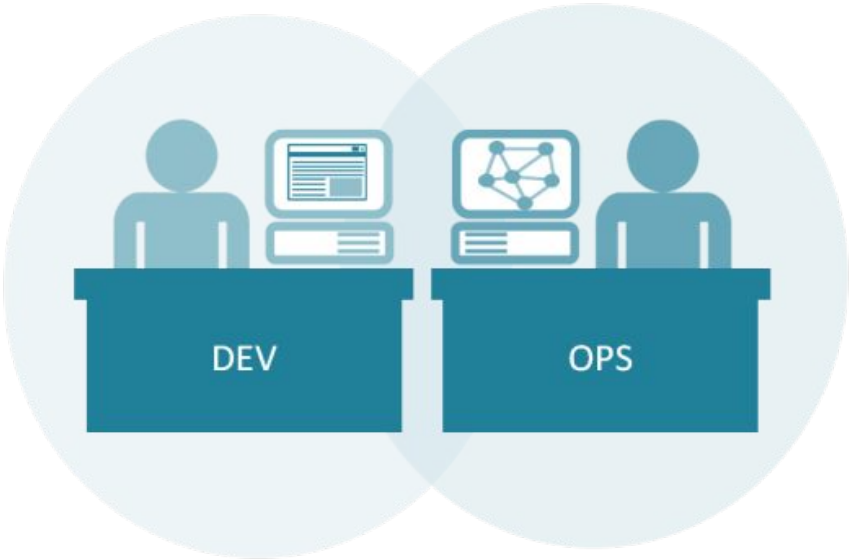


Microservices Engineering Boot Camp

{ TECHTOWN } training

64600M_3.0_2018

Welcome!

- Logistics (breaks, facilities, lunch, etc.)
- Rules of Engagement
- Introductions
- Let’s Get Started!

Who is your instructor today?

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.

Part 1:
Introduction to Microservices

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

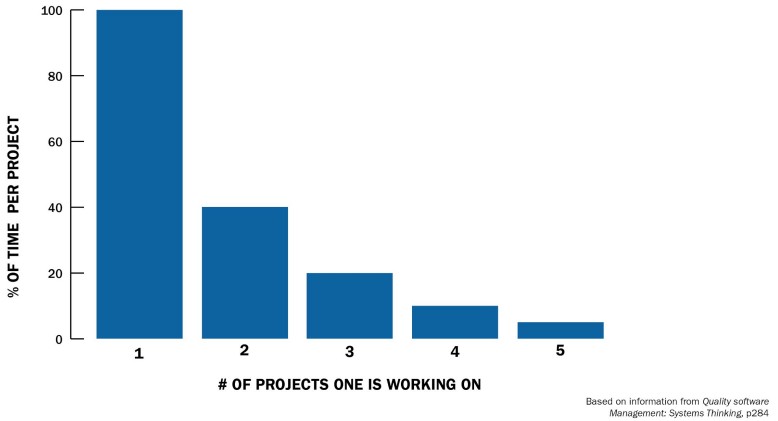
Lets imagine a medium sized enterprise app that has a team with roughly

- **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**

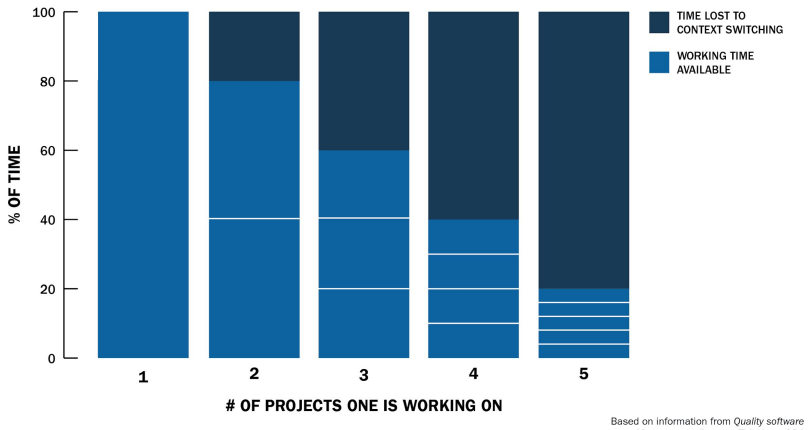
36 individuals total, with 1/3 partially allocated to this and other projects

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.

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.

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

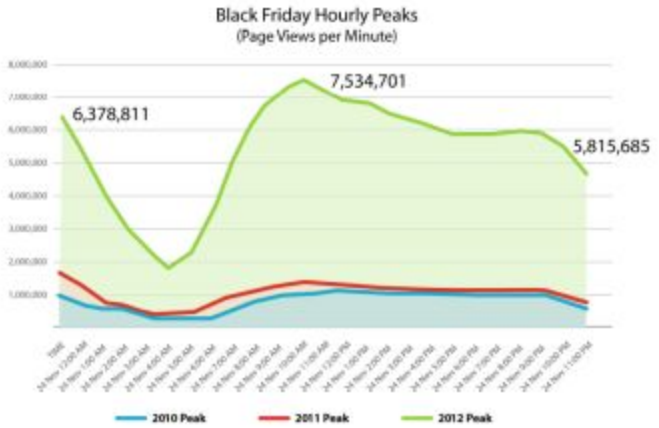
- A platform is an ecosystem of APIs that third parties may build functionality on top of.
- Notable Microservice Platform examples: Hootsuite, Gilt Group, SoundCloud, Amazon Web Services, Netflix, Spotify, Google Cloud, Box, and more!

Walmart Canada Case Study

Walmart Canada: Situation

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

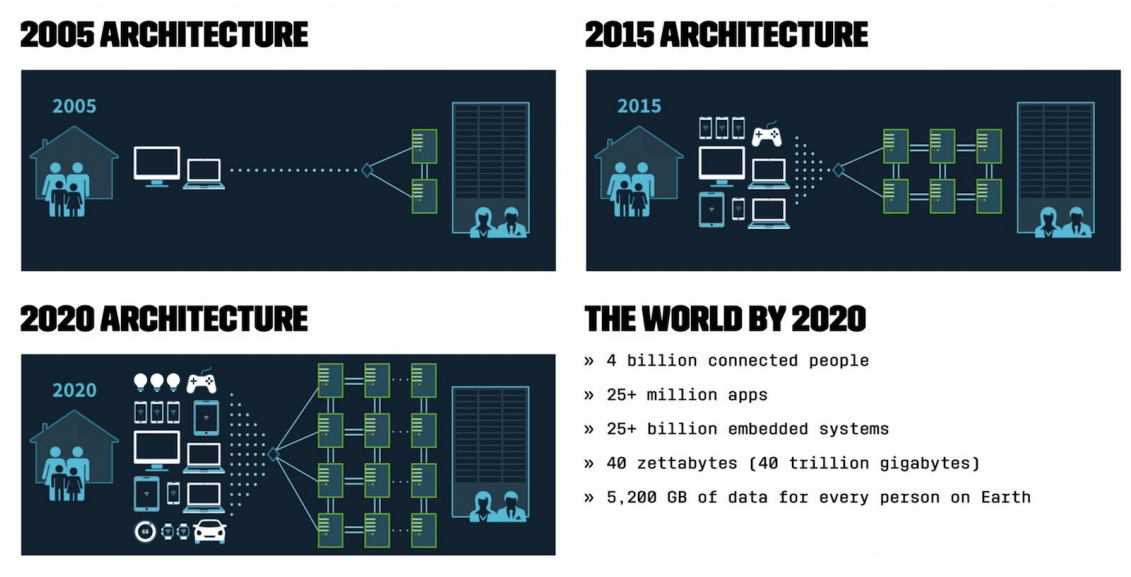
- Traffic was increasing every year, keeping the team off balance
- Customer backlash, competitive pressures (i.e. Amazon) made this a top priority



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

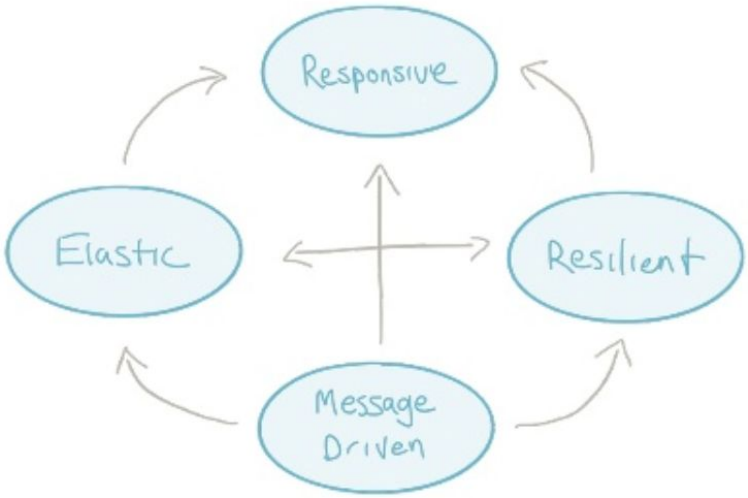
The ultimate maturity model
for microservices.

» Responsive

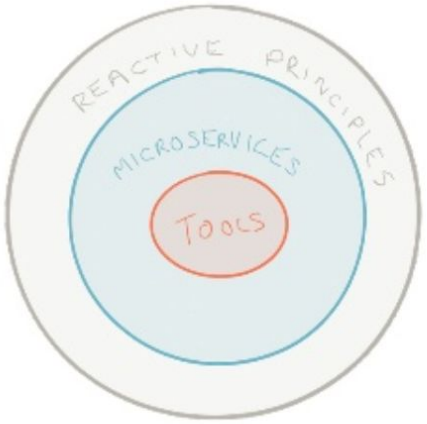
» Resilient

» Elastic

» Message-driven



Walmart Canada: Action



WHY, WHAT, HOW

Clip slid

Reactive (principles)

» responsive, resilient, elastic, message-driven

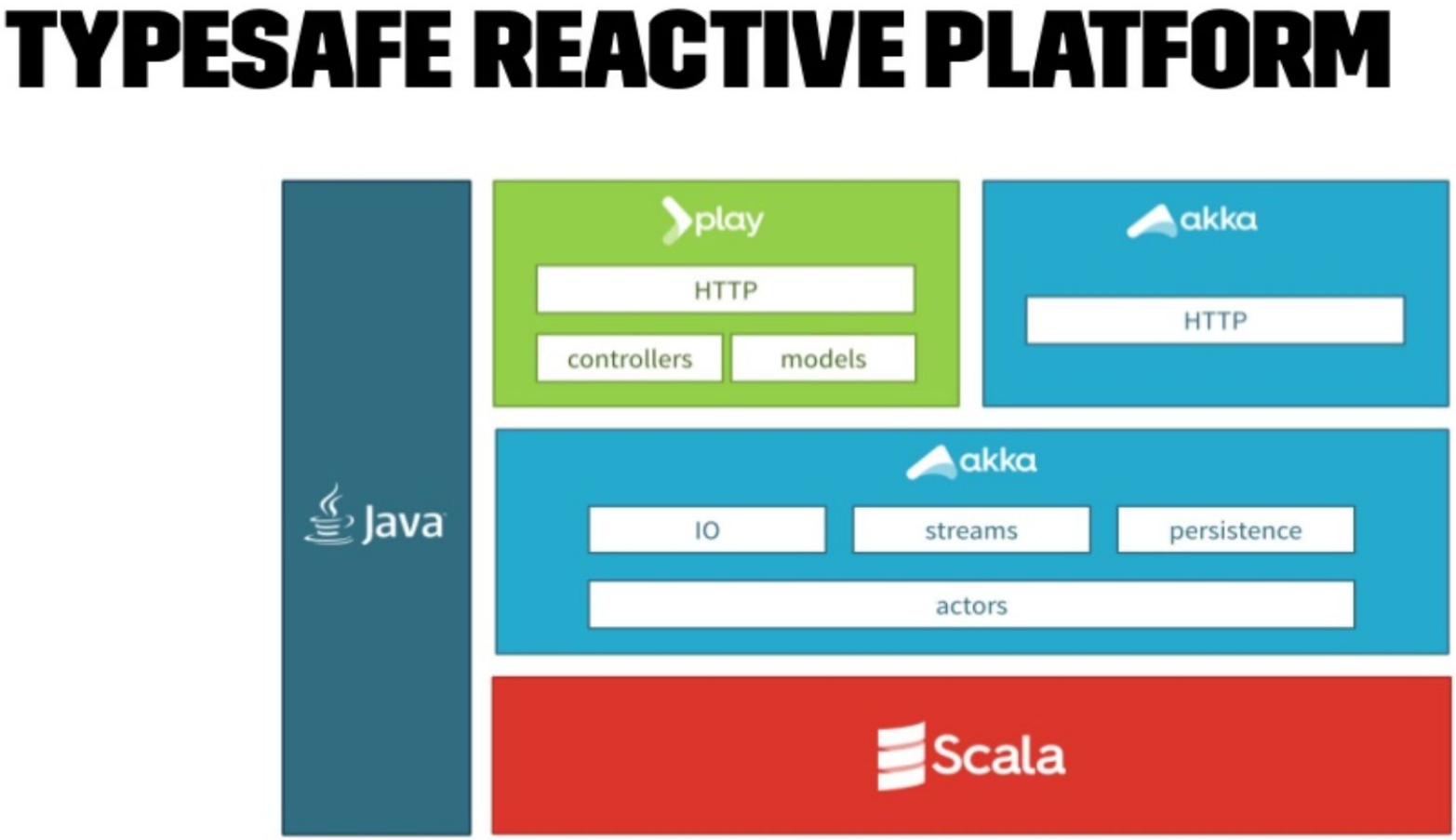
Microservices (strategy)

