team off balance
- Customer backlash, competitive pressures (i.e. Amazon) made this a top priority

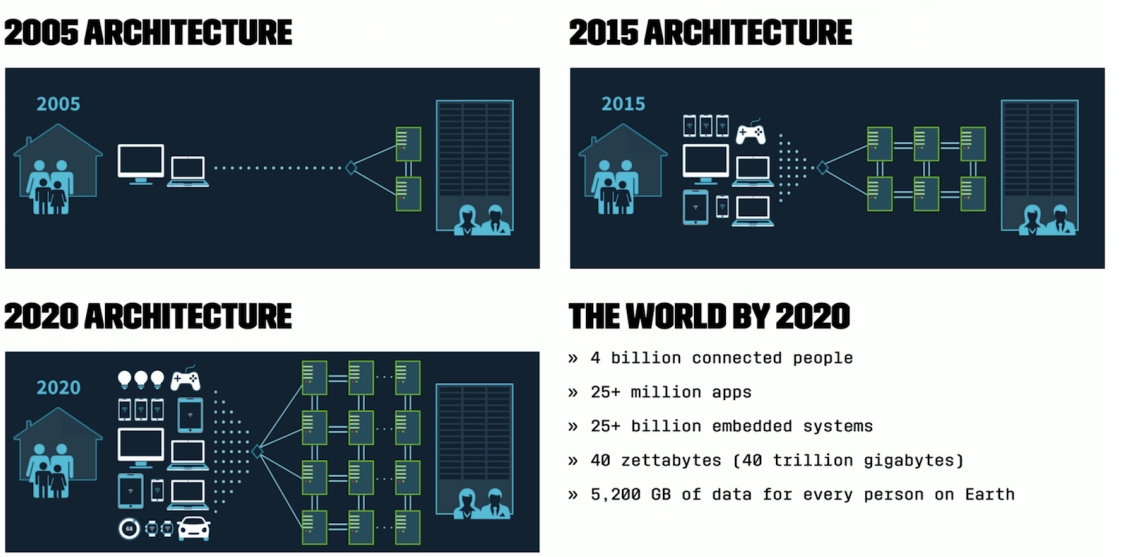




Walmart Canada: Action

With traffic likely to continue to increase, goals were set

- Near 100% availability
- Consistent responsiveness under varying load conditions
- Predictable spikes of traffic e.g. Black Friday
- Less predictable spikes e.g. marketing campaign



Walmart Canada: Action

Re-architected solution

REACTIVE

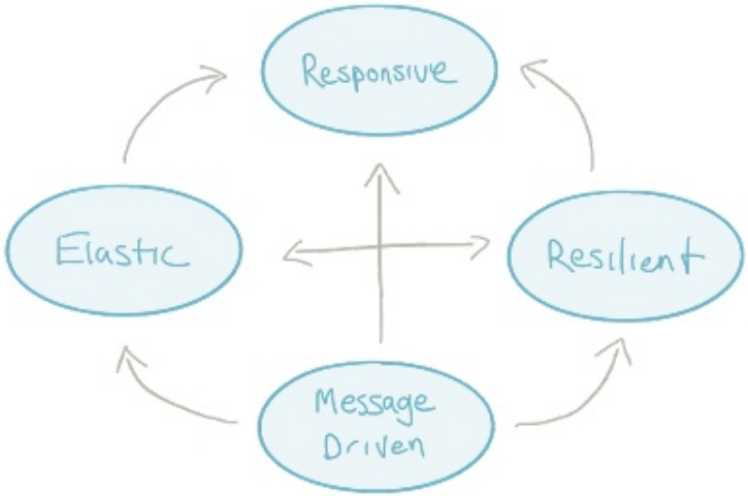
The ultimate maturity model for microservices.

» Responsive

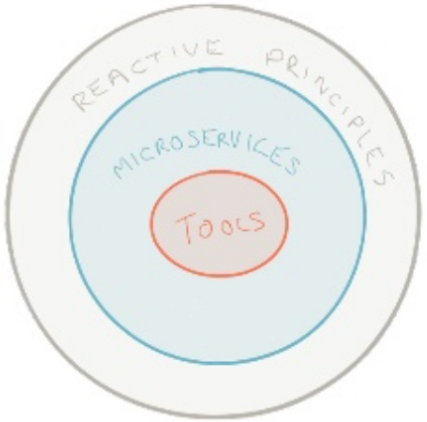
» Resilient

» Elastic

» Message-driven



Walmart Canada: Action



WHY, WHAT, HOW

Clip slid

Reactive (principles)

» responsive, resilient, elastic, message-driven

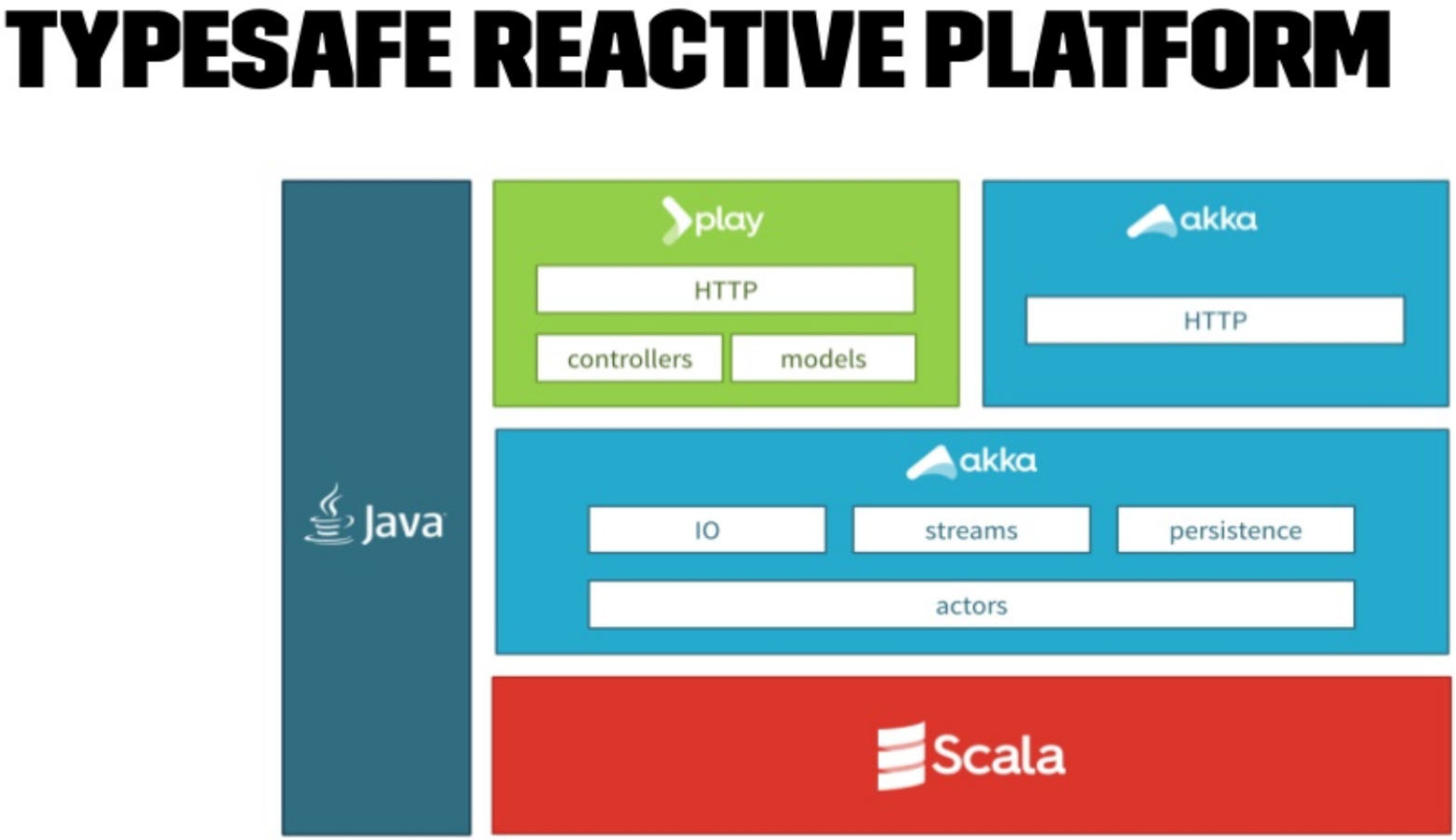
Microservices (strategy)

» bounded contexts (DDD), event sourcing, CQRS, eventual consistency

Tools (implementation)

» Typesafe Reactive Platform (Play, Akka, Spark)

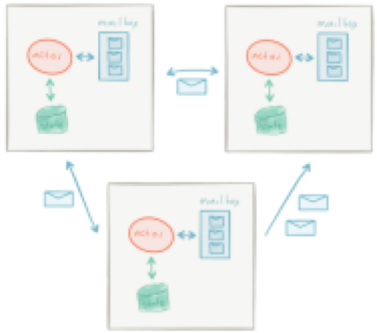
Walmart Canada: Action

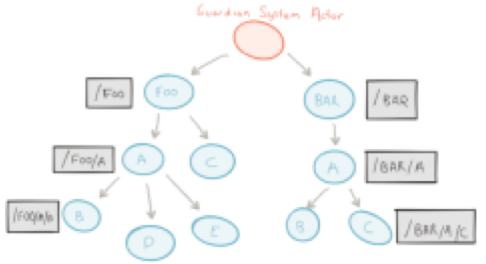


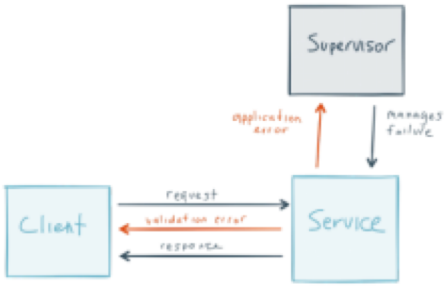
Walmart Canada: Action

Re-architected solution

- **Message Driven**
  - Distribution
  - Location transparency
  - Isolation
- **Resilient**
  - Supervision
  - Dedicated separate error channel

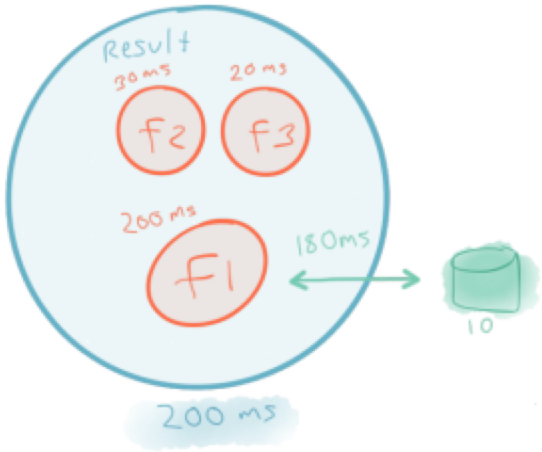






Walmart Canada: Action

- **Elastic**
  - **Scale up**
    - Async
    - Non-blocking
  - **Scale out**
    - Immutable data
    - Share nothing
- **Responsive**
  - **Responsive to events, load failure, users**
  - **Distribution of data**
  - **Circuit Breakers**



Walmart Canada: Result

Business Uplift

- Conversions up 20%
- Mobile orders up 98%
- No downtime on Black Friday or Boxing Day

Operational Savings

- Moved off expensive hardware
- Cheap virtual x86 servers
- 20-50% cost savings
- ~40% compute cycles

How to break an application into Microservices?

Based on existing system and natural architectural boundaries

- **Database can be exposed as a Microservice for example**
- **Any slow process behind a queue can be considered a Microservice**
- **Language boundaries are an opportunity for Microservice creation**

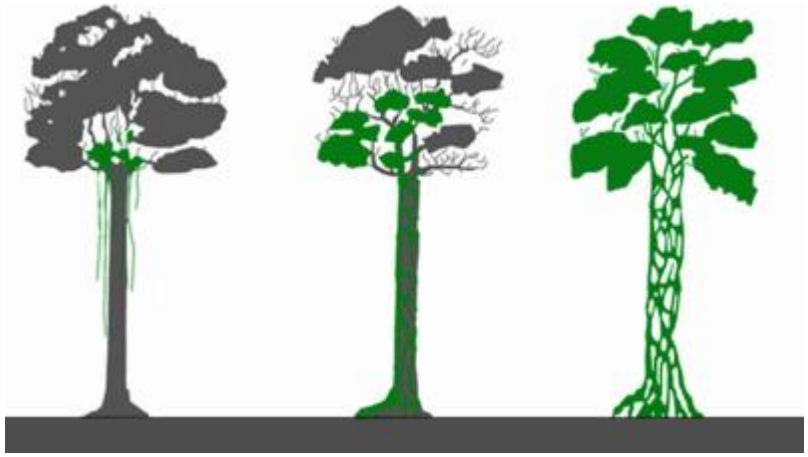
Strategy for Legacy Monoliths: The Strangler Pattern

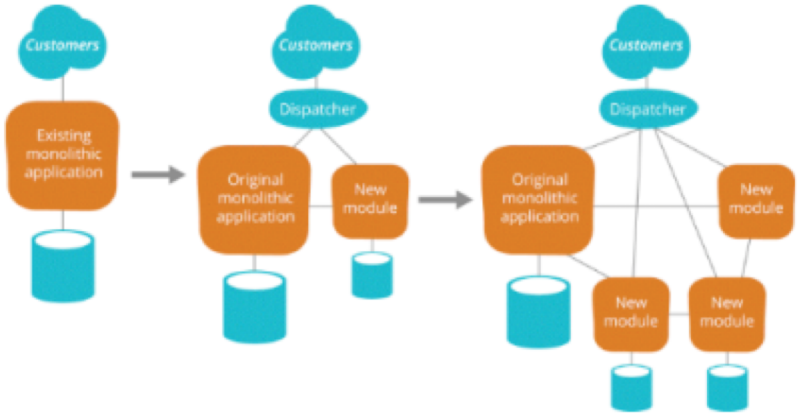
https://flic.kr/p/nMWBkR

Strategy for Legacy Monoliths: The Strangler Pattern

Gradually replace a legacy system by implementing new features of an existing system with those in one or several new systems

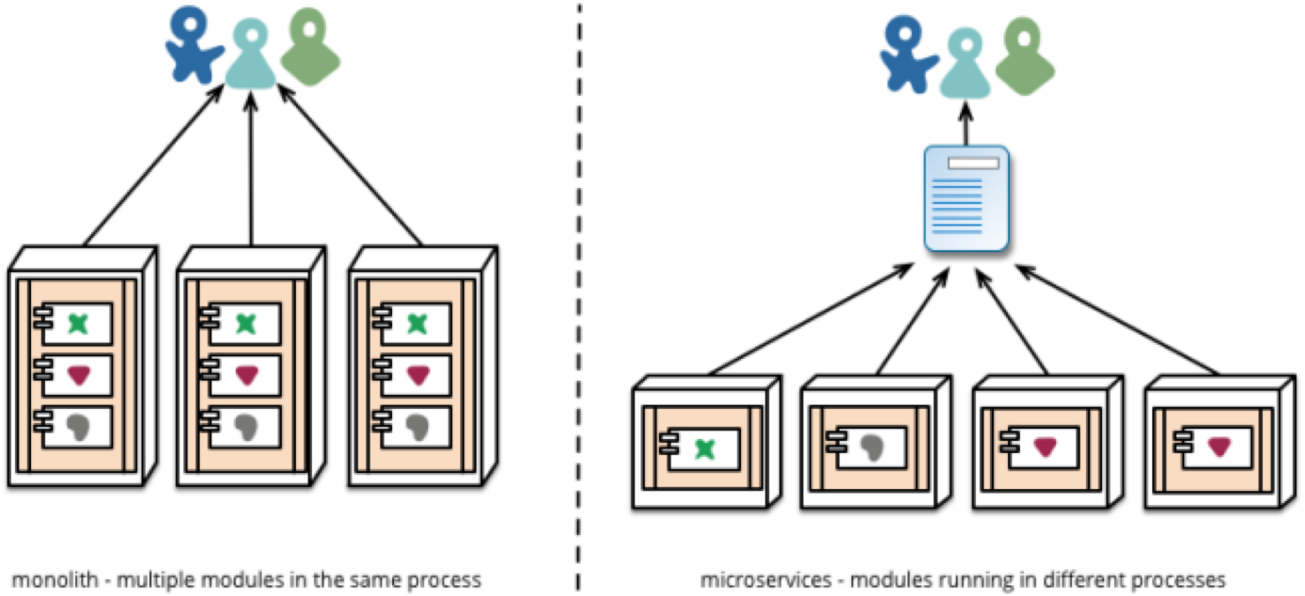
- Rather than trying to replace a large, expensive legacy system all at once you can iterate and improve in small amounts.
- This can be implemented for example by a load balancer in front the targeted functionality
- Over time, the old application is ‘strangled’ like a tree that has hosted a Strangler Fig.
- Focus on quick wins, needs that were not satisfied by existing software to deliver the most value quickly.





What is a Microservice?

- **A service that can be <u>independently</u> deployed**

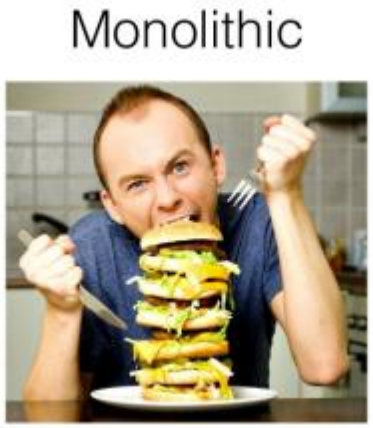


monolith - multiple modules in the same process

microservices - modules running in different processes

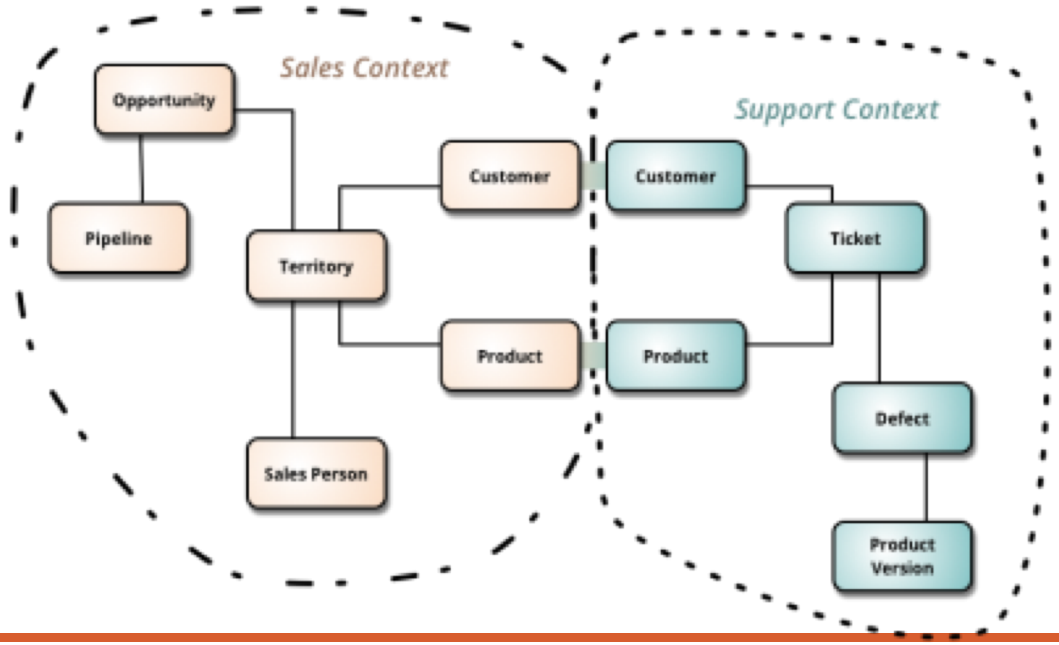
How big is a Microservice?

- A **single programmer** can design, implement, deploy and maintain a Microservice – Fred George
- Software that **fits in your head** – Dan North



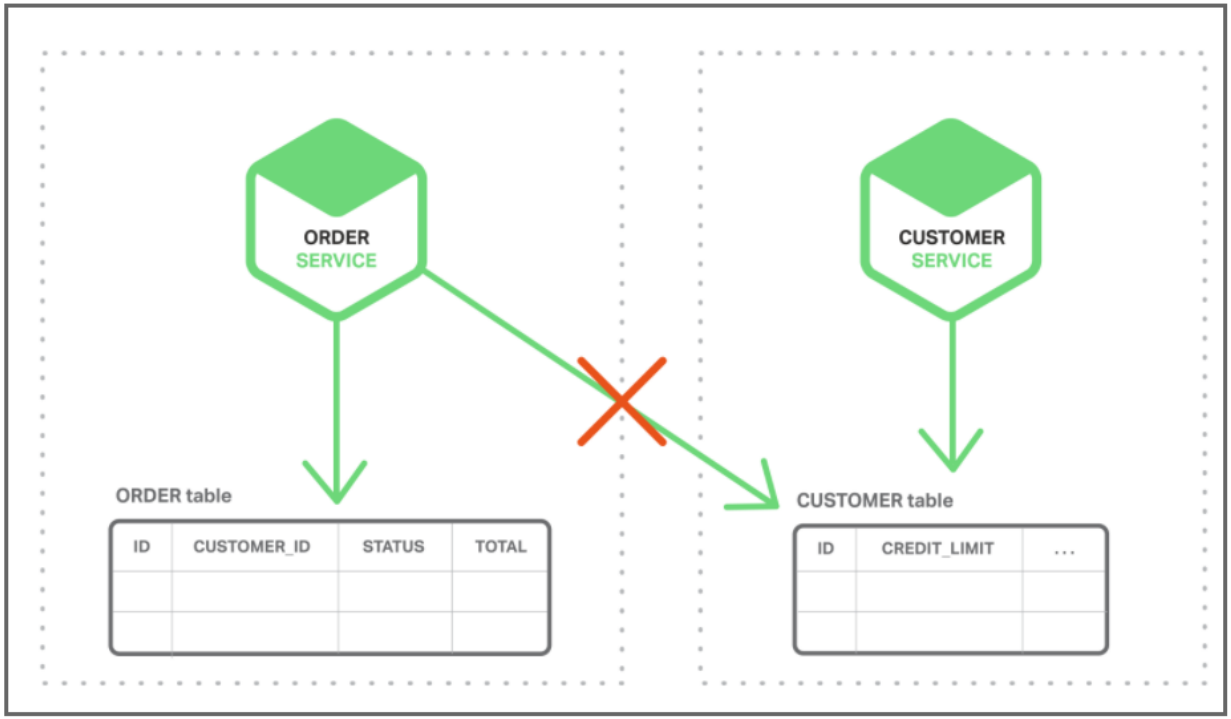
Where is the data for a Microservice?

- A single logical **database per service** – Chris Richardson
- A Microservice implements a single **Bounded Context**; Data model is driven by the domain model (DDD)
- **Event driven** data Management



Event Driven Data Management

- Implement business transactions that maintain consistency across multiple services
- Implement queries that retrieve data from multiple services

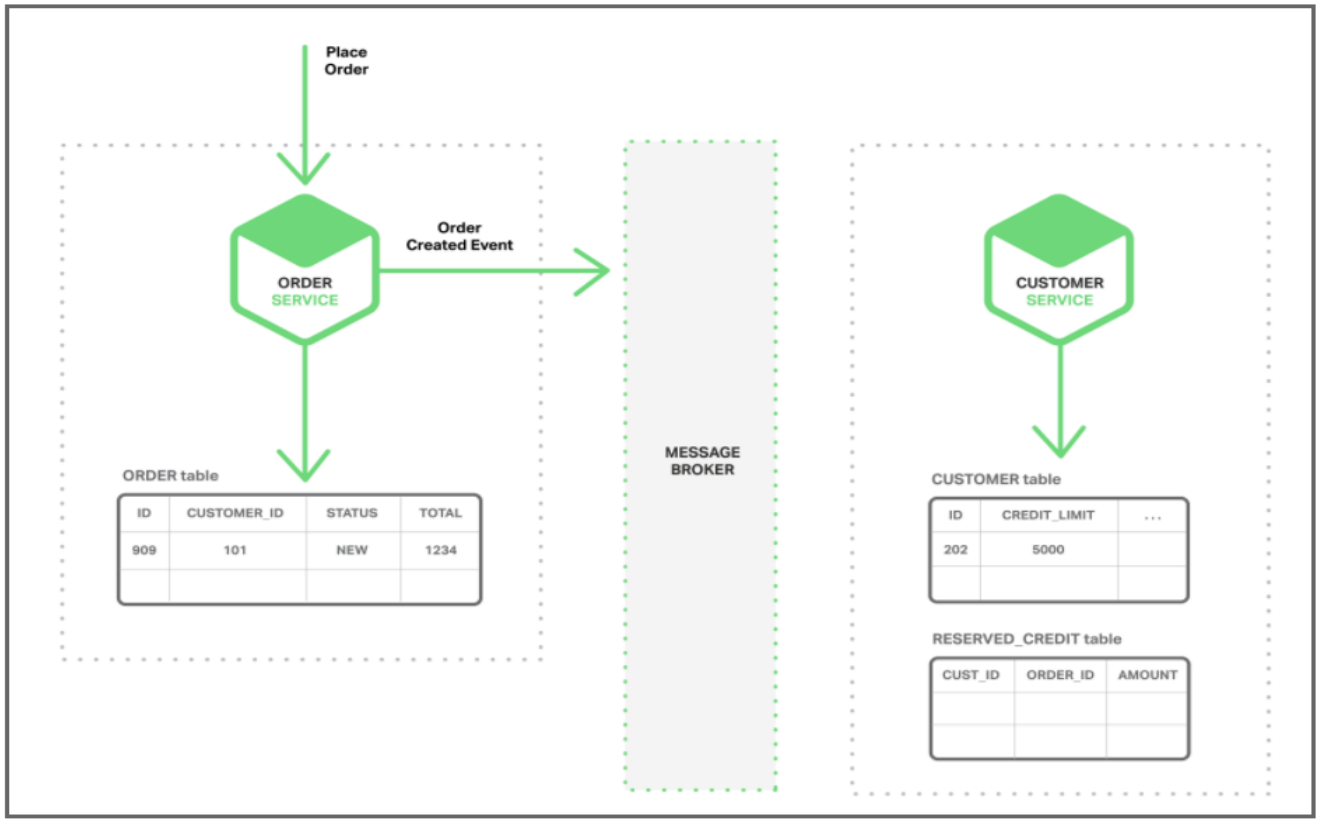


Event Driven Data Management

- **Microservice publishes an event when something notable happens, such as when it updates a business entity. Other microservices subscribe to those events. When a microservice receives an event it can update its own business entities, which might lead to more events being published.**
  - **Use events to implement business transactions that span multiple services**

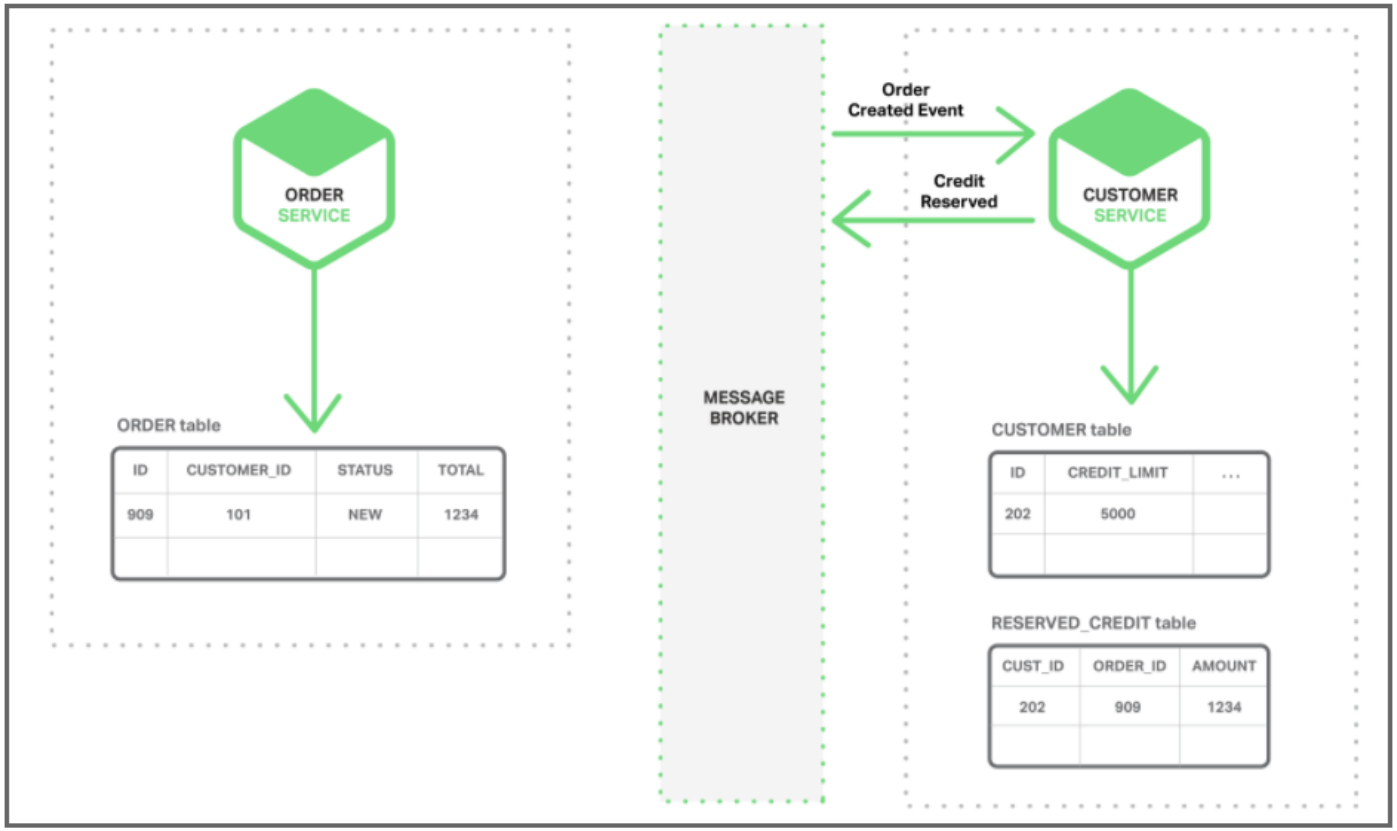
Event Driven Data Management

1. The Order Service creates an Order with status NEW and publishes an Order Created event.



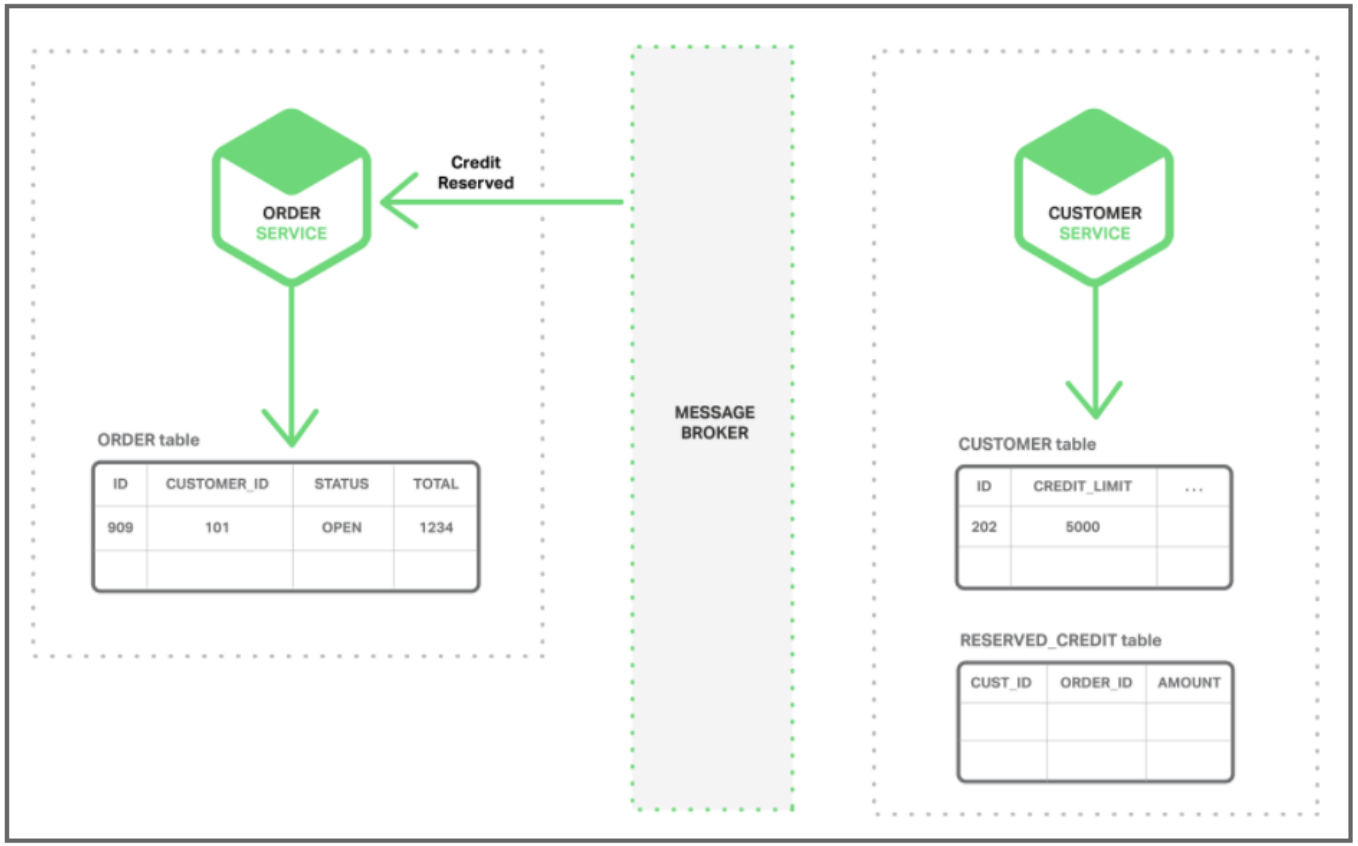
Event Driven Data Management

2. The Customer Service consumes the Order Created event, reserves credit for the order, and publishes a Credit Reserved event.



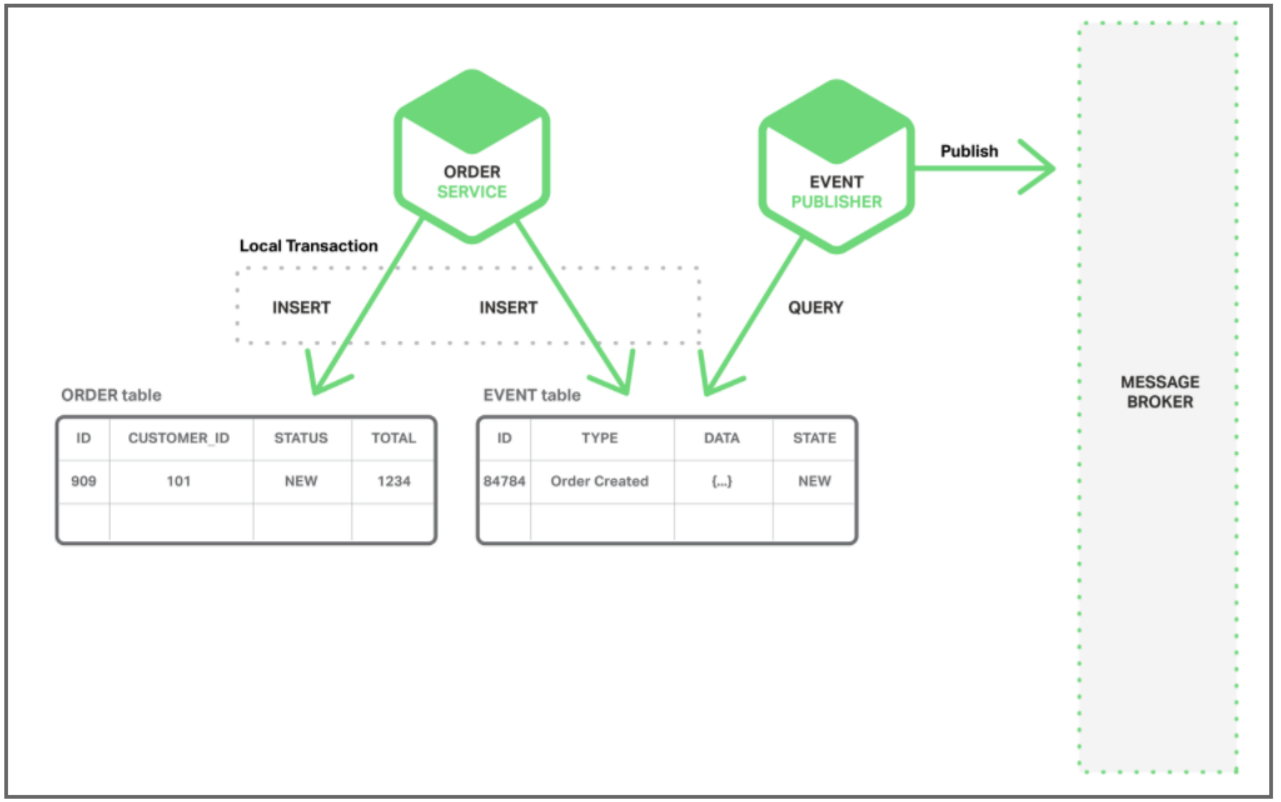
Event Driven Data Management

3. The Order Service consumes the Credit Reserved event, and changes the status of the order to OPEN.



Event Driven Data Management - Preserve Atomicity

EVENT table – Functions as a message queue, in the database that stores the state of the business entities. The application begins a (local) database transaction, updates the state of the business entities, inserts an event into the EVENT table, and commits the transaction. A separate application thread or process queries the EVENT table, publishes the events to the Message Broker, and then uses a local transaction to mark the events as published.



How to maintain Microservices?

- Addressable through a **service discovery** system – Chris Richardson
- Microservices **built & deployed independently. Stateless**, with state as backing services – 12Factor.net

** Service Discovery – Will discuss in “Patterns”

How to architect Microservices?

Typically

- **CRUD – Create, Read, Update and Delete API implementations**
- **Range between RPC, Message and Stream processing**
- **Evolve from a Monolith 1st iteration**

Microservice: Advantages

- Simple services that are focused on doing one thing well
- Each service can be built using the best tool for the job
- Systems built this way are inherently loosely coupled
- Teams can deliver and deploy independently from each other
- Enable continuous delivery but allowing frequent releases while the rest of the system continues to be stable

Microservices

Solve organizational problems

- Teams blocked by or waiting on other teams
- Communication overhead
- Slow velocity

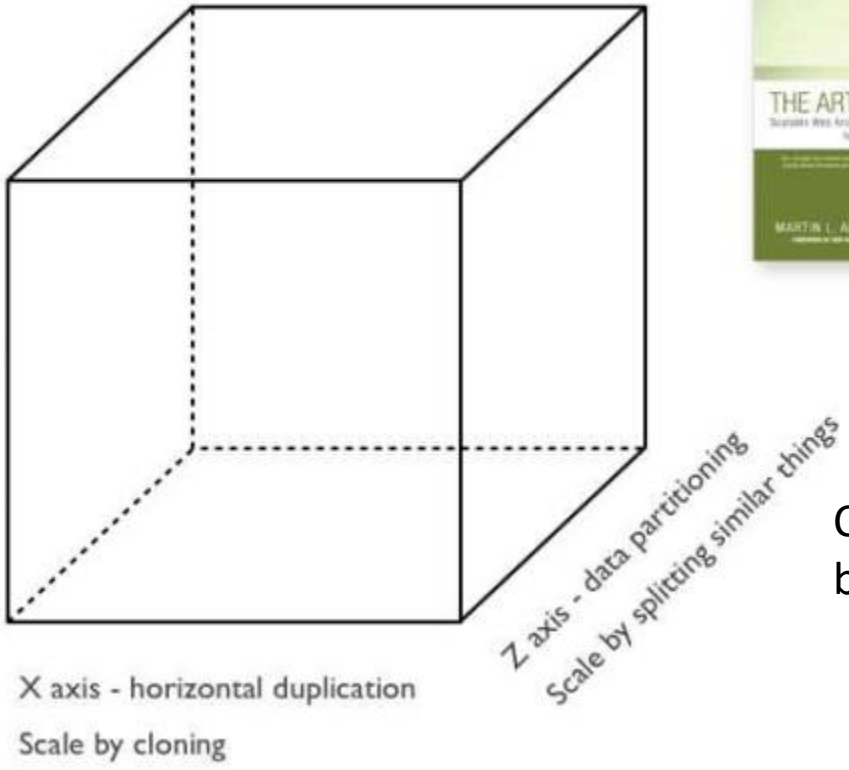
Cause Technical Problems

- Requires Devops, devs deploying and operating their services
- Need well defined business domains
- More distributed systems
- Need an orchestration layer

The Art of Scaling

3 dimensions to scaling

Decrease complexity
More resource efficient



Complex structure,
but enables scalability

Doesn't decrease complexity,
but scales

Self Service Mandate (Build a Platform)

1. **All teams will expose their data and functionality through self service Api interfaces**
2. **No other forms of communication allowed, no tickets, direct linking, reads of data stores, no back doors, only service interface calls over the network**
3. **All services must be designed to be externalizable. The team must plan and design to be able to expose the api's to developers in the outside world, creating a platform.**

The now-famous Jeff Bezos rant, leading to formation of Amazon Web Services
https://gist.github.com/chitchcock/1281611

Devops = Optimize for Speed

“Therefore, broadly speaking, to achieve DevOps outcomes we need to reduce the effects of functional orientation (“optimizing for cost”) and enable market orientation (“optimizing for speed”).”