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

Tools (implementation)

» Typesafe Reactive Platform (Play, Akka, Spark)

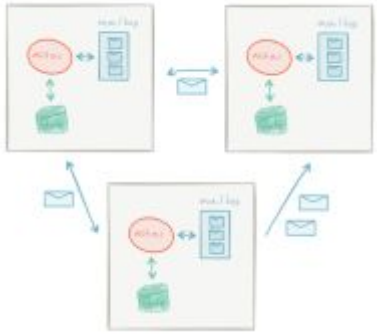
Walmart Canada: Action

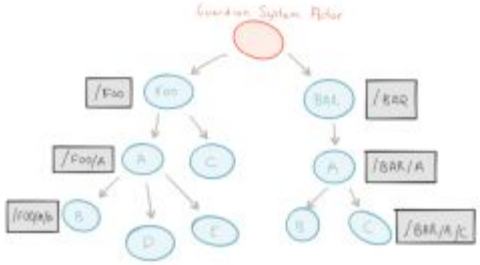


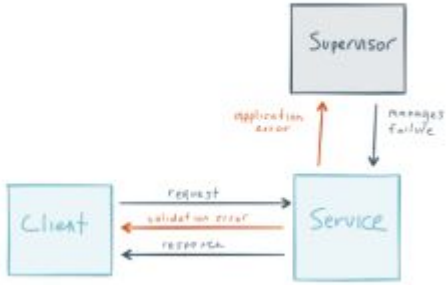
Walmart Canada: Action

Re-architected solution

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

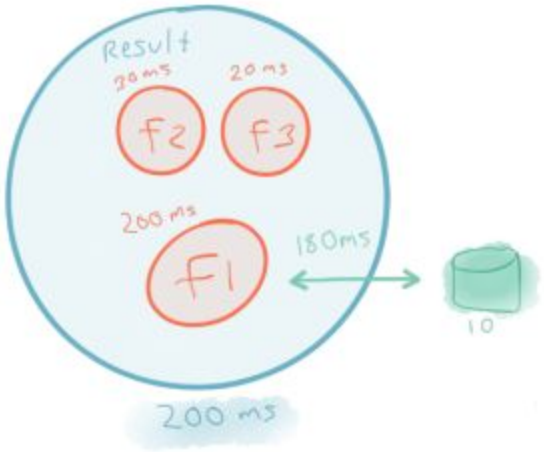


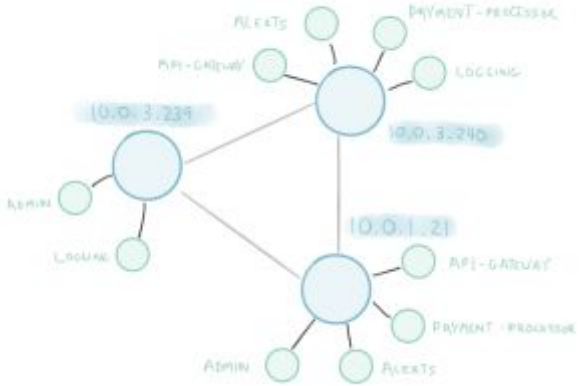




Walmart Canada: Action

- **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

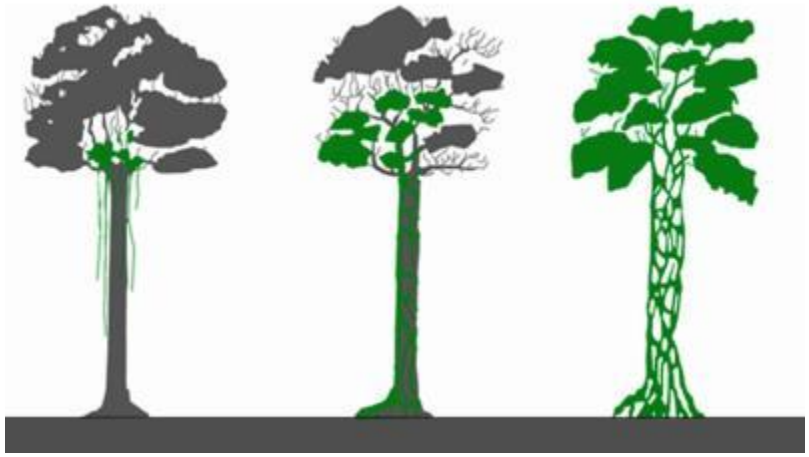
Strategy for legacy monoliths: The Strangler Pattern

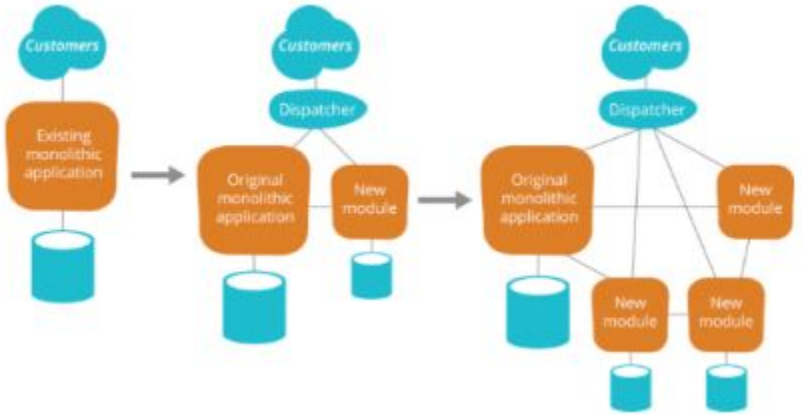
https://flic.kr/p/nMWBkR

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

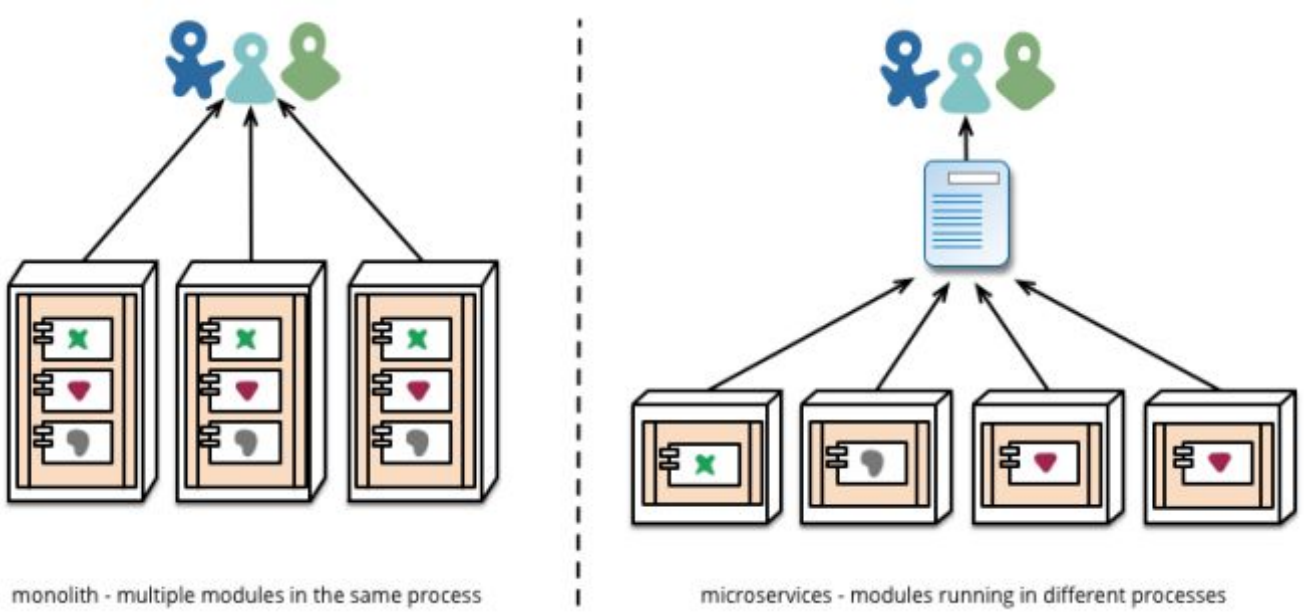
- Rather than trying to replace a large, expensive legacy system all at once you can iterate and improve in small amounts.
- This can be implemented for example by a load balancer in front the targeted functionality
- Over time, the old application is ‘strangled’ like a tree that has hosted a Strangler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





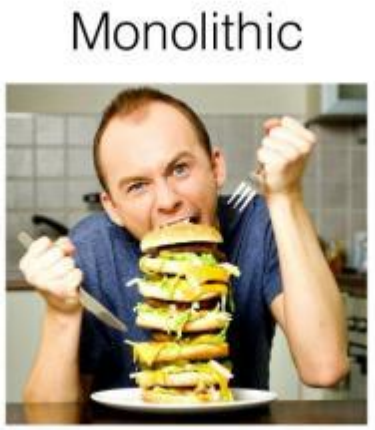
What is a Microservice?

- A service that can be independently deployed



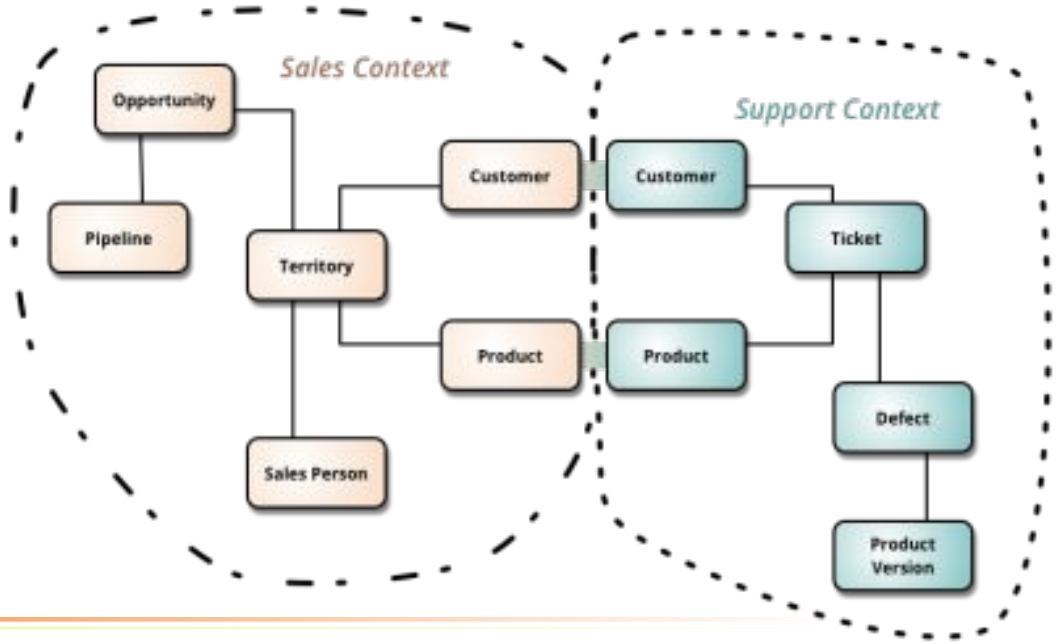
How big is a Microservice?

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



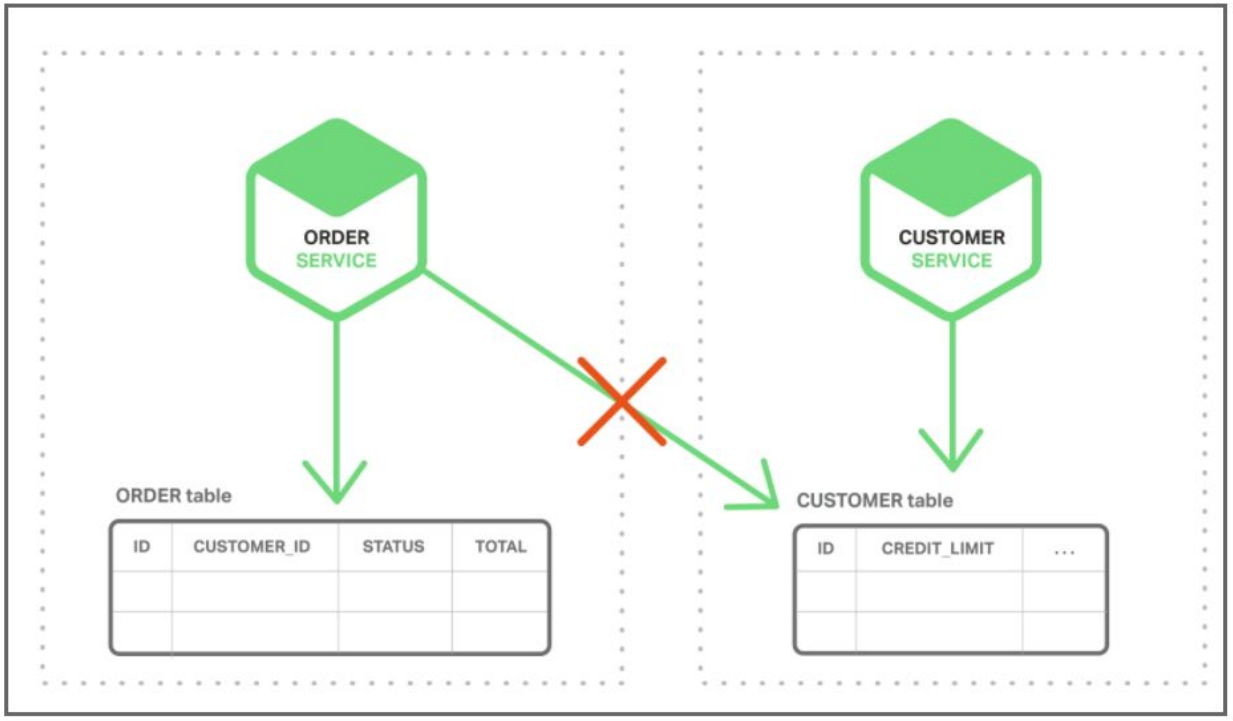
Where is the data for a Microservice?