-Gene Kim

- **This consists of having many small teams work independently, rapidly delivering customer value**
- **Teams are cross-functional and independent. Each team, consisting of 8 or fewer resources, deploys independently.**
- **Market oriented teams are responsible for feature development, testing, securing, deploying, and supporting service in production**

Microservices = Optimize for Speed

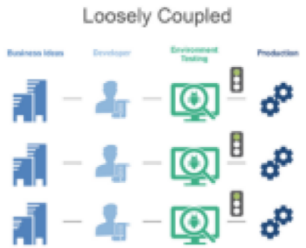
"Organizations with these types of service-oriented architectures, such as Google and Amazon, have incredible flexibility and scalability. These organizations have tens of thousands of developers where small teams can still be incredibly productive."

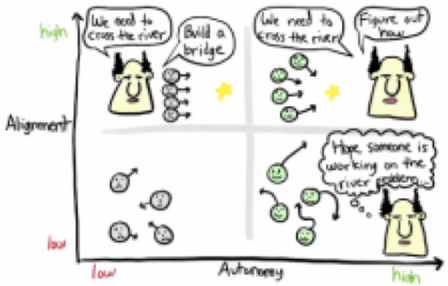
-Randy Shoup, former Engineering Director for Google App Engine

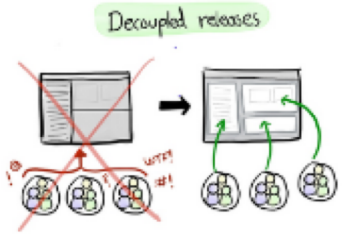
- **This consists of having many small teams work independently, rapidly delivering customer value**

Small Team Size

- Build platform with small teams of 6-8 engineer teams
- **Self Sufficient**: Each team completely owns one or more services and has everybody they need to develop functionality
- **Automate Interfaces:** Teams connect to each other through API's. Teams don't hire an army of program managers to manage dependencies.
- **Decentralized:** Teams plan, develop and deploy autonomously







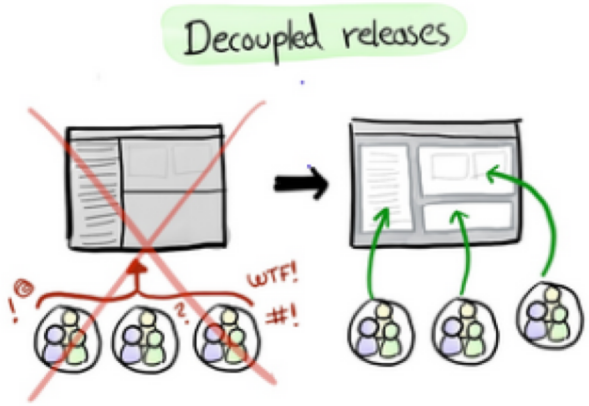
Starting and Scaling DevOps in the Enterprise by Gary Gruver
https://www.amazon.com/dp/B01M332BN2/ref=dp-kindle-redirect?_encoding=UTF8&btkr=1

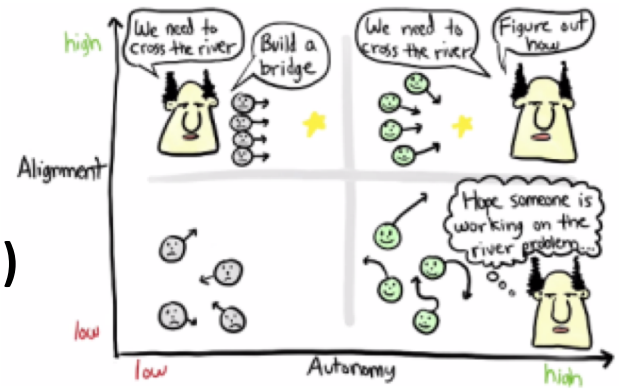
Why Small Team Size? Scaling the Organization

Autonomy

- Decentralizes power and creates autonomy which is critical to scaling the organization
- Each team can focus on the business metric they are responsible for and act autonomously to maximize the metric.

Amazon CTO Werner Vogels explained the advantages of this structure to Larry Dignan of *Baseline* in 2005 (see notes accompanying this slide)



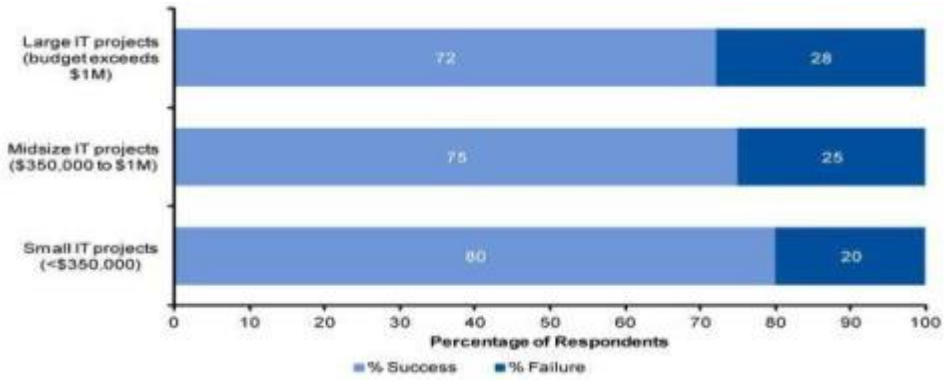


Science behind why two-pizza team works http://blog.idonethis.com/two-pizza-team/
ITRevolution, Conways Law by Gene Kim http://itrevolution.com/conways-law/

Why do Microservices?

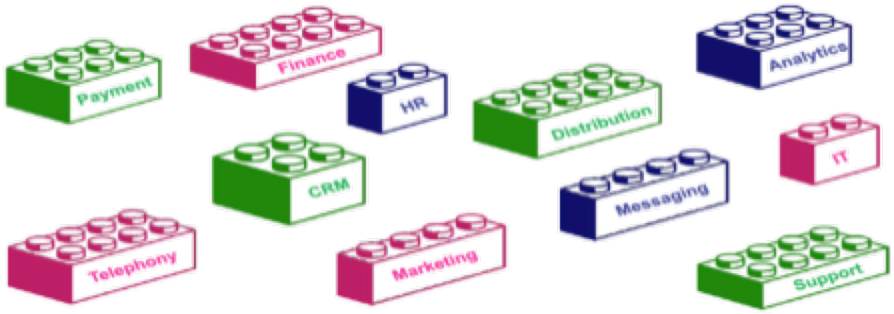
Reduce complexity, increase success

- According to a Gartner survey of 150 participants in 2011, the failure of projects exceeding $1 million was found to be almost 50% higher than for projects with budgets below $350,000.
- This result was reinforced by prior Gartner IT projects which found small IT projects to experience a one-third lower failure rate than large projects



https://thisiswhatgoodlookslike.com/2012/06/10/gartner-survey-shows-why-projects-fail/

Microservice Building Blocks



APIs provide the flexibility for businesses to grow by adopting new business models and enable accelerated development of new applications.

It is like picking different LEGO blocks to build a toy house.

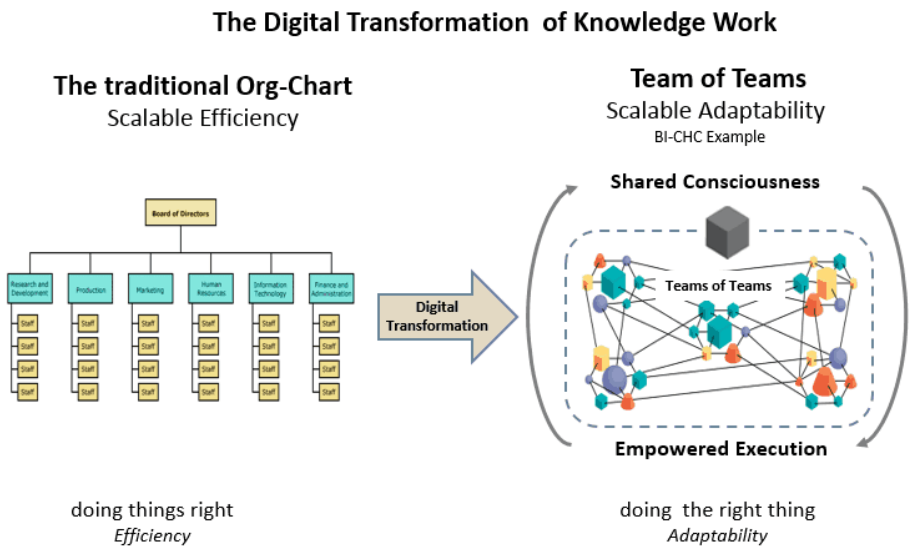
Resolution: Service Oriented Architecture

This concept aligns with General Stanley McChrystal's recent work Team of Teams.

- **As McChrystal writes: "To defeat a network, we had to become a network."**
- **A network in this context is a collection of small cross-functional teams that have been empowered to self-organize, self-manage, and self-execute.**
- **Teams are enabled to become a 'network' by a Service Oriented Architecture (SOA) and have done so at various adopted of SOA such as AWS, Netflix et tal**
- **Cost, time to market, and adaptiveness are vastly improved**

What is needed?

Changing Management Structures by moving from scalable Efficiency to scalable Adaptability in order to succeed



External Benchmarks

2016 IT Performance by Cluster

	High IT Performers	Medium IT Performers	Low IT Performers
Deployment frequency *For the primary application or service you work on, how often does your organization deploy code?*	On demand (multiple deploys per day)	Between once per week and once per month	Between once per month and once every 6 months
Lead time for changes *For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code commit to code successfully running in production)?*	Less than one hour	Between one week and one month	Between one month and 6 months
Mean time to recover (MTTR) *For the primary application or service you work on, how long does it generally take to restore service when a service incident occurs (e.g., unplanned outage, service impairment)?*	Less than one hour	Less than one day	Less than one day*
Change failure rate *For the primary application or service you work on, what percentage of the changes either result in degraded service or subsequently require remediation (e.g., lead to service impairment, service outage, require a hotfix, rollback, fix forward, patch)?*	0-15%	31-45%	16-30%

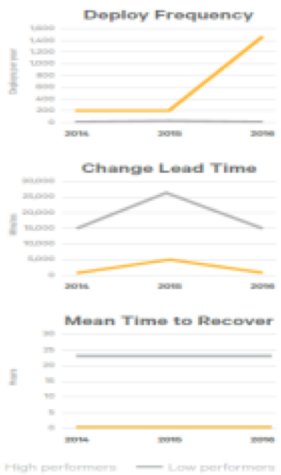
* Low performers were lower on average (at a statistically significant level), but had the same median as the medium performers.

*** Source: Puppet Labs Report**

Competitive Advantage

High performing teams are breaking away from the pack and the status quo of even three years ago is a dangerous assumption currently.

A team which had state of the



Microservices Patterns

Before diving into Microservices Patterns, let’s walk through a Sample Monolithic Application.

Later we will convert the same application to Microservices and deploy in Kubernetes

Exercise 1.1 Monolithic App in Java

CLASSROOM WORK 101 (45 minutes)

1. **Maven Build and Run application**
2. **Redis data store**
3. **Jersey Rest API**
4. **Docker compose**

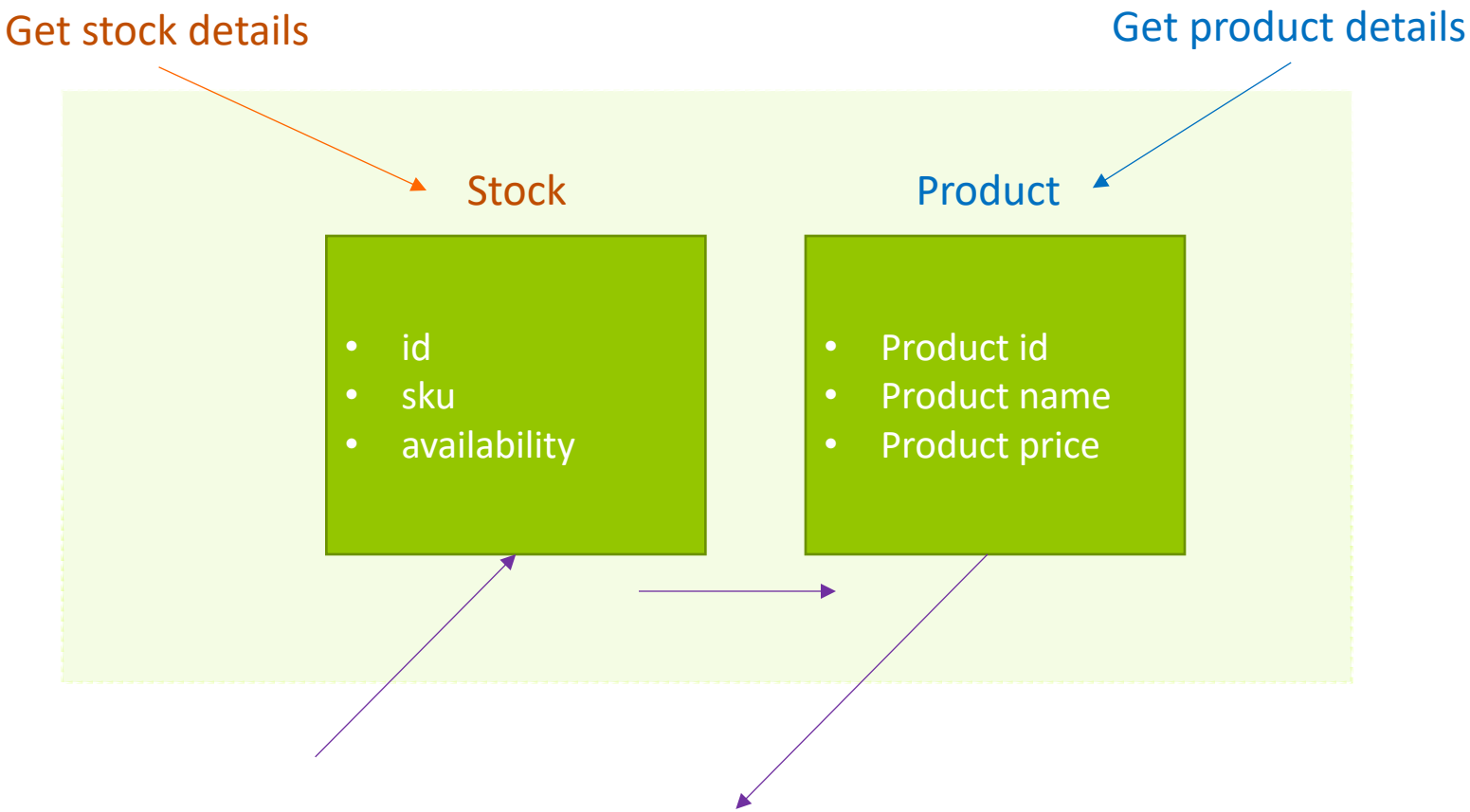
Step by Step Instruction to complete the exercise

https://github.com/techtown-training/microservices-bootcamp/blob/master/exercise/exercise1.1-monolithic_java_project.md

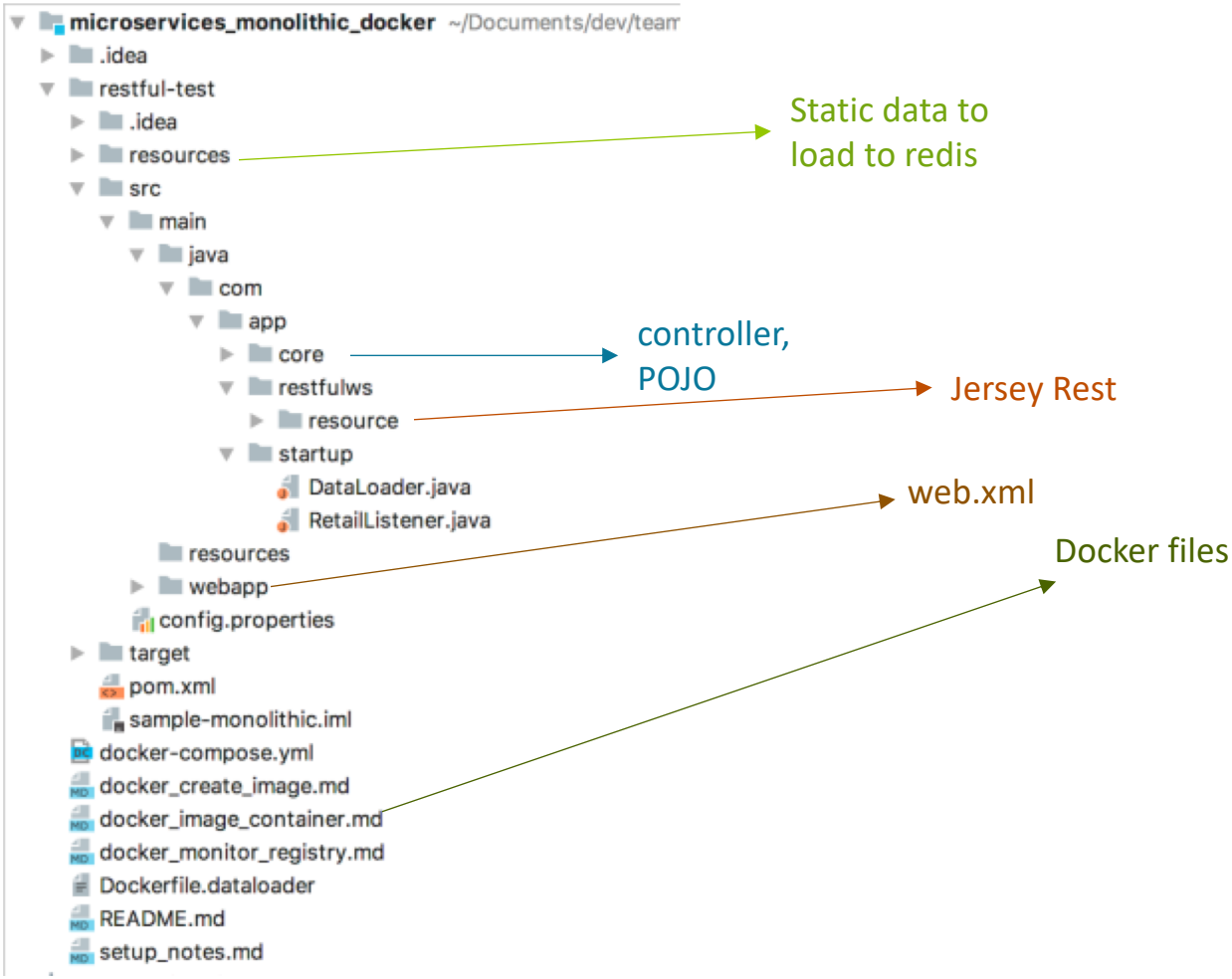
The source code can be found here !!

https://github.com/techtown-training/microservices-bootcamp/tree/master/exercise/src_code/microservices_monolithic_docker

Sample Monolithic App in Java



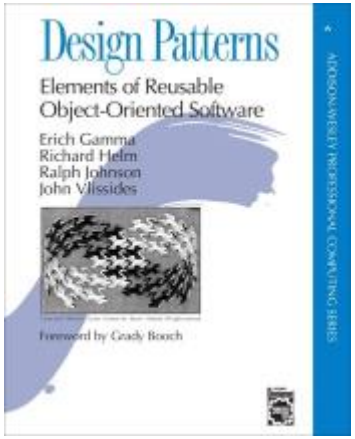
Project Structure



https://github.com/techtown-training/microservices-bootcamp/blob/master/exercise/src_code/microservices_monolithic_docker/README.md

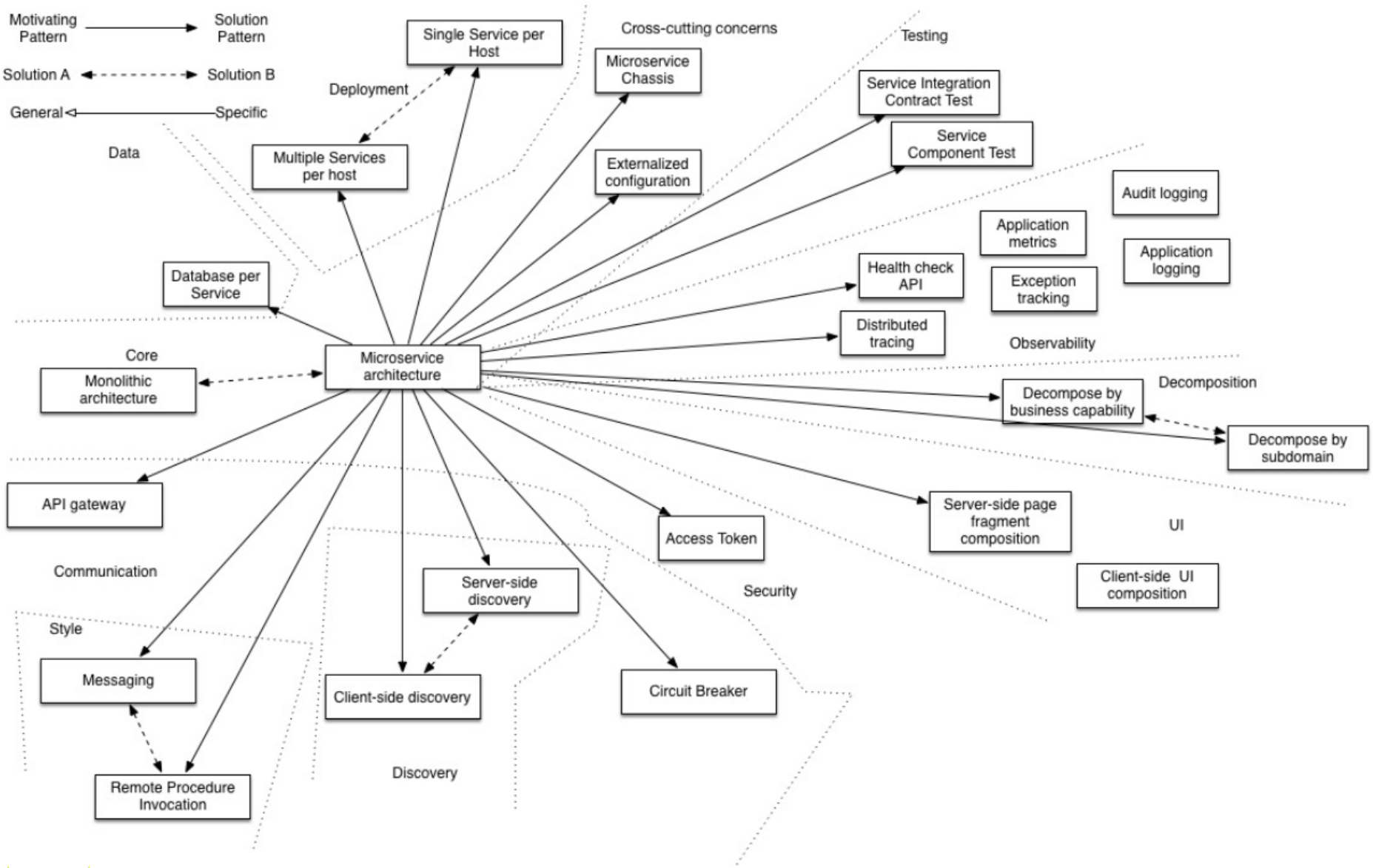
Patterns

What is a pattern?



Reusable **solution** to a **problem** occurring in a particular **context**

Microservice Architectures



Pattern - Categories

- **Data Management**
- Communication
- Deployment
- Discovery
- Reliability
- Observability
- Testing

Data Management Patterns

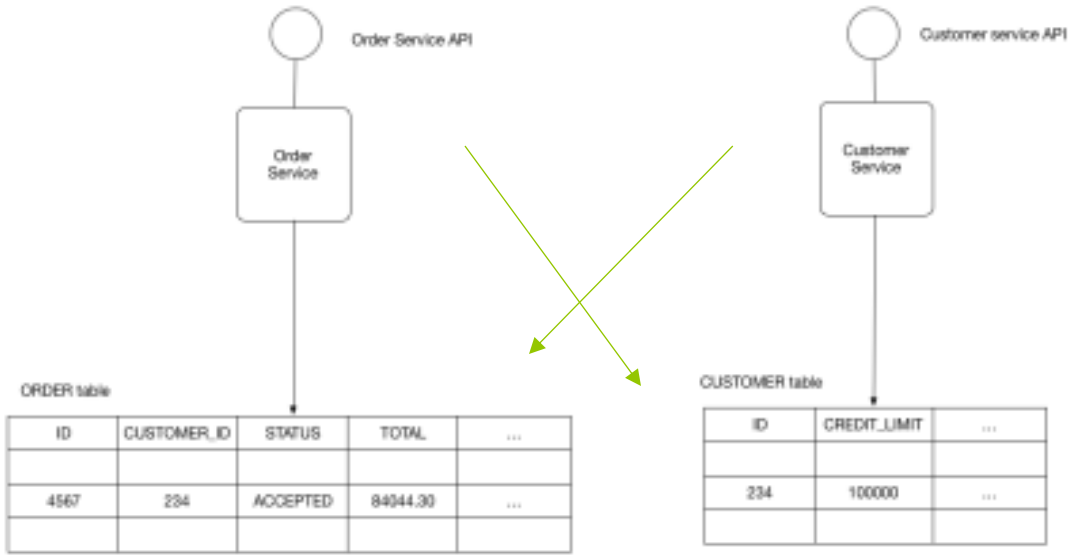
- Shared database
- Database per Service
- Saga
- Event Sourcing
- CQRS (Command Query Responsibility Segregation)
- API Composition

Shared Database

Shared Database
Use a (single) database shared by multiple services. Local ACID transactions.

Benefits:
- Straightforward ACID transactions to enforce data consistency
- Single database is simpler to operate

Drawbacks:
- Since all services access the same database, interference and transaction locks on the table.
- Single database might not satisfy the data storage and requirements of all services.

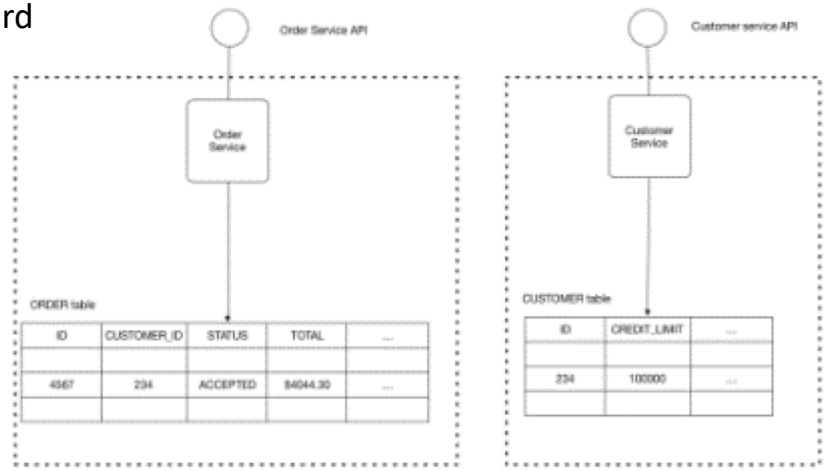


Database Per Service

Database Per Service
Every Microservice has its own persistent data and accessible only via its API
E.g. Private-tables-per-service; Schema-per-service; Database-server-per-service

Benefits:
- Ensures loose coupling
- Each service can use the type of database that is best suited to its needs. E.g., text searches could use ElasticSearch; social graph could use Neo4j.

Drawbacks:
- Transactions involving multiple services not straight forward

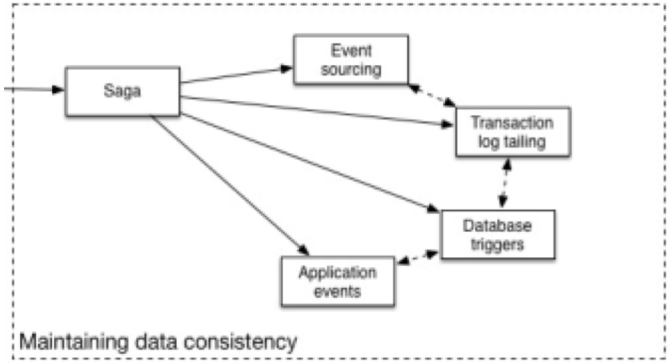


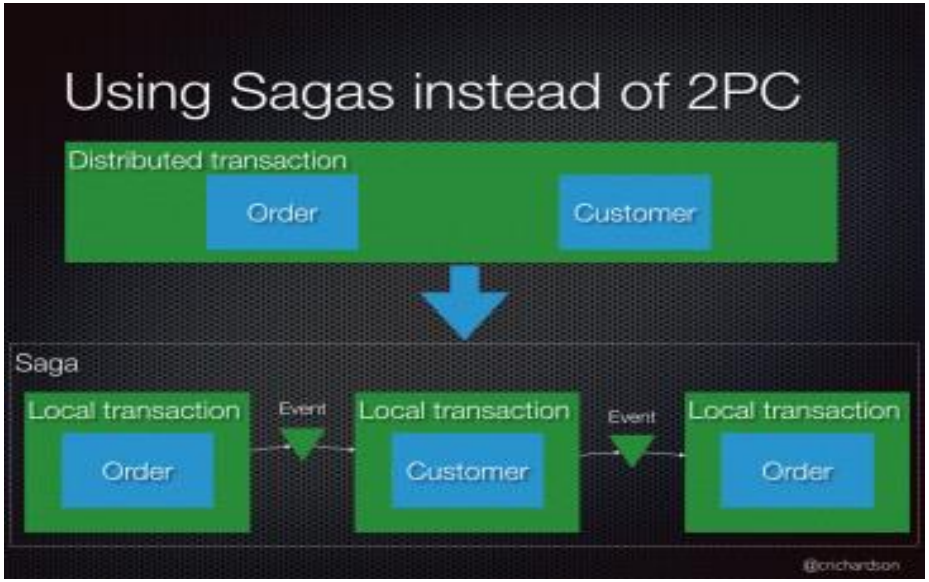
- For consistency; may need to implement capability to roll back (**Saga** pattern)
- Interaction is achieved by message/event brokers

Saga Pattern – For Data Consistency

Saga Pattern

- **Grouping transactions with the capability to undo changes if the transaction fails**
- **Enables an application to maintain data consistency across multiple services**





How to reliably/atomically publish events whenever state changes? Solution can be **Event Sourcing**

Event Sourcing

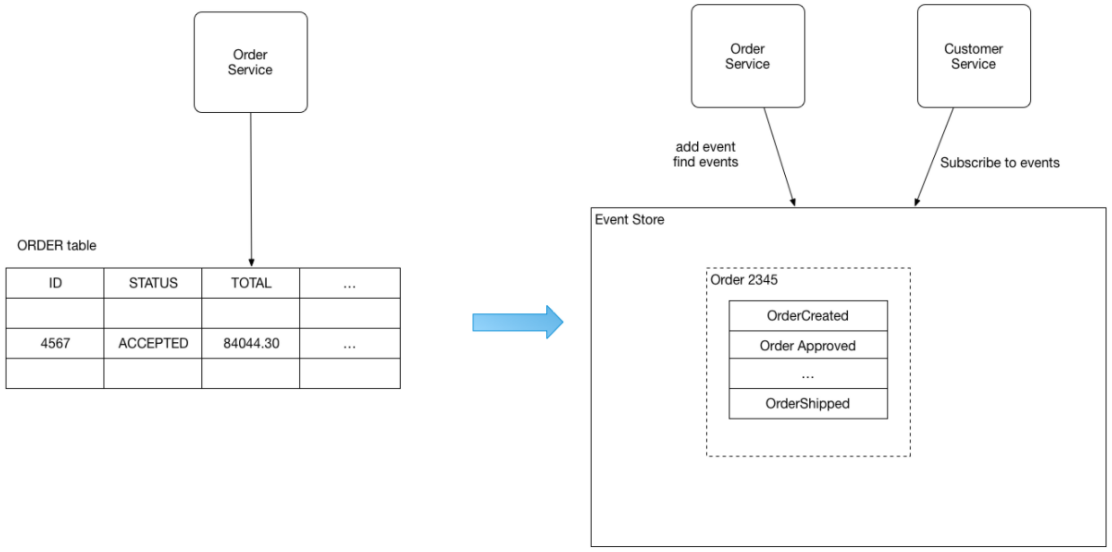
- Persists the state of a business entity as a sequence of state-changing events – in Event Store
- The store has APIs for adding, retrieving an entity's events and subscribe to events.
- The store is like a message broker; A new event is delivered to all interested subscribers.
- E.g. the Order is saved as sequence of events, and customerService subscribe to those events

Benefits:

- Reliably publish events whenever state changes

Drawbacks:

- Difficult to design (unconventional)
- The event store is difficult to query since it requires typical queries to reconstruct the state of the business entities - complex and inefficient --- **use CQRS to query**

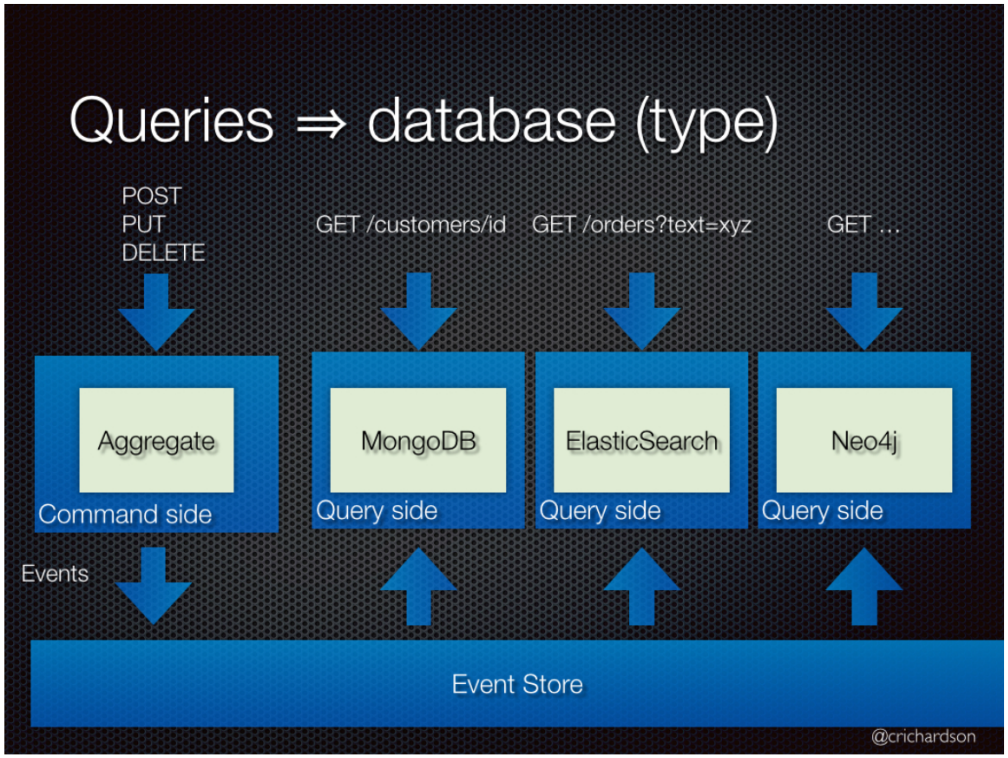


CQRS – Command Query Responsibility Segregation

Split the application into two parts:

- **Command-side**: handles create, update, and delete requests and emits events when data changes.
- **Query-side:** handles queries by executing them against one or more materialized views that are kept up to date by subscribing to the stream of events emitted when data changes.

A Command cannot return data and a Query cannot change the data.



Use Case

- Get data from multiple different Aggregates
- Application has an imbalance in responsibility (read/writes)

How to

- Two models talking to same date store (applications that require synchronous processing and immediate results.)
- Two models talking to different data store (Eventual Consistency)

API Composition

Perform Queries in Microservices Architecture

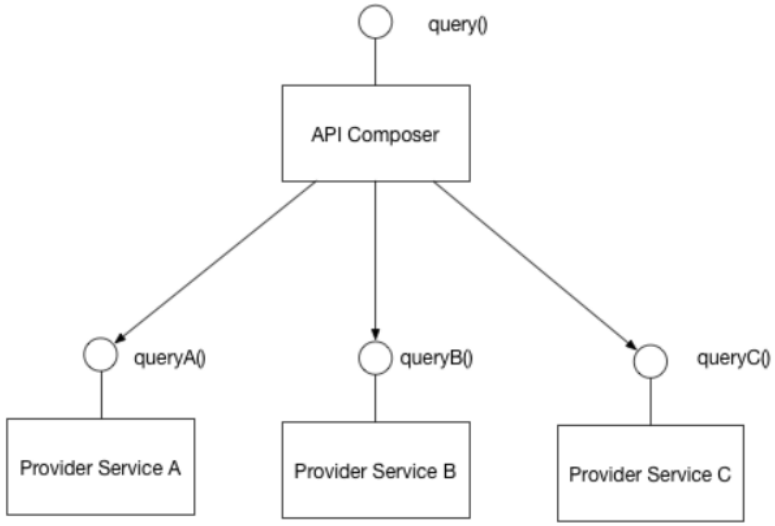
- Implement a query by defining an *API Composer*, which invoking the services that own the data and performs an in-memory join of the results.

Benefits:

- It a simple way to query data in a Microservice architecture

Drawbacks:

- Some queries would result in inefficient, in-memory joins of large datasets.



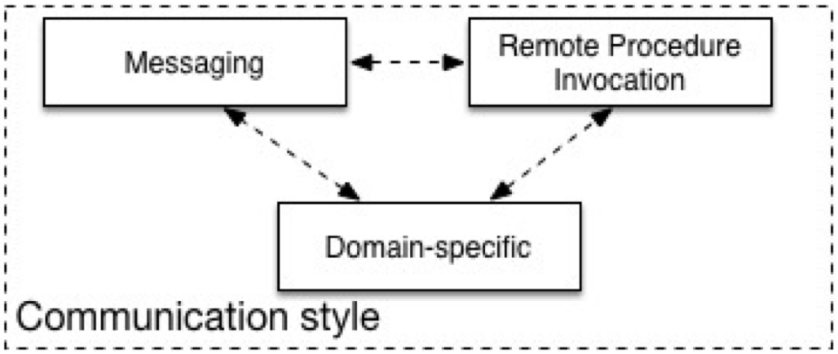
Pattern - Categories

- **Data Management**
- **Communication**
- Deployment
- Discovery
- Reliability
- Observability
- Testing

Communication Style

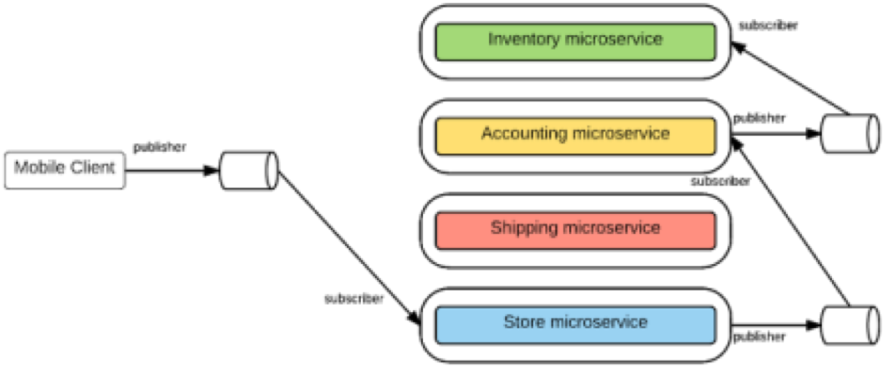
Messaging

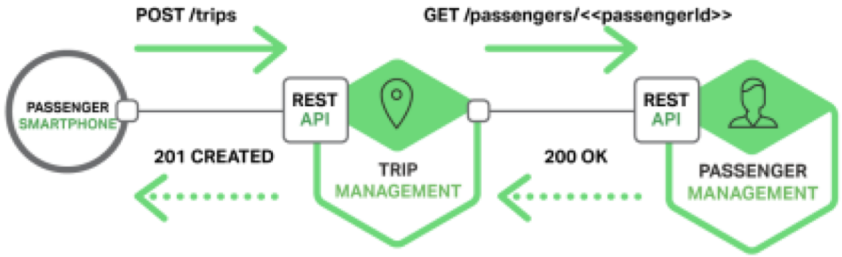
- Services communicate by sending asynchronous messages
- Decouples client & service
- Must also install and maintain a broker (Kafka, RabbitMQ, etc)
- Complex and should be highly available



Remote Procedure Invocation

- Simple and familiar
- Both client and service must be available
- Only simple request and reply. Notifications, publish/subscribe, async communication not supported
- E.g. Rest, Thrift





Pattern - Categories

- **Data Management**
- **Communication**
- **Deployment**
- Discovery
- Reliability
- Observability
- Testing

Deployment Patterns

- Service instance per host
- Multiple Service instance per host
- Service instance per VM
- Service instance per container
- Server-less

Service Instance per Host

Deploy each single service instance on its own host

Benefits:

- **Services instances are isolated from one another**
- **There is no possibility of conflicting resource requirements or dependency versions**
- **A service instance can only consume at most the resources of a single host**
- **It is straightforward to monitor, manage, and redeploy each service instance**

Drawbacks:

- **Potentially less efficient resource utilization**

Multiple Service Instance per Host

Run multiple instances of different services on a host (Physical or Virtual machine).