- 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

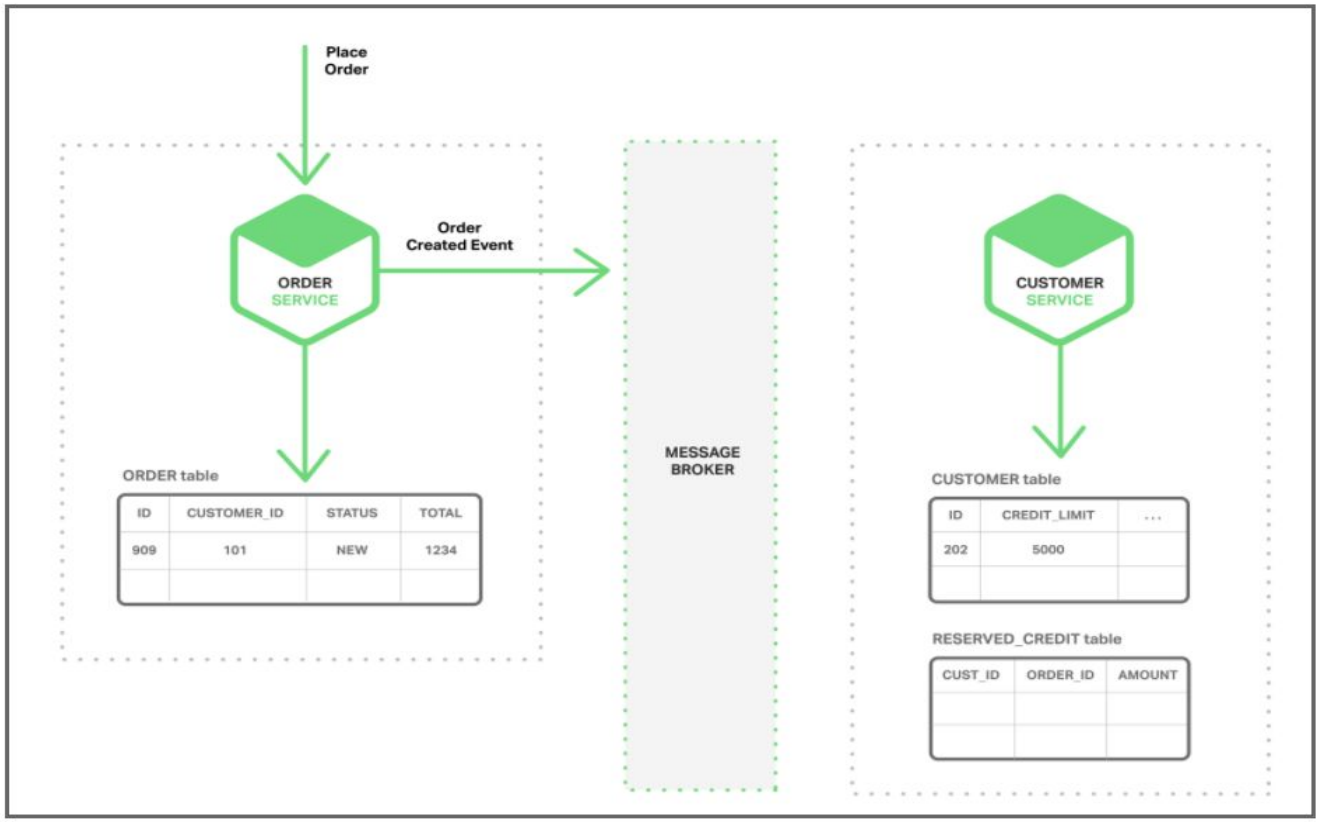


Event Driven Data Management

- 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

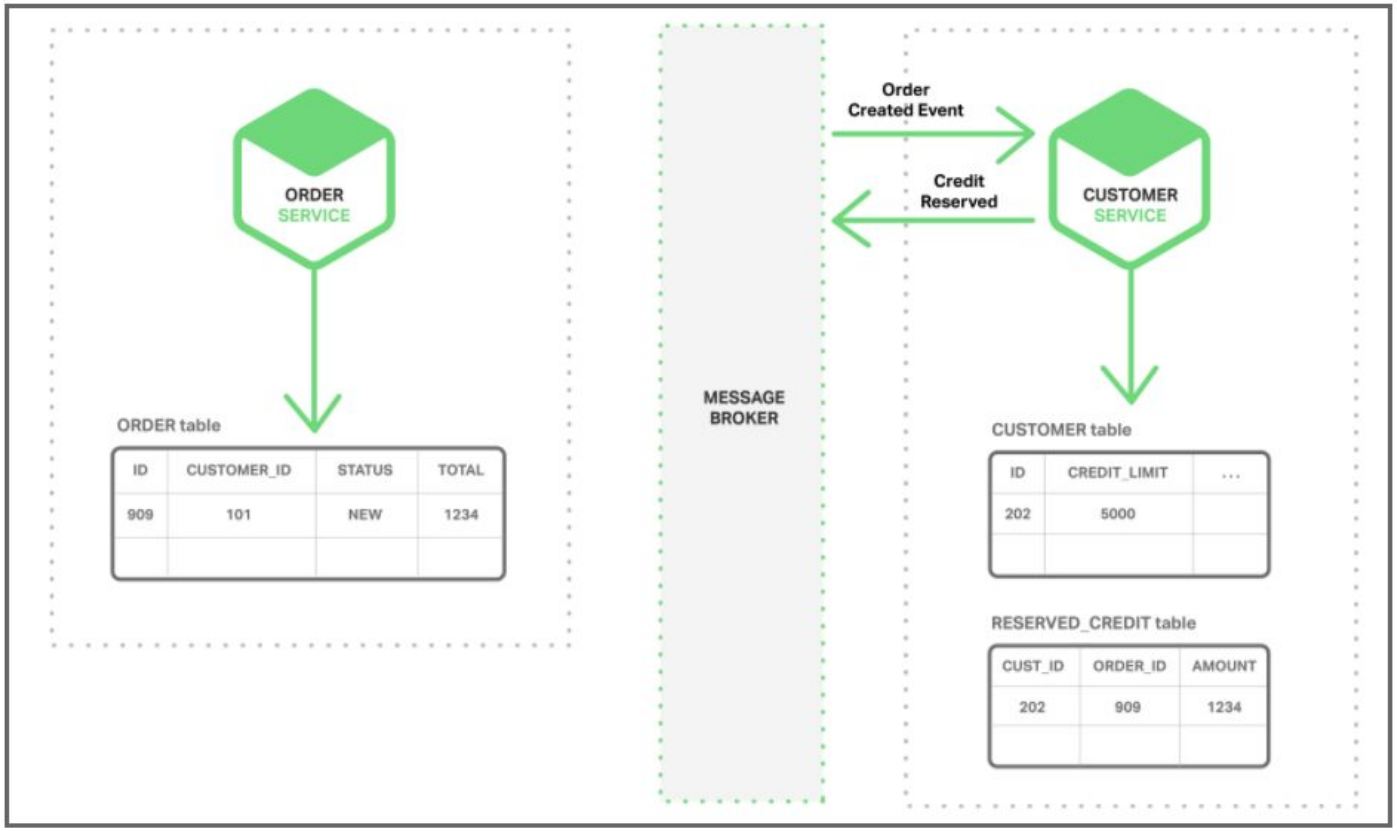
Event Driven Data Management

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



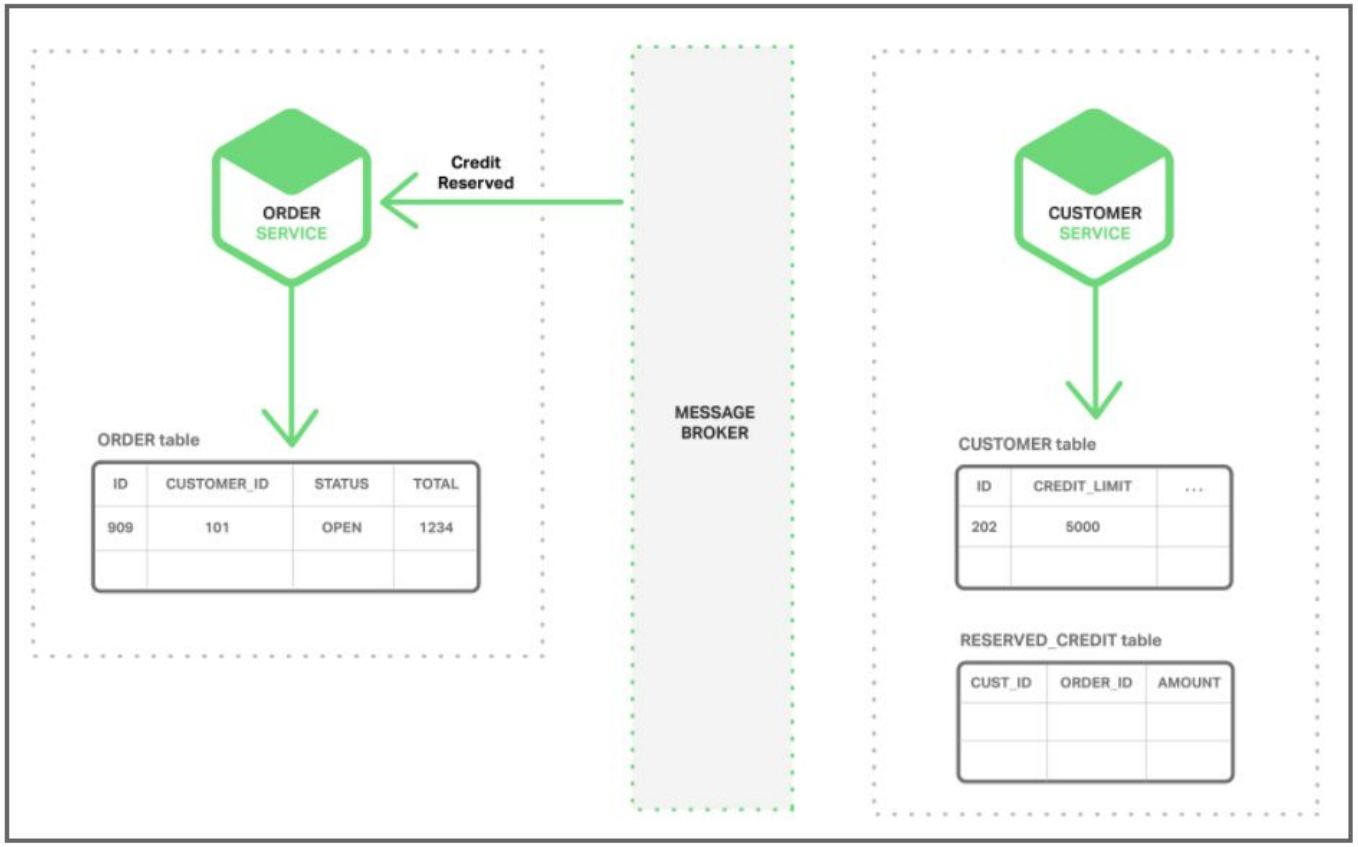
Event Driven Data Management

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



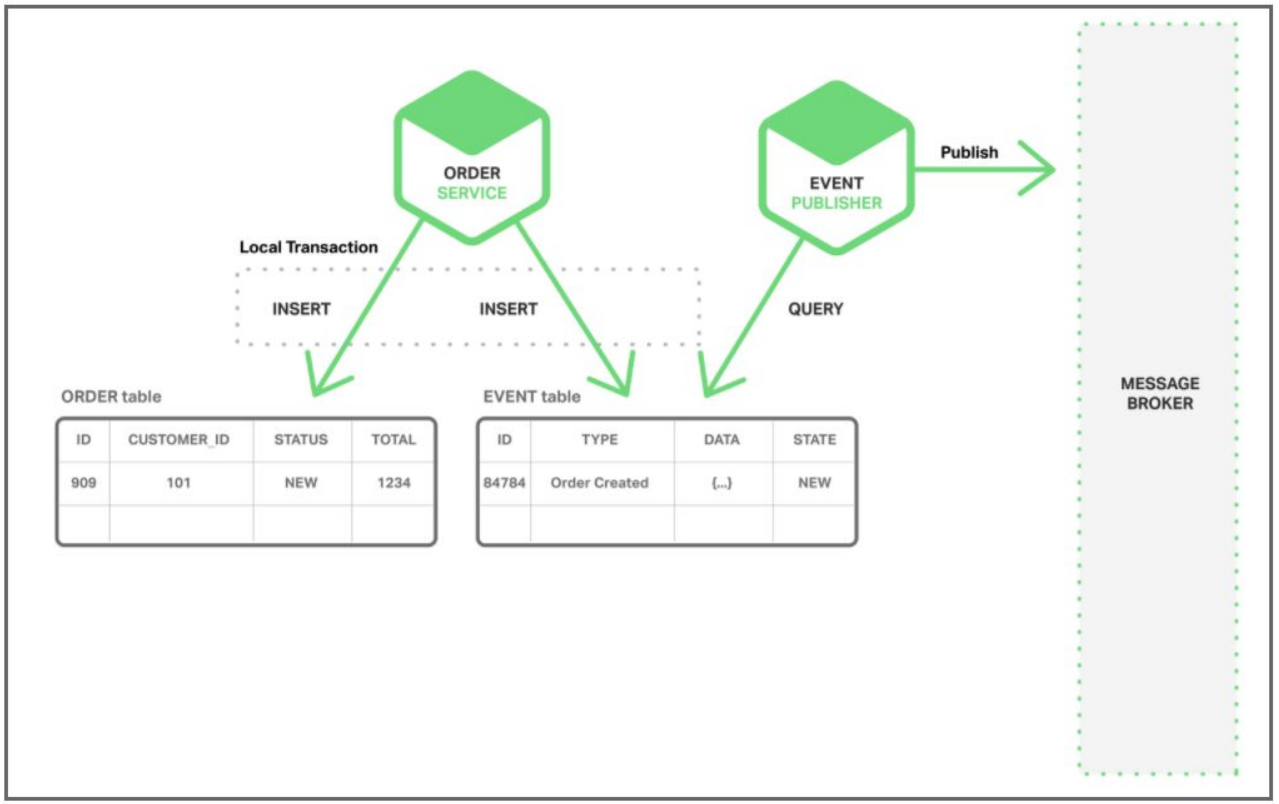
Event Driven Data Management

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

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

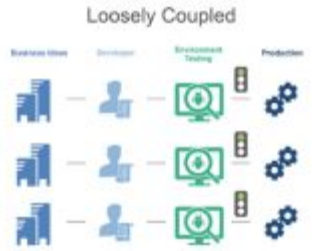
"Organizations with these types of service-oriented architectures, such as Google and Amazon, have incredible flexibility and scalability. These organizations have tens of thousands of developers where small teams can still be incredibly productive."

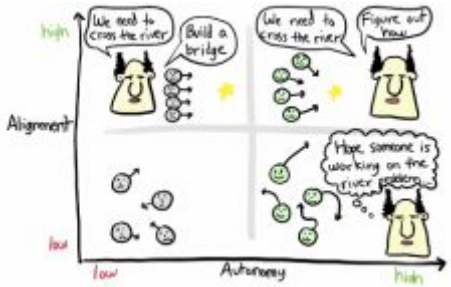
-Randy Shoup, former Engineering Director for Google App Engine

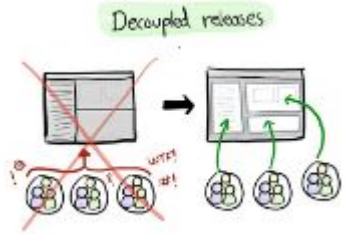
- This consists of having many small teams work independently, rapidly delivering customer value

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







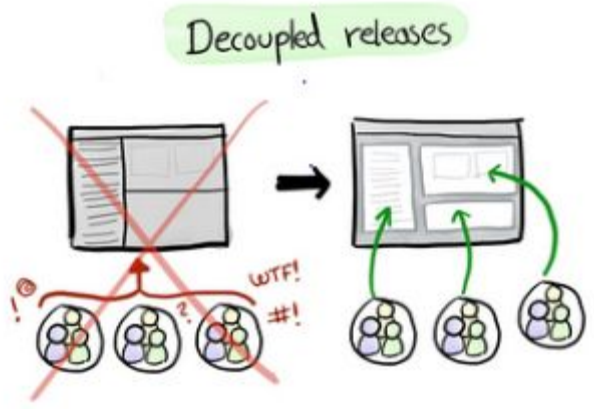
Starting and Scaling DevOps in the Enterprise by Gary Gruver
https://www.amazon.com/dp/B01M332BN2/ref=dp-kindle-redirect?_encoding=UTF8&btkr=1

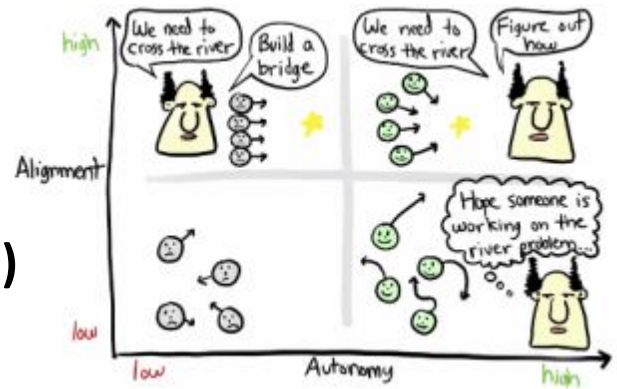
Why Small Team Size? Scaling the Organization

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)



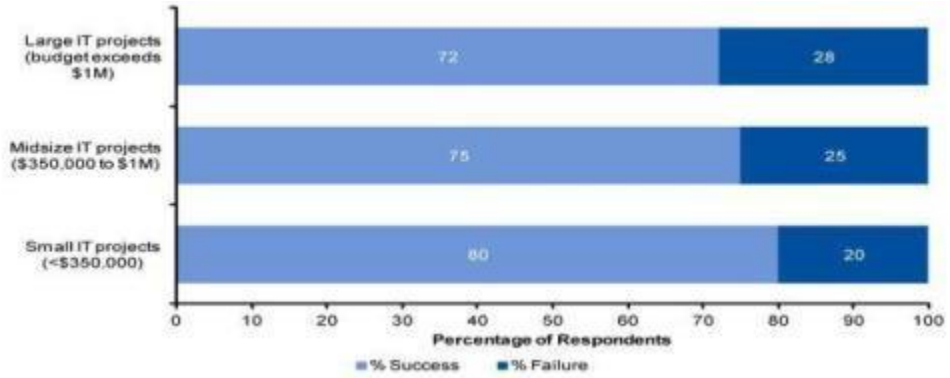


Science behind why two-pizza team works http://blog.idonethis.com/two-pizza-team/
ITRevolution, Conways Law by Gene Kim http://itrevolution.com/conways-law/

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.

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

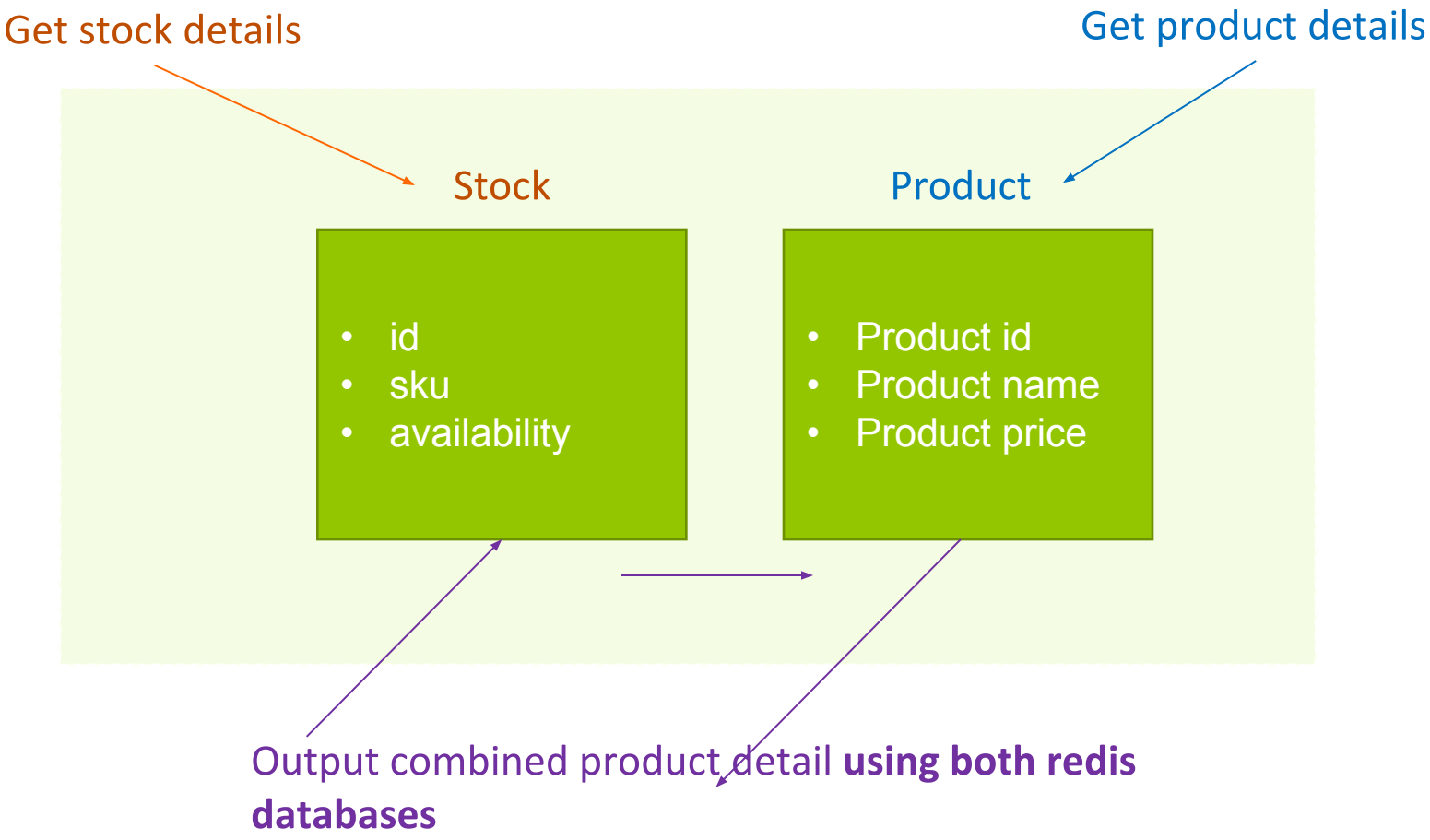
High performing teams are breaking away from the pack and the status quo of even three years ago is a dangerous assumption currently.

Microservices Patterns

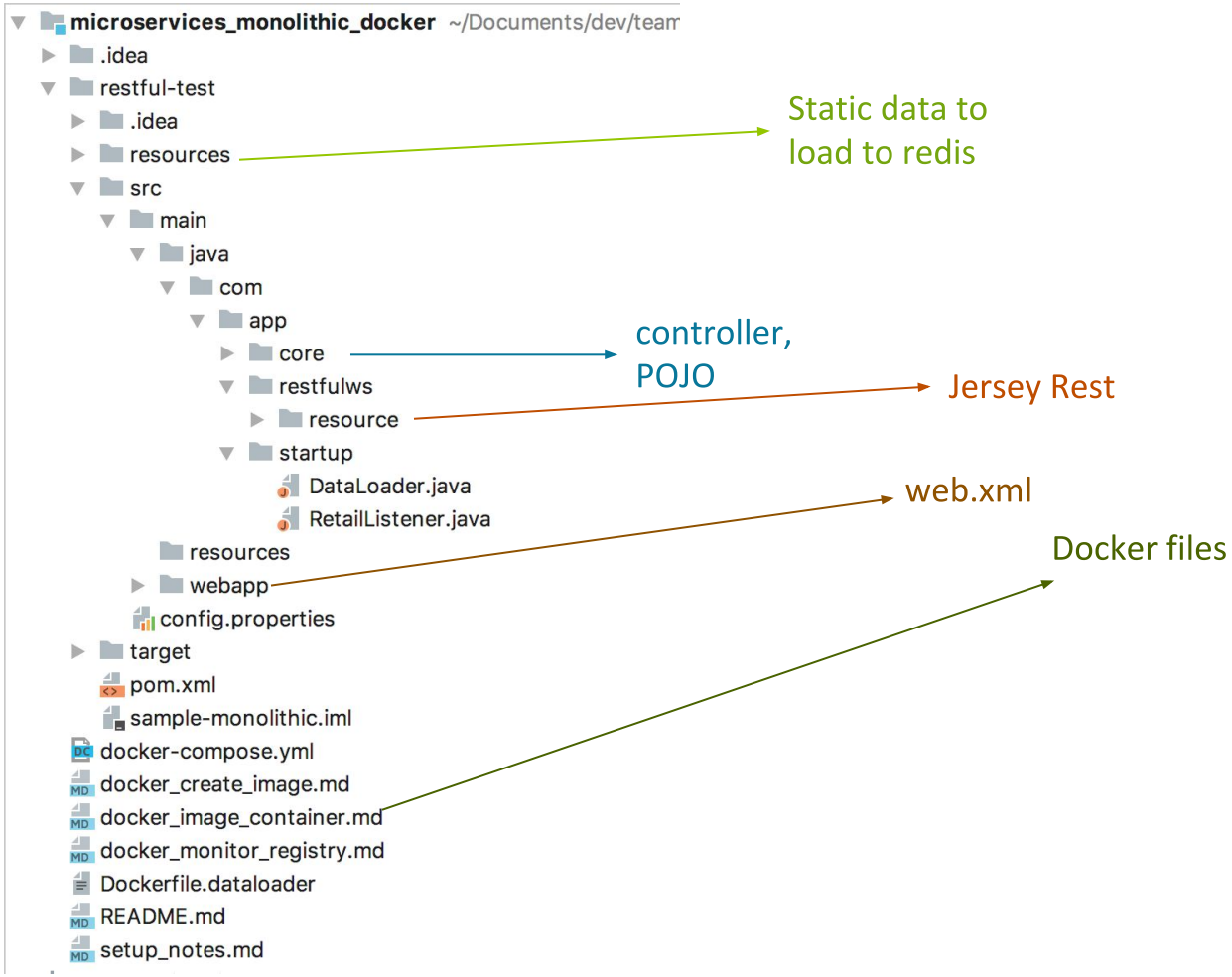
Before diving into Microservices Patterns, let’s walk through a Sample Monolithic Application.

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

Sample Monolithic App in Java



Project Structure



https://github.com/shekhar2010us/microservices_monolithic_docker/blob/master/README.md

Setup – Running Java Monolithic

- Login to virtual machine
- Install docker, maven and git
- Pull code
- Build code
- Build docker image
- Run docker-compose

https://github.com/shekhar2010us/microservices_monolithic_docker/blob/master/setup_notes.md

Writing Monolithic - Java

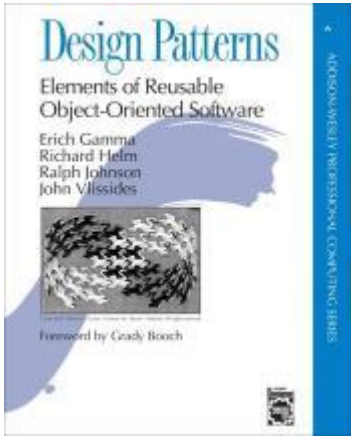
CLASSROOM WORK 101 (45 minutes)

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

https://github.com/shekhar2010us/microservices_monolithic_docker

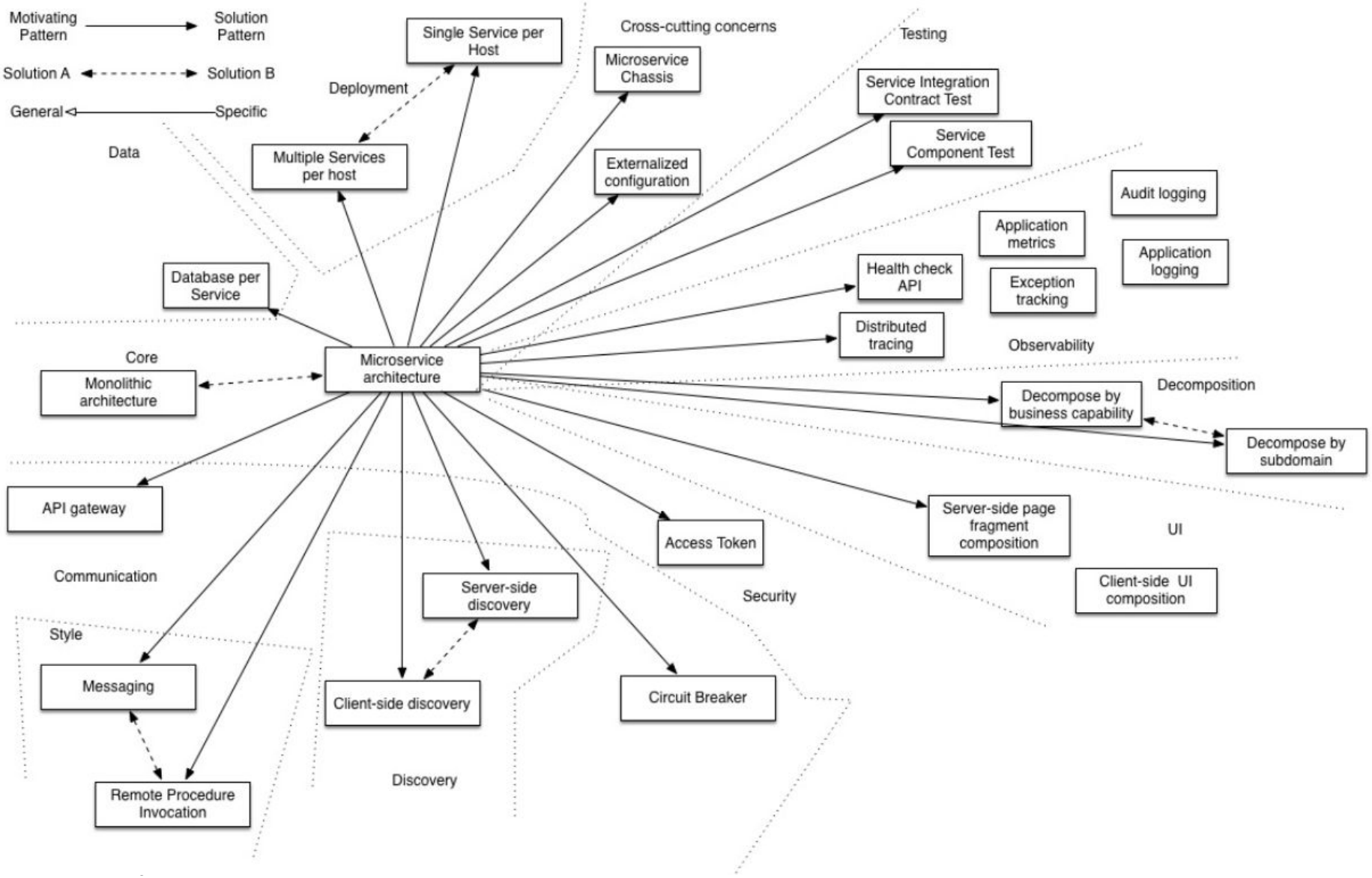
Patterns

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

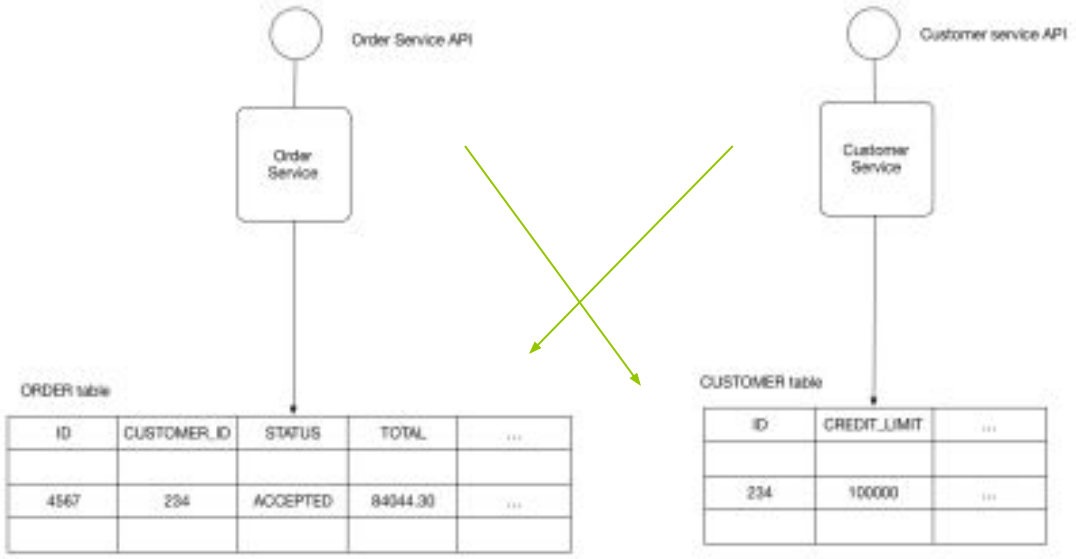
- Shared database
- Database per Service
- Saga
- Event Sourcing
- CQRS (Command Query Responsibility Segregation)
- API Composition

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.