- **Deploy each service instance as a JVM process. For example, a Tomcat or Jetty instances per service instance.**
- **Deploy multiple service instances in the same JVM. For example, as web applications or OSGI bundles.**

Benefits:

- **More efficient resource utilization**

Drawbacks:

- **Risk of conflicting resource requirements**
- **Risk of conflicting dependency versions**
- **Difficult to limit the resources consumed by a service instance**
- **If multiple services instances are deployed in the same process then its difficult to monitor the resource consumption of each service instance. Its also impossible to isolate each instance**

Service Instance per VM

Package the service as a virtual machine image and deploy each service instance as a separate VM

Benefits:

- **Straightforward to scale the service by increasing the number of instances. Amazon Autoscaling Groups can even do this automatically based on load.**
- **The VM encapsulates the details of the technology used to build the service. All services are, for example, started and stopped in exactly the same way.**
- **Each service instance is isolated**
- **A VM imposes limits on the CPU and memory consumed by a service instance**

Drawbacks:

- **Building a VM image is slow and time consuming**

Service Instance per Container

Package the service as a (Docker) container image and deploy each service instance as a container

E.g. Kubernetes, DC/OS, Marathon/Mesos, Docker Swarm

Benefits:

- **Easy to scale up and down a service by changing the number of container instances.**
- **The container encapsulates the details of the technology used to build the service.**
- **Each service instance is isolated**
- **A container imposes limits on the CPU and memory consumed by a service instance**
- **Containers are extremely fast to build and start. E.g. 100x faster to package an application as a Docker container than it is to package it as an AMI.**

Drawbacks:

- **Less mature than VMs**

Server-less

Deployment infrastructure hides the concept of servers and takes the service's code & desired performance characteristics and runs the service.

The deployment infrastructure is a utility operated by a public cloud provider. It typically uses either containers or virtual machines to isolate the services.
E.g. AWS Lambda, Google Cloud Functions, Azure functions

Benefits:

- **Eliminates the need to spend time on the heavy lifting of managing low-level infrastructure.**
- **Extremely elastic. It automatically scales your services to handle the load.**
- **You pay for each request rather than provisioning what might be under utilized VMs or containers.**

Drawbacks:

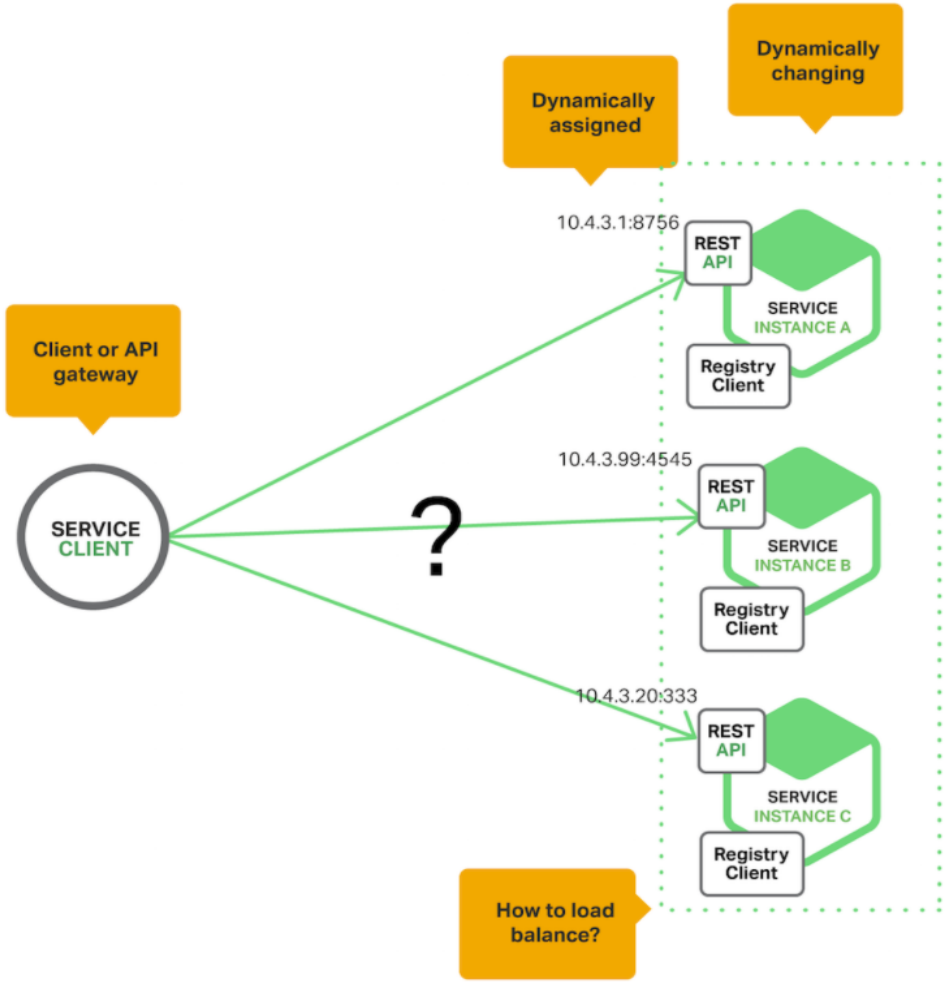
- **Has far more constraints than a VM/Container-based infrastructure. E.g., language support.**
- **Only suitable for deploying stateless applications, not for long running stateful or message broker.**
- **Risk of high latency - the time it takes for the infrastructure to provision an instance and for the function to initialize might result in significant latency.**
- **Can only react to increases in load, cannot proactively pre-provision capacity.**

Pattern - Categories

- **Data Management**
- **Communication**
- **Deployment**
- **Discovery**
- Reliability
- Observability
- Testing

Service Discovery

- Service instances have **dynamically assigned network locations**. Moreover, the set of service instances changes dynamically because of autoscaling, failures, and upgrades. Consequently, the client code needs to use a more elaborate **service discovery mechanism**.



Service Registry

- Key of **service discovery**
  - Database containing the network locations of service instances.
  - Needs to be highly available and up to date.
  - Clients can cache network locations obtained from the service registry -- that information eventually becomes out of date
  - A service registry consists of a cluster of servers that use a replication protocol to maintain consistency.

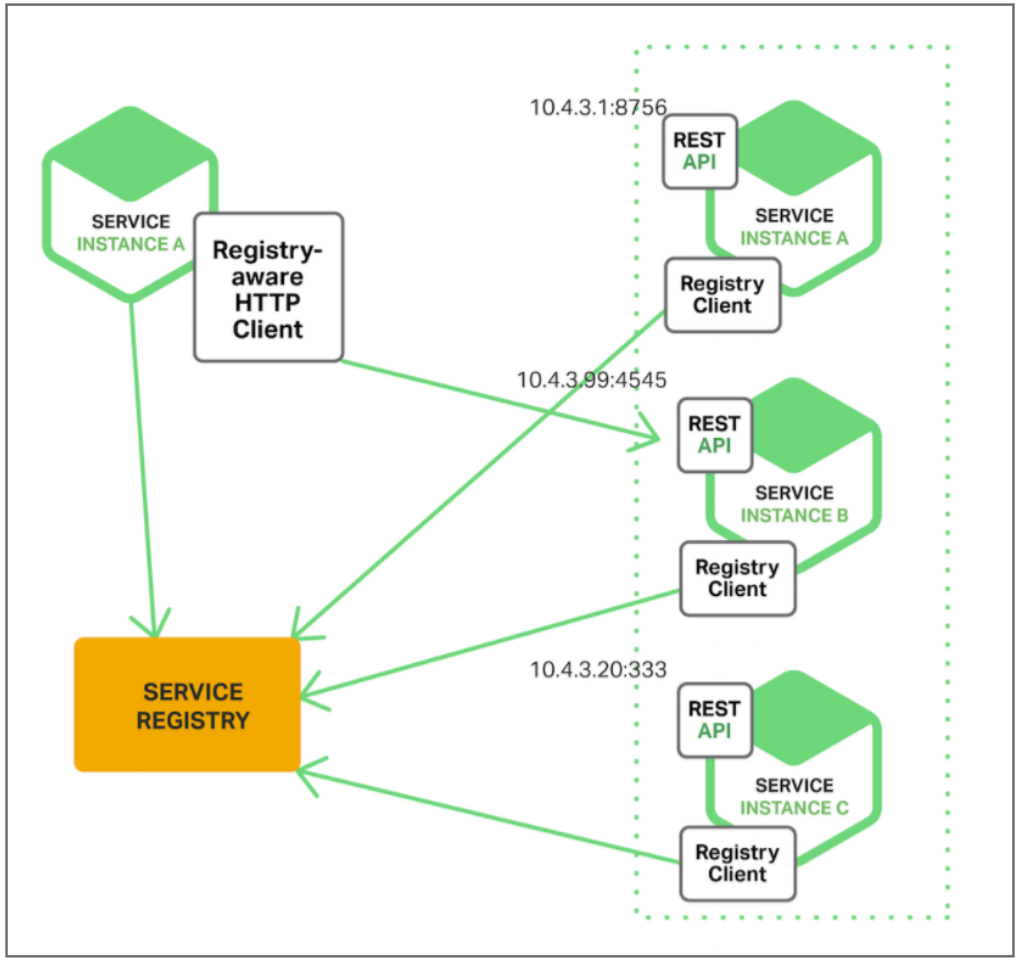
  E.g. Netflix Eureka, consul, Zookeeper

  https://github.com/hashicorp/consul-template -- dynamically reconfigure NGINX nginx.conf

Client-Side Service Discovery Pattern

The client queries a **service registry**, which is a database of available service instances. The client then uses a load-balancing algorithm to select one of the available service instances and makes a request.

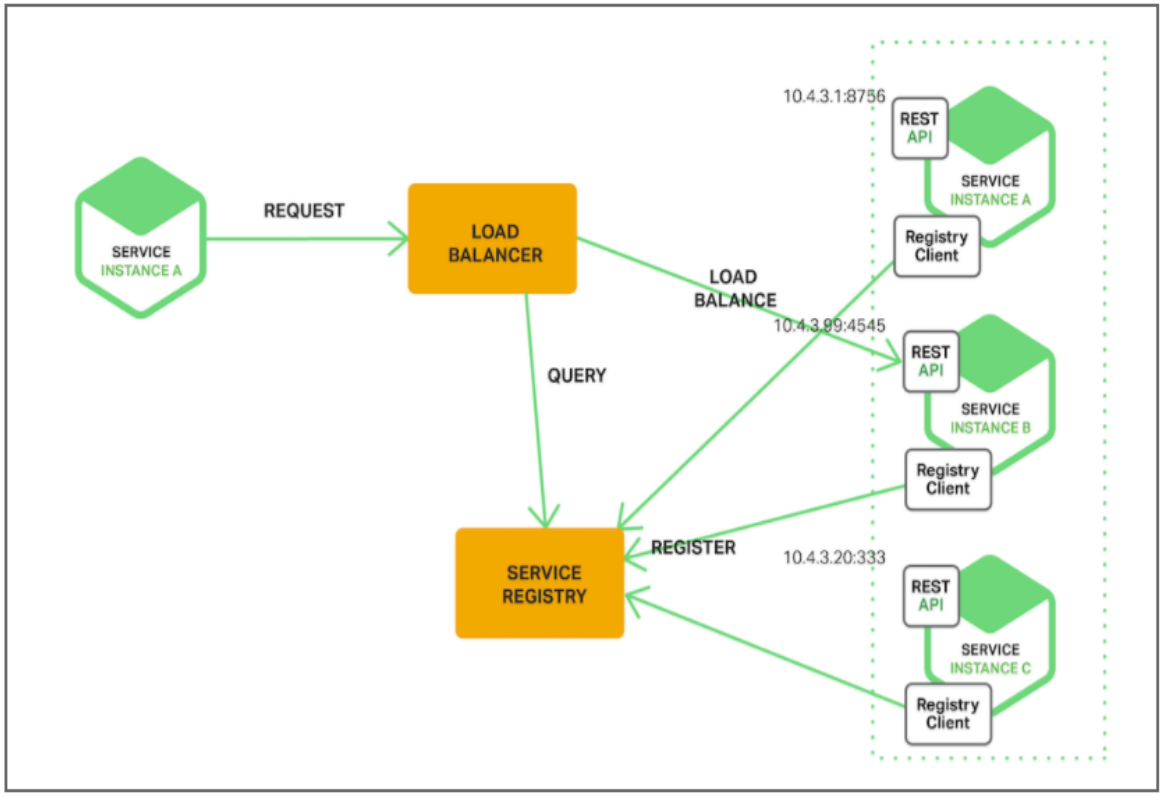
E.g. Netflix OSS



Server-Side Service Discovery Pattern

The client makes a request to a service via a load balancer. The load balancer queries the **service registry** and routes each request to an available service instance. As with client-side discovery, service instances are registered and deregistered with the service registry.

E.g. Kubernetes, Mesos, AWS Elastic LoadBalancer



API Gateway

Single entry point for all clients. The API gateway handles requests in one of two ways:

- Proxied/routed to the appropriate service
- Fanned out to multiple service

The API gateway can expose a different API for each client.
E.g., **Netflix API** gateway runs client-specific adapter code

Benefits:

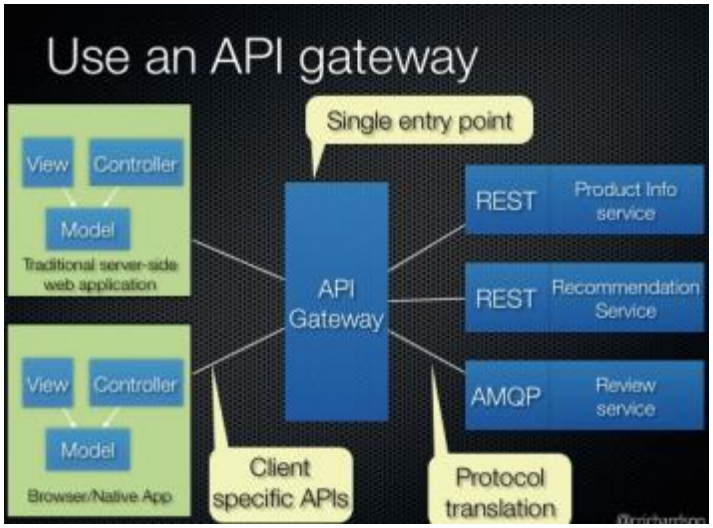
- Insulates the clients from how the application is partitioned into microservices
- Insulates the clients from the problem of determining the locations of service instances
- Provides the optimal API for each client

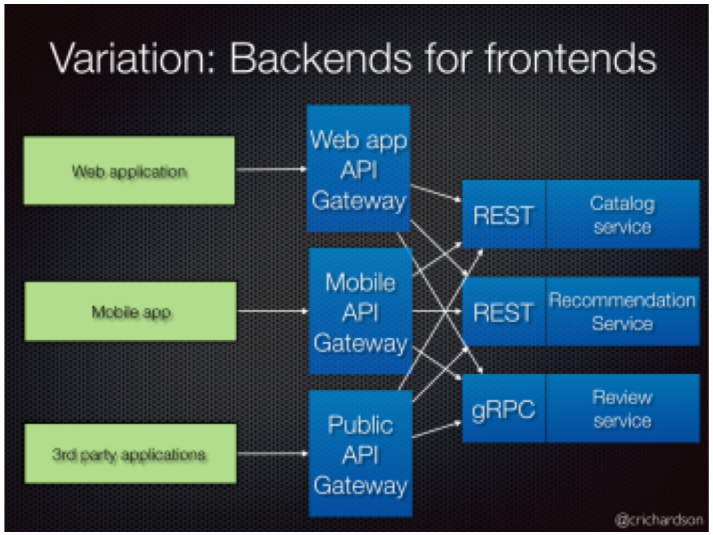
Drawbacks:

Increased complexity - another moving part to develop, deployed and managed

Increased response time due to the additional network hop through the API gateway.

E.g. Netty





Pattern - Categories

- **Data Management**
- **Communication**
- **Deployment**
- **Discovery**
- **Reliability**
- Observability
- Testing

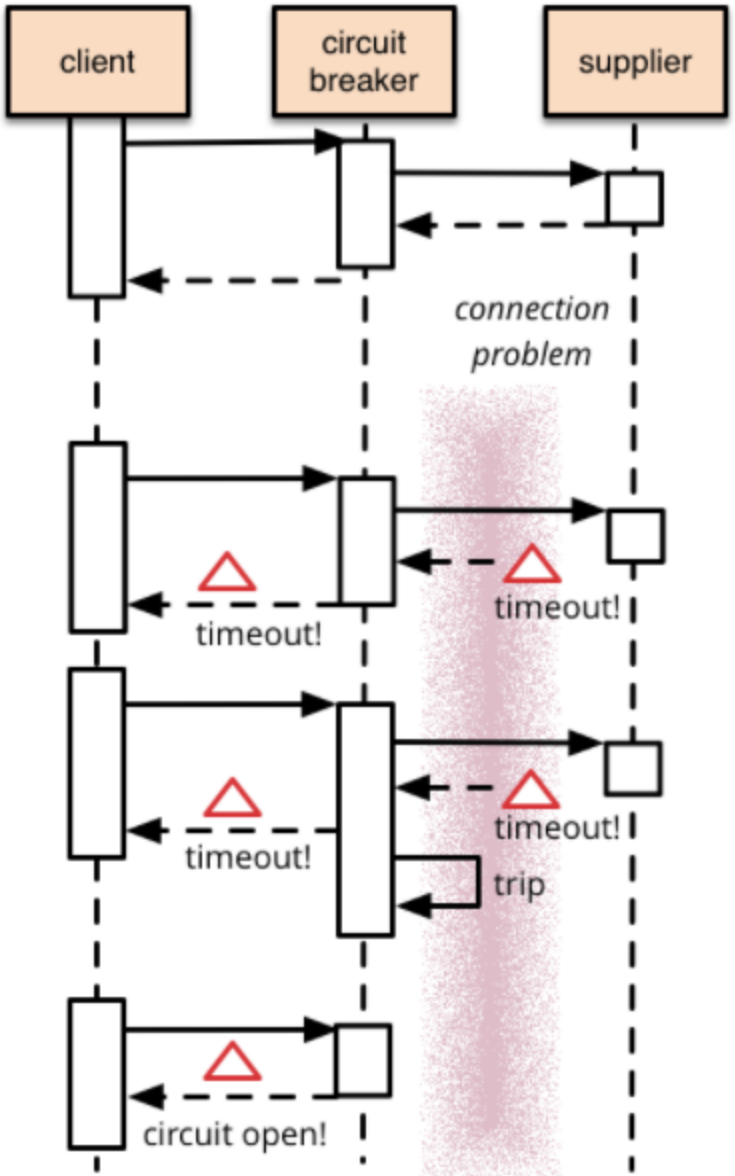
Circuit Breaker

Reason:

- **Remote calls can fail, reach timeout due to unresponsive supplier.**
- **Increased number of calls may lead to cascading failures across multiple systems**

What is it:

- **Wrap a protected function call in a circuit breaker object, which monitors for failures. Once the failures reach a certain threshold, the circuit breaker trips, and all further calls to the circuit breaker return with an error, without the protected call being made at all.**



Pattern - Categories

- **Data Management**
- **Communication**
- **Deployment**
- **Discovery**
- **Reliability**
- **Observability**
- Testing

Observability Patterns

- Application Metrics
- Health check API
- Distributed Tracing
- Exception Tracking
- Log aggregation
- Audit Logging

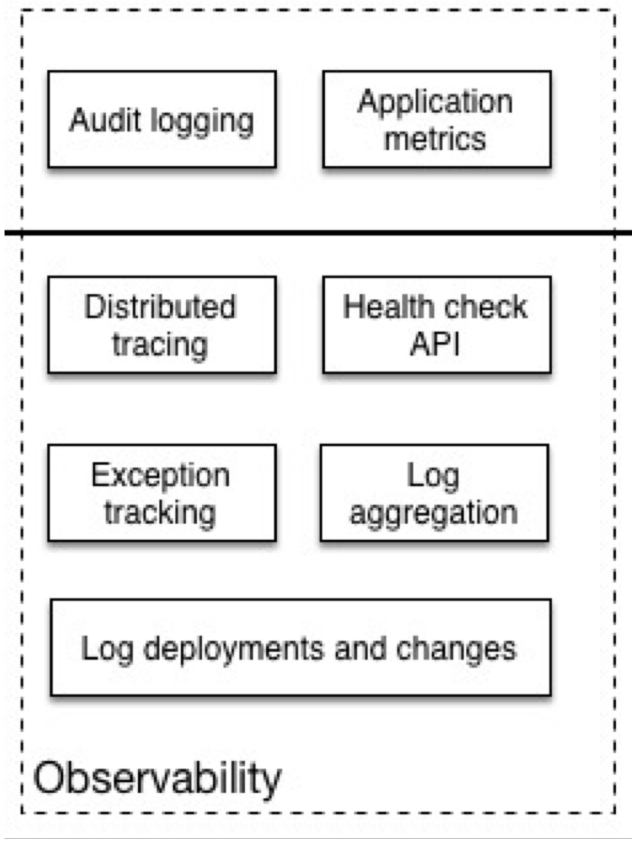
Observability (Metrics)

Health Check API

A health check client - a monitoring service - periodically invokes the endpoint to check the health of the service instance

- Ideally, you have a service that can ensure your service is active and accepting requests, a 'Health Check'

Health check examples included Consul, Kubernetes, and various libraries such as Sprint Boot, Gokit, etc.



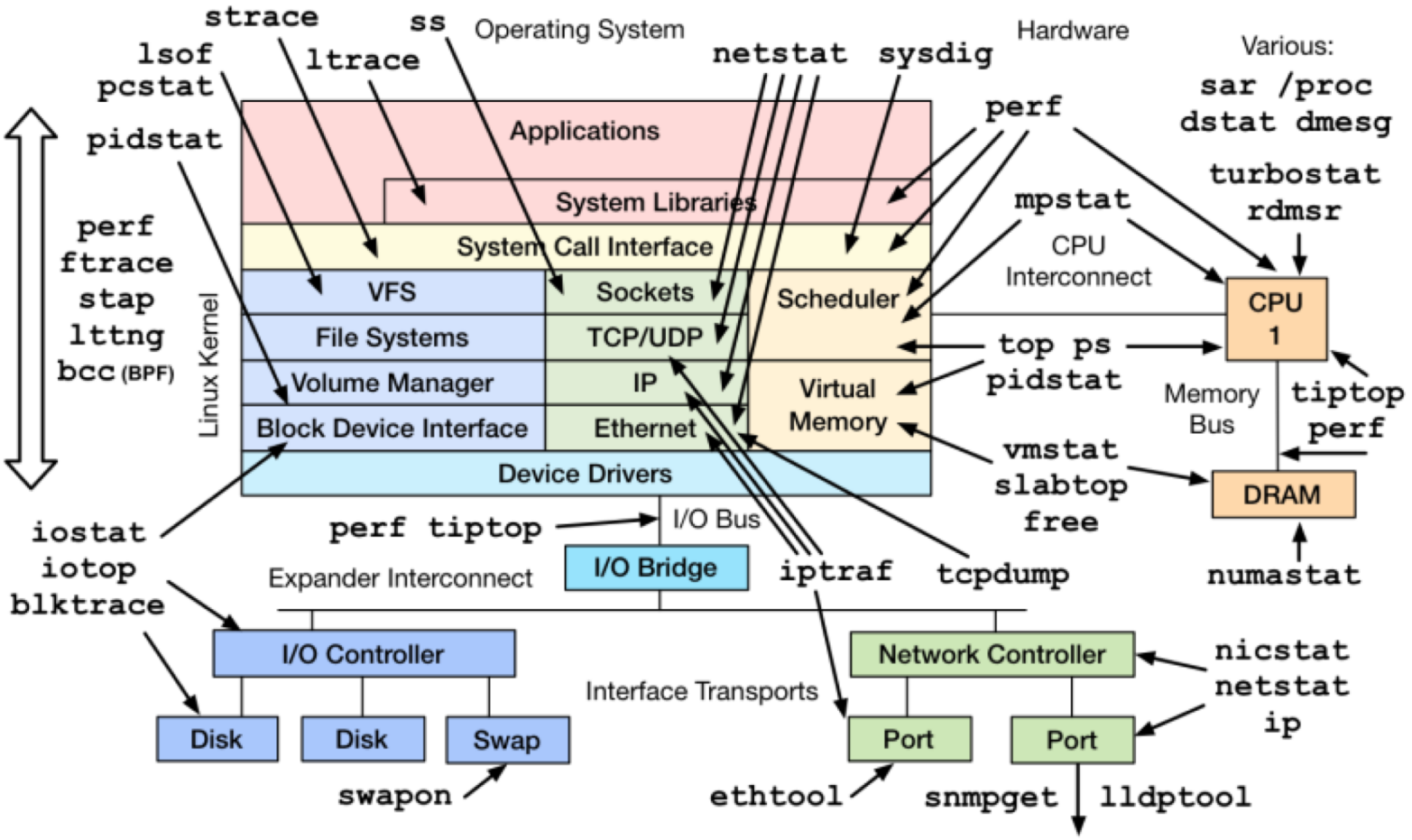
Application Metrics

- Need to understand the behavior of an application so performance and other problems can be proactively managed
- Typically come in two models:
  - Push – service pushes metrics (typically an agent) from the service
  - Pull – Service pulls metrics from the service (typically a dashboard)
- Examples include Prometheus, AWS Cloud Watch, Sysdig, etc

Observability Metrics

- **USE** method (Brendan Gregg)
- For every resource, check
  - **Utilization**: average time that the resource was busy servicing work
    - Example: One disk is running at 90%
  - **Saturation:** the degree to which the resource has extra work which it can't service, often queued
    - Example: The CPUs have an average run queue length of four
  - **Error count** (rate): the count of error events
    - Example: This network interface has had fifty late collisions
- Useful for resources such as queues, CPU's, memory, interconnects, etc.

Linux Performance Observability Tools



http://www.brendangregg.com/linuxperf.html 2017

Observability Metrics

RED method (Tom Wilkie)

For every service, check that

- **Request count** (rate)
- **Error count** (rate)
- **Duration**

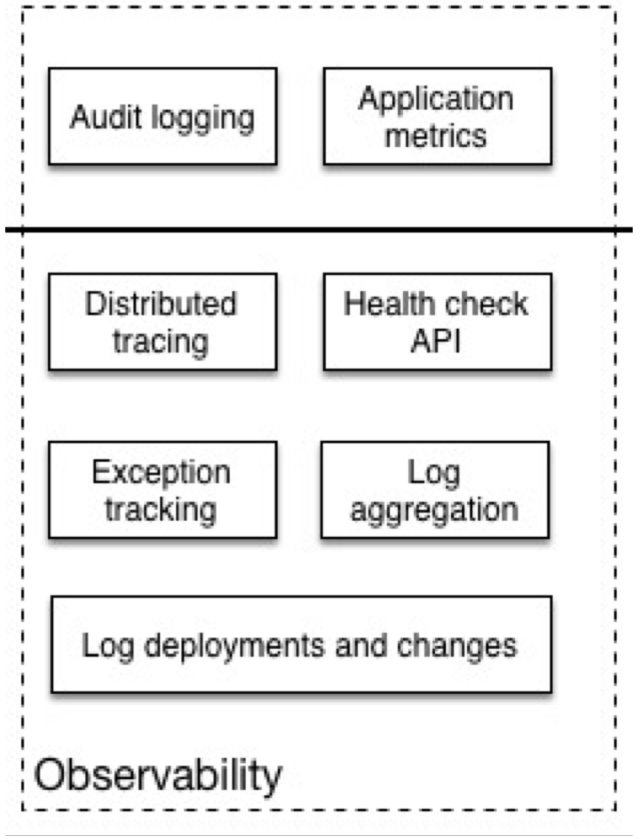
Note:

Observability = Logging + Monitoring + Tracing + Visualization

All good instrumentation libraries (Prometheus, Graphite, etc.) have at least three main primitives

- Counter: records events that happen such as incoming requests, bad requests, errors, etc.
- Gauge: records things that fluctuate over time such as the size of a thread pool
- Histogram: records observations of scalar quantities of events such as request durations

Observability (Tracing)



Distributed Tracing

- It becomes quite important to understand what broke and why

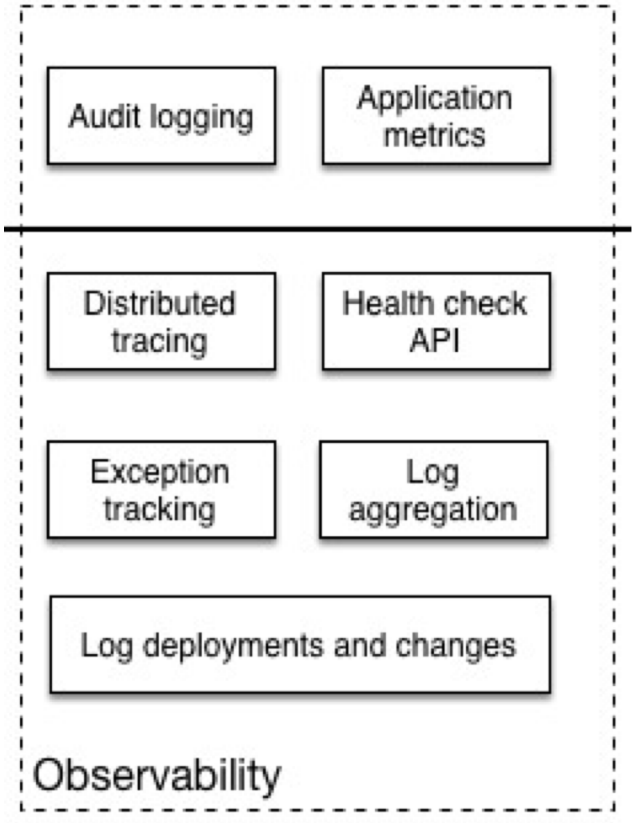
Common Distributed Tracing functionality

- Assign each external request a unique ID
- Pass ID to all services involved with handling the request
- Include id in all log messages
- Record information (start/end time) about requests and operations performed

Observability (Exceptions and Logging)

Exception Tracking

- It is crucial to save errors and the corresponding stack trace
- Ideally, exceptions are de-duplicated and recorded for further aggregation and investigation



Log deployments and changes

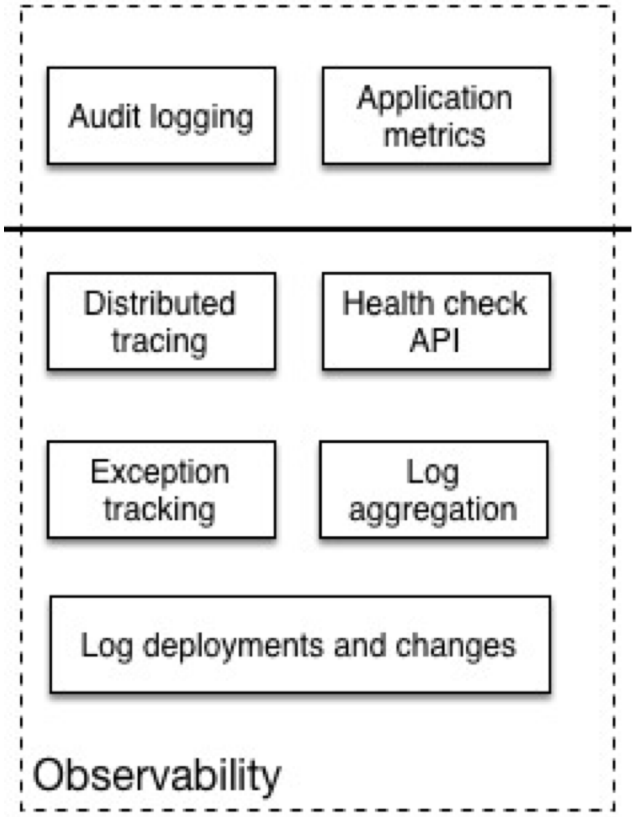
- Track changes as they are opportunities for failure
- For example, tracking deployments and changes so they can be easily correlated with issues later for faster resolution

*Etsy graphing error rates against new code deployments

Observability (Aggregation and Audit)

Log Aggregation

- Requests span multiple instances over multiple machines. It is important to gather information in a standardized format
- Log files contain errors, warnings, and debug information
- It is important to aggregate, search and analyze such logs
- Popular solutions for this include the ELK stack (Elastic Search, Logstash and Kibana) and AWS Cloud Watch



Audit Logging

- It is vital to know actions a user has performed.
- For security and compliance, this is often a must have
- Average time to detect intrusions is 98 days for financial services, 197 days for retail, it should/can be much lower

Logging Best Practices

- Log output as a set of JSON key/value pairs instead of pure text
- Easier for computer to test and aggregate
- Easier to make rules and query
- Comprehensive

- Do this:
  - “Timestamp: 2017-09-01 1:25”, ”caller”:”main.go:5”, “transport”:”HTTP”,”addr”:”8080”
- Not this
  - “2017-09-01 1:25 main.go 5 HTTP 8080”

Pattern - Categories

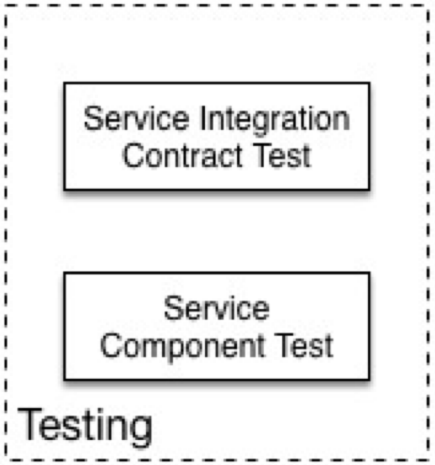
- Data Management
- Communication
- Deployment
- Discovery
- Observability
- Testing

Service Integration/Component Testing

Service Integration Contract Tests

- End-to-end testing is slow and expensive
- Test the APIs (functionality) provided by services

- **Black box testing**



Service Component Tests

- Test the components of the service
- A test suite that tests a service in isolation using test doubles for any services that it invokes.

- **White box testing**

End-To-End tests Vs Unit Test

End-to-end tests: time taking and expensive

- The dev team might have to spend weeks finding all the issues breaking the end-to-end test
- They would be better served with a large base of unit test that find bugs quickly

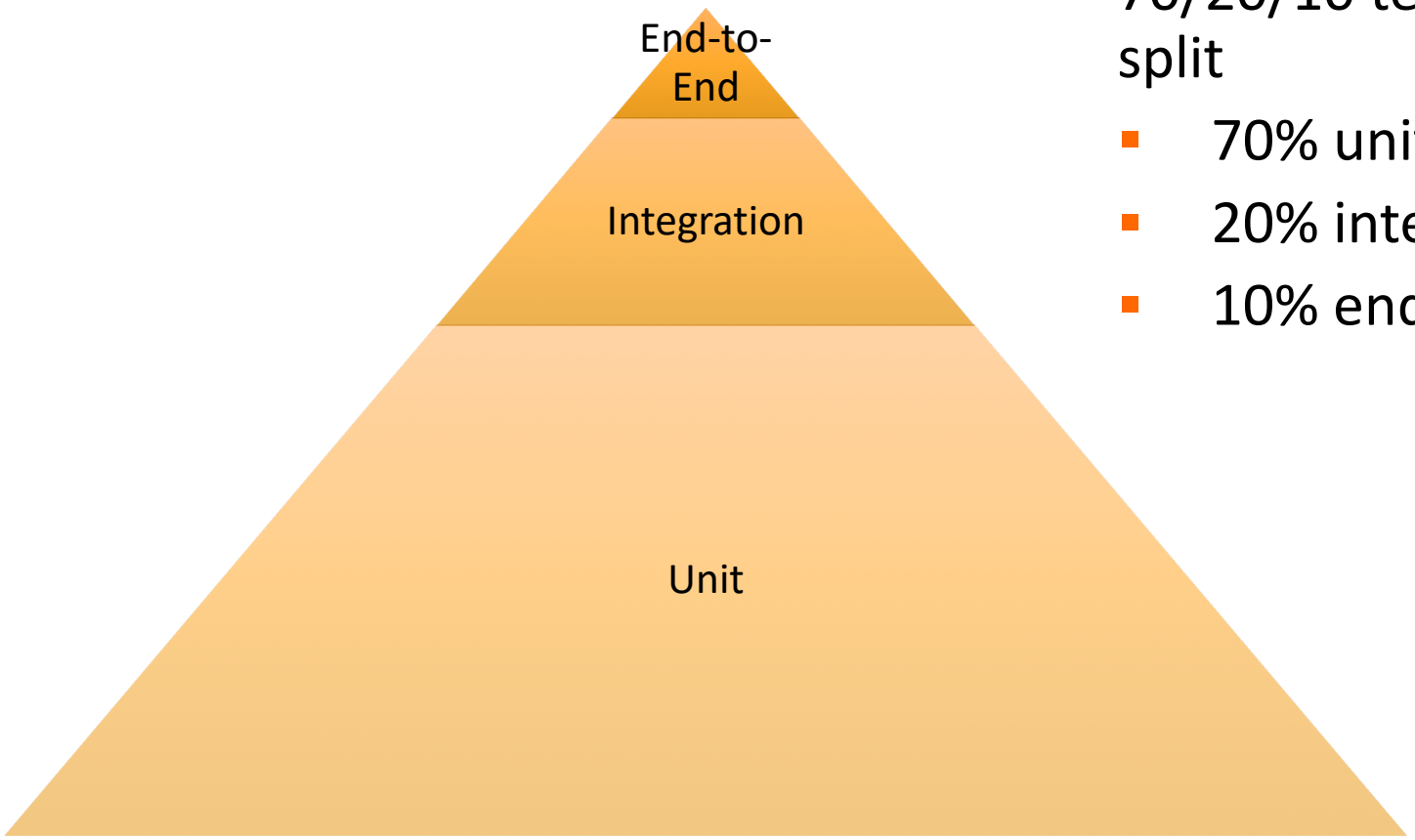
- Slow
- No isolated failures
- Simulates a real user

Unit Tests: Fast and cheap

- Unit tests should form the majority of tests as they quickly identify bugs and represent a fast feedback loop
- When a Unit test fails, the function and area that fails is much narrower in scope than broad end-to-end tests

- Fast
- Isolated Failures
- But do not simulates a real user

Testing Pyramid



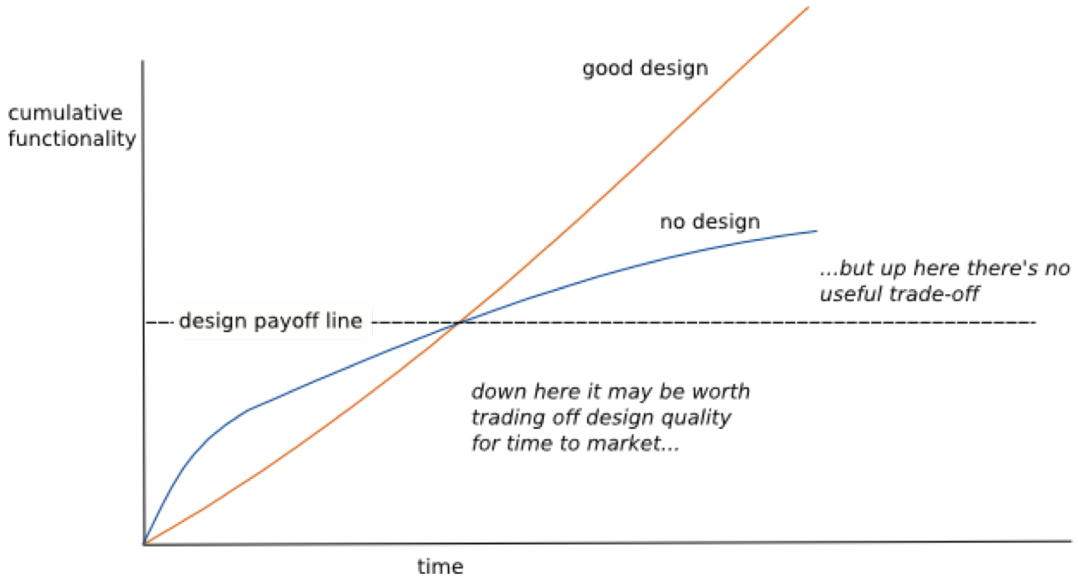
Google recommends a 70/20/10 test composition split

- 70% unit tests
- 20% integration tests
- 10% end-to-end tests

Technical Debt Payback

Addressing **Technical Debt** makes an **assumption** that the team will get a **payoff** on their **investment**

- Have more unit tests, helps in continuous deployment, helps in fast tracing of errors and building a resilient product – This reduces technical debt
- The interest from technical debt can be used to reduce even more technical debt (compounding)

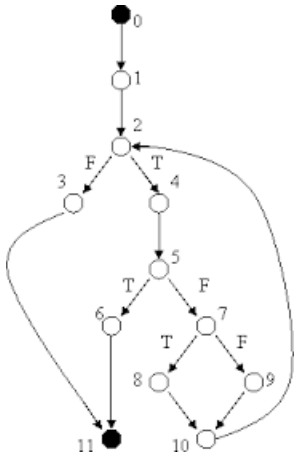


Static Code Metrics

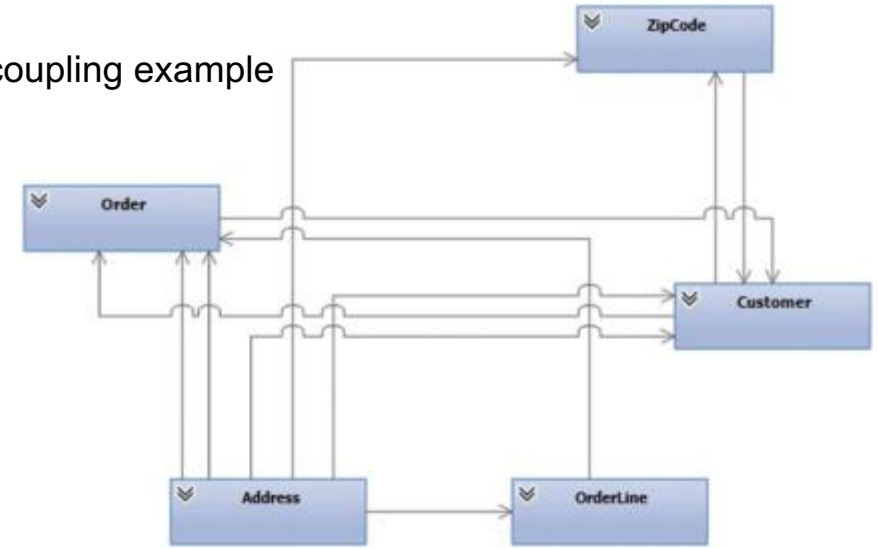
Coding Techniques to Improve Architecture Agility

- Code Complexity – recommend about 20 lines per method
- Cyclomatic Complexity (# of linearly independent paths) – recommend keep it low
- Coupling (measure of how many classes a single class uses), keep it low
- Inheritance, no deeper than 6.