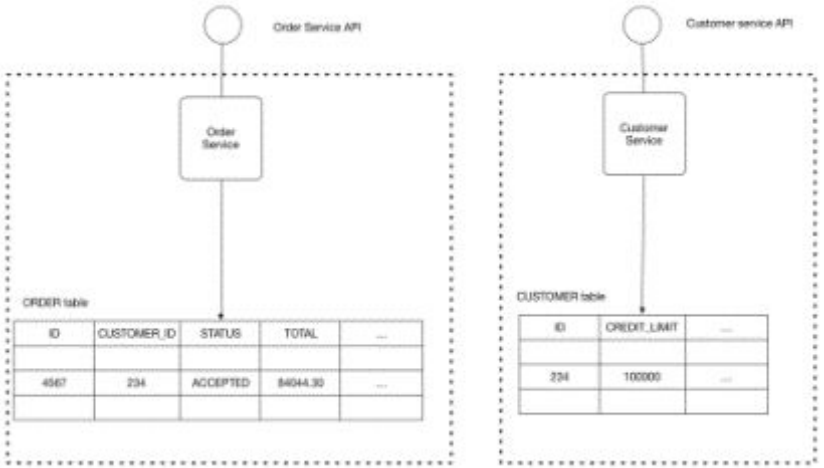
Database Per Service

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

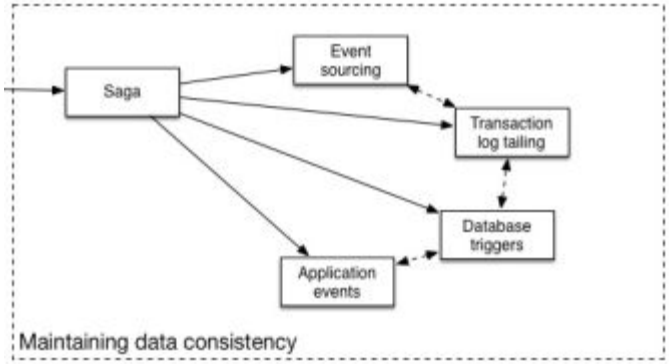
- ❑ For consistency; may need to implement capability to roll back (**Saga** pattern)
- ❑ Interaction is achieved by message/event brokers

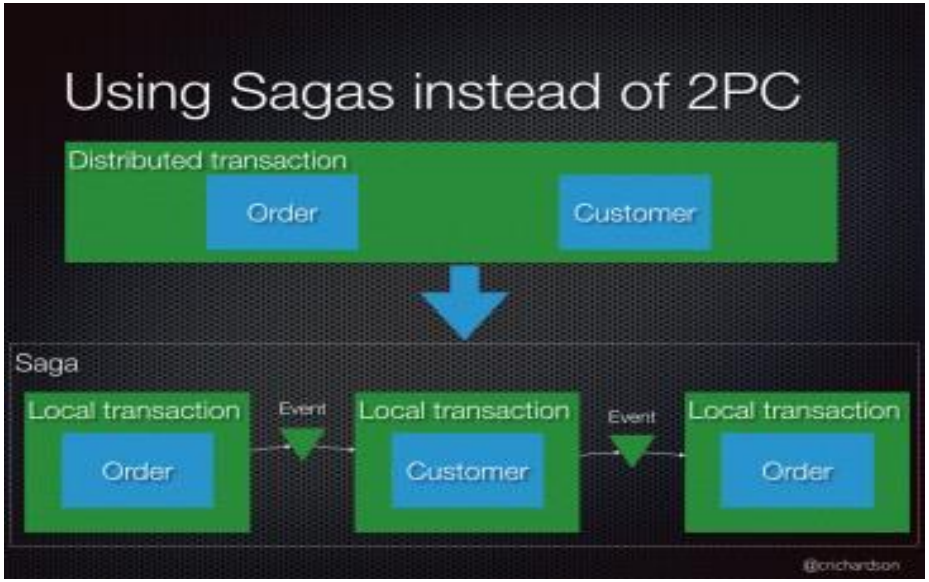


Saga Pattern – For Data Consistency

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

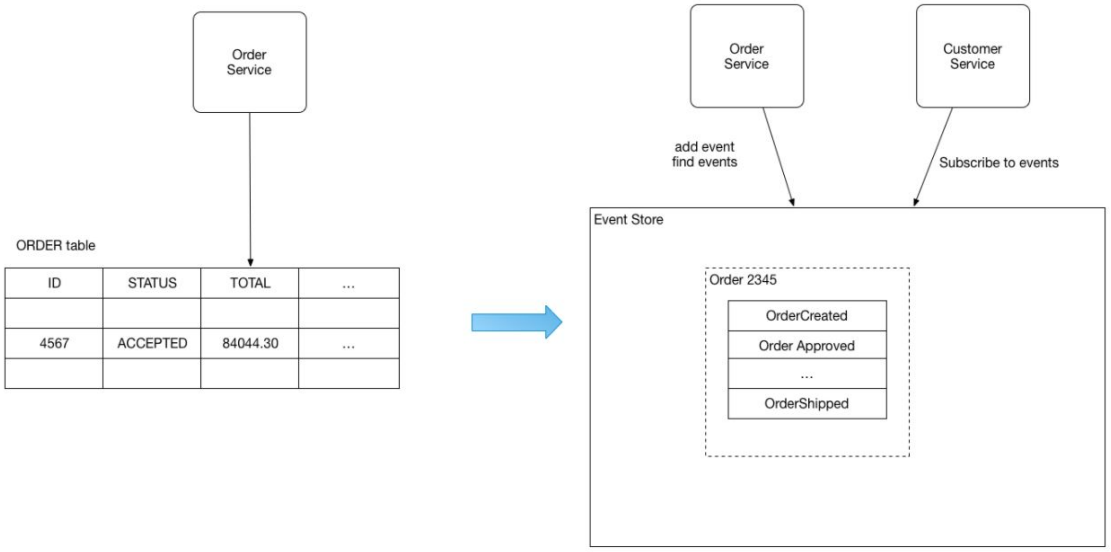
- Persists the state of a business entity as a sequence of state-changing events – in Event Store
- The store has APIs for adding, retrieving an entity's events and subscribe to events.
- The store is like a message broker; A new event is delivered to all interested subscribers.
- E.g. the Order is saved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

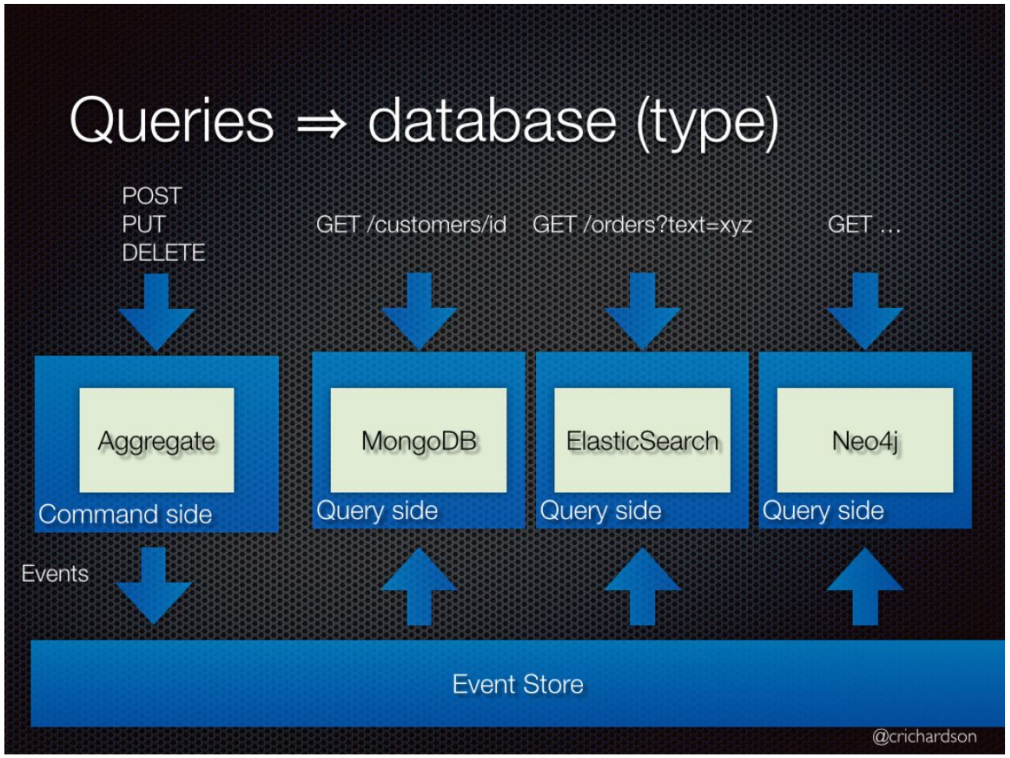


CQRS – Command Query Responsibility Segregation

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 datastore (applications that require synchronous processing and immediate results.)
- Two models talking to different datastore (Eventual Consistency)

API Composition

Perform queries in Microservices architecture

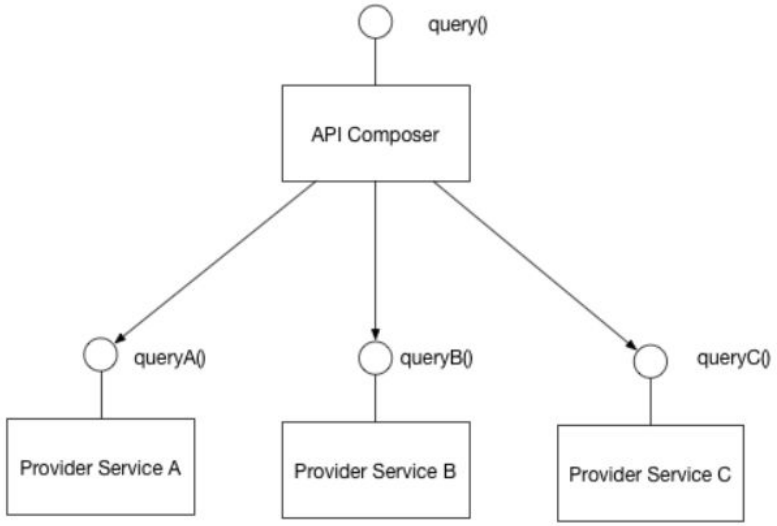
- Implement a query by defining an *API Composer*, which invoking the services that own the data and performs an in-memory join of the results.