Cyclomatic Complexity



Class coupling example



EXTERNAL BENCHMARKS (Velocity & Stability Metrics)

2016 IT Performance by Cluster

	High IT Performers	Medium IT Performers	Low IT Performers
Deployment frequency *For the primary application or service you work on, how often does your organization deploy code?*	On demand (multiple deploys per day)	Between once per week and once per month	Between once per month and once every 6 months
Lead time for changes *For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code commit to code successfully running in production)?*	Less than one hour	Between one week and one month	Between one month and 6 months
Mean time to recover (MTTR) *For the primary application or service you work on, how long does it generally take to restore service when a service incident occurs (e.g., unplanned outage, service impairment)?*	Less than one hour	Less than one day	Less than one day*
Change failure rate *For the primary application or service you work on, what percentage of the changes either result in degraded service or subsequently require remediation (e.g., lead to service impairment, service outage, require a hotfix, rollback, fix forward, patch)?*	0-15%	31-45%	16-30%

* Low performers were lower on average (at a statistically significant level), but had the same median as the medium performers.

*** Source: Puppet Labs Report**

Competitive Advantage

High performing teams are breaking away from the pack and the status quo of even three years ago is a dangerous assumption currently.

A team which had state of the

Like Inflation, technical debt can be gauged by a basket of indicators

If a theoretical **team** spent **zero time** addressing **technical debt**, we would expect these **thresholds** to be crossed

Static Code Metrics
- Code Complexity =<20 lines per method
- Cyclomatic Complexity =<1-5 paths
- Coupling =< 9
- Inheritance =< 6

Security Vulnerabilities
- Critical =< 0 fix immediately
- Med/Low =< fix within 30 days

Open Defects
- Average time open <= 30 days

Code Velocity
- Lead (Cycle) time <= 6 weeks
- Deployments per month >= 2

Code Stability
- Uptime >= 99.9 %
- Mean Time To Recovery <= 4 hours

Tests
- Unit Coverage >= 80%

Microservices - Definition of Done

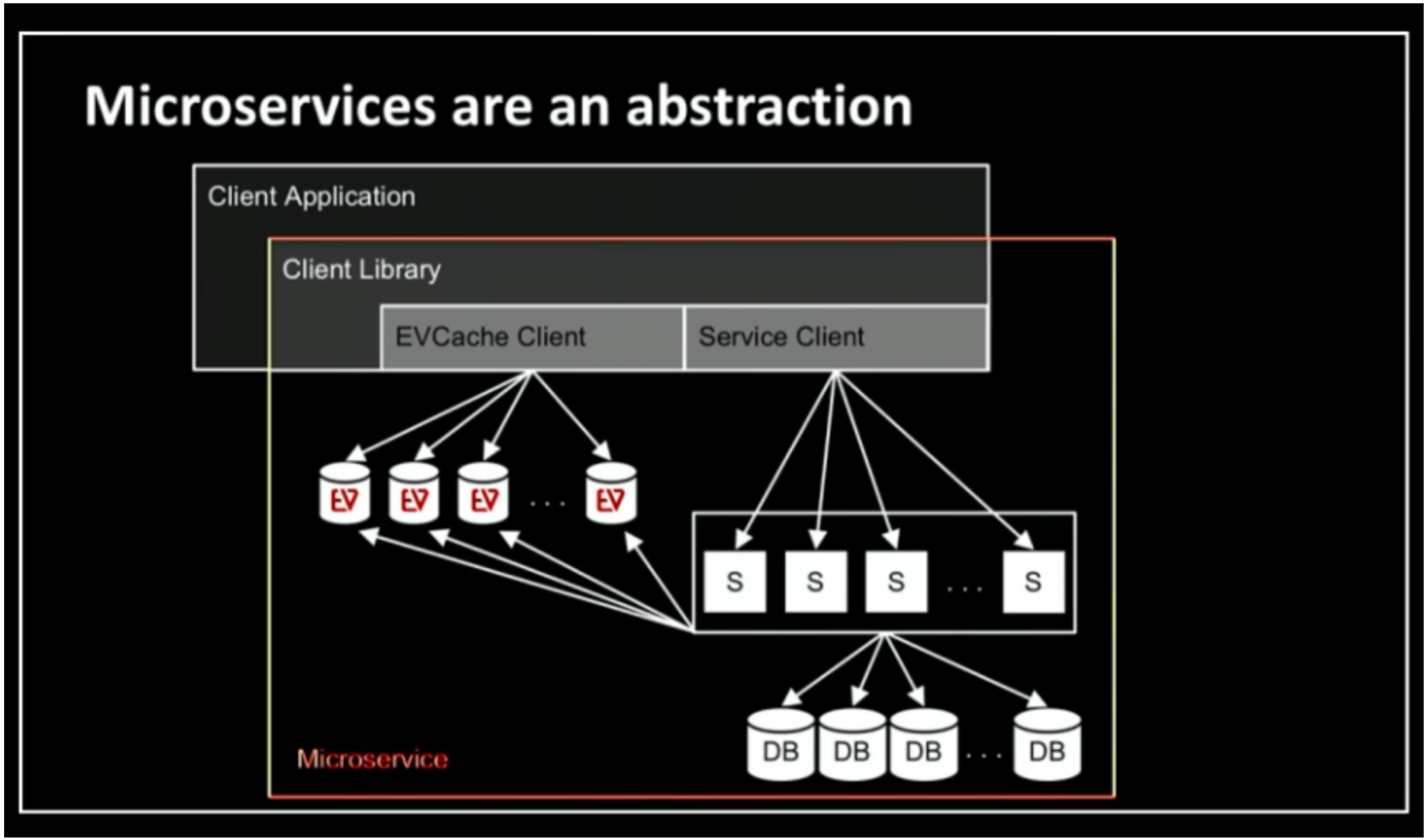
- Can be deployed, replaced, and upgraded independently (Services are autonomous, loosely coupled)
- Published service interface (Services are discoverable)
- Bounded Context (Specific responsibility enforced by explicit boundaries)
- Externally accessible API
  - No other forms of communication allowed, no tickets, direct linking, reads of data stores, no back doors, only service interface calls over the network

Microservices in Netflix

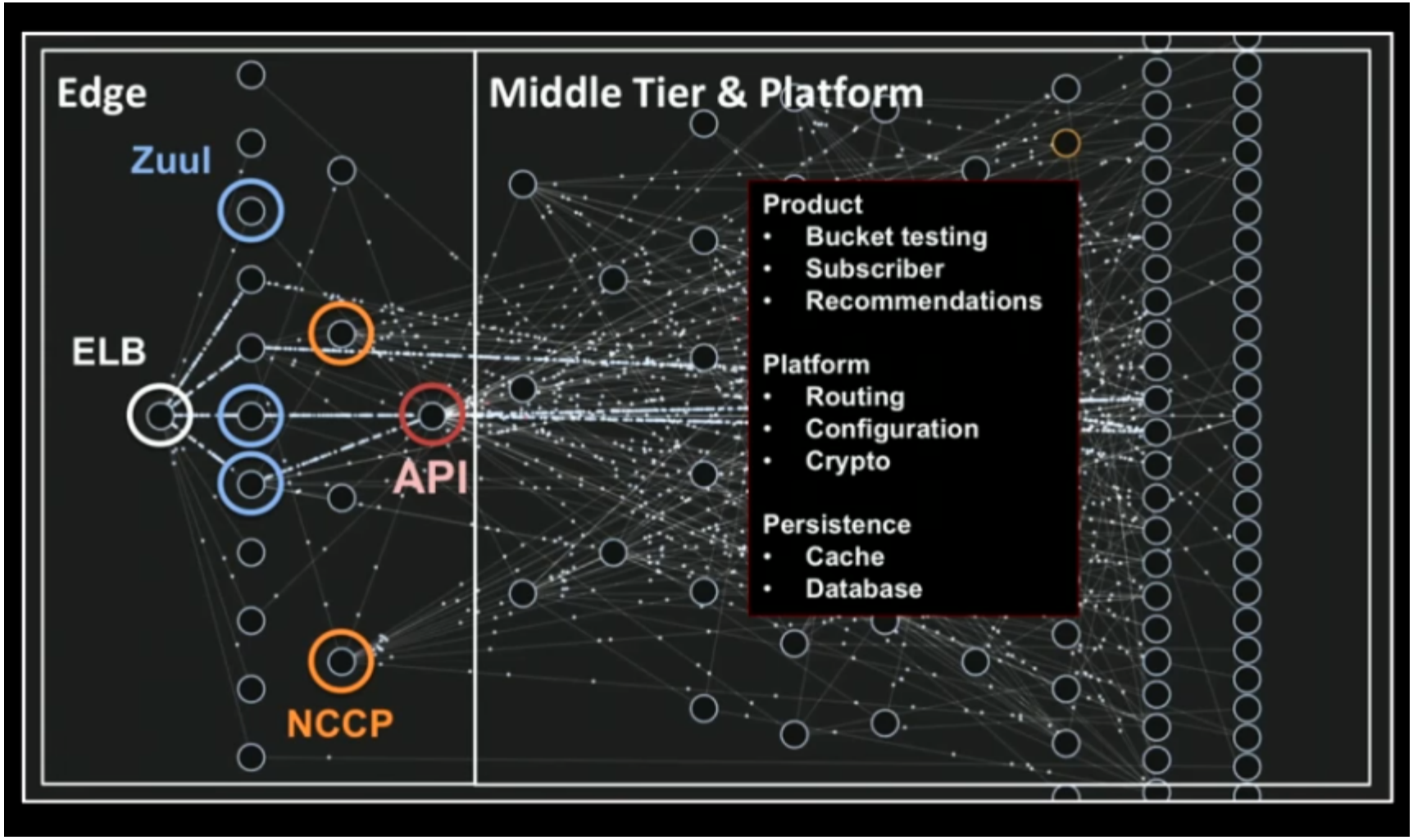
- By Josh Evans (Director of Operational Engg.)
- At Qcon, San Francisco, 2016

https://www.youtube.com/watch?v=CZ3wIuvmHeM

Microservices in Netflix



Microservices in Netflix



Microservices in Netflix

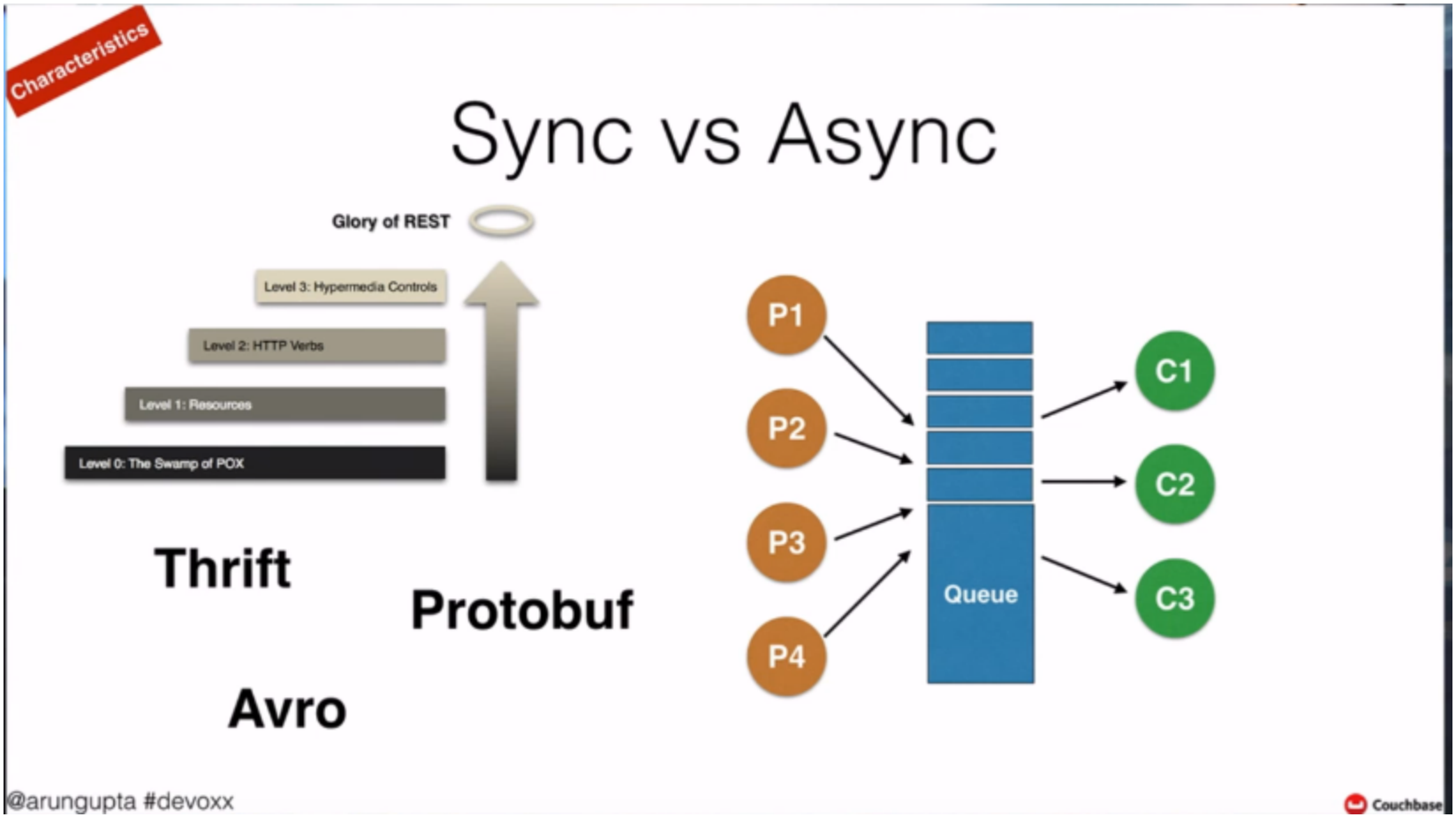
- ELB
- Dependencies
- Eventual consistency, CAP
- Circuit breakers
- Workload partitioning
- Auto-scaling
- Multi-region failure strategy

Microservices in Couchbase

- By Arun Gupta (VP Engg.)
- At Devoxx, Belgium, 2015

https://www.youtube.com/watch?v=iJVW7v8O9BU

Microservices in Couchbase

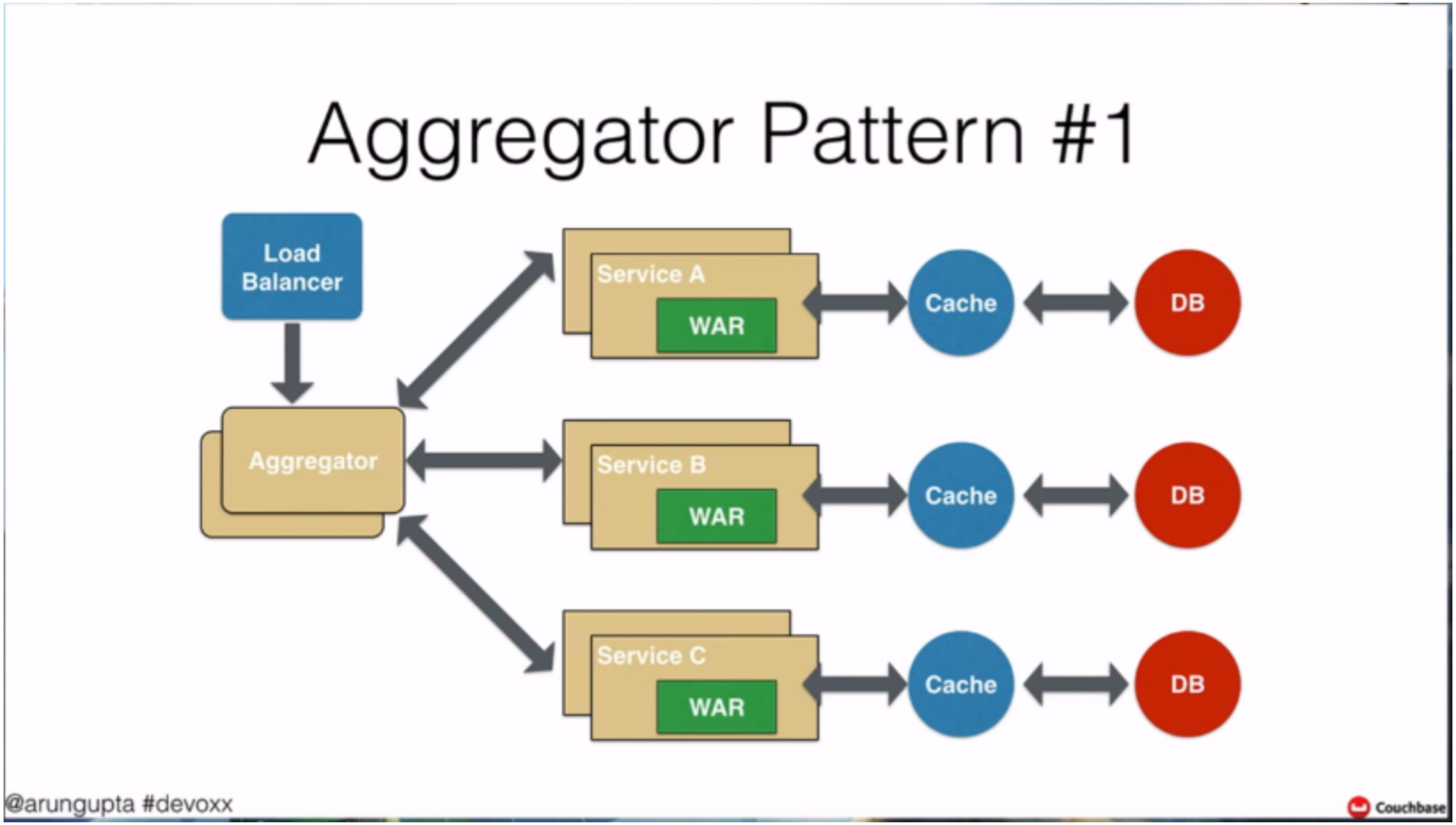


Microservices in Couchbase

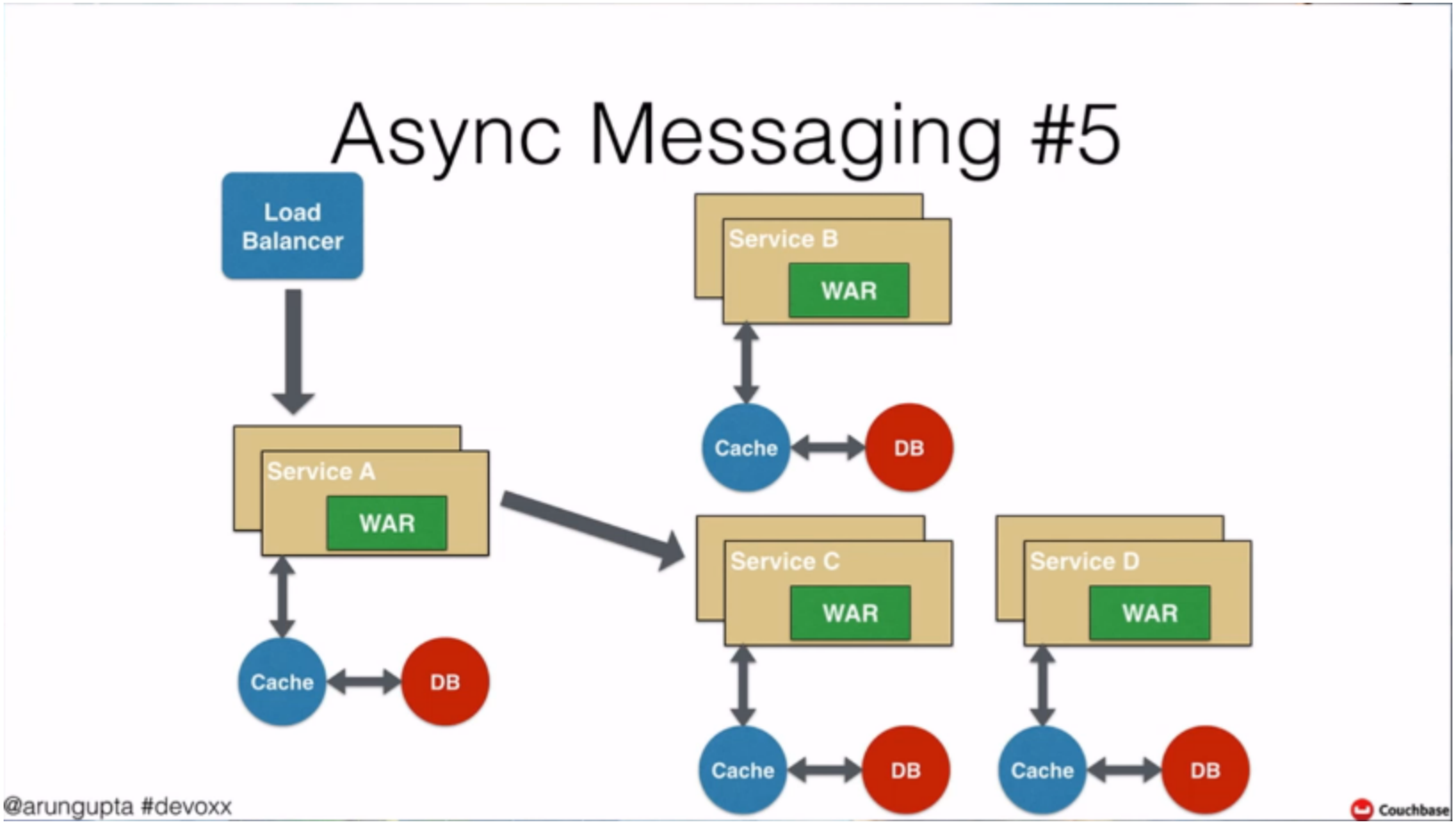
Strategies for decomposing

- Verb or usecase - e.g. Checkout UI
- Noun - e.g. Catalog product service
- Bounded context
- Single Responsible Principle - e.g. Unix utilities

Microservices in Couchbase



Microservices in Couchbase



Microservices in Couchbase

- Microservices architecture and Design patterns
- An example (in Java EE), for a shopping cart application.
  - Monolithic → Microservices concept and code
- Service Registry: FixedURI, Zookeeper
- Event Sourcing, CQRS

Microservices in Development

Part 2

Notes

Optimize for Time to Value

"Today, when organizations measure and optimize their activities, time to value is becoming a dominant metric"

–Adrian Cockcroft, Devops thought leader reknown for architecture work at Netflix and VP of AWS

Overview of Docker Containers

Docker installation as done in Classroom work 101

install docker

```
curl -sSL https://get.docker.com/ | sh
```

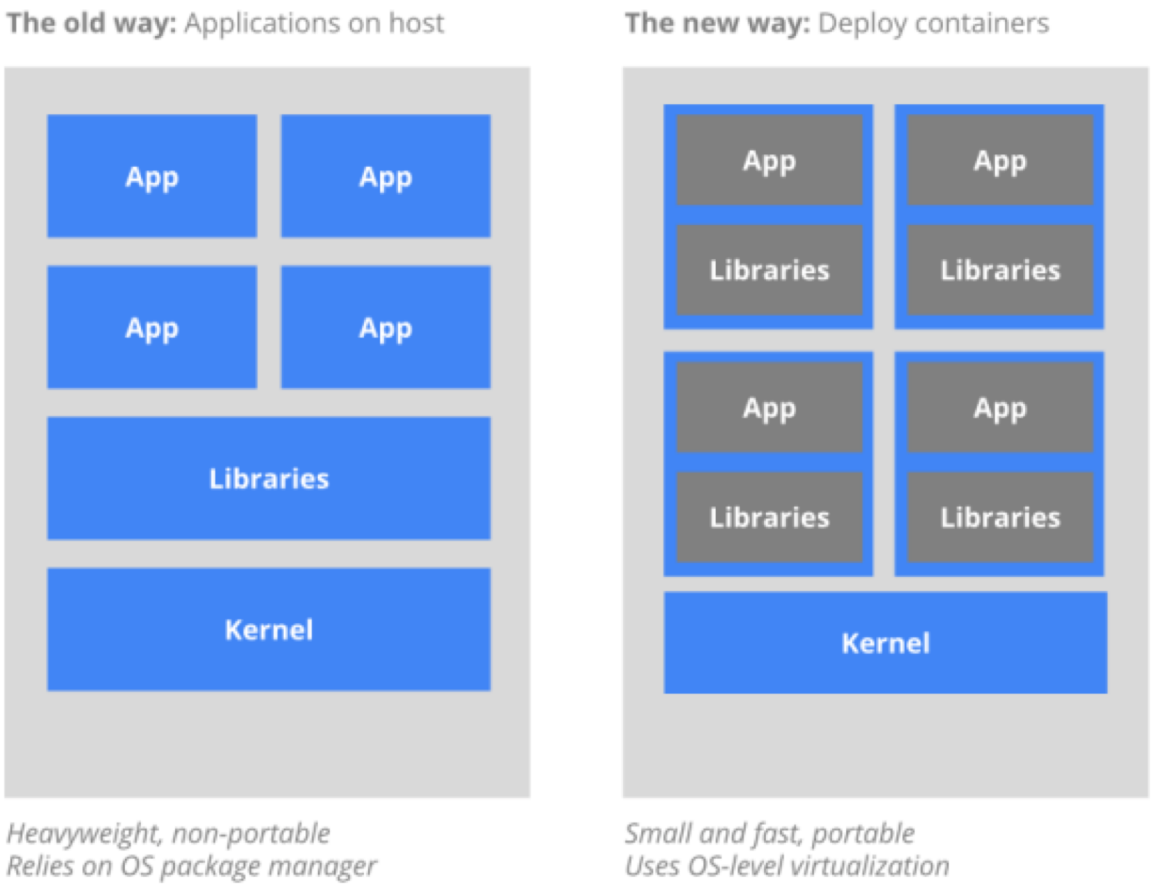
add sudo access to the user

```
sudo groupadd docker
sudo usermod -aG docker $USER
#exit and login again
```

install docker-compose

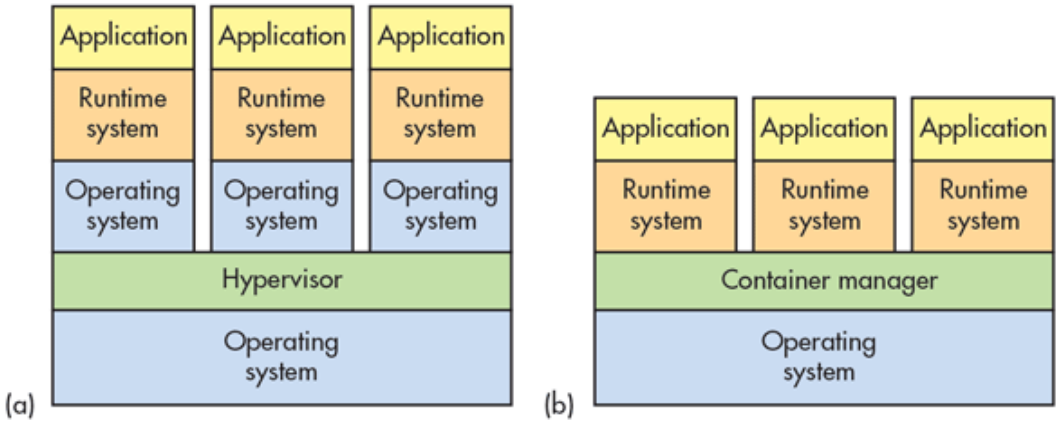
```
sudo curl -L https://github.com/docker/compose/releases/download/1.21.0/docker-compose-`uname -s`-`uname
sudo chmod +x /usr/local/bin/docker-compose
```

Containers

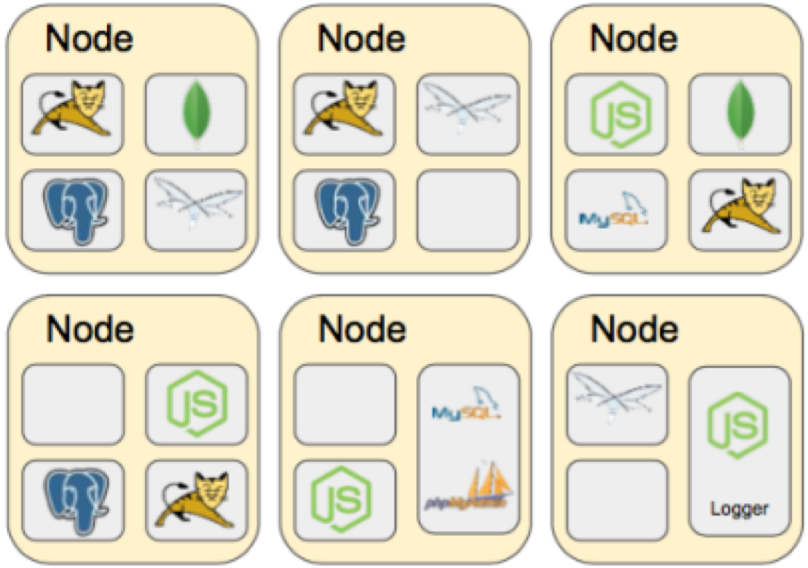

The old way: Applications on host
App
App
App
App
Libraries
Kernel
Heavyweight, non-portable
Relies on OS package manager
The new way: Deploy containers
App
App
Libraries
Libraries
App
App
Libraries
Libraries
Kernel
Small and fast, portable
Uses OS-level virtualization

Container Advantages

- Portable
- Isolated
- Lighter footprint & overhead (vs VMs)
- Simplify Devops practices
- Speed up Continuous Integration
- Empower Microservice architectures and adoption



Challenges with Multiple Containers



- How to scale?
- Once I scale, where are they?
- How do my containers find each other?
- How should I manage port conflicts?
- What if a host fails?
- How to update them? Health checks?
- How will I track their logs?

The Need for Container Orchestration

- Containers are becoming the standard unit of deployment
- Each container image has
  - Code
  - Binaries
  - Configuration
  - Libraries
  - Frameworks
  - Runtime
- Developers and Operators love containers

The Need for Container Orchestration

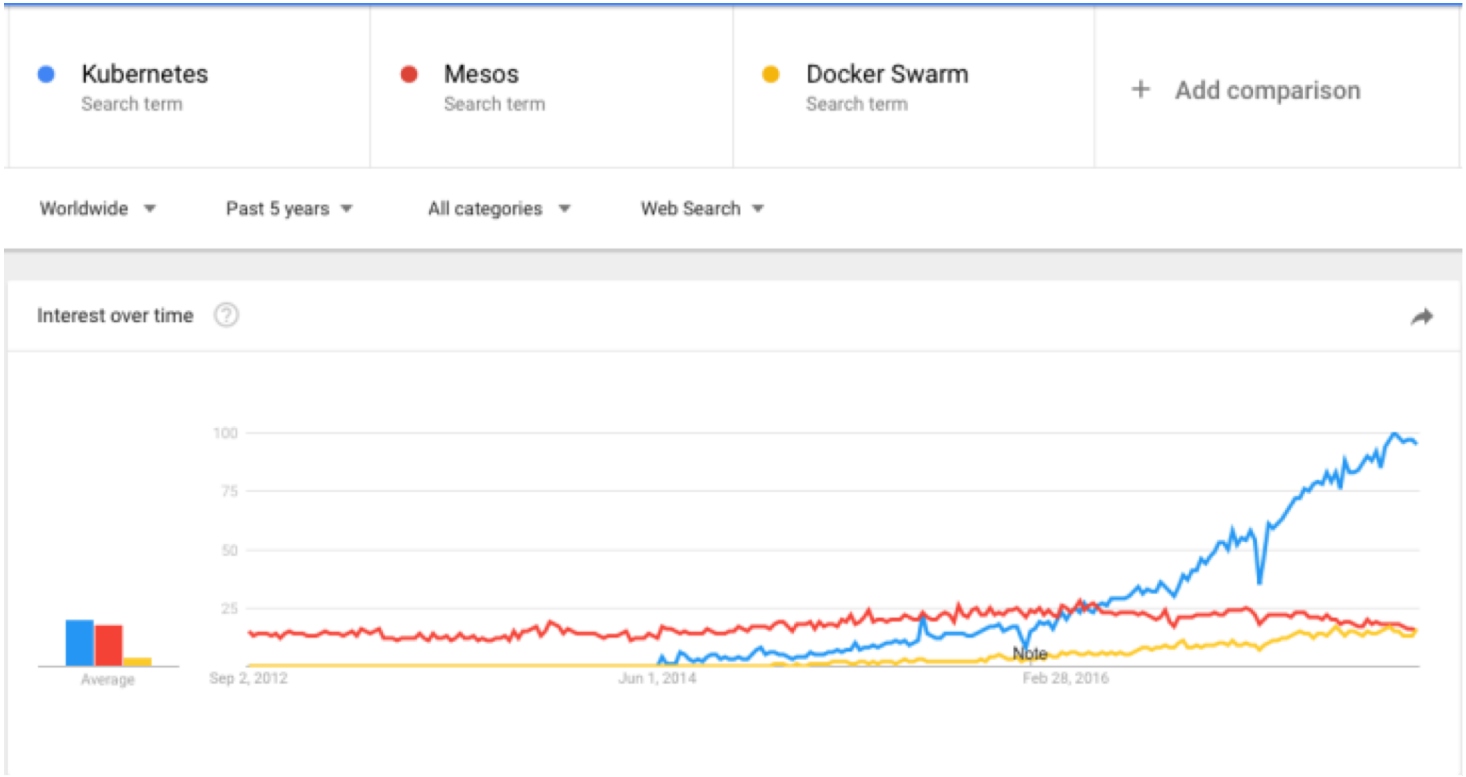
- Docker has solved the problem of packaging, deploying and running containerized applications
- Docker has 3 key components
  - Container Engine
  - Registry
  - Tools (like docker-cli, docker-compose, docker-machine, etc.)
- Docker is great for managing a few containers running on a fewer machines
- Production applications deal with dozens of containers running on hundreds of machines

The Need for Container Orchestration

- The unit of deployment is changing from a machine to a container
- Infrastructure has become immutable
- Emphasis on treating the datacenter as a large server (cluster)
- Tools are evolving to manage the new datacenter infrastructure
  - Docker Swarm
  - Kubernetes
  - Mesosphere DC/OS
- Manage the lifecycle of containerized applications running in production
- Automate the distribution of applications

Container Orchestration Tools

- The three most popular are: Kubernetes, Docker Swam & Mesos
- Kubernetes has become the unofficial standard

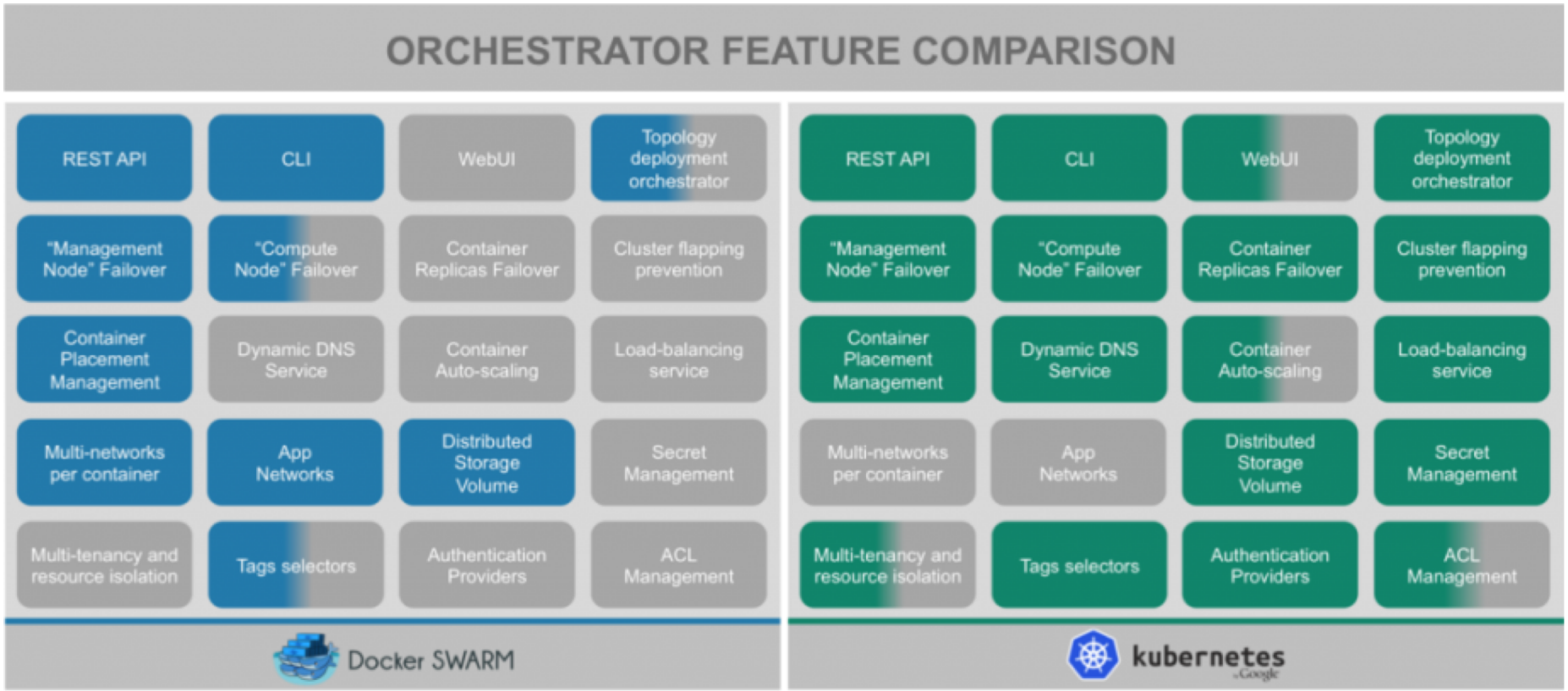


Kubernetes Stats

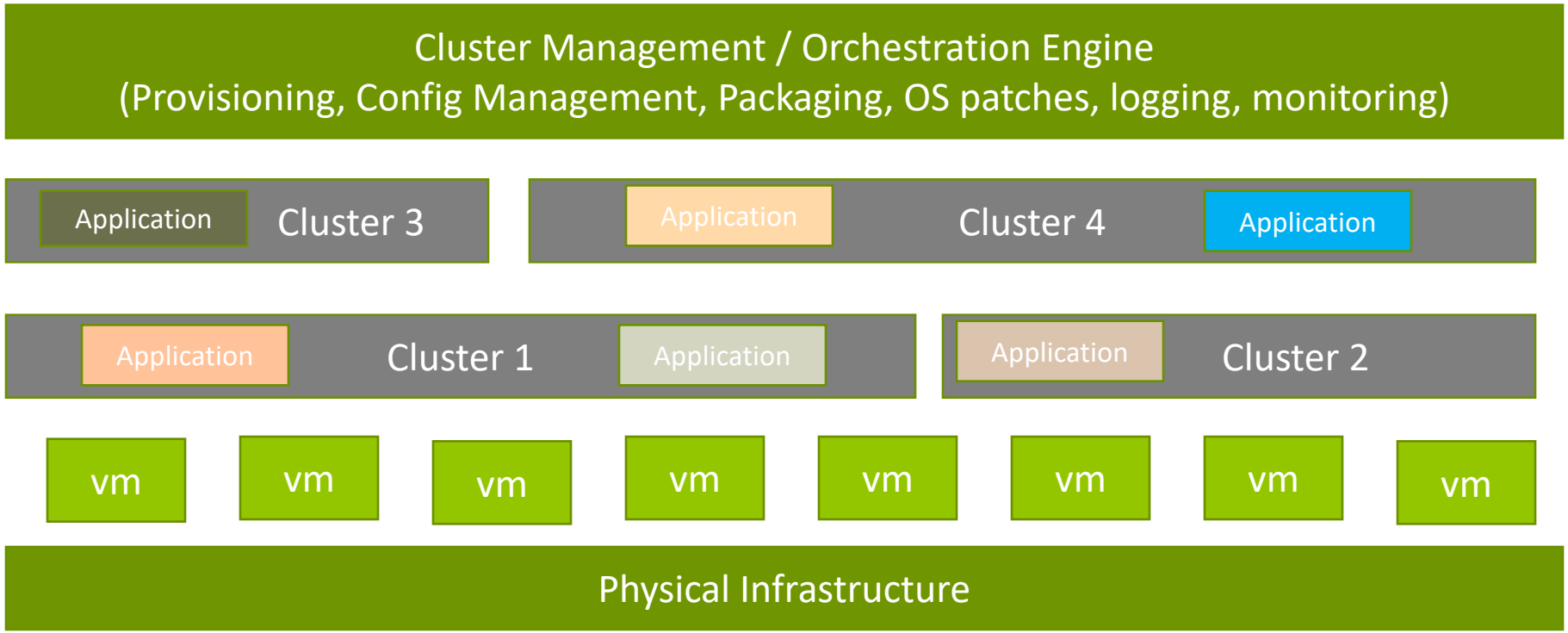
- **Top 100 project by stars**
- **22k LinkedIn professionals**
- **Largest container management ecosystem**

Github		
54k+ Commits	280+ Releases	1365+ Contributors
Top 100 Forked Github Project	Top 2 Starred Go Project	Top 0.01% Starred Github Project

Feature Comparison (March 2016)



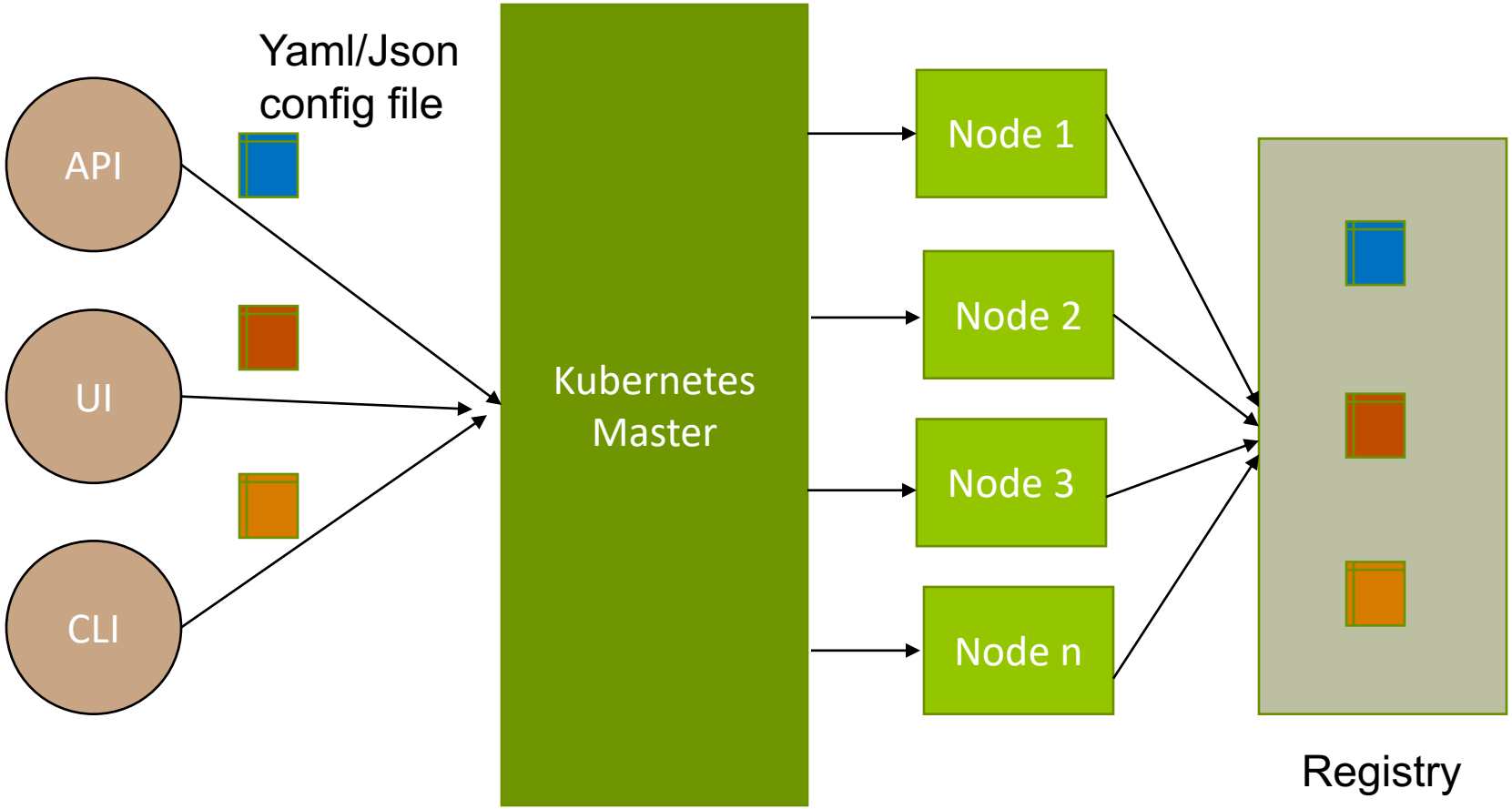
Architecture – Bird's Eye View



What is Kubernetes?

- Kubernetes is inspired from an internal Google project called Borg
- Open source project managed by the Linux Foundation
- Unified API for deploying web applications, batch jobs, and databases
- Decouples applications from machines through containers
- Declarative approach to deploying applications
- Automates application configuration through service discovery
- Maintains and tracks the global view of the cluster
- APIs for deployment workflows
  - Rolling updates, canary deploys, and blue-green deployments

Kubernetes Architecture



Configuring Kubernetes

- Minikube (Portable, One VM)
  - Simplest way to get Kubernetes cluster up and running
  - Supports Microsoft Windows and Mac OS X
- Kubernetes Multi-Node Cluster
  - Emulates production environment
  - Good for testing advanced scenarios
- Google Container Engine (low-friction, no-touch deployment)
  - Hosted and managed by Google
  - Powered by Google Compute Engine
- DC/OS with Mesos and Marathon
  - Hosted and managed by Mesosphere

Getting Started with Minikube

- Install Oracle VirtualBox for Mac
- Install Docker Toolbox for Mac
- Install Docker version manager
- Install the latest version of Minikube for Mac OS X
- Download the latest version of kubectl
- Run the following commands from the directory where kubectl is downloaded
  - chmod +x ./kubectl
  - sudo mv kubectl /usr/local/bin
- Launch minikube
  - minikube --start –vm-driver=virtualbox
- Test minikube installation
  - minikube status
  - Kubectl get cs

Local Kubernetes

Minikube is a option for developing locally, but some features are not available and require a cloud provider.

1. Install Hypervisor (Virtualbox, etc)
2. Stand up minikube & test

Google Compute Engine Kubernetes Connect

GCE Kubernetes Quickstart

1. Create project
2. Install gcloud (CLI)
3. Install kubectl (gcloud components install kubectl)
4. Standup Hello World

AWS Kubernetes Connect

AWS Kubernetes Setup

- AWS Does not have a managed K8s service.
- Two third party options include Heptio's quickstart or 'Kops'

Azure Kubernetes Connect

Azure Kubernetes Setup

1. Login Azure
2. Download CLI and login with CLI
    1. Powershell may be used for this example, but the CLI will be more versatile and used in future examples
    2. Azure Powershell may be used if limitations are understood: https://docs.microsoft.com/en-us/azure/cloud-shell/limitations

https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough

https://github.com/Azure-Samples/azure-voting-app-redis

Exercise 2.1 Kubernetes Master Installation

CLASSROOM WORK 20 minutes (required)

** This is an important exercise, if this exercise is not completed, you won't be able to run any kubernetes related exercise

1. **Install docker and docker-compose**
2. **Install kubectl**
3. **Install kubeadm**
4. **Install pod networking; Flannel**
5. **Initialize kubelet to create master node**

https://github.com/techtown-training/microservices-bootcamp/blob/master/exercise/exercise2.1-install_kube_master.md

Kubernetes Installation - Pointers

- Swap is disabled on all nodes
- SELinux is disabled on all nodes
- Firewall is disabled on all nodes
- Bridge configuration
- Docker 17.06.2 is installed
- Storage : Overlay
- Cgroup : Systemd
- Install crictl
- Install pod-network-cidr
- Setup private registry

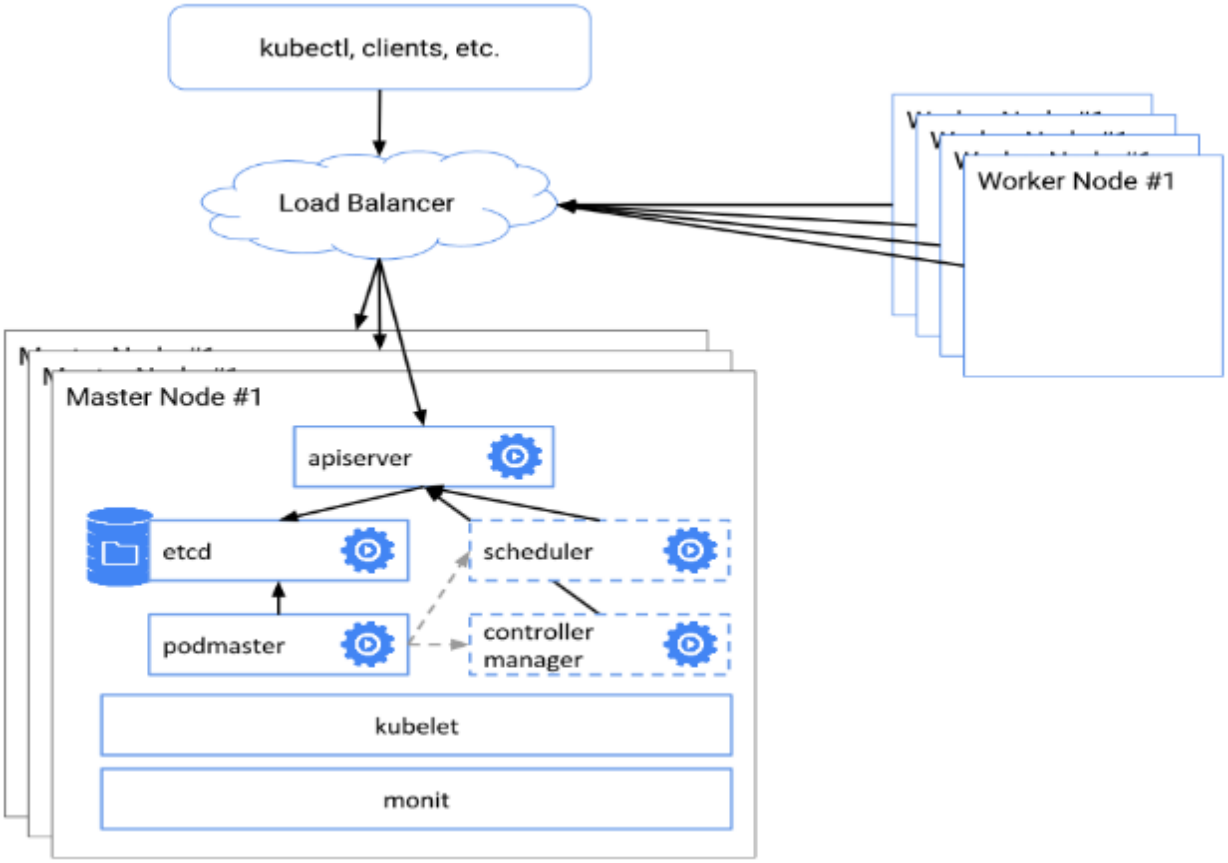
Installation includes:

- kubeadm
- kubelet
- kubectl

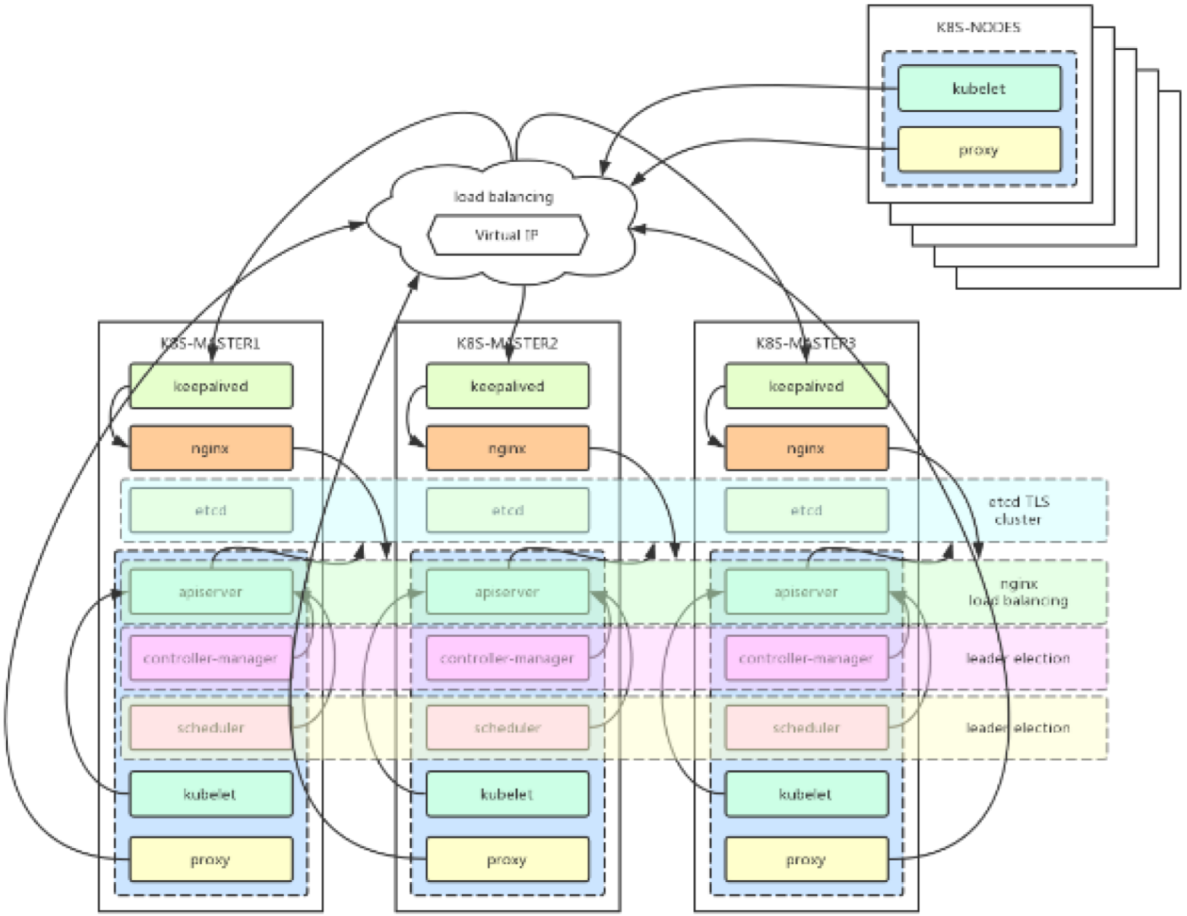
HA Kubernetes Installation
(high availability)

- **Creating the HA Kubernetes includes:**
  - **Creating reliable constituent nodes that collectively form the HA master implementation**
  - **Setting up a redundant, reliable storage layer with clustered etcd**
  - **Starting replicated, load balanced Kubernetes API servers**
  - **Setting up master-elected Kubernetes scheduler and controller-manager daemons**

HA Kubernetes Installation
(high availability)



HA Kubernetes Installation
(high availability)



Option:

- **Create a containerized cluster of «etcd»**
- **Keepalived to provide a VIP to the 3 instances of apiserver and nginx as a load balancer**

Amazon Case

Amazon Web Services: Situation

In 2002, Amazon was hitting a wall when trying to scale their website to grow with demand.

- **The system was limited by its databases and a large monolith "big ball of mud" system named Obidos.**
- **In response, Jeff Bezos sent a memo to technical staff which decreed the creation of a service-oriented architecture.**

Amazon Web Services: Action

- **Jeff Bezos 2002, SOA decree**

  1) **All teams will henceforth expose their data and functionality through service interfaces**
  2) **Teams must communicate with each other through these interfaces**
  3) **There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network**
  4) **It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols – doesn't matter**
  5) **All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions**

1 Google Platforms Rant https://plus.google.com/+RipRowan/posts/eVeouesvaVX

Amazon Web Services: Action

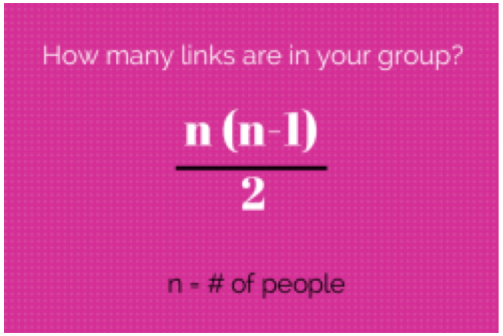
1. If a team wanted data or services from another team, they could no longer book their developers into a meeting. They had to query their API's.
2. Services are managed by small teams, which act like small independent companies focused on the customers of their service.

- "Each of these services require a strong focus on who their customers are, regardless whether they are external or internal" Amazon CTO Werner Vogels
- "We can scale our operation independently, maintain unparalleled system availability, and introduce new services quickly without the need for massive reconfiguration." Amazon CTO Werner Vogels

Complication: Communication is Expensive

Communication grows n^2 with team size

- One reason is that according to organizational psychologist Richard Hackman, the number of links between people grows rapidly with team size.
  - n(n-1)/2 where n is the number of people
- Coordination, communication and relating costs lower individual productivity as teams grow
- Ideal team size 6 to 8 people. Navy Seals identify 4 as the optimal combat team



Conway's Law

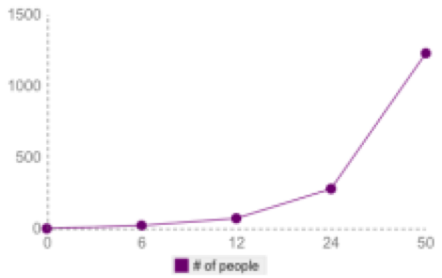
Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations

Complication: Communication is Expensive (Continued)

As groups size increases, the connections increase rapidly

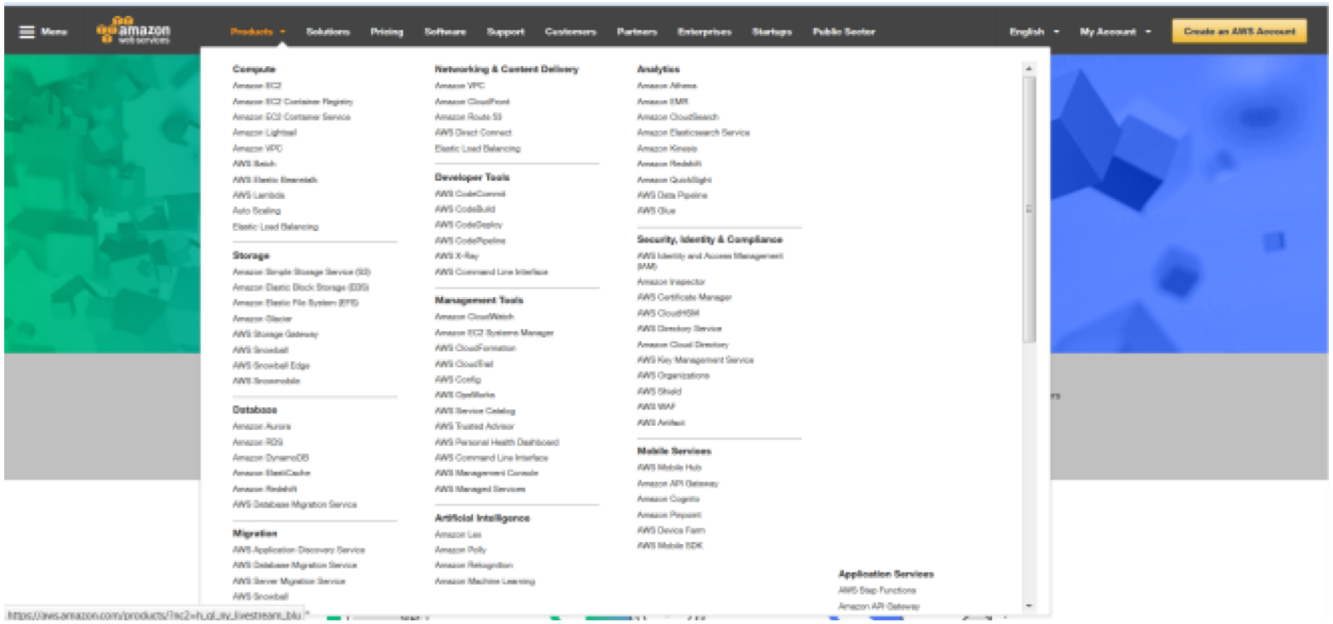
- **A team of 5 has 10 links**
- **A team of 10 has 45 links to maintain**
- **A small company of 60 people has 1770 links to manage**
- **Data Lake has 91 people which means 4095 links to manage**

William Kimball, Professor of Organizational Behavior, explains "The larger a group, the more process problems members encounter carrying out their collective work – social loafing, etc."



Amazon Web Services: Result

- Service Oriented Architecture can be observed in Amazon Web Services homepage
- Every service listed is an independent ‘2 Pizza Team’ that can plan and develop independently
- Additional teams can be readily added due to decentralized layout



Amazon Web Services: Result

1. By focusing on scalability and service independence, Amazon was able to greatly increase speed of development and scale their development
2. By having a ‘2-pizza rule’, or team sizes limited to those which can be fed by 2 pizzas (6-10 people), Jeff Bezos asserts that each team will be as productive as possible.

- This laid the foundation and DNA for Amazon Web Services, which has occupied > 80% of the market share for cloud hosting and disrupted nearly every incumbent enterprise provider.

Exercise 2.2 Kubernetes Connect

CLASSROOM WORK (15 minutes)

Kubernetes Simple Example

1. Start a deployment (pod will start automatically)
2. Expose deployment as service

https://github.com/techtown-training/microservices-bootcamp/blob/master/exercise/exercise2.2-k8s_simple_example.md

** Note – later we will discuss about pods, deployment, service in details. This exercise is just to start running containers, but a lot will be clear in later slides

GE Case

GE Oil and Gas: Situation

- GE Oil and Gas committed in 2013 to a wide ranging initiative with several goals
  - Improve productivity of shop floors
  - Build applications and solutions that reduce downtime and improve operations
  - Cut costs
  - Improve speed and agility of IT processes and Infrastructure

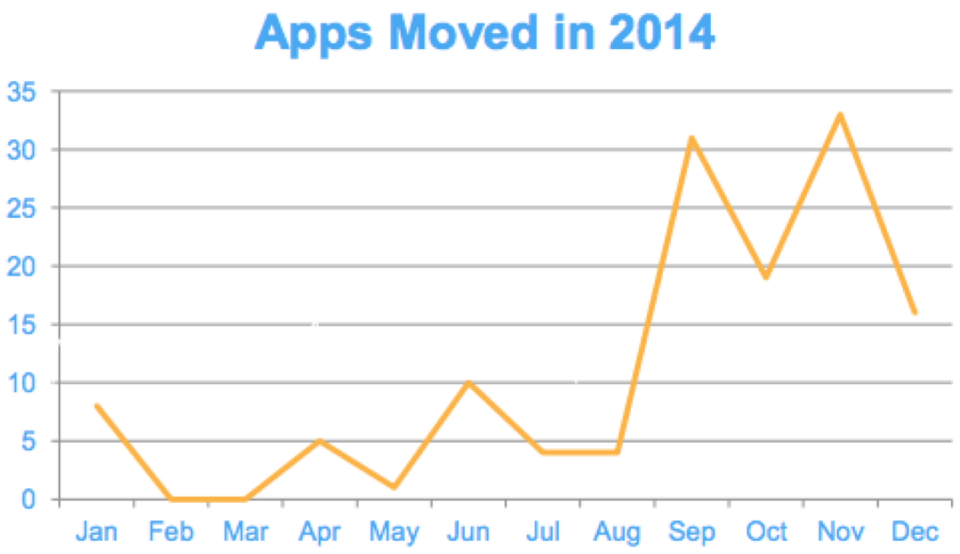
- 45,000 employees
- 11 different regions
- 7 research centers

- 85% of worlds offshore oil rigs use GE drilling systems
- $5B annual spend on energy research and development
- 900 applications (9000 applications in greater GE)
  - Many Apps have 100 or fewer users

GE Oil and Gas: Action

In order to reduce outages and improve automation, elected to more app portfolio to cloud

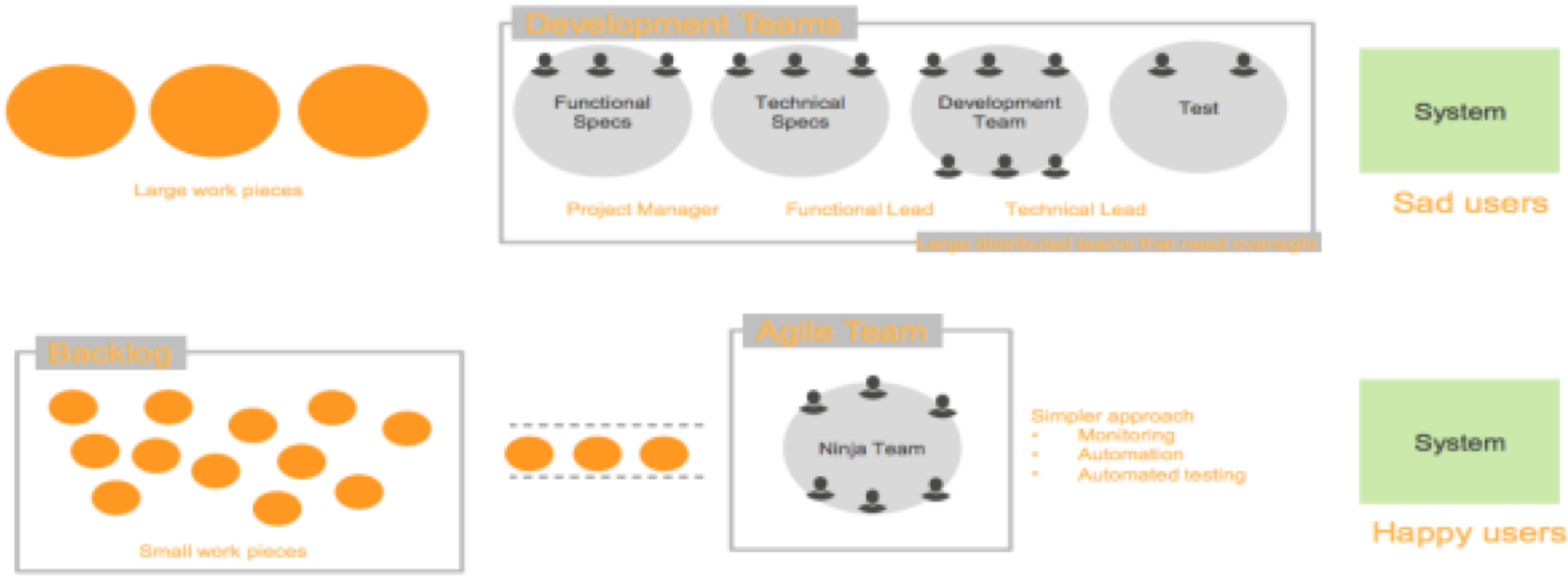
- **While doing so, encountered organizational challenges and had to do things differently in order to achieve their goal**
- **GE achieve massive improvements in productivity by changing fundamentally how they worked**



GE Oil and Gas: Action

GE achieved a massive increase in productivity by working differently

- **They broke down silos and united the team to a single cross functional ownership.**
- **They were able to do this by shrinking the work pieces into smaller deliverables**
- **Finally, the teams delivered more frequent artifacts to users, resulting in 'Happier' users**



GE Oil and Gas: Result

Adopting Devops best-practices and moving to cloud

- GE received a 52% reduction in TCO (total cost)
  - Bot-enabled automation
  - Self service
  - Dynamic storage
  - VM cost optimization
  - Migration from Oracle to Amazon Aurora
- Big increase in productivity
  - 50% reduction in tickets
  - 98% reduction in impactful business outages and incidents

Business Agility	Operational Resilience	Cost Avoidance	Workforce Productivity	Operational Costs
• 77% faster to deliver business applications	• 98% reduction in P1/P0's	• 52% average TCO savings	• 15 automated bots developed	• 35% reduction in compute assets (792)
• Rapid experimentation	• Improved security posture	• 80% cloud first adoption	• 8 cloud migration parties	• 50 applications decommissioned
• Reduced technical debt	• 15 cloud services created		• Shift to self-service culture	• $14M YOY Savings
• Streamlined M&A activity	• Improved performance		• DevOps in Practice	

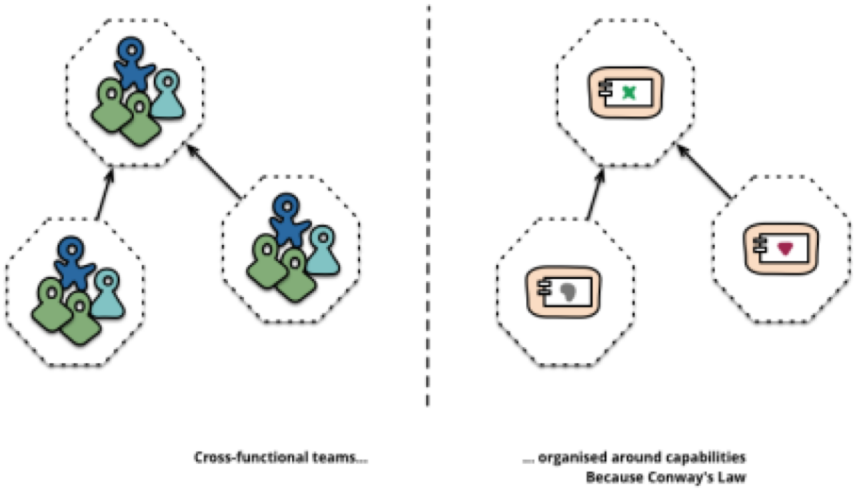
$14.2M Investment + 18 Months + Progress Focus = 311 Apps in Cloud & $14M YOY Savings

Conway's Law

"Organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organizations."

–Melvin Conway

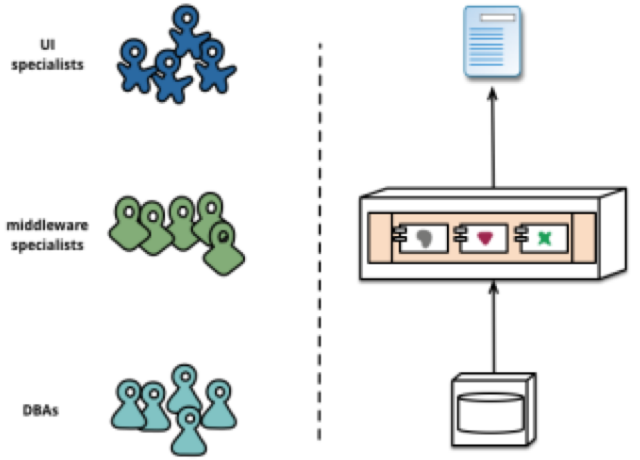
Do This



Cross-functional teams…

… organised around capabilities
Because Conway's Law

Not that



Siloed functional teams…

… lead to silod application architectures.
Because Conway's Law

Kubernetes Terminology - Pods

- The smallest unit that can be scheduled to be deployed through K8s is called a *pod*.
- Homogeneous group of containers.
- This group of containers would share storage, Linux namespaces, cgroups, IP addresses. These are co-located, hence share resources and are always scheduled together.
- Pods are not intended to live long. They are created, destroyed and re-created on demand, based on the state of the server and the service itself.

Kubernetes Terminology - Service

- As pods have a short lifetime, there is no guarantee about the IP address they are served on. This could make the communication of microservices hard.

- Hence K8s has introduced the concept of a *service*, which is an abstraction on top of a number of pods, typically requiring to run a proxy on top, for other services to communicate with it via a Virtual IP address.

- This is where you can configure load balancing for your numerous pods and expose them via a service.

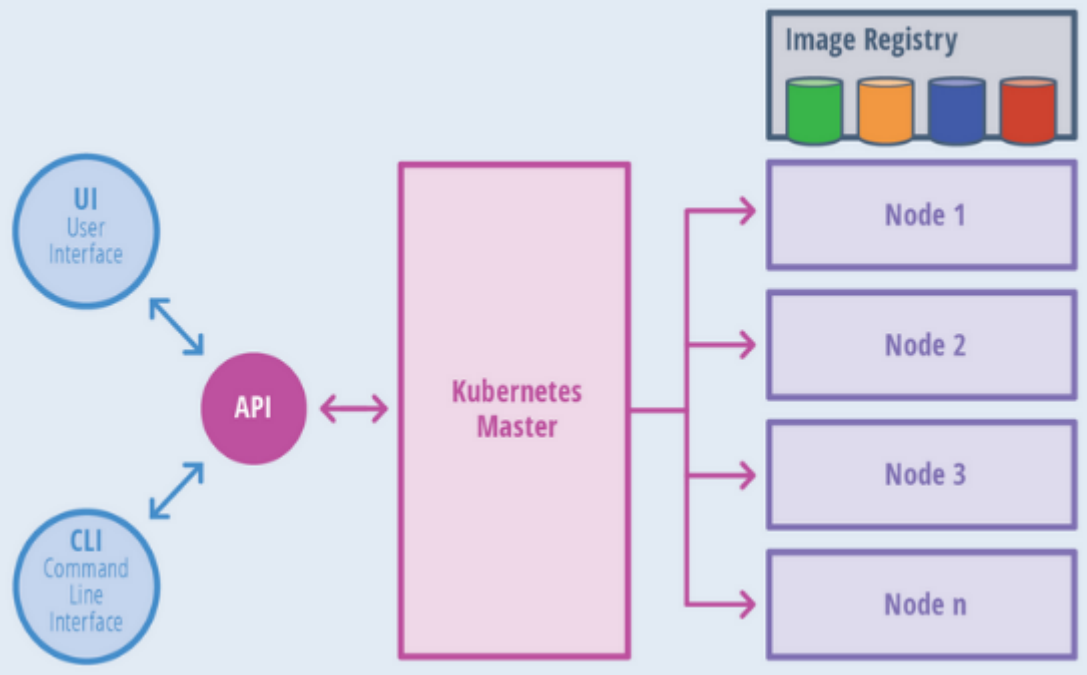
Kubernetes Terminology – Master Node

- The master node is responsible for the management of Kubernetes cluster. This is the entry point of all administrative tasks. The master node is the one taking care of orchestrating the worker nodes, where the actual services are running.
  - **API Server** - The API server is the entry points for all the REST commands used to control the cluster.
  - **Etcd Storage** - is a simple, distributed, consistent key-value store. It's mainly used for shared configuration and service discovery. Example of data stored by Kubernetes in etcd is jobs being scheduled, created and deployed, pod/service details and state, namespaces and replication information, etc.
  - **Scheduler** - The scheduler has the information regarding resources available on the members of the cluster, as well as the ones required for the configured service to run and hence is able to decide where to deploy a specific service.
  - **Controller-Manager** - A controller uses apiserver to watch the shared state of the cluster and makes corrective changes to the current state to change it to the desired one. An example is the **Replication-controller** which takes care of the number of pods in the system.

Kubernetes Terminology - Others

- **Nodes**
  - **Hosts that run Kubernetes applications**
- **Containers**
  - **Units of packaging**
- **Labels**
  - **Key-Value pairs for identification**
- **Kubelet - gets the configuration of a pod from the apiserver and ensures that the described containers are up and running.**
- **Kube-proxy - acts as a network proxy and a load balancer for a service on a single worker node.**

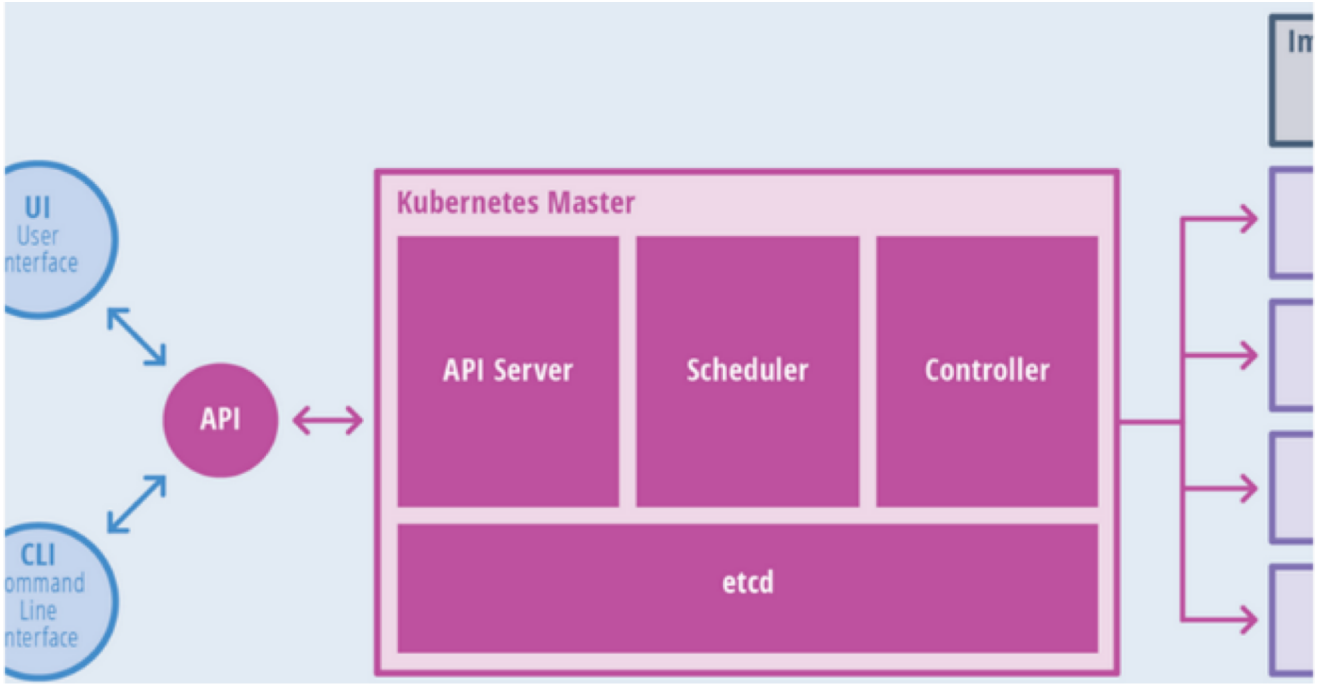
General Architecture - Kubernetes



The **master** is responsible for
Exposing the API interface
Scheduling the deployments
Managing the cluster

Each **node** runs a container runtime, such as Docker, along with an agent that communicates with the master.
Nodes run additional components for logging, monitoring, service discovery and optional add-ons
Nodes expose compute, networking and storage resources to applications

Kubernetes Master



Components of master:

- **API Server**
- **Scheduler**
- **Controller**
- **Etcd**

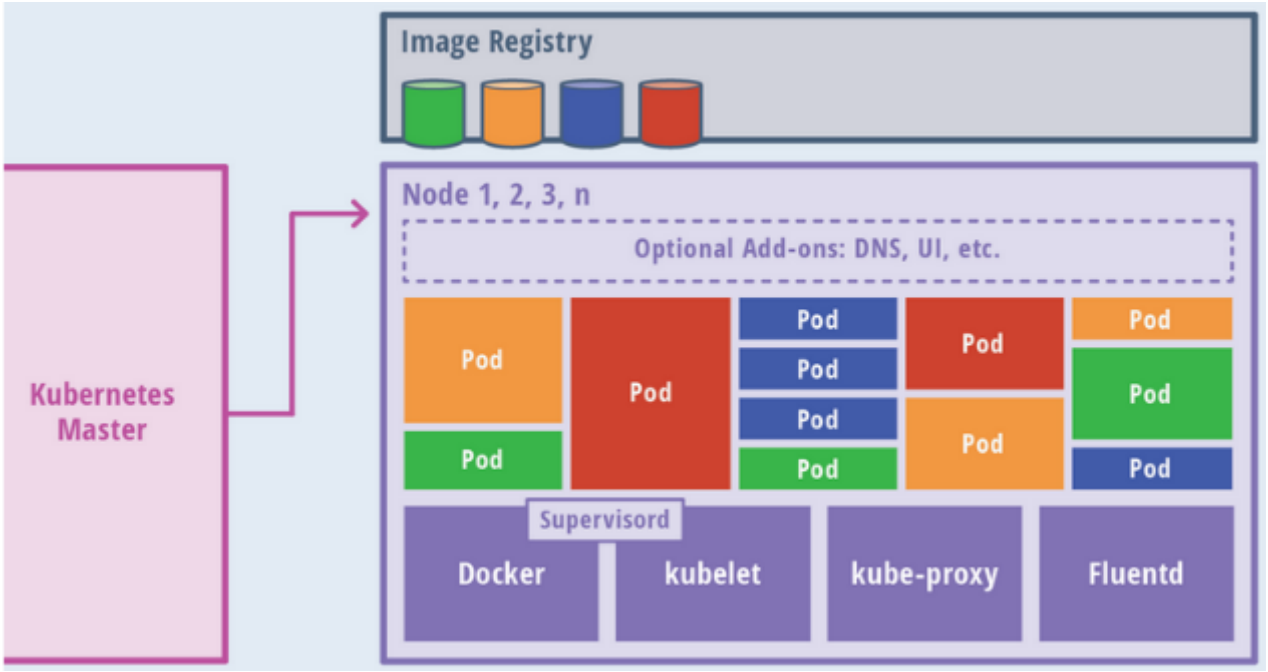
Kubernetes Master

- **The API server** is the entry points for all the REST commands used to control the cluster. It processes REST requests, validates them, and executes the bound business logic. The result state has to be persisted in the “etcd” component.
- **Etcd** is an open source, distributed key-value database; it acts as a single source of truth (SSOT) for all components of the Kubernetes cluster. Masters query etcd to retrieve various parameters of the state of the nodes, pods and containers. Etcd is considered a metadata service in Kubernetes.

Kubernetes Master (Continued)

- **Controller Manager** is responsible for most of the collectors that regulate the state of the cluster. In general, a controller can be considered a daemon that runs in nonterminating loop and is responsible for collecting and sending information to the API server. It works toward getting the shared state of cluster and then making changes to bring the current status of the server to the desired state. The key controllers are replication controller, endpoint controller, namespace controller, and service account controller. The controller manager runs different kind of controllers to handle nodes, endpoints, etc.
- **Scheduler** is one of the key components of Kubernetes master. It is responsible for distributing the workload, tracking resource utilization on cluster nodes and selecting the nodes for the workloads to run. In other words, this is the mechanism responsible for allocating pods to available nodes.

Kubernetes Nodes



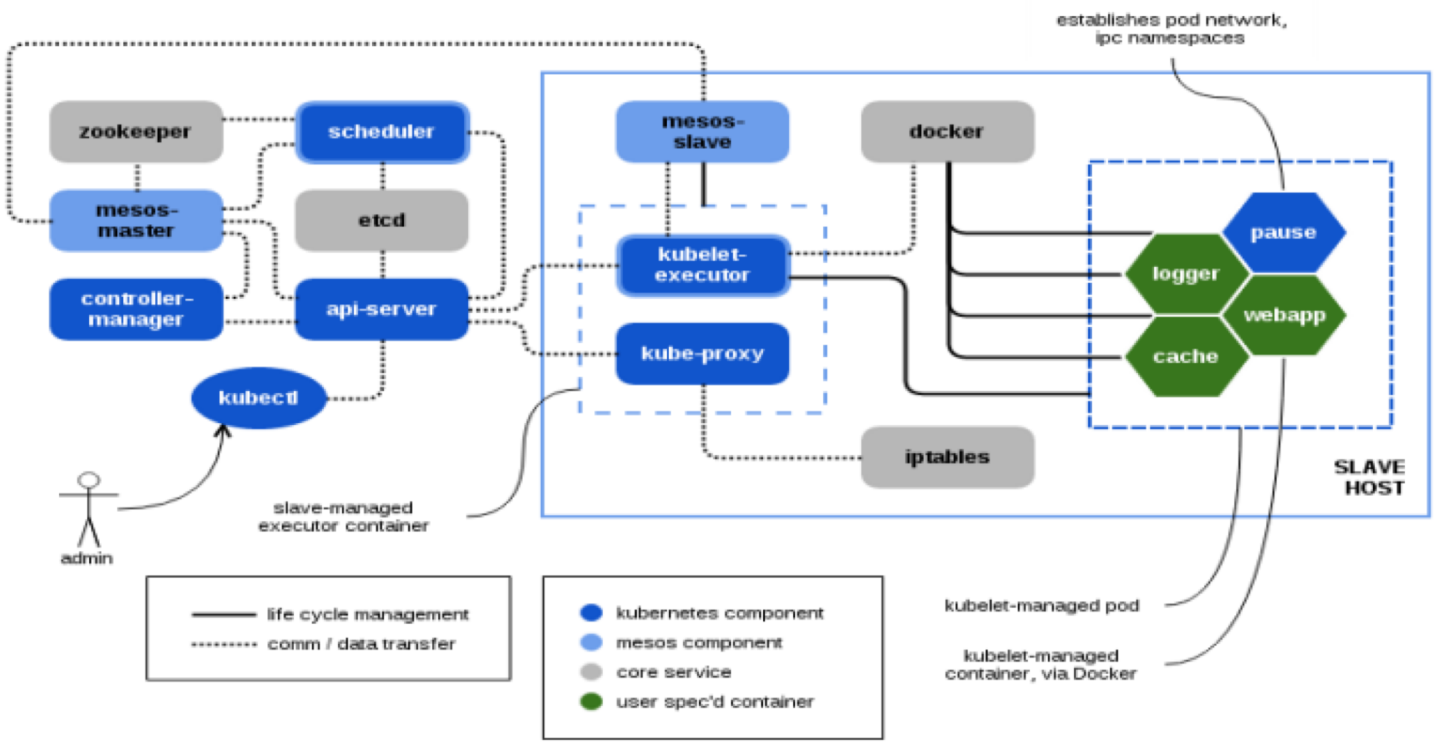
Components of a node:

- **Kubelet**
- **Kube-proxy**
- **Docker**
- **Fluentd**

Kubernetes Node Components

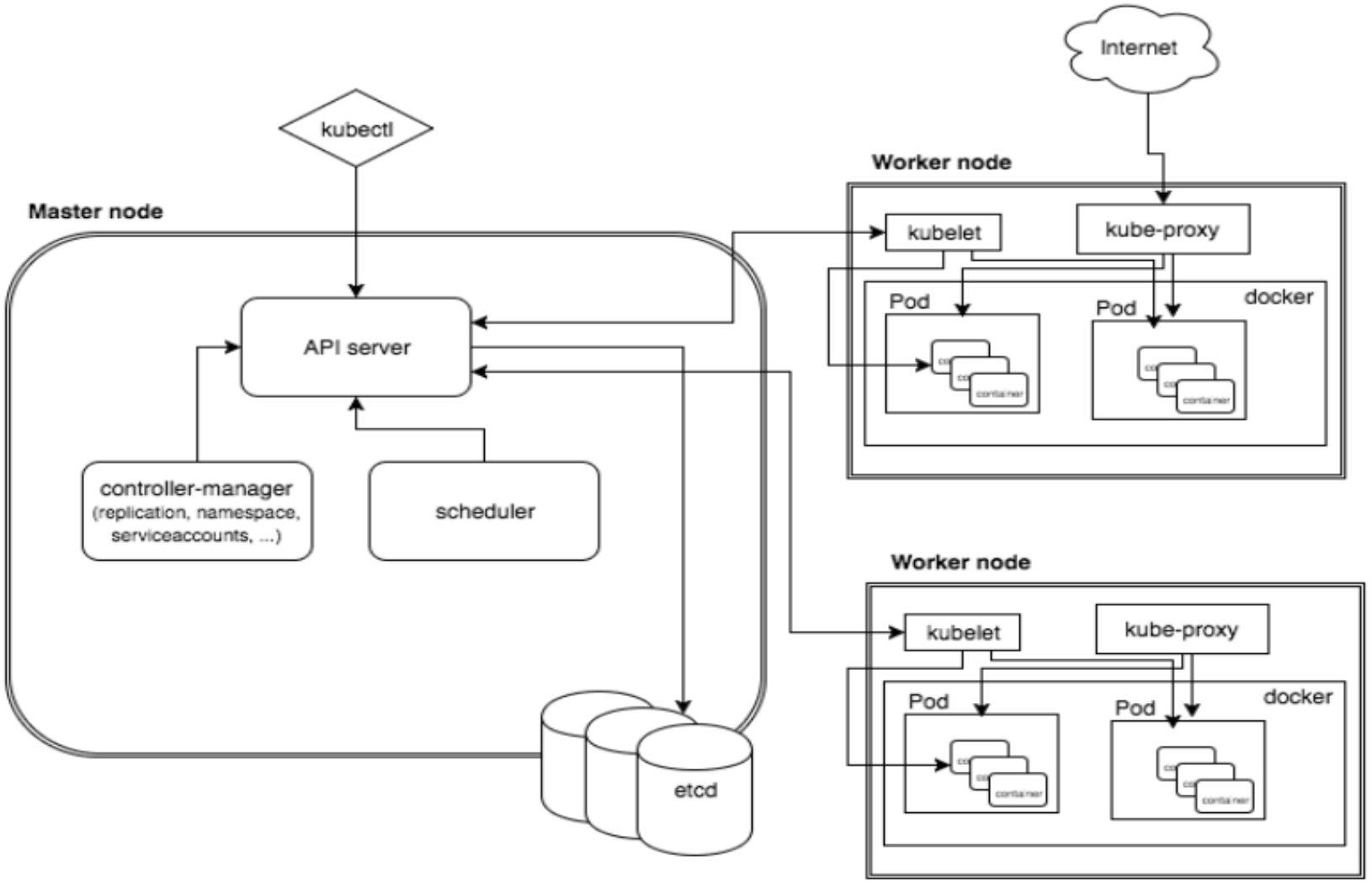
- **Kubelet Service** on each node is responsible for relaying information to and from the control plane service. It interacts with etcd store to read configuration details and to write values. It communicates with the master component to receive commands and work. The kubelet process then assumes responsibility for maintaining the state of work and the node server. It manages network rules, port forwarding, etc.
- **Kubernetes Proxy Service** is a proxy service which runs on each node and helps in making services available to the external host. It helps in forwarding the request to correct containers and is capable of performing primitive load balancing. It makes sure that the networking environment is predictable and accessible and at the same time it is isolated as well. It manages pods on node, volumes, secrets, creating new containers' health checkup, etc.

DC/OS Integration with Kubernetes



Mesosphere partnered with Google to leverage the extensibility of Kubernetes and build Kubernetes-Mesos. The open source project that integrates with the Kubernetes scheduling API and the Mesos scheduler runtime, as well as provides an executor component that integrates kubelet services and the Mesos executor runtime.

K8s Architecture - Detailed



Exercise 2.3 Kubernetes Examples

CLASSROOM WORK

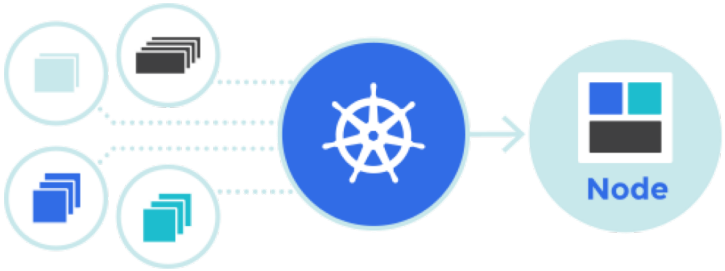
Kubernetes 101 **(Part 1 – run only the below section)**

1. Running nginx
2. Pod IPs
3. Pod containers

https://github.com/techtown-training/microservices-bootcamp/blob/master/exercise/exercise2.3-k8s_simple_example.md

Kubernetes (K8's for short)

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications



Roles and Advantages of using K8s:

- Mounting storage systems
- Distributing secrets
- Checking application health
- Replicating application instances
- Using Horizontal Pod Autoscaling
- Naming and discovering
- Balancing loads
- Rolling updates
- Monitoring resources
- Accessing and ingesting logs
- Debugging applications
- Providing authentication and authorization

Kubernetes: Clusters

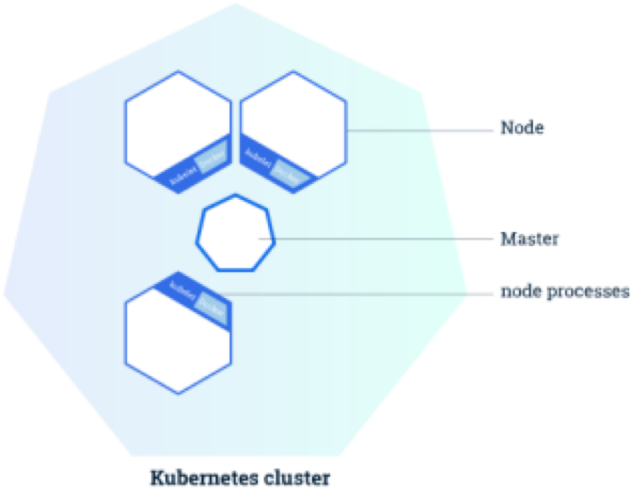
Allows you to deploy containerized applications to a cluster without tying them specifically to individual machines.

Masters

- Coordinates the cluster
  - **Schedules applications**
  - **Maintains application state**
  - **Scales applications**
  - **Rolls out updates**
- Can be duplicated when HA (High Availability) is desired.
- Often run as a service by public cloud providers (Azure/Google Kubernetes as a Service)

Nodes

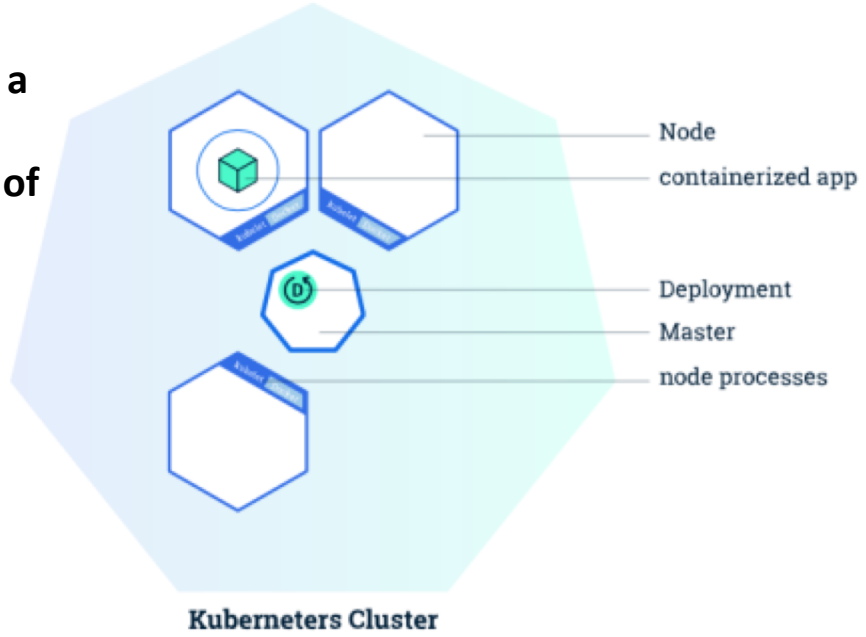
- House workers that run the applications
- **Three node minimum for production clusters**



Kubernetes: Deployments

Allows you to deploy a (self-healing) instance of an application.

- **Self Healing: Continuously monitors and replaces instances if necessary**
- **Provides declarative updates for Pods and ReplicaSets**
  - **Updates actual state to desired state at a controlled rate**
  - **For example: Current state, 3 instances of Tomcat. Desired state, 5 instances of Tomcat. -> Create 2 more instances of Tomcat**
- **Relay on declarative manifest to determine intent (i.e. yaml/json file most commonly)**



$ kubectl create

```
peter@Azure:~$ cat pod.yaml
# Copy of pod.yaml without file extension for test
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
peter@Azure:~$ kubectl create -f pod.yaml
pod "nginx" created
peter@Azure:~$ kubectl get pods
NAME      READY     STATUS    RESTARTS   AGE
nginx     1/1       Running   0          17s
```

Note: This is just creation of the pod. In order to expose it, a service must be created. In order for it to be respawned, a deployment or replication control must be created.

https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#creating-a-deployment

Kubernetes: Pods

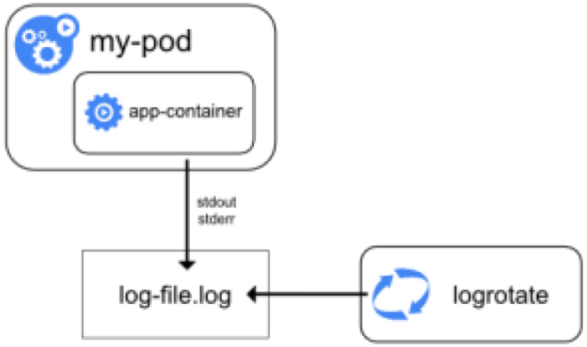
Allows you to create a group of one or more containers and some shared resources for those containers

- **Pod Resource Types**
  - **Networking, as a unique cluster IP address**
  - **Container information:**
    - Container image version
    - Specific ports
    - Etc.
- **Atomic unit of Kubernetes**
- **Tied to node which is it scheduled**
- **Always co-located and co-scheduled**
- **Containers in a pod share**
  - **IP address and port space**
  - **Filesystem**
  - **Storage (Volumnes)**
  - **Labels**
  - **Secrets**

$ kubectl get pods

```
peter@azure:~$ kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
azure-vote-back-3048739398-dtklm    1/1       Running   0          9m
azure-vote-front-837696400-j9jh5    1/1       Running   0          9m
peter@azure:~$ kubectl logs azure-vote-back-3048739398-dtklm
1:C 06 Sep 12:24:58.141 # oooOoooOoooo Redis is starting oooOoooOoooo
1:C 06 Sep 12:24:58.141 # Redis version=4.0.1, bits=64, commit=00000000, modified=0, pid=1, just started
1:C 06 Sep 12:24:58.141 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
1:M 06 Sep 12:24:58.143 * Running mode=standalone, port=6379.
1:M 06 Sep 12:24:58.143 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
1:M 06 Sep 12:24:58.143 # Server initialized
1:M 06 Sep 12:24:58.143 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent
_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled.
1:M 06 Sep 12:24:58.143 * Ready to accept connections
```

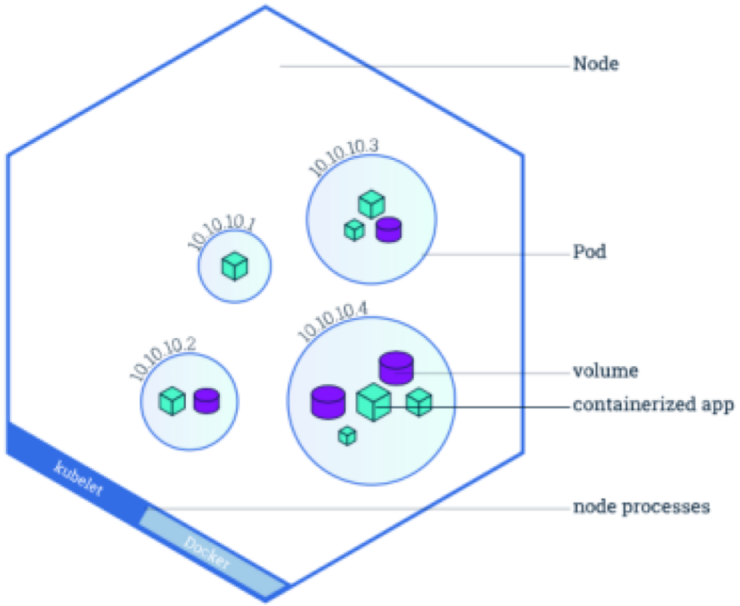
Note: Logs from pods are readily available with a **kubectl logs <podname>** command.



Kubernetes: Nodes

Allows Pods to be scheduled. A basic worker physical or virtual machine of Kubernetes.

- **Must be managed by a master**
- **May host multiple pods**
- **Internal IP Address endpoint**
- **Can be tagged and filtered using labels**
- **Must be running two processes**
  - **Kubelet – Kubernetes client process. Communicates with master and schedules pods**
  - **Container runtime – Docker, Rocket, etc. Pulls and runs containers from registry**



$ kubectl get nodes

```
peter@Azure:~$ kubectl get nodes
NAME                    STATUS                     AGE       VERSION
k8s-agent-743d5653-0    Ready                      21h       v1.6.6
k8s-agent-743d5653-1    Ready                      21h       v1.6.6
k8s-agent-743d5653-2    Ready                      21h       v1.6.6
k8s-master-743d5653-0   Ready,SchedulingDisabled   21h       v1.6.6
```

Kubernetes Command Line Tool: kubectl

Most common access point for launching workloads on Kubernetes clusters.

- **kubectl [command] [TYPE] [NAME] [flags]**
- Very popular commands
  - **kubectl get** - list resources about a target.
    - kubectl get services
    - kubectl get pods
  - **kubectl describe** - show basic information about a resource
    - Kubectl describe pods my-pod
  - **kubectl logs** - print the logs from a container in a pod. Extremely useful
    - Kubectl logs my-pod
- Also useful:
  - **kubectl exec** - execute a command on a container in a pod
  - **Kubectl top** – Show metrics for a node

Kubernetes Connect

CLASSROOM WORK

Kubernetes 101 **(Part 2 – Run only below section)**

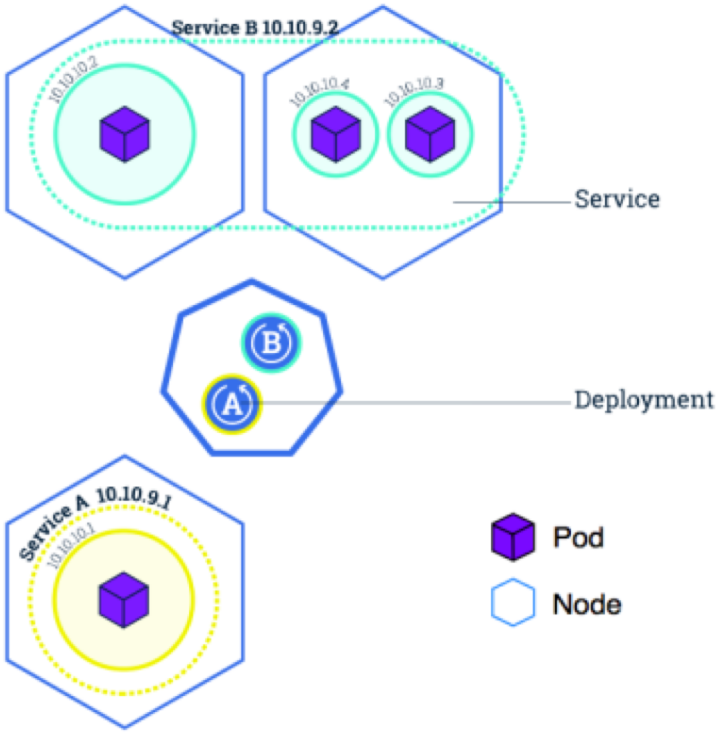
1. Deployments and Replica sets
2. Services
3. Back to Deployments

https://github.com/techtown-training/microservices-bootcamp/blob/master/exercise/exercise2.3-k8s_simple_example.md

Kubernetes: Service

Abstraction regarding a set of pods which enables load balancing, traffic exposure, load balancing and service discovery.

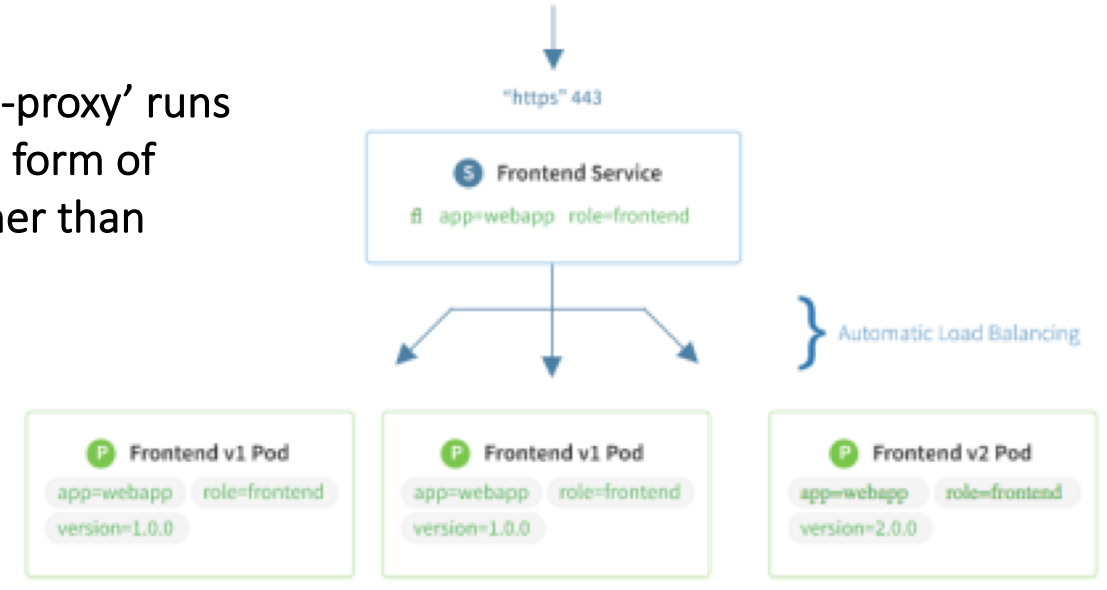
- **Abstraction that allows pods to die and replicate in Kubernetes without affecting your application.**
- **Enable loose coupling between different Pods.**
- **Provides a stable virtual IP and port**
- **Services allow Pods to receive traffic.**
  - **Each Pod has a unique IP but those IP addresses are not exposed outside the Pod without a service.**



$ kubectl get services

```
peter@Azure:~$ kubectl get services --all-namespaces
NAMESPACE     NAME                   CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
default       kubernetes             10.0.0.1       <none>        443/TCP          22h
kube-system   heapster               10.0.74.107    <none>        80/TCP           22h
kube-system   kube-dns               10.0.0.10      <none>        53/UDP,53/TCP    22h
kube-system   kubernetes-dashboard   10.0.69.57     <nodes>       80:31476/TCP     22h
kube-system   tiller-deploy          10.0.103.65    <none>        44134/TCP        22h
```

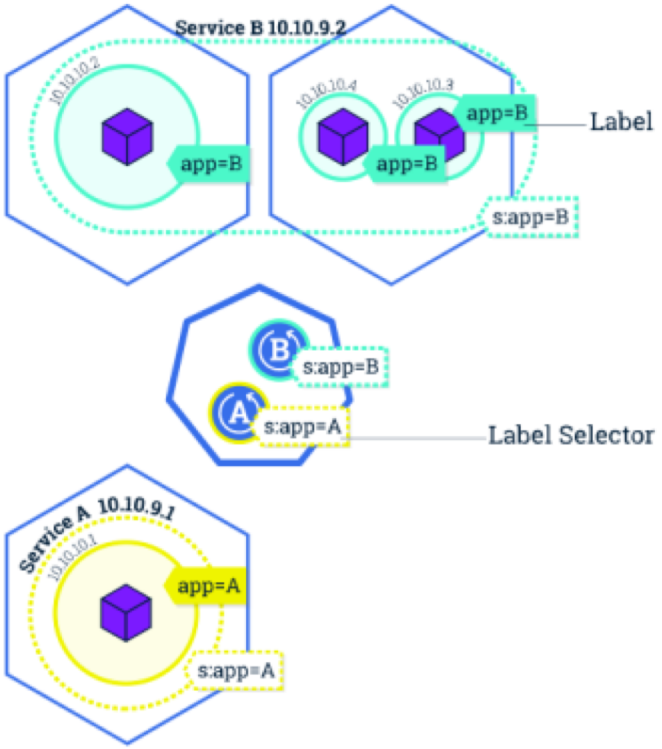
Note: A component called ‘kube-proxy’ runs on each node and implements a form of virtual IP for services of type other than ‘ExternalName’



Kubernetes: Label

Key/Value pairs that can be attached to objects to enable a variety of use cases.

- **Designate objects for specific environments such as development, test and production**
- **Set versions to objects**
- **Set roles or other arbitrary information to classify objects**
- **Can be added or modified at any time**



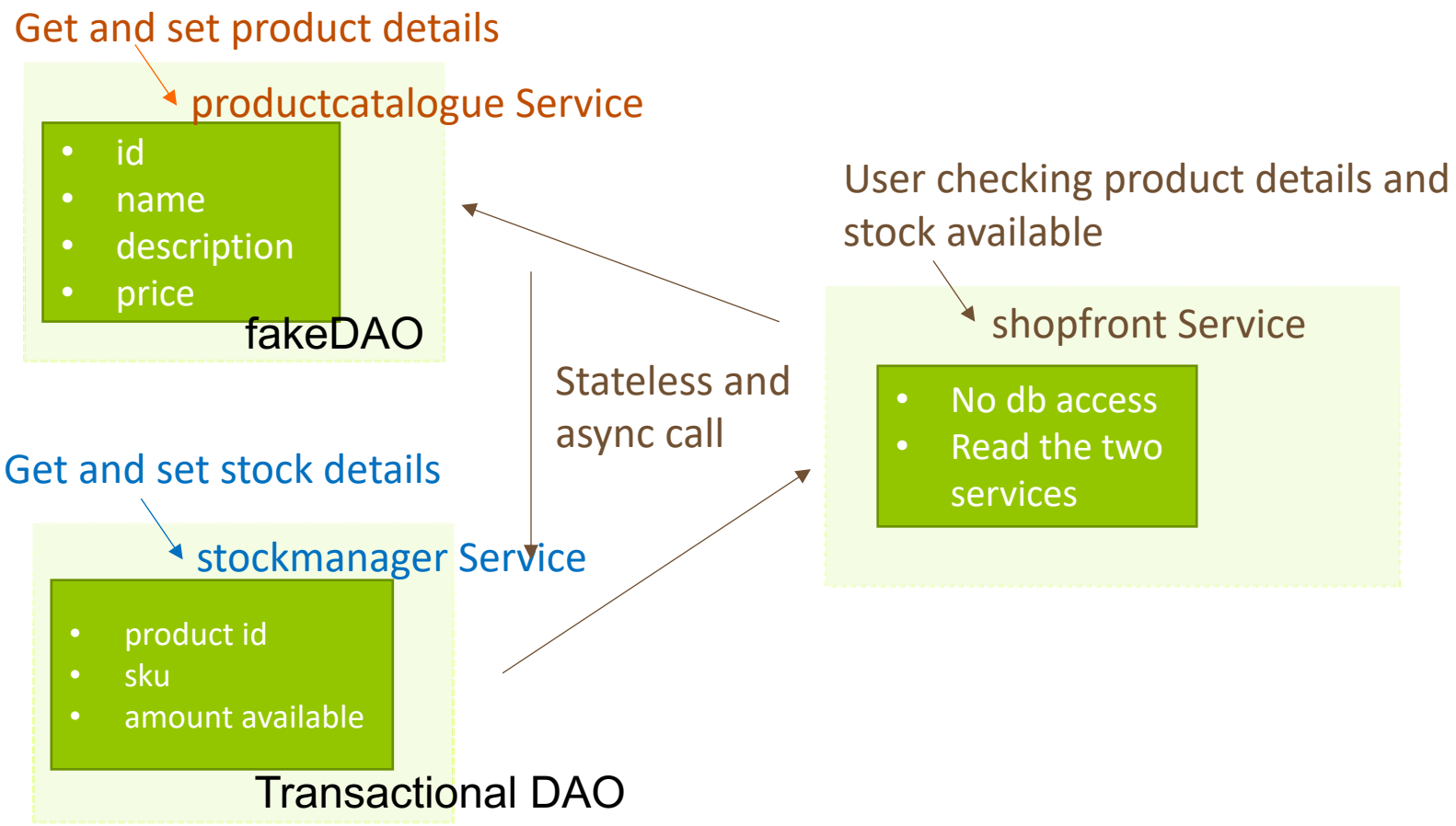
$ kubectl get pod –l name=<label>

```
peter@Azure:~$ kubectl run nginx --image nginx --port 80
deployment "nginx" created
peter@Azure:~$ kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
nginx-158599303-04h8d    1/1       Running   0          4s
peter@Azure:~$ kubectl label pods nginx-158599303-04h8d new-label=awesome
pod "nginx-158599303-04h8d" labeled
peter@Azure:~$ kubectl get pods -l new-label=else
No resources found.
peter@Azure:~$ kubectl get pods -l new-label=awesome
NAME                     READY     STATUS    RESTARTS   AGE
nginx-158599303-04h8d    1/1       Running   0          3m
peter@Azure:~$ kubectl get pods --show-labels
NAME                     READY     STATUS    RESTARTS   AGE       LABELS
nginx-158599303-04h8d    1/1       Running   0          3m        new-label=awesome,pod-template-hash=158599303,run=nginx
```

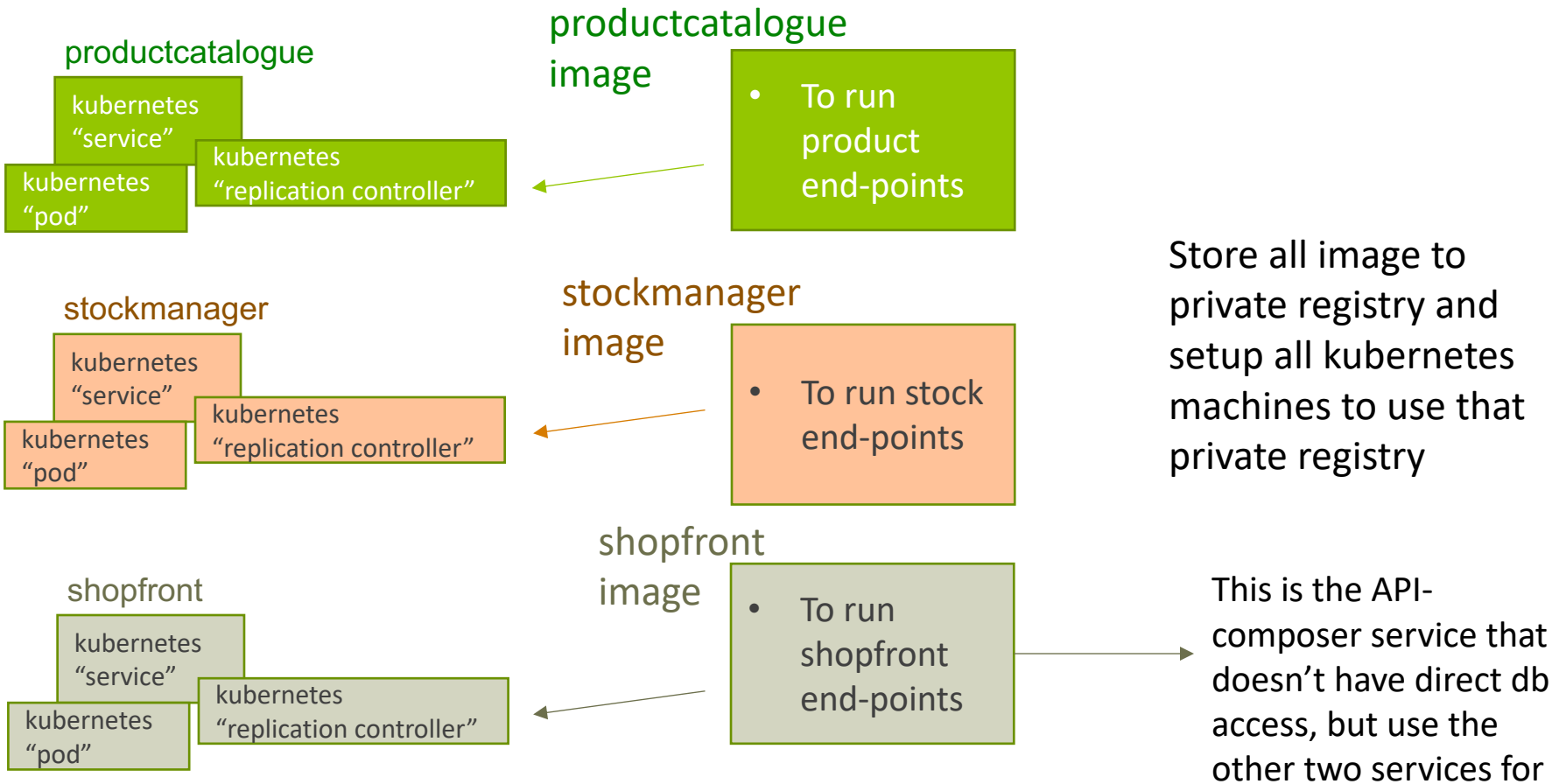
Note: Best mechanism to organize Kubernetes objects

- **Let’s break the monolithic application from Day 1 into Microservices**

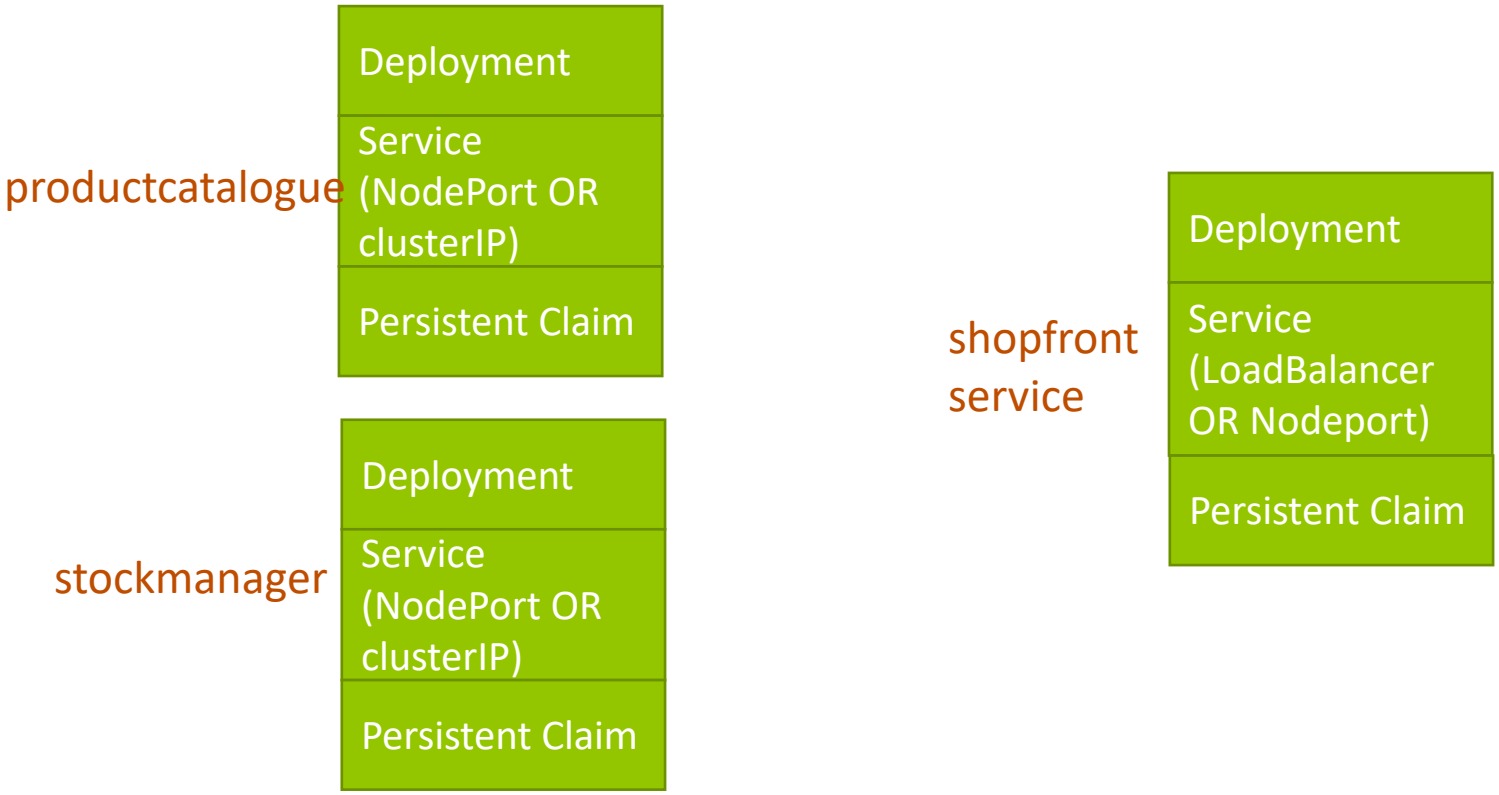
Microservice App in Java
(Taken from Day 1)



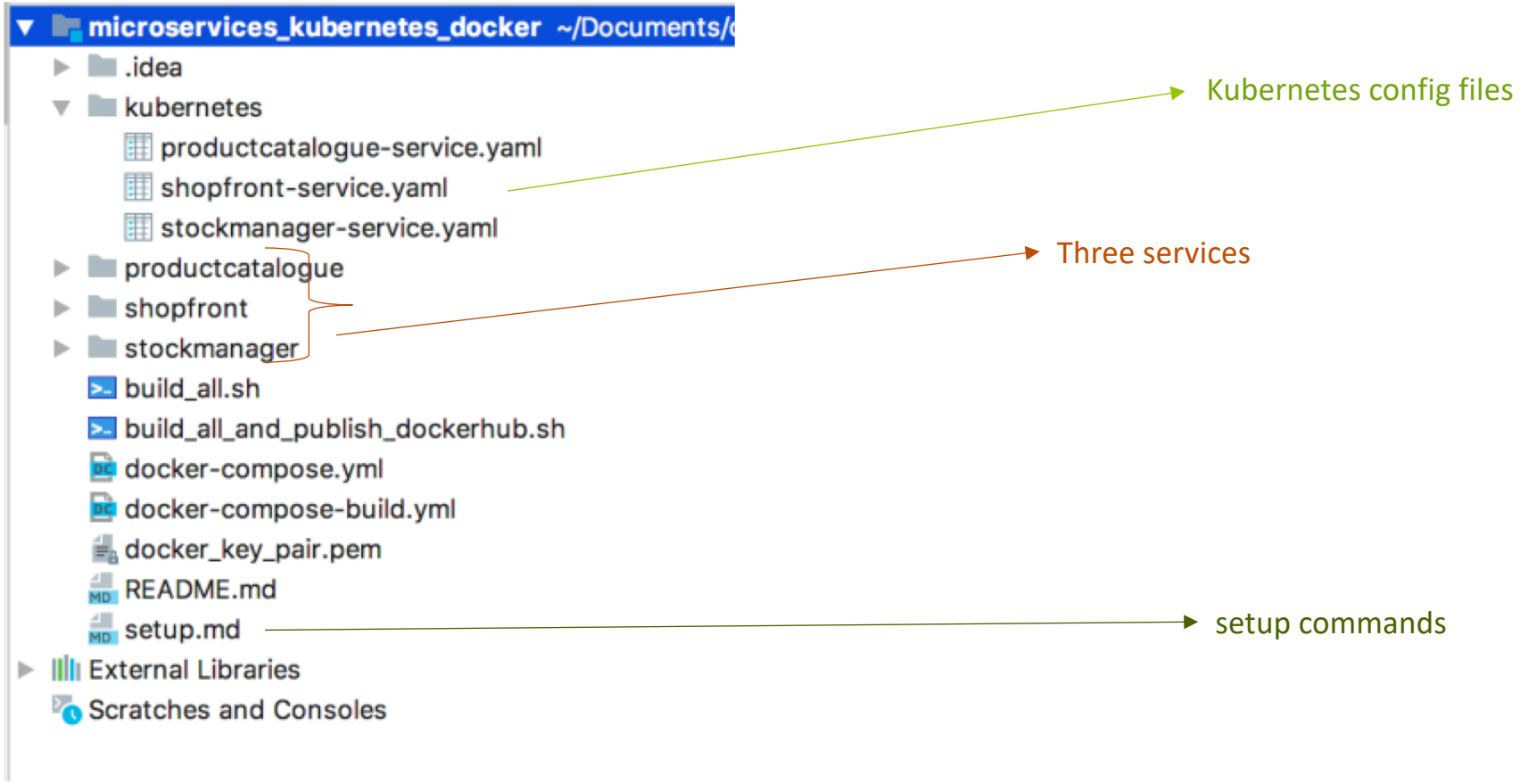
Docker Image & Kubernetes Requirements
(To run the App)



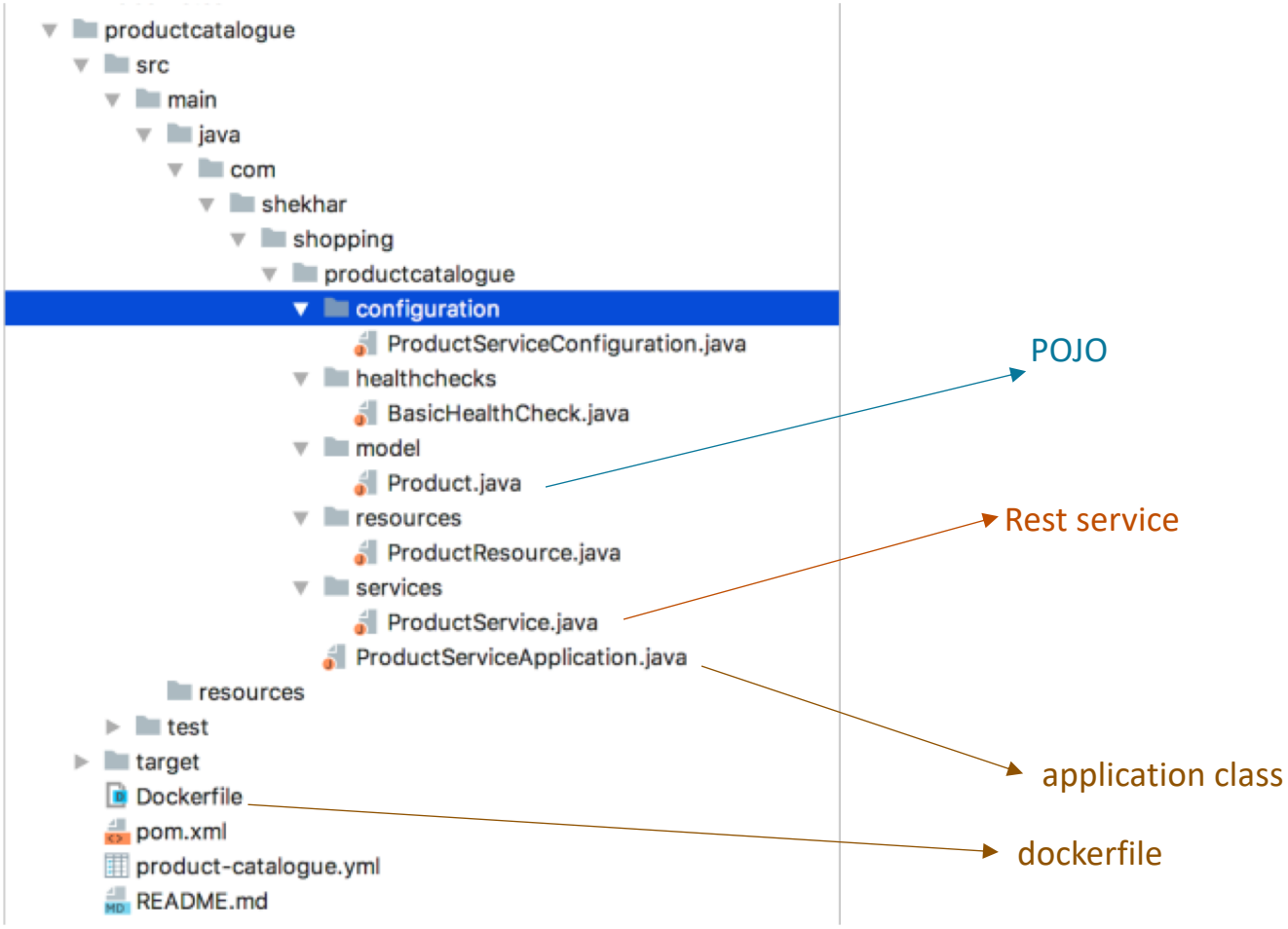
Kubernetes Resources
(To run the App)



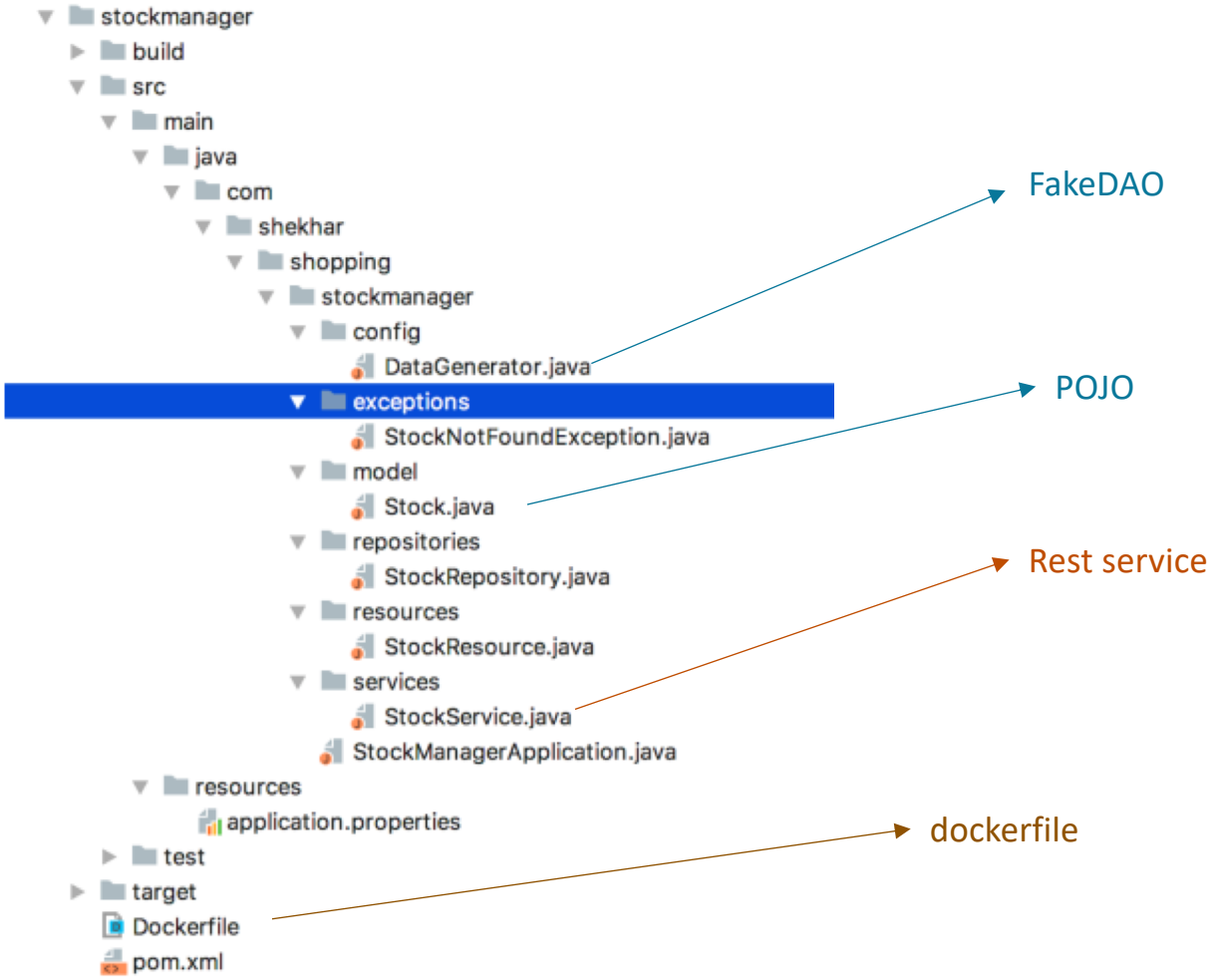
Project Structure – Project and Kubernetes



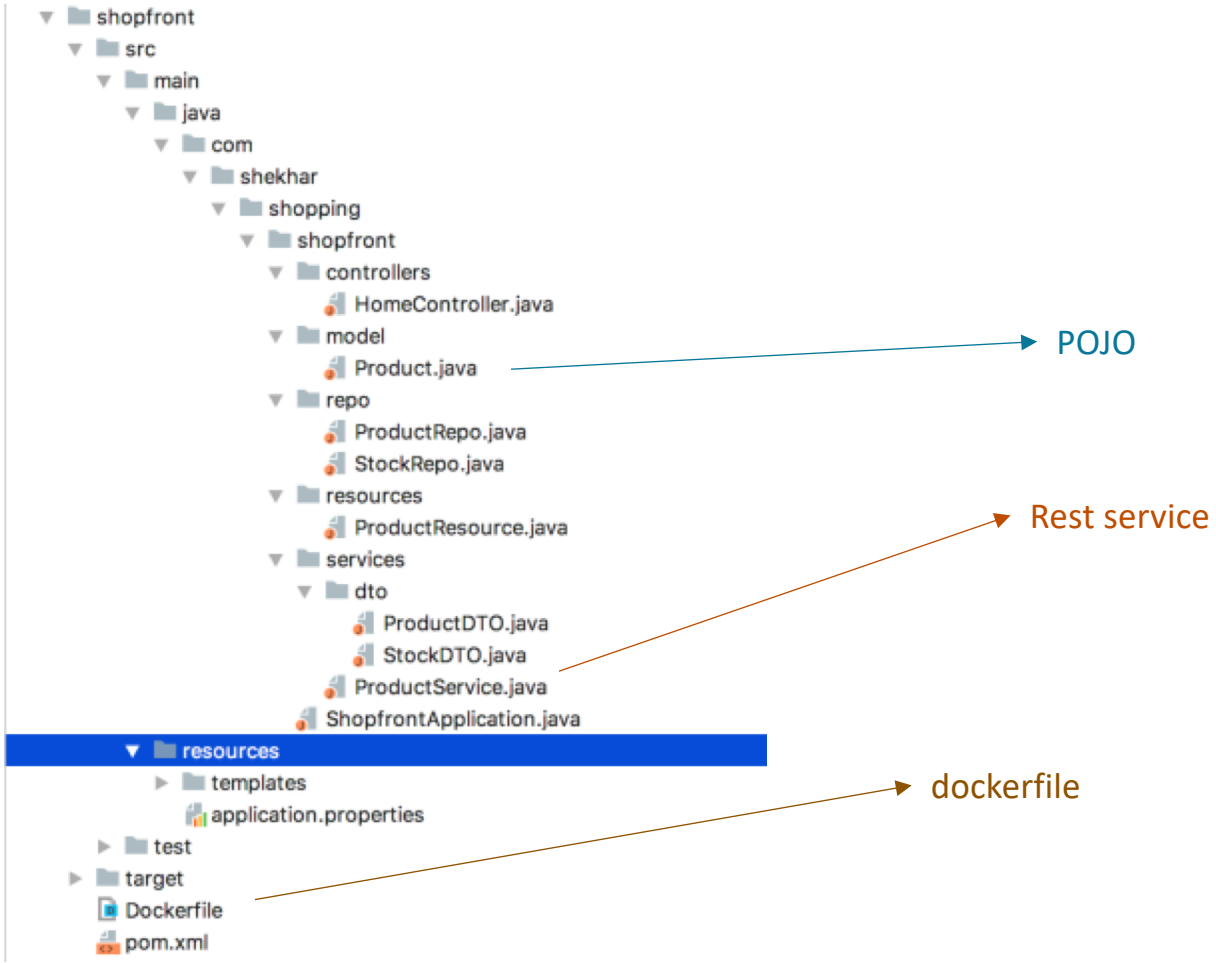
Project Structure – productcatalogue Service



Project Structure – stockmanager Service



Project Structure – shopfront Service



Build Project and Docker Image - productcatalogue

```
# Build the project

cd productcatalogue
mvn clean install -U
```

```
# docker build -t shekhar/productcatalogue:1.0 .

FROM openjdk:8-jre
ADD target/productcatalogue-1.0.jar app.jar
ADD product-catalogue.yml app-config.yml
EXPOSE 8020
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","app.jar", "server",
"app-config.yml"]
```

Build Project and Docker Image - stockmanager

```
# Build the project

cd stockmanager
mvn clean install -U
```

```
# docker build -t shekhar/stockmanager:1.0 .

FROM openjdk:8-jre
ADD target/stockmanager-1.0.jar app.jar
EXPOSE 8030
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

Build Project and Docker Image - shopfront

```
# Build the project

cd shopfront
mvn clean install -U
```

```
# docker build -t shekhar/shopfront:1.0 .

FROM openjdk:8-jre
ADD target/shopfront-1.0.jar app.jar
EXPOSE 8010
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

Kubernetes Entities
(productcatalogue service and replicationcontroller)

```
apiVersion: v1
kind: Service
metadata:
  name: productcatalogue
  labels:
    app: productcatalogue
spec:
  type: NodePort
  selector:
    app: productcatalogue
  ports:
  - protocol: TCP
    port: 8020
    name: http
```

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: productcatalogue
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: productcatalogue
    spec:
      containers:
      - name: productcatalogue
        image:
shekhar/productcatalogue:1.0
        ports:
        - containerPort: 8020
        livenessProbe:
          httpGet:
            path: /healthcheck
            port: 8025
          initialDelaySeconds: 30
          timeoutSeconds: 1
```

Kubernetes Entities

- stockmanager service and replicationcontroller

```yaml
apiVersion: v1
kind: Service
metadata:
  name: stockmanager
  labels:
    app: stockmanager
spec:
  type: NodePort
  selector:
    app: stockmanager
  ports:
  - protocol: TCP
    port: 8030
    name: http
```

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: stockmanager
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: stockmanager
    spec:
      containers:
      - name: stockmanager
        image: shekhar/stockmanager:1.0
        ports:
        - containerPort: 8030
        livenessProbe:
          httpGet:
            path: /health
            port: 8030
          initialDelaySeconds: 30
          timeoutSeconds: 1
```

Kubernetes Entities
- shopfront service and replicationcontroller

```yaml
apiVersion: v1
kind: Service
metadata:
  name: shopfront
  labels:
    app: shopfront
spec:
  type: NodePort
  selector:
    app: shopfront
  ports:
  - protocol: TCP
    port: 8010
    name: http
```

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: shopfront
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: shopfront
    spec:
      containers:
      - name: shopfront
        image: shekhar/shopfront:1.0
        ports:
        - containerPort: 8010
        livenessProbe:
          httpGet:
            path: /health
            port: 8010
          initialDelaySeconds: 30
          timeoutSeconds: 1
```

Setup – Running Java Microservices

- Break monolith to multiple services
- Build individual service
- Create required docker images
- Push all images to private registry
- Create kubernetes deployment, pod, services, rc, and persistent claim
- Start all kubernetes entities

https://github.com/techtown-training/microservices-bootcamp/blob/master/exercise/exercise2.4-microservices_java_project.md

Exercise 2.4 Converting Monolithic to Microservices - Java

CLASSROOM WORK 103 (75 minutes)

1. Break the existing monolith into logical multiple piece
2. Build docker image for individual services
3. Select the Microservice Pattern (here, we used Api-Composer)
4. Create necessary pod, deployment, service, persistent claim and volume in Kubernetes to deploy all services

Exercise steps:

https://github.com/techtown-training/microservices-bootcamp/blob/master/exercise/exercise2.4-microservices_java_project.md

Source code:

https://github.com/techtown-training/microservices-bootcamp/tree/master/exercise/src_code/microservices_kubernetes_docker

Successfully converted a Monolith to Microservices and deployed in Kubernetes.

Additional practice exercise

Exercise – 2.5 (RC, RS, Deploy)
20 minutes

- Exercise to create Replication Controllers, Replica Sets and Deployments and describe their functions
- Check the results and describe the objects

- sudo su –
- You already have the cloned the github project [https://github.com/techtown-training/microservices-bootcamp.git]
- Navigate to 'exercise/src_code/kubernetes_additional_exercise/ex2.5/' – for all config files required for this exercise
- Exercise instruction:-

https://github.com/techtown-training/microservices-bootcamp/blob/master/exercise/exercise2.5-rs_rc_deploy.md

Exercise – 2.6 (Pods, Deploy, Services) 15 minutes

- For all the concepts discussed until now
- Create pods, Services, Deployments
- Various ways to use the create command

We already have the git repo cloned in our machines.

Navigate to ‘exercise/src_code/kubernetes_additional_exercise/ex2.6/’ – for all config files required for this exercise

Exercise instruction:-

https://github.com/techtown-training/microservices-bootcamp/blob/master/exercise/exercise2.6-pod_deploy_service.md

Exercise – 2.7 (Kubernetes Labels)
15 minutes

- Create a series of pods with different labels
- Query the pods based on various labels
- Delete pods based on certain selectors

We already have the git repo cloned in our machines.

- Navigate to ‘exercise/src_code/kubernetes_additional_exercise/ex2.7/’ – for all config files required for this exercise
- Exercise instruction:-

https://github.com/techtown-training/microservices-bootcamp/blob/master/exercise/exercise2.7-label_usage.md

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

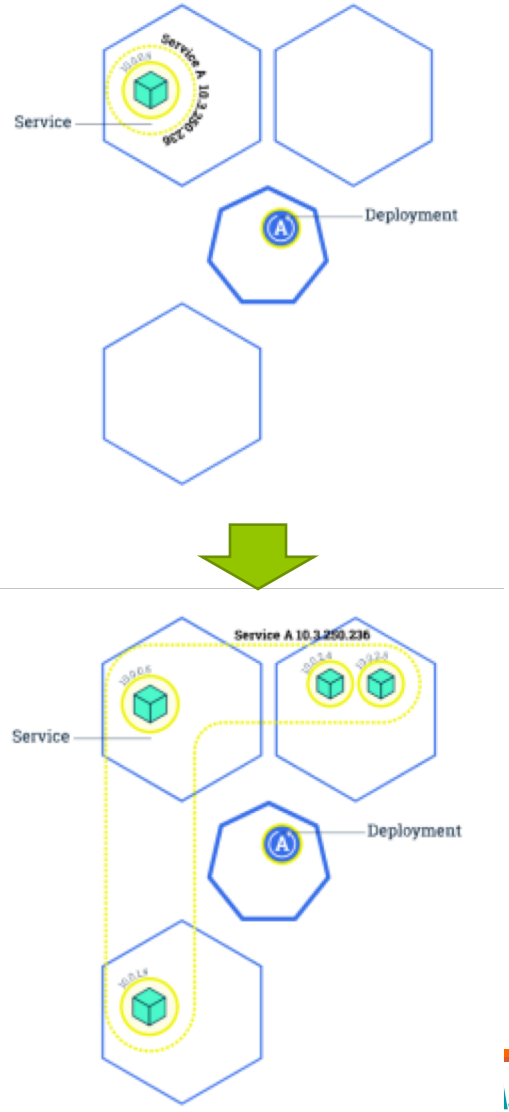
Microservices in Production

Part 3

Kubernetes: Scaling

Changing the number of resources to meet a desired state

- **Accomplished by adjusting the number of replicas in a deployment**
- **Accommodates both scaling up and scaling down to a minimum of 0**
- **Traffic is automatically sent to newly created instances through the service load balancer**
- **Can be used to enable rolling updates without downtime**



$ kubectl scale deployments/<deployment> --replicas=<new num>

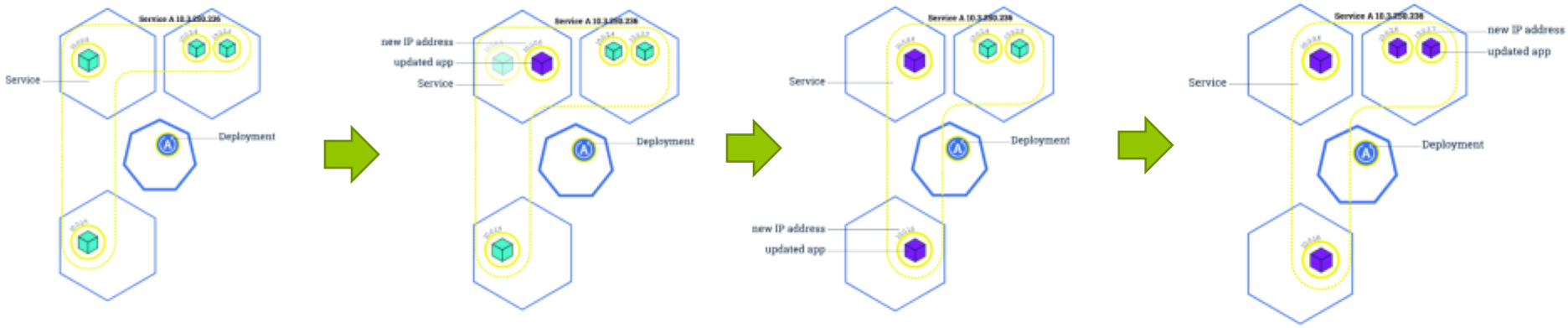
```
peter@Azure:~$ kubectl run nginx --image nginx --port 80
deployment "nginx" created
peter@Azure:~$
peter@Azure:~$ kubectl get deployments
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx     1         1         1            1           39s
peter@Azure:~$ kubectl scale deployments/nginx --replicas=2
deployment "nginx" scaled
peter@Azure:~$ kubectl get deployments
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx     2         2         2            1           1m
```

Note: Make sure to delete the deployments or replicasets when trying to clear out pods.

Kubernetes: Rolling Update

Updates the pods in a serial manner will load balancing traffic to available instances

- **If deployment is exposed publicly, the service will load-balance traffic to only active and available pods**
- **Used to enable zero downtime updates**
- **Allows greater application update frequency**
- **Can also be leveraged to rollback to previous versions**



```
root@ip-172-31-0-112:~/code# kubectl run nginx --image nginx:1.12.1 --port 80
deployment.apps/nginx created
root@ip-172-31-0-112:~/code# kubectl set image deployments/nginx nginx=nginx:latest
deployment.extensions/nginx image updated
root@ip-172-31-0-112:~/code# kubectl describe deployment nginx
Name:                   nginx
Namespace:              default
CreationTimestamp:      Thu, 19 Jul 2018 04:00:55 +0000
Labels:                 run=nginx
Annotations:            deployment.kubernetes.io/revision=2
Selector:               run=nginx
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  run=nginx
  Containers:
   nginx:
    Image:        nginx:latest
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-6c486b77db (1/1 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  43s   deployment-controller  Scaled up replica set nginx-7f9bc86464 to 1
  Normal  ScalingReplicaSet  11s   deployment-controller  Scaled up replica set nginx-6c486b77db to 1
  Normal  ScalingReplicaSet  10s   deployment-controller  Scaled down replica set nginx-7f9bc86464 to 0
```

```
root@ip-172-31-0-112:~/code# kubectl rollout undo deployments/nginx
deployment.extensions/nginx
root@ip-172-31-0-112:~/code# kubectl describe deployments/nginx
Name:                   nginx
Namespace:              default
CreationTimestamp:      Thu, 19 Jul 2018 04:00:55 +0000
Labels:                 run=nginx
Annotations:            deployment.kubernetes.io/revision=3
Selector:               run=nginx
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  run=nginx
  Containers:
   nginx:
    Image:        nginx:1.12.1
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-7f9bc86464 (1/1 replicas created)
Events:
  Type    Reason             Age                From                   Message
  ----    ------             ----               ----                   -------
  Normal  ScalingReplicaSet  1m                 deployment-controller  Scaled up replica set nginx-6c486b77db to 1
  Normal  ScalingReplicaSet  1m                 deployment-controller  Scaled down replica set nginx-7f9bc86464 to 0
  Normal  ScalingReplicaSet  18s (x2 over 2m)   deployment-controller  Scaled up replica set nginx-7f9bc86464 to 1
  Normal  DeploymentRollback 18s                deployment-controller  Rolled back deployment "nginx" to revision 1
  Normal  ScalingReplicaSet  17s                deployment-controller  Scaled down replica set nginx-6c486b77db to 0
```

Note: Rollbacks can also be also enacted with zero-downtime

kubectl rollout status deployment nginx

kubectl rollout history deployment nginx

kubectl rollout history deployment/nginx --revision=3

kubectl rollout undo deployment/nginx --to-revision <num>

Kubernetes Cheat Sheet (CRUD snapshot)

Creating Objects

Kubernetes manifests can be defined in json or yaml. The file extension `.yaml`, `.yml`, and `.json` can be used.

```
$ kubectl create -f ./my-manifest.yaml           # create resource(s)
$ kubectl create -f ./my1.yaml -f ./my2.yaml     # create from multiple files
$ kubectl create -f ./dir                        # create resource(s) in all manifest files in dir
$ kubectl create -f https://git.io/vPieo         # create resource(s) from url
$ kubectl run nginx --image=nginx                # start a single instance of nginx
$ kubectl explain pods,svc                       # get the documentation for pod and svc manifests
```

Viewing, Finding Resources

```
# Get commands with basic output
$ kubectl get services                          # List all services in the namespace
$ kubectl get pods --all-namespaces             # List all pods in all namespaces
$ kubectl get pods -o wide                      # List all pods in the namespace, with more details
$ kubectl get deployment my-dep                 # List a particular deployment

# Describe commands with verbose output
$ kubectl describe nodes my-node
$ kubectl describe pods my-pod

$ kubectl get services --sort-by=.metadata.name # List Services Sorted by Name
```

Deleting Resources

```
$ kubectl delete -f ./pod.json                      # Delete a pod using the type and name specified in pod.json
$ kubectl delete pod,service baz foo                # Delete pods and services with same names "baz" and "foo"
$ kubectl delete pods,services -l name=myLabel      # Delete pods and services with label name=myLabel
$ kubectl -n my-ns delete po,svc --all              # Delete all pods and services in namespace my-ns
```

Exercise 3.1 Wordpress on Kubernetes

CLASSROOM WORK 30 minutes

1. **Create persistent volume**
2. **Create secret for MySQL password**
3. **Launch Wordpress**

Source code:
https://github.com/techtown-training/microservices-bootcamp/tree/master/exercise/src_code/kubernetes_additional_exercise/ex3.1

Step by step instruction
https://github.com/techtown-training/microservices-bootcamp/blob/master/exercise/exercise3.1-create_pv.md

https://github.com/techtown-training/microservices-bootcamp/blob/master/exercise/exercise3.1-create_volumes.md

Spotify Case

Spotify: Situation

Founded in April 2006 and launch in Oct 2008, Spotify was immediately competing with incumbent behemoth Apple music.

- **In addition to a variety of other streaming services, Spotify also had to content with the entry of Google music in 2011 and even Amazon music in 2017.**
- **How does Spotify survive and win with essentially a commodity service while competing with tech giants that have more capital, bigger budgets and access to talent in a larger variety of geographic locations including silicon valley?**
  - **Spotify was founded in Stockholm Sweden.**

Spotify: Action

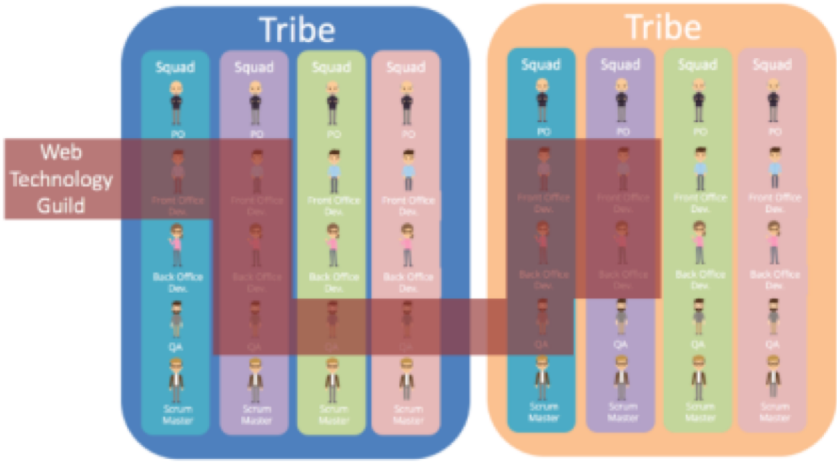
"We have chosen to optimize for speed of delivery," said Spotify's principal engineer, Niklas Gustavsson. "The main reason behind this is that we are in a highly competitive and complicated market and we think that we can win by being faster than our competitors."

- **Optimize for speed**
- **Separated the developer teams into fully autonomous groups of 7 +- 2 individuals**
  - "By making teams as independent as we can, they won't be blocking each other. Each team can execute on their own." - **Gustavsson**
- **Responsible to design, develop, test, and release into production**
- **Self organizing (Kanban, Scrum, mix)**

Spotify: Action

Instead of centrally mandating solutions such as a type of agile or code editor, each team is allowed the freedom to investigate and choose what will make them most productive.

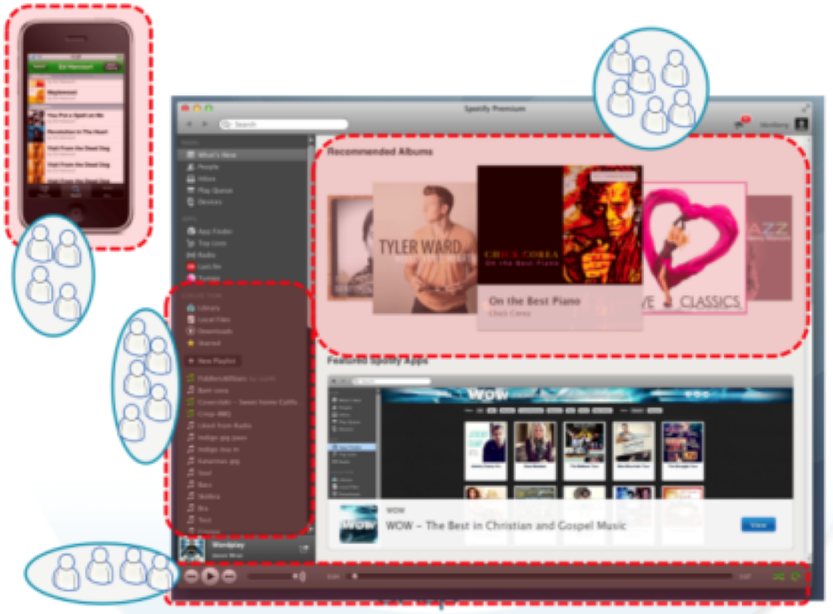
- **Autonomy leads to cross pollination of best practices**
- **Guild process (Regular meetings of groups of like minded individuals) also encourages cross pollination of best practices**
- **Once best practices are widely adopted then they become defacto standards**



Spotify: Action

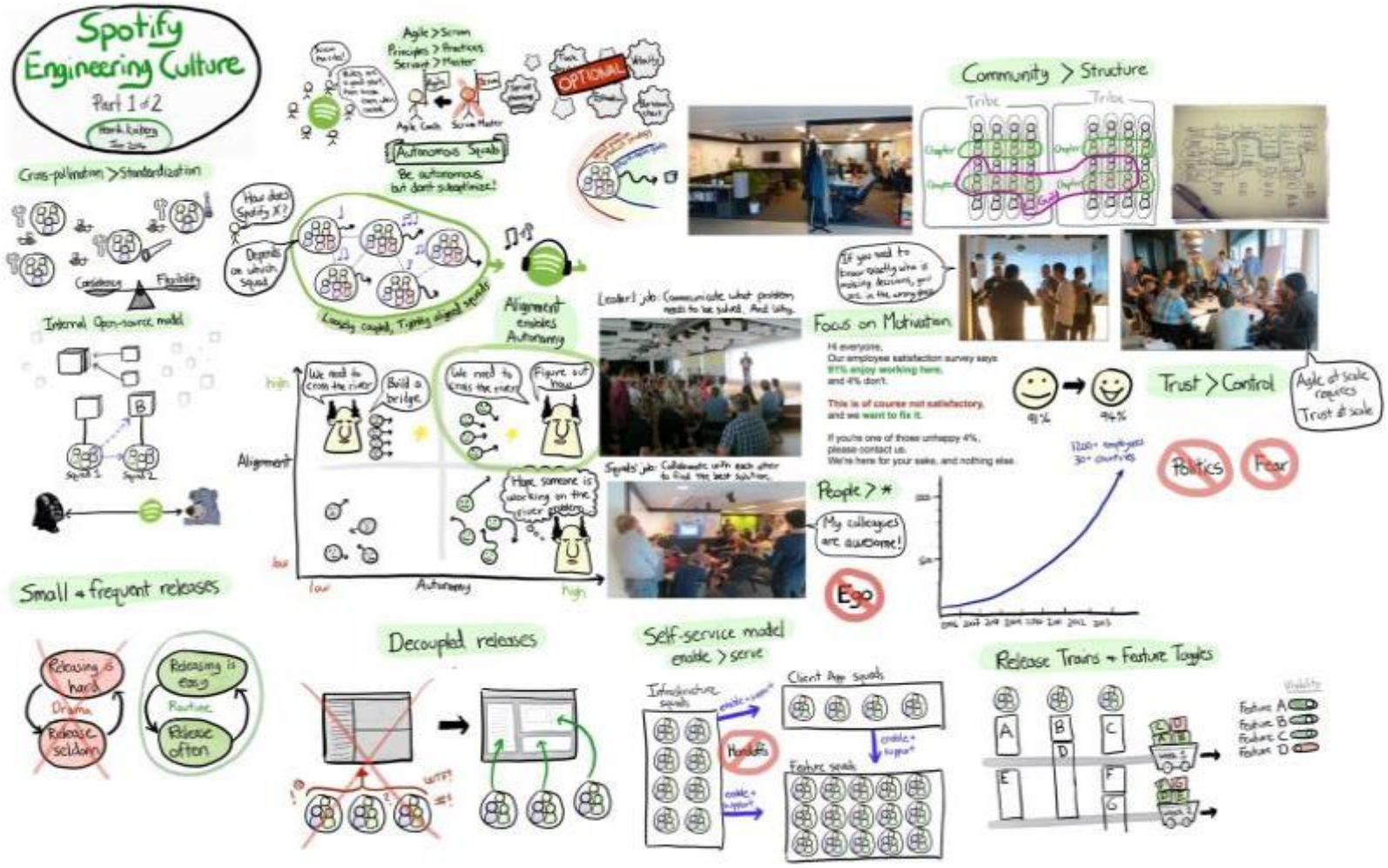
Each squad has a well defined mission. “"That might be to make the best search product in the world, or the best music stream quality product in the world. Or it might be things like growing our subscriber base or building APIs for third parties.” Gustavsson

- **Clear long term mission per squad**
- **Lean startup/MVP and validated learning encouraged**
- **Metrics and A/B testing to verify hypothesis**
- **Rapid releases encouraged**

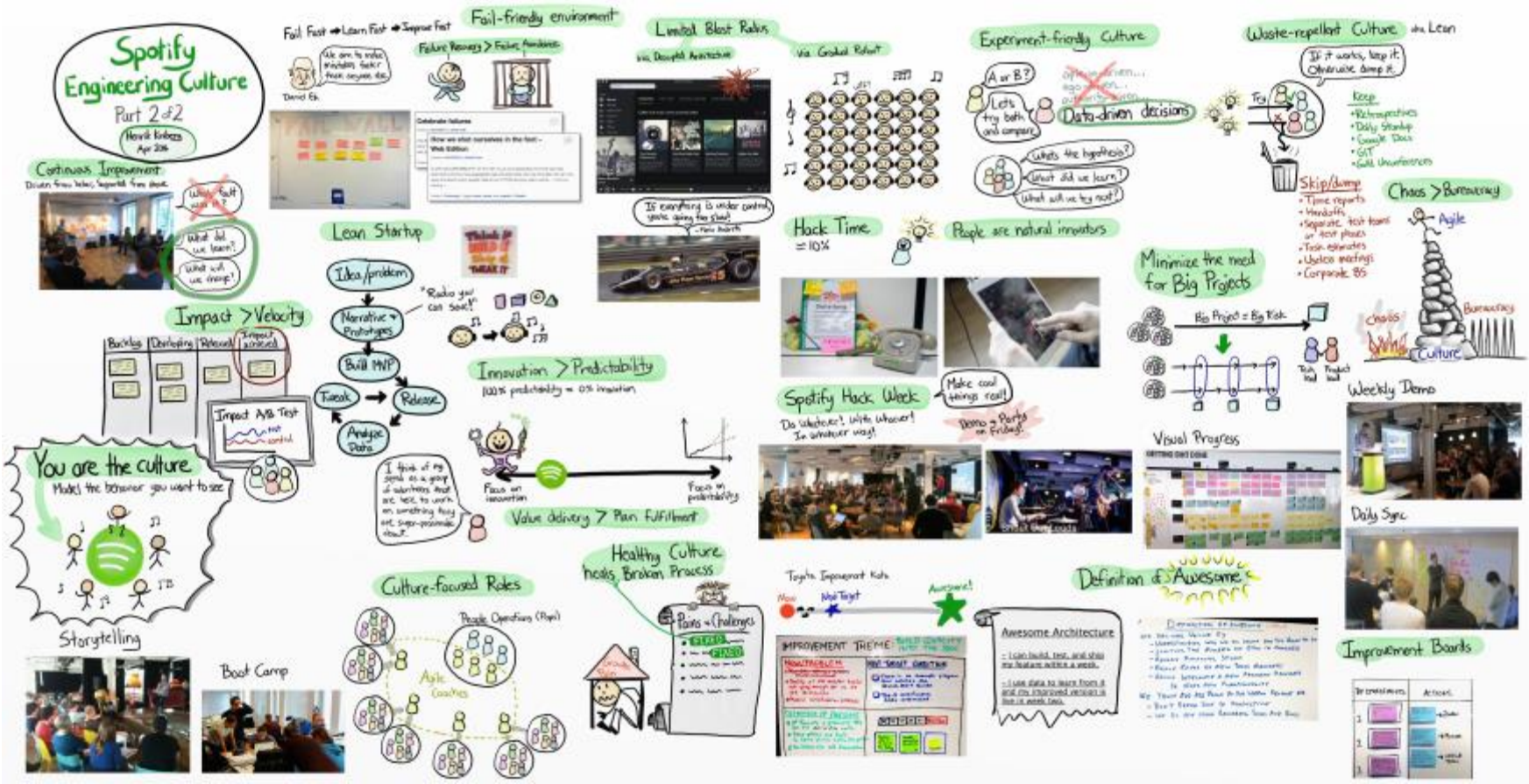


How different squads may own different parts of the user experience

Spotify: Action Summary (1/2)



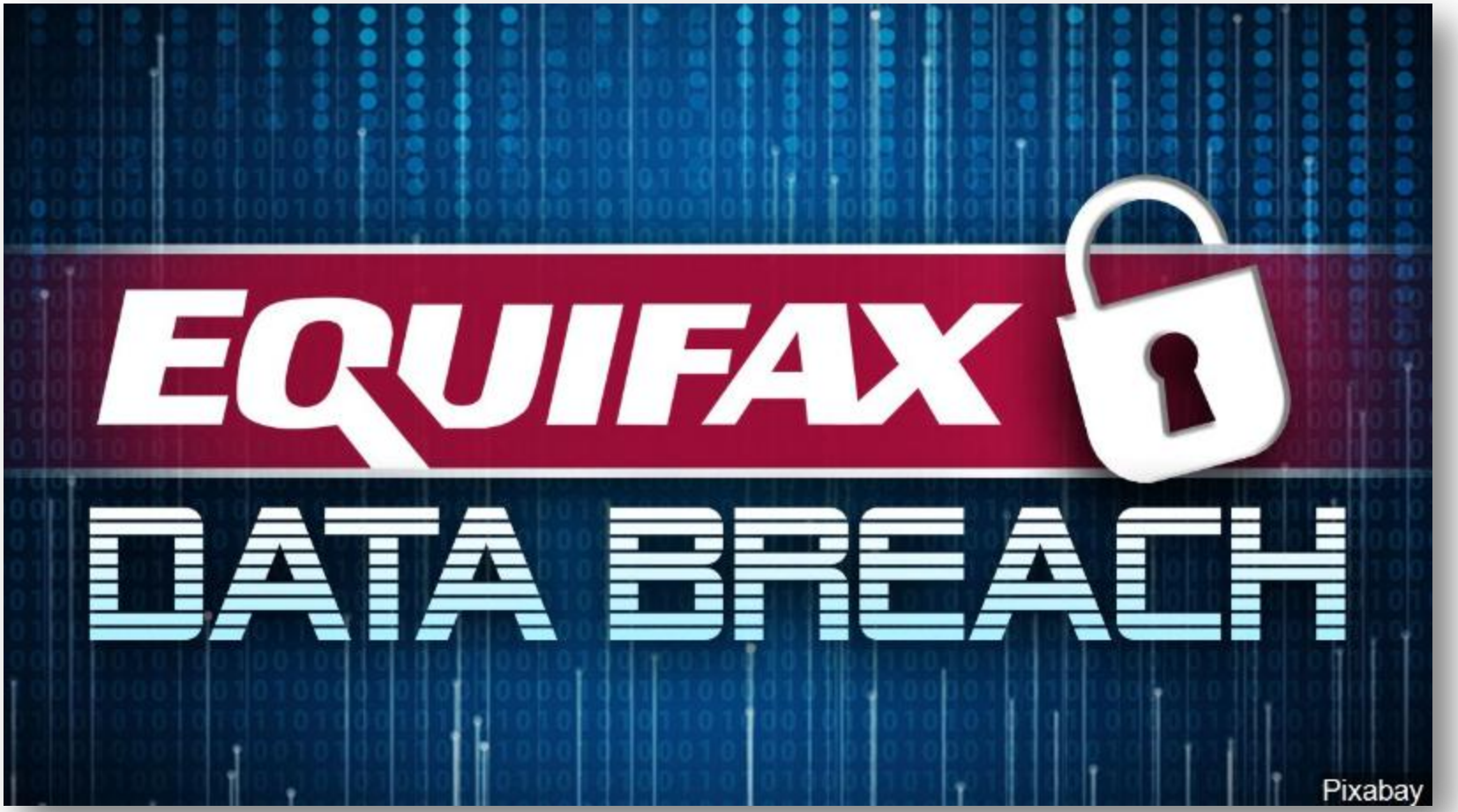
Spotify: Action Summary (2/2)



Spotify: Result

- Scaled to over 1600 employees
- Over $2B in revenue
- 08/17 Spotify was the most downloaded music App in the United States
- Over 140M active users with 60M paying subscribers
- Planned IPO in the near future

Equifax Case



Situation

Facts

- 143 million breach victims had their names, social security numbers, birth days, addresses and driver licenses stolen by attackers in May 2017
- The site was hacked in mid-May, and attacked continued to access the data until late July 2017 (breach discovered)
- The Equifax breach notification page was using a free shared CloudFlare SSL, causing many browsers to think it was a phishing site (not terrible, but not helping)
- Signing up for Equifax's identity theft protection forces uses to accept a terms of service that waive ability to sue Equifax. (Also not helping)
- Finally, an employee portal in Argentina had admin/admin as the username/password. (indicates a lax security posture)

Action: Equifax Tech Stack

What was the tech stack?

- Apache Struts
- IBM WebSphere
- Java

Apache Struts was compromised by an announced vulnerability 2 months prior (Apache Struts CVE-2017-5638)

- Failure to track and update vulnerabilities is a common OWASP (open web application security) issue (A9)

The personal information of the entire US adult population was within reach of this vulnerable web server

- Over reliance on perimeter security is absolutely suspect

Action: How to prevent easy attacks?

Learn the basics! OWASP Top 10 application vulnerabilities

OWASP Top 10 – 2013 (Previous)	OWASP Top 10 – 2017 (New)
A1 – Injection	A1 – Injection
A2 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References - Merged with A7	A4 – Broken Access Control (Original category in 2003/2004)
A5 – Security Misconfiguration	A5 – Security Misconfiguration
A6 – Sensitive Data Exposure	A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control - Merged with A4	A7 – Insufficient Attack Protection (NEW)
A8 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards - Dropped	A10 – Underprotected APIs (NEW)

Notes

Action: How to prevent easy attacks?

Learn the basics!

8 secure design principles from Saltzer and Schroeder:

- **Economy of mechanism: Keep the design simple**
- **Fail-safe defaults: Fail towards denying access**
- **Complete mediation: Check authorization of every access request**
- **Open design: Assume attack knows the system internals**
- **Separation of privilege: require two separate keys or other ways to check authorization (2 factor auth)**
- **Least privilege: Give only necessary rights**
- **Least common mechanisms: Ensure failures stay local**
- **Psychological acceptability: design security mechanism that are easy to use**

Action: A layered architecture (Microservices) would have likely defeated this attack

- Apache Software Foundation Response
- "Establish security layers. It is good software engineering practice to have individually secured layers behind a public-facing presentation layer"
- "A **breach into the presentation layer** should **never empower access** to significant or even **all back-end** information resources. "
- Principle of Least Privilege
- Implementing rate limiting between services would have been effective
- Normal use case for data retrieval required retrieving a single record per session and retrieving the entire database should have set off alarms

Action: Monitoring between layers would have also defeated the attack

- Apache Software Foundation
- “Establish monitoring for unusual access patterns to your public Web resources ”
- Queries could have been monitored looking for suspicious behavior
  - Many open and commercial tools will look for common attach patterns and alert
  - A firewall could have been placed between services
- **Basically, don’t put all of America’s PII behind one compromised web server**
  - And if you do, try to monitor and halt the breach before several months pass

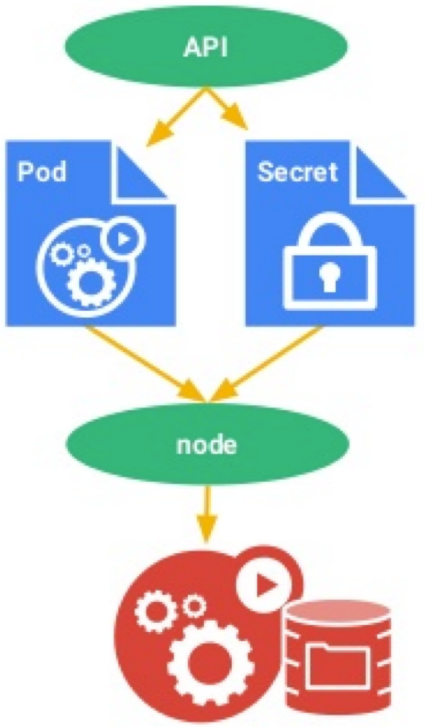
Result: Security basics for Microservices and beyond

1. OWASP Top 10
2. 8 Secure Design Principles from Saltzer and Schroeder
3. Understand and keep updated libraries and frameworks used. Know any known security vulnerabilities. Roll out updates rapidly.
4. Monitor for intrusions and run scenarios if something you imagine cannot be hacked was indeed exploited, what would happen?
5. Have a layered security defense when possible

Kubernetes: Secrets

An object that contains a small amount of sensitive data such as a password, token or key.

- **According to 12-factor configuration comes from the environment**
  - **Config is everything that is likely to vary between deployments**
- **Can be used by pods and the underlying Kubelet when pulling images**
- **Secrets belong to a specific Kubernetes Namespace**
- **Secret size cannot exceed 1MB**



Kubernetes: Secrets

```
peter@Azure:~/secret$ echo -n "A19fh68B001j" > ./apikey.txt
peter@Azure:~/secret$ ls
apikey.txt  config.json  vault
peter@Azure:~/secret$ kubectl create secret generic apikey --from-file=./apikey.txt
secret "apikey" created
peter@Azure:~/secret$ kubectl describe secrets/apikey
Name:         apikey
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
apikey.txt:  12 bytes
peter@Azure:~/secret$ kubectl create -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/secrets/pod.yaml
W1103 11:54:36.624294     136 factory_object_mapping.go:423] Failed to download OpenAPI (the server could not find the requested
 resource), falling back to swagger
pod "consumesec" created
peter@Azure:~/secret$ kubectl exec consumesec -c shell -i -t -- bash
[root@consumesec /]# mount | grep apikey
tmpfs on /tmp/apikey type tmpfs (ro,relatime)
[root@consumesec /]# cat /tmp/apikey/apikey.txt
```

Note: Can be mounted as data volumes or exposed as environment variables

Exercise 3.2 Kubernetes Secrets

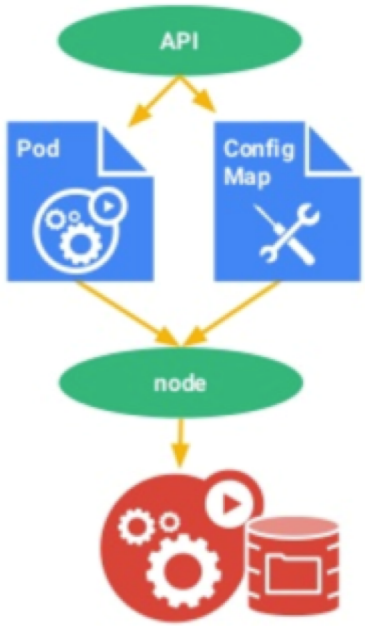
CLASSROOM WORK - Optional

1. **Create Secret config file and pods**
2. **Install and Launch Vault**

https://github.com/techtown-training/microservices-bootcamp/blob/master/exercise/exercise3.2-secrets.md

Kubernetes: Configmap

A set of key-value pairs that serve as configuration data for pods. They solve the problem of how to pass config data such as environment variables to pods.



Config maps are commonly composed of:

- **Command line arguments**
- **Environment variables**
- **Files in a volume**

Config maps function similar to secrets, but values are stored as strings and are more readily readable.

$ kubectl create -f etcd-config.yml

```
apiVersion: extensions
kind: ConfigMap
metadata:
  name: etcd-env-config
data:
  discovery_url: http://etcd-discovery:2379
  etcdctl_peers: http://etcd:2379
```

Exercise 3.3 Kubernetes Configmap

CLASSROOM WORK

1. **Create Configmap**
2. **Consume Configmap in environment variables**
3. **Set command line using Configmap**

Source code:
https://github.com/techtown-training/microservices-bootcamp/tree/master/exercise/src_code/kubernetes_additional_exercise/ex3.3

Step by step instruction:
https://github.com/techtown-training/microservices-bootcamp/blob/master/exercise/exercise3.3-configmap.md

Kubernetes: Volumes

Storage that has an explicit lifetime, the same as the pod that encloses it.

- **A directory accessible to containers in a pod**
- **Volumes outlive containers that run within a pod**
- **May be passed between pods**
- **Data is preserved across container restarts**
- **Azure Disk & Azure File supported**

Kubernetes: Volumes

```
peter@Azure:~$ kubectl exec sharevol -c c1 -i -t -- bash
[root@sharevol /]# mount | grep xchange
/dev/sda1 on /tmp/xchange type ext4 (rw,relatime,discard,data=ordered)
[root@sharevol /]# echo 'some data' > /tmp/xchange/data
[root@sharevol /]# exit
exit
peter@Azure:~$ kubectl exec sharevol -c c2 -i -t -- bash
[root@sharevol /]# mount | grep /tmp/data
/dev/sda1 on /tmp/data type ext4 (rw,relatime,discard,data=ordered)
[root@sharevol /]# cat /tmp/data/data
some data
[root@sharevol /]#
```

Note: **Volumes are based on a wide assortment of underlying storage provides. Every major public cloud (including Azure) has several options for volume storage**

Kubernetes: Volumes

```
peter@Azure:~$ kubectl exec sharevol -c c1 -i -t -- bash
[root@sharevol /]# mount | grep xchange
/dev/sda1 on /tmp/xchange type ext4 (rw,relatime,discard,data=ordered)
[root@sharevol /]# echo 'some data' > /tmp/xchange/data
[root@sharevol /]# exit
exit
peter@Azure:~$ kubectl exec sharevol -c c2 -i -t -- bash
[root@sharevol /]# mount | grep /tmp/data
/dev/sda1 on /tmp/data type ext4 (rw,relatime,discard,data=ordered)
[root@sharevol /]# cat /tmp/data/data
some data
[root@sharevol /]#
```

Note: **Volumes are based on a wide assortment of underlying storage provides. Every major public cloud (including Azure) has several options for volume storage**

Kubernetes: Persistent Volumes

Persistent storage in a cluster

- **Separate to PersistentVolumeClaim, which is a request for storage.**
- **May be created ahead of time in a static manner for use by a cluster**
- **May be created dynamically from available, unclaimed resources**
- **May be freed and passed from user to user so care should be taken to delete contents when done with use**
- **Critical for Stateful containers**

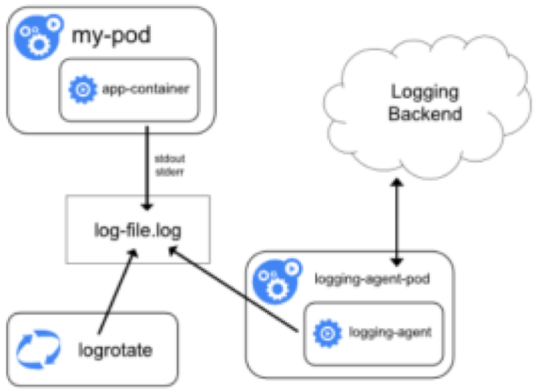
See example in the wordpress example :: **kind: PersistentVolumeClaim**

Kubernetes: Logging

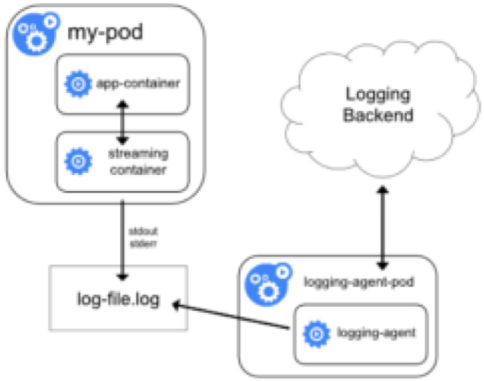
Aggregate, access and print logs from containers

- **Logs can be accessed from a currently run container or any number or previously run containers, accessed by container name**
- **Containerized apps write logs to stdout and stderr**
- **Cluster level logging can be performed using several different techniques including node level agents, container sidecars, or even having containers push directly to an application**
- **Node level logging is the most common approach, typically using Elasticsearch.**

Using a node logging agent



Streaming sidecar container



Kubernetes: Logging

```
peter@Azure:~$ kubectl create -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/logging/pod.yaml
W1009 12:28:03.089702     121 factory_object_mapping.go:423] Failed to download OpenAPI (the server could not find the requested resource), falling b
ack to swagger
pod "logme" created
peter@Azure:~$ kubectl logs --tail=5 logme -c gen
Mon Oct 9 12:28:18 UTC 2017
Mon Oct 9 12:28:19 UTC 2017
Mon Oct 9 12:28:19 UTC 2017
Mon Oct 9 12:28:20 UTC 2017
Mon Oct 9 12:28:20 UTC 2017
peter@Azure:~$ kubectl logs -f --since=10s logme -c gen
```

```
peter@Azure:~$ kubectl create -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/logging/oneshotpod.yaml
W1009 12:46:45.895922     139 factory_object_mapping.go:423] Failed to download OpenAPI (the server could not find the requested resource), falling b
ack to swagger
pod "oneshot" created
peter@Azure:~$ kubectl logs -p oneshot -c gen
9
8
7
6
5
4
3
2
1
```

Note: An external system can perform the log rotation by calling logrotate, which would result in kubectl logs returning an empty result

Kubernetes: Logging

```
peter@Azure:~$ kubectl create -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/logging/pod.yaml
W1009 12:28:03.089702     121 factory_object_mapping.go:423] Failed to download OpenAPI (the server could not find the requested resource), falling b
ack to swagger
pod "logme" created
peter@Azure:~$ kubectl logs --tail=5 logme -c gen
Mon Oct 9 12:28:18 UTC 2017
Mon Oct 9 12:28:19 UTC 2017
Mon Oct 9 12:28:19 UTC 2017
Mon Oct 9 12:28:20 UTC 2017
Mon Oct 9 12:28:20 UTC 2017
peter@Azure:~$ kubectl logs -f --since=10s logme -c gen
```

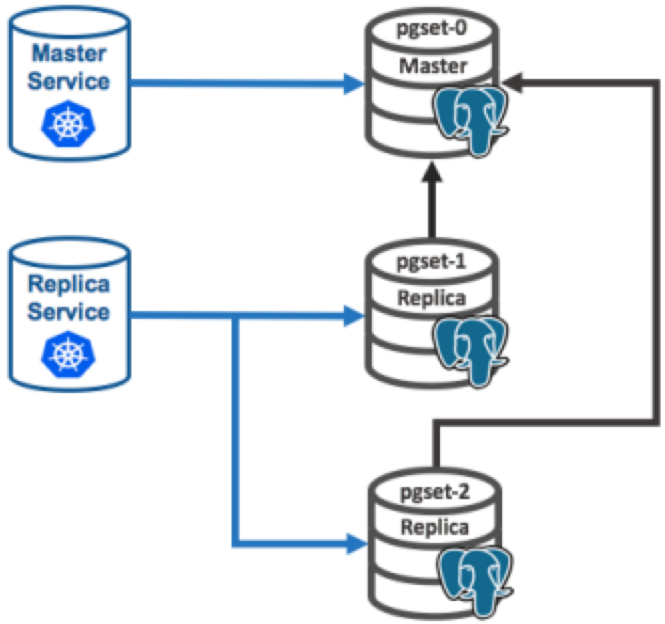
```
peter@Azure:~$ kubectl create -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/logging/oneshotpod.yaml
W1009 12:46:45.895922     139 factory_object_mapping.go:423] Failed to download OpenAPI (the server could not find the requested resource), falling b
ack to swagger
pod "oneshot" created
peter@Azure:~$ kubectl logs -p oneshot -c gen
9
8
7
6
5
4
3
2
1
```

Note: An external system can perform the log rotation by calling logrotate, which would result in kubectl logs returning an empty result

Kubernetes: Stateful Sets

Functionality of deployments with guarantees about ordering and unique identities per Pod.

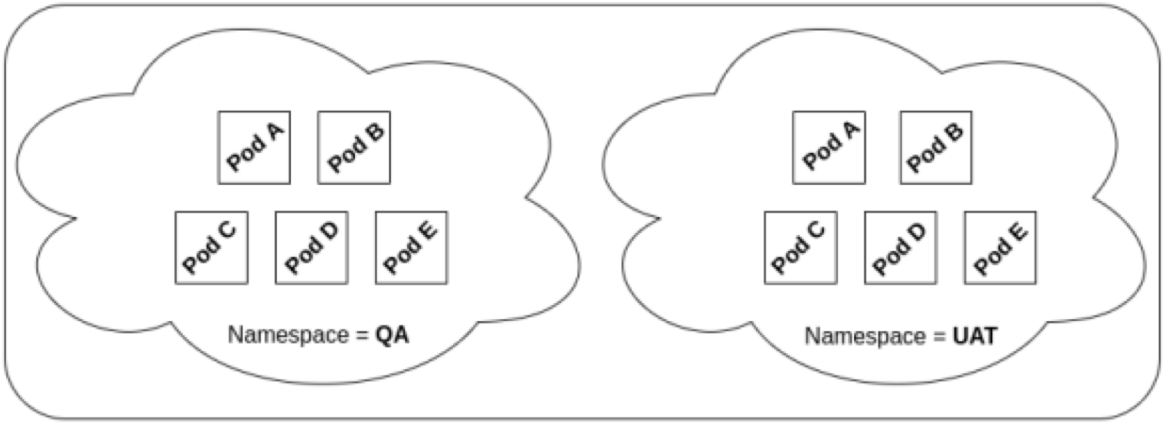
- Pods in a stateful set are not interchangeable
- Each Pod has a persistent identifier
- Define desired state object, controller performs updates to desired state
- Useful for applications that require persistent storage, unique network identifiers, ordered deployment, deletes or updates.



Kubernetes: Namespace

Partitions the names of API objects

- **Resources are segmented within namespaces**
  - **Sandbox per developer for example**
- **Pods will route dns to default (own) namespace**
  - **api.default.svc.cluster.local**
  - **Api.foo…**
- **The same app could run independently in different namespaces**



Kubernetes: Namespace

```
peter@Azure:~$ kubectl get ns
NAME          STATUS    AGE
default       Active    23d
kube-public   Active    23d
kube-system   Active    23d
peter@Azure:~$ kubectl create -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/ns/ns.yaml
W1006 13:59:41.630630     134 factory_object_mapping.go:423] Failed to download OpenAPI (the server could not find the requested resource), falling back to swagger
namespace "test" created
peter@Azure:~$ kubectl get ns
NAME          STATUS    AGE
default       Active    23d
kube-public   Active    23d
kube-system   Active    23d
test          Active    19s
peter@Azure:~$ kubectl create --namespace=test -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/ns/pod.yaml
W1006 14:00:18.027941     145 factory_object_mapping.go:423] Failed to download OpenAPI (the server could not find the requested resource), falling back to swagger
pod "podintest" created
peter@Azure:~$ kubectl get pods --namespace=test
NAME         READY     STATUS              RESTARTS   AGE
podintest    0/1       ContainerCreating   0          11s
```

Note: Namespaces create an excellent mechanism to subdivide resources and provide isolation. This enables using a cluster for multiple projects or multiple teams

Security

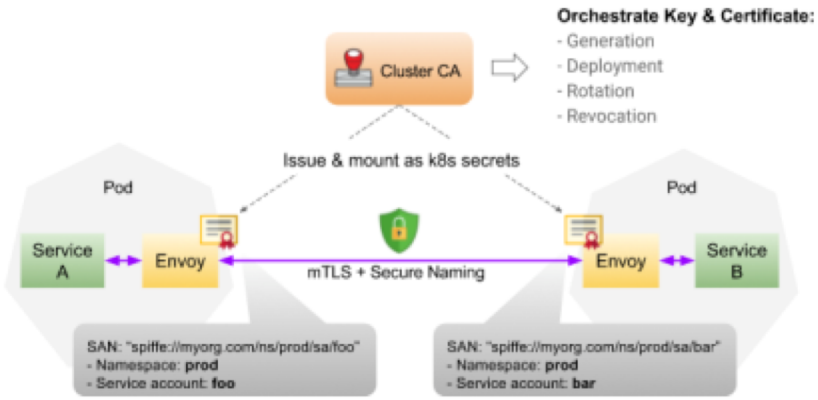
Secure Traffic

Secure traffic between Microservices

- HTTPS
  - Reconfigure ELBs (on AWS)
  - Add certificate (letsencrypt.org)

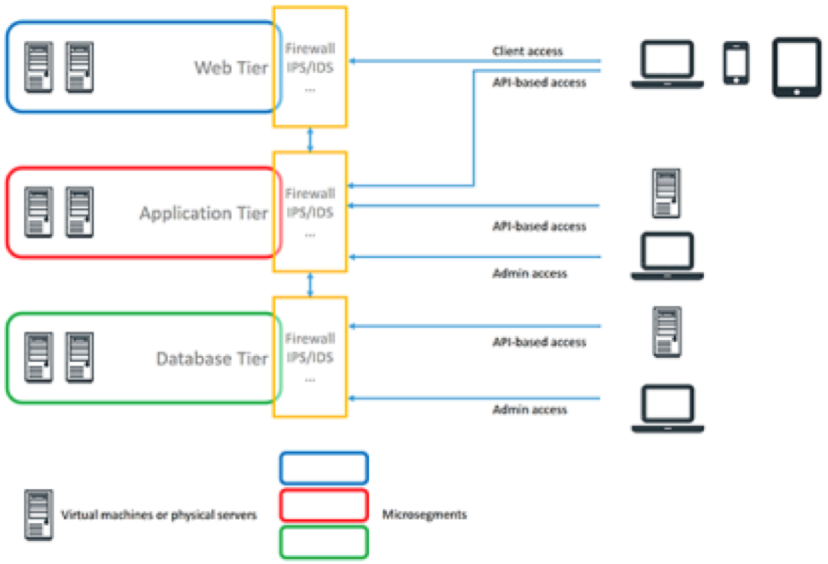
Secure traffic for users

- HTTPS
- Secure Identity with SPIFFEE & Istio



Security - Microsegmentation

- **Segment the network**
  - **Only allow white-listed traffic between pods**
- **Common tools**
  - **Istio**
  - **Calico**
  - **Weave**
  - **OpenContrail**



Security Actions

- Ensure attacker who gains access to web container can't get into the database
- Create separate DB users for each Microservice
- Strong passwords
  - K8's secrets and/or Vault
  - Only accessed by containers who need them

Security – Least Privilege

"Each subject in a system be granted the most restrictive set of privileges"

- **Disable public IP access for any machine that doesn't need it**
- **VPN only for admin access**

Security Actions

- Limit admin access
- “Attribute based access control” mode by default
- Access control by token
  - Can allow/deny API operations
  - Can be used to only allow certain actions
- Webhook (optional)
  - Add url for k8s to verify authorization

Course Summary and Supplemental Lab Work

Part 4

Deploy Cassandra with Stateful Sets

CLASSROOM WORK

1. Create Cassandra Headless Service
2. Create a Cassandra Ring
3. Validate the Stateful Set
4. Modify the Stateful Set

Helm (package manager) on Kubernetes

CLASSROOM WORK

1. **Verify Helm is working**
2. **Install nginx-ingress**

Windows on Kubernetes

CLASSROOM WORK

1. Verify Connectivity to Kubernetes cluster (skip steps if reusing previous cluster)
2. Create windows yaml file (iis.ysml)
3. Stand up windows machines and verify connectivity

Squads, Chapters and Guilds

SQUAD

Primary home (like a scrum team)

"Basically a mini startup"

Sit together.
One long-term mission.
Have all the skills/tools needed to design, develop, test & release to production.
Self-organising.
Lean, MVP, Validated Learning.
Become experts on product area.
Product Owner prioritises work.

Example: The Fashion Squad **Mission**: Create an awesome experience that makes it easy & fast to find & buy fashion

TRIBE

Collection of squads within business area

"Incubators for the mini startups"

Squads within tribe sit in same area.
<100 people per tribe.
Shared lounges promote inter-squad contact.
Regular informal get-togethers to share what working on, demos etc.
Tribe leader responsible for providing right habitat.

Example: The **Mobile** tribe includes the iPad, iPhone, Android and Touch squads.

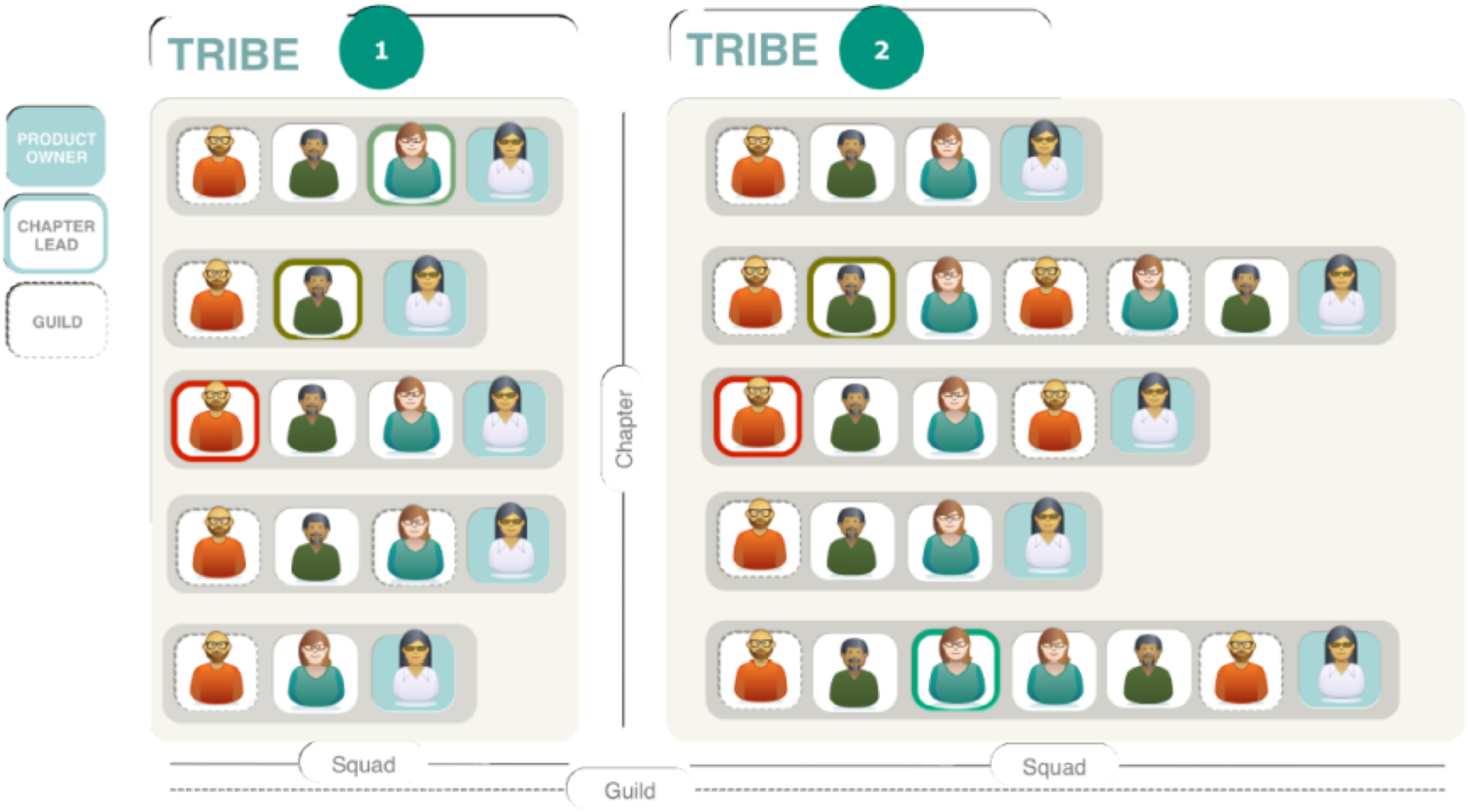
CHAPTER

Secondary home (within tribe)

People who do similar work (design, testing etc)

Meet regularly to discuss expertise and challenges.
Chapter lead is line manager for Chapter members.
Chapter Lead is responsible for developing people & setting salaries but remains part of squad & does day-to-day work.

Example: Within a tribe there will be design, testing, development and other chapters.


TRIBE 1
TRIBE 2
PRODUCT OWNER
CHAPTER LEAD
GUILD
Chapter
Squad
Squad
Guild

GUILD

A group of people from a across the organisation who want to share knowledge, tools, code & practices. Each has a Guild Co-ordinator. Some examples are the web technology guild, the test automation guild, and the agile coaching guild.

Communities of interest

Problem-solving & support

Quarterly Survey
Talk about where improvements and support are needed

Dependencies Reviews
Regularly review how squads see their dependencies. Reorganise work as necessary to eliminate inter-tribe and blocking dependencies.

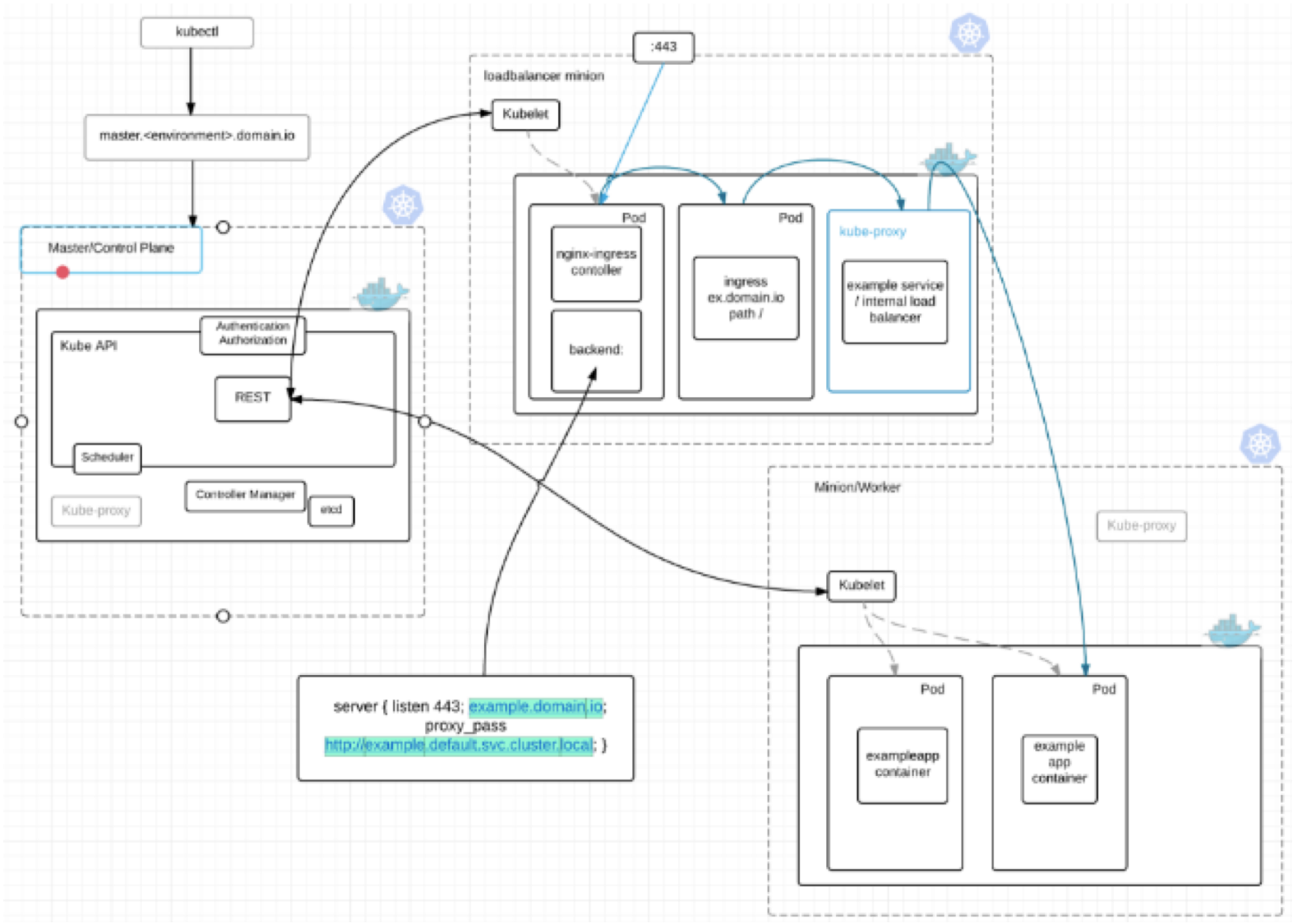
Scrum of Scrums Only On Demand
For example, when a large project requires the co-ordinated work of multiple squads for a few months you may choose to have daily sync meetings.

Operations Team
Gives squads the support they need to release code themselves: in the form of infrastructure, scripts, and routines.

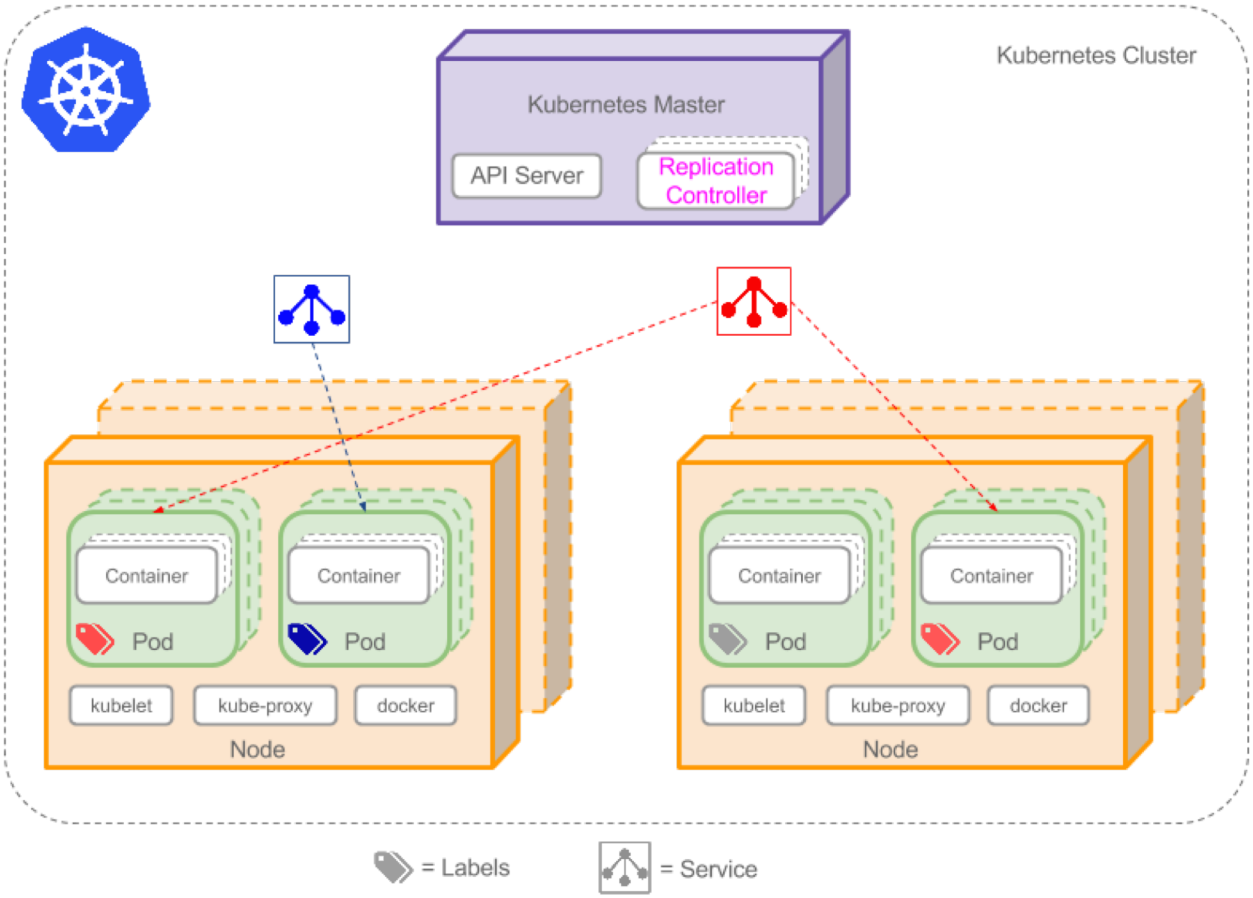
Each squad has access to Analytics expertise and an Agile Coach.

This is a summary of Spotify's whitepaper "Scaling Agile @Spotify" by Henrik Kniberg and Anders Ivarsson. Thanks for sharing and for the inspiration!

K8s Architecture – Networking View



K8s Architecture – Simplified



Elastic Search on K8s

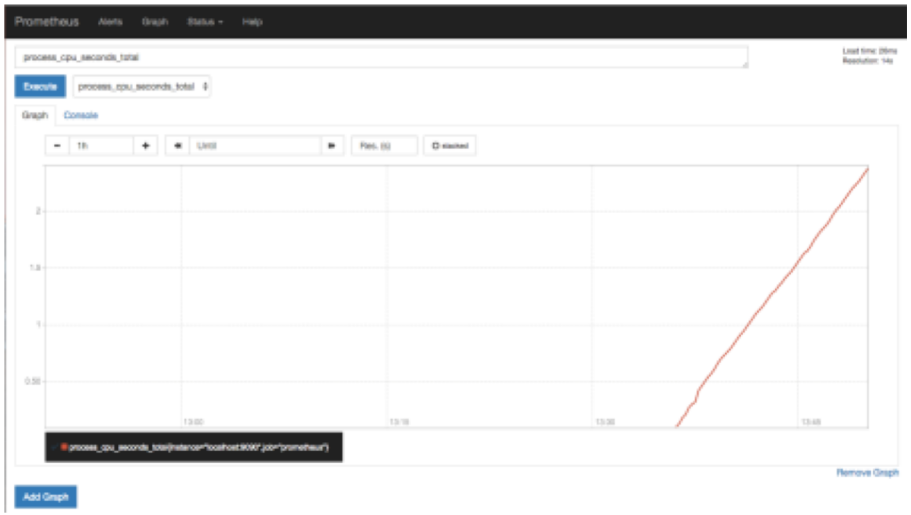
CLASSROOM WORK – No need

1. Clone the examples repo
2. Deploy Elasticsearch
3. Verify functionality

Kubernetes & Prometheus (Metrics)

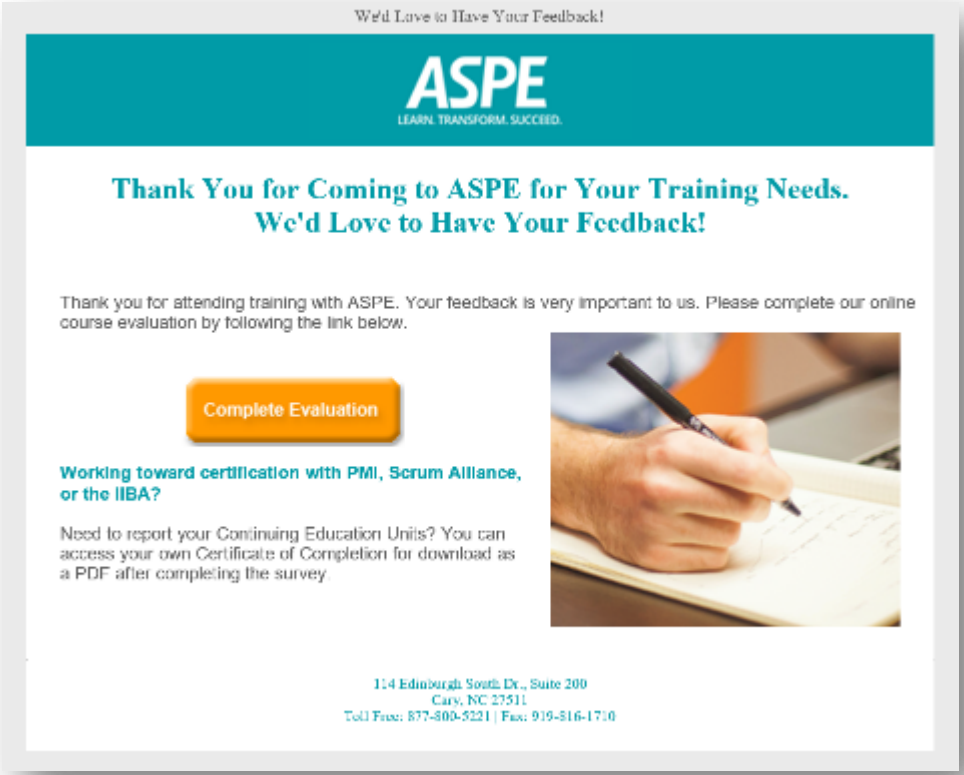
CLASSROOM WORK

1. **Stand up Prometheus**
2. **Gather and monitor metrics**



Feedback Email

- If you haven't already, you will receive an email similar to the one here asking for your feedback on the course.
- This Evaluation will ask your feedback on many aspects of your training, including the course materials, instructor facilitation and labs where applicable.
- We appreciate you completing this survey to help us continue to better our content and deliveries in order to serve you better.



Questions