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.



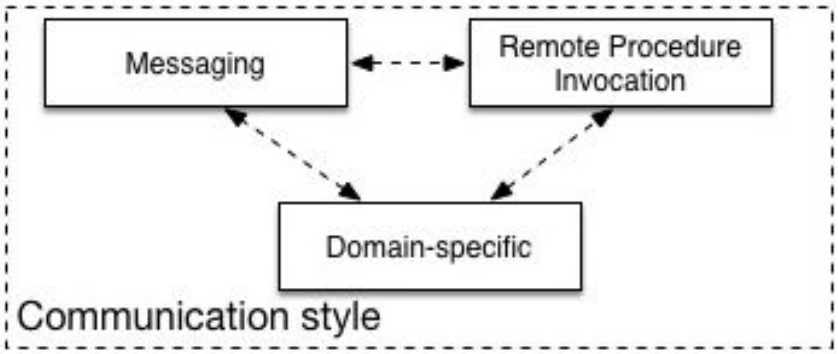
Pattern - Categories

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

Communication Style

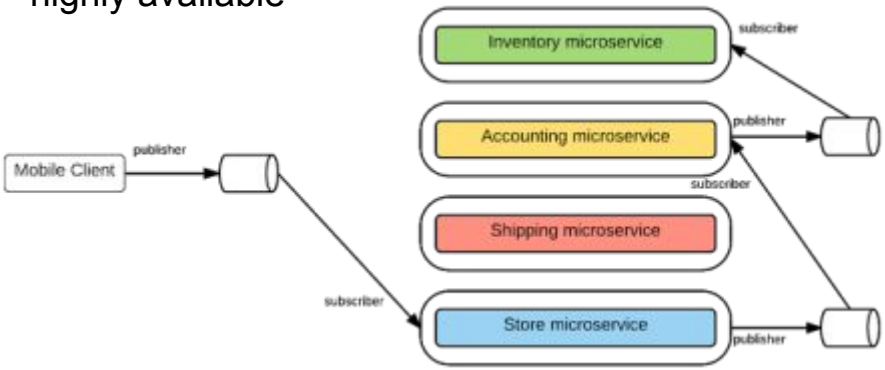
Messaging

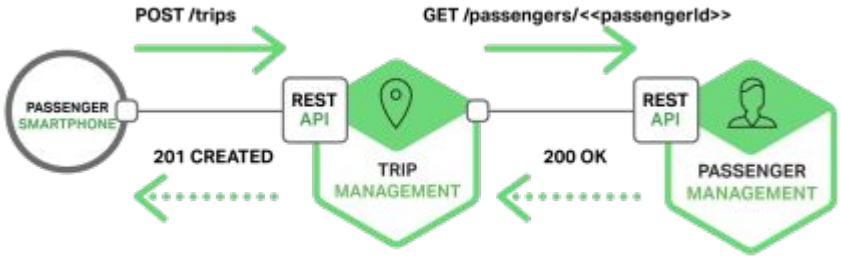
- Services communicate by sending asynchronous messages
- Decouples client & service
- Must also install and maintain a broker (Kafka, RabbitMQ, etc)
- Complex and should be highly available



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 re-deploy 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
- If multiple services instances are deployed in the same process then its difficult to monitor the resource consumption of each service instance. It's 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:

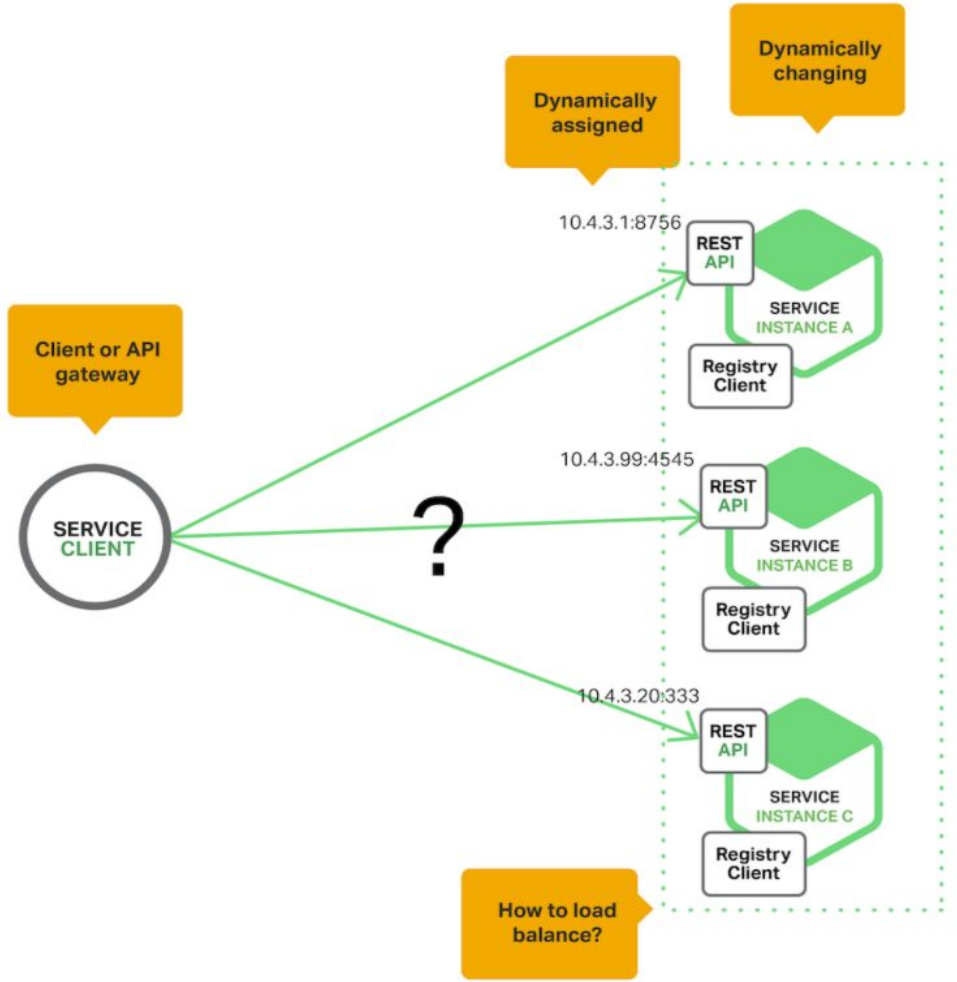
- Has far more constraints than a VM/Container-based infrastructure. E.g., language support.
- Only suitable for deploying stateless applications, not for long running stateful or message broker.
- Risk of high latency - the time it takes for the infrastructure to provision an instance and for the function 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.

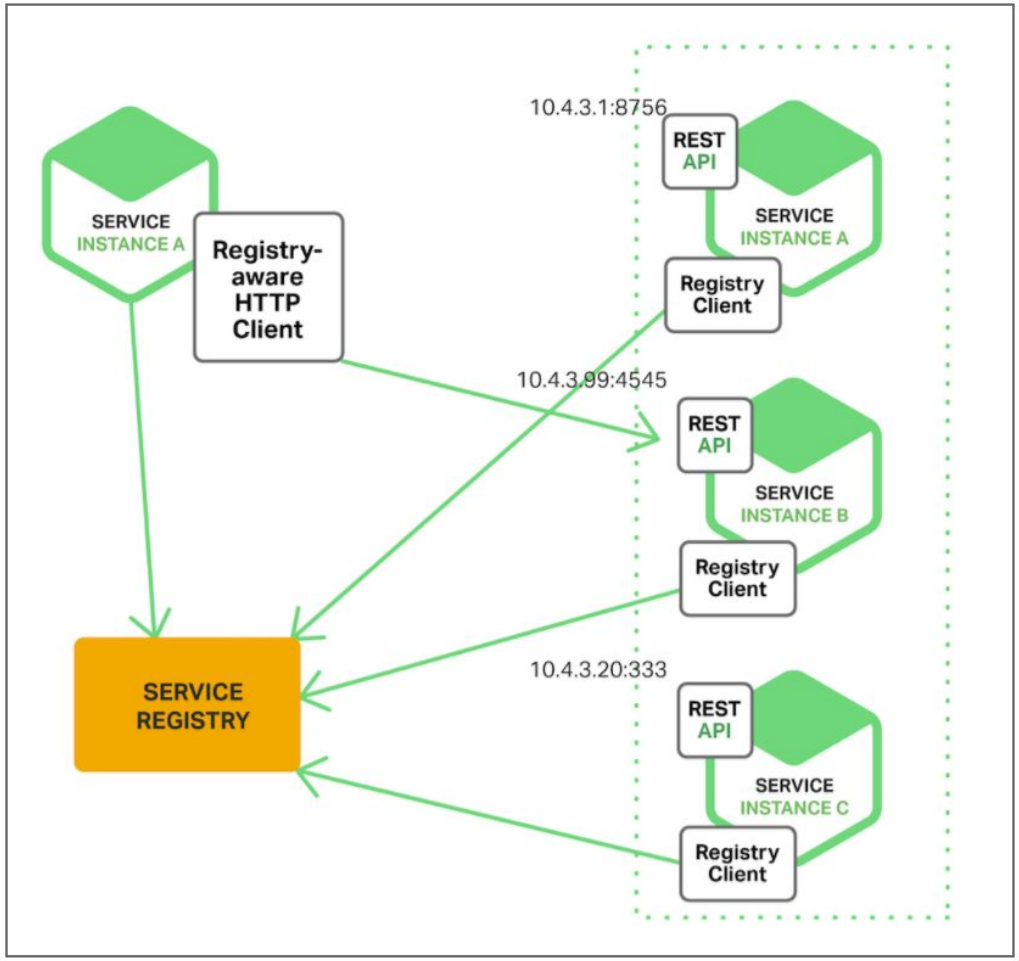
  E.g. Netflix Eureka, consul, Zookeeper

  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.

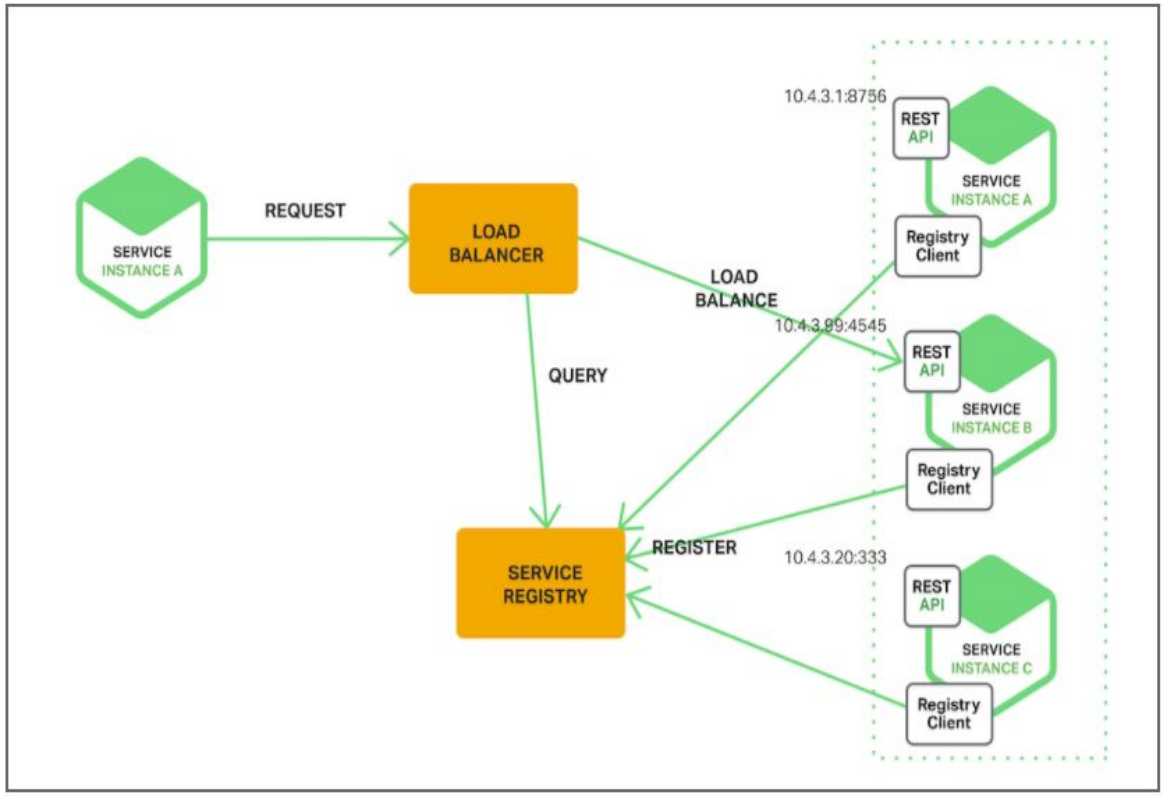
E.g. Netflix OSS



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 Load Balancer



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:

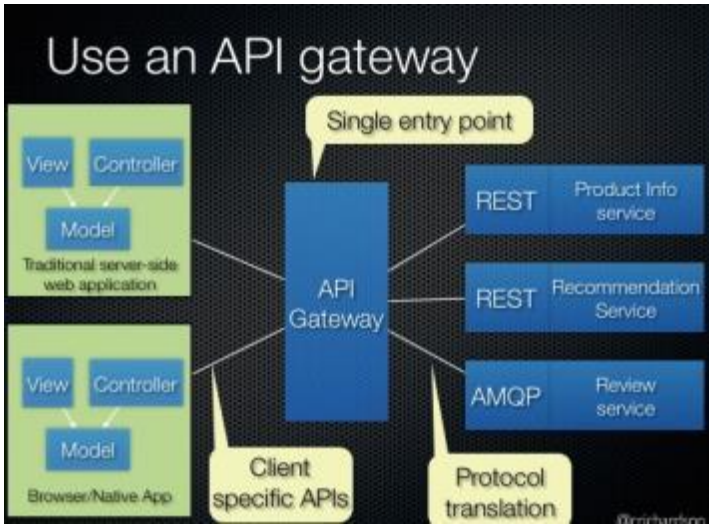
- Insulates the clients from how the application is partitioned into microservices
- Insulates the clients from the problem of determining the locations of service instances
- Provides the optimal API for each client

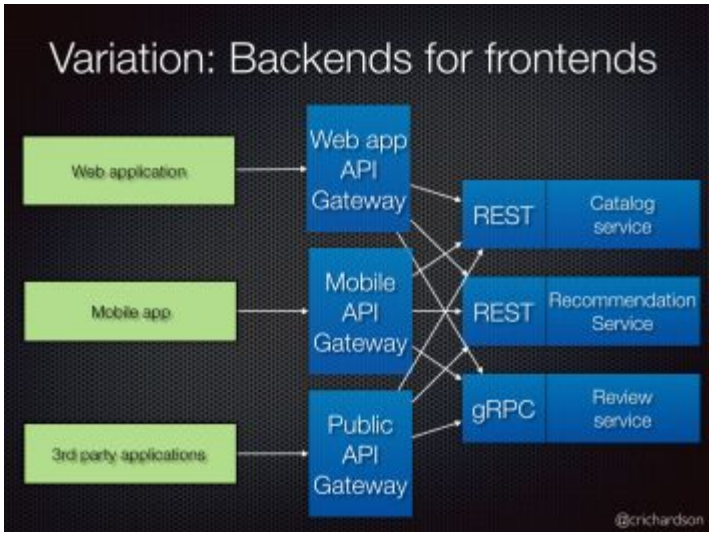
Drawbacks:

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





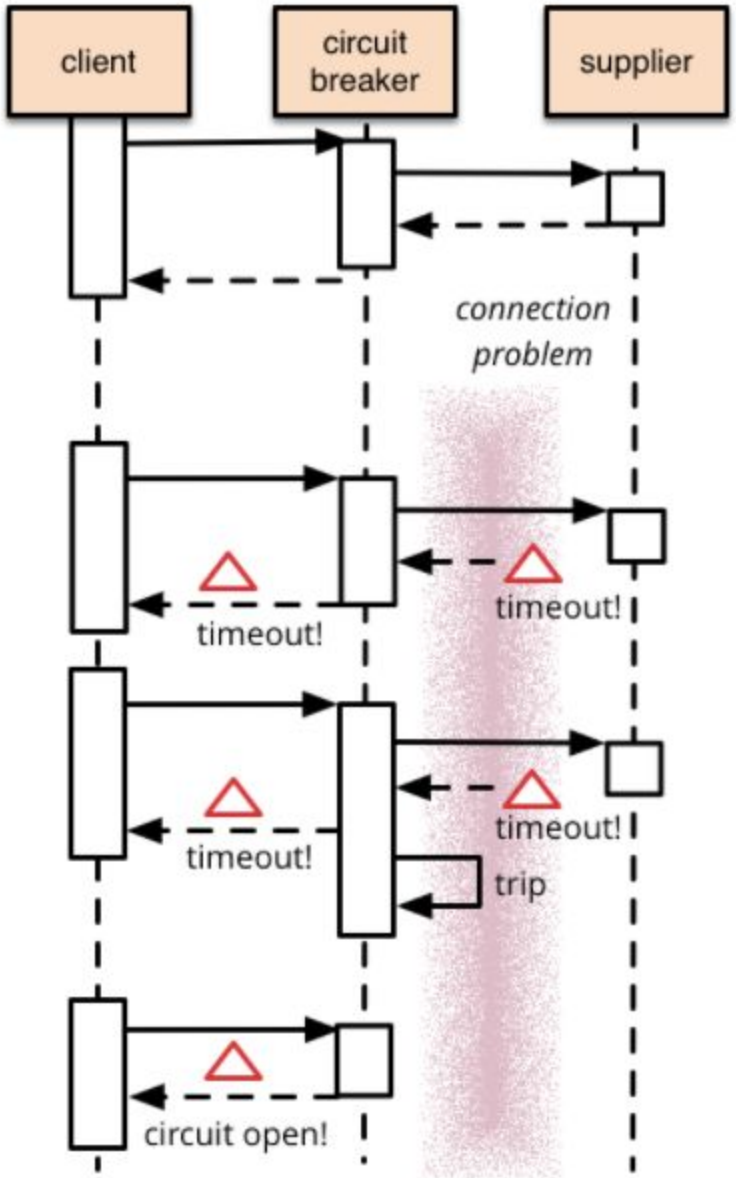
Pattern - Categories

- **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

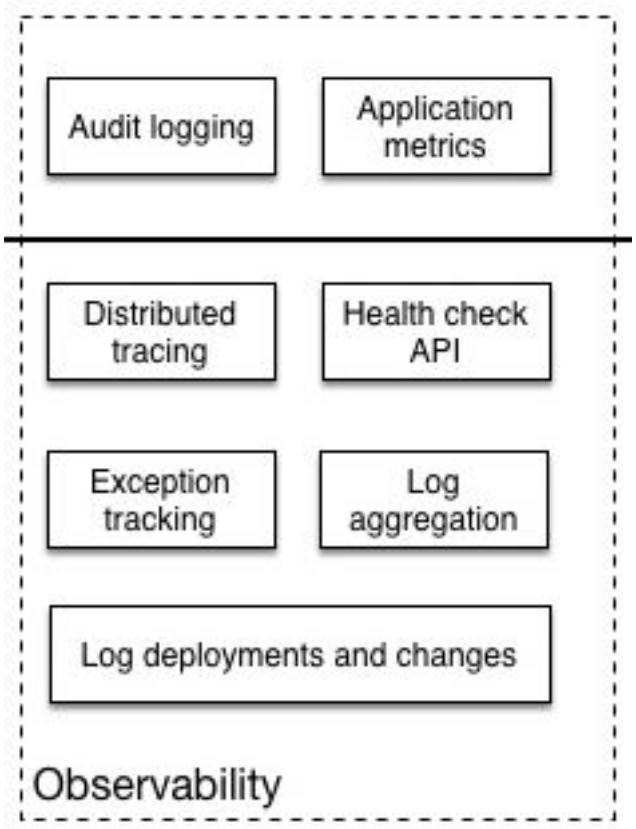
Observability (Metrics)

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.



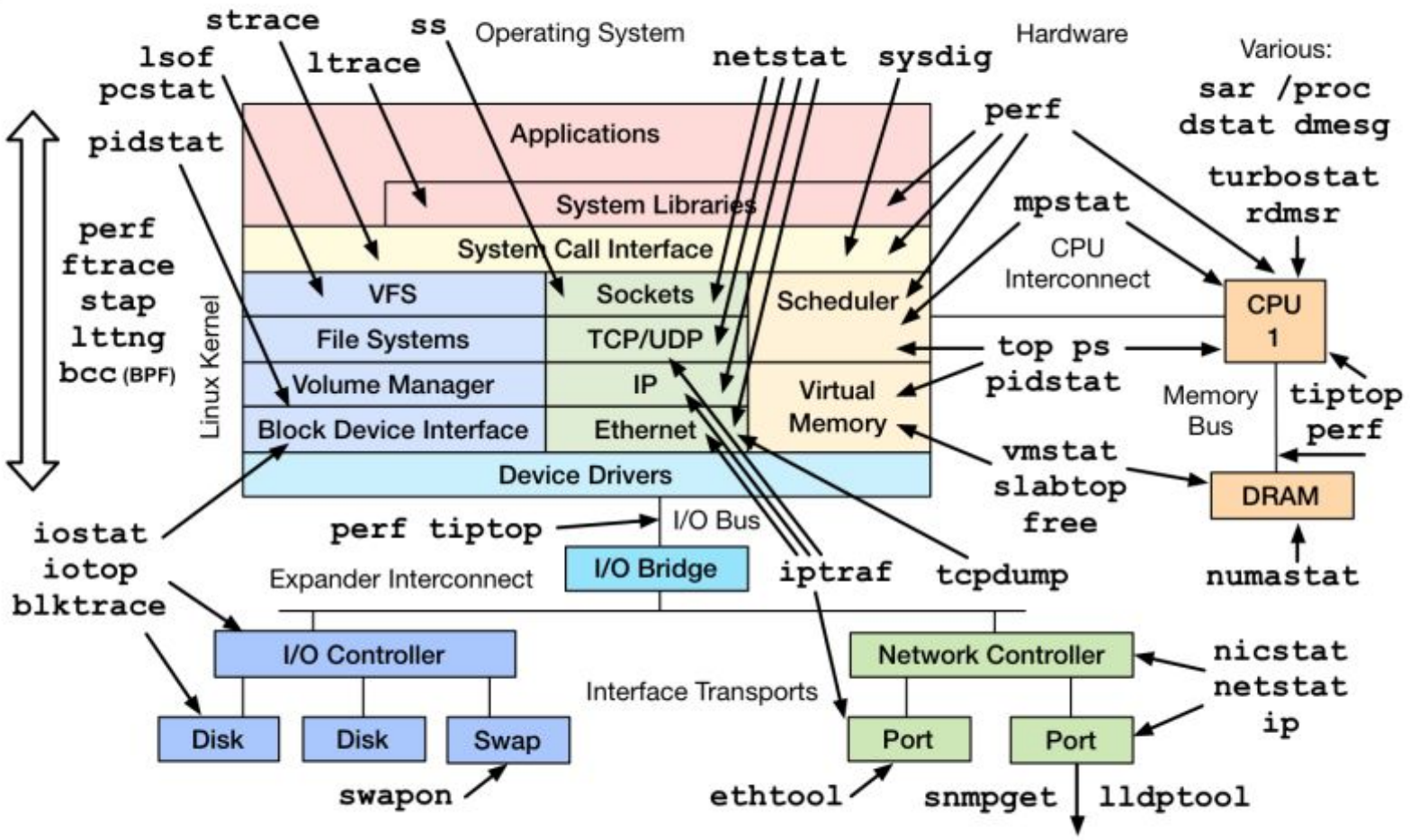
Application Metrics

- 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**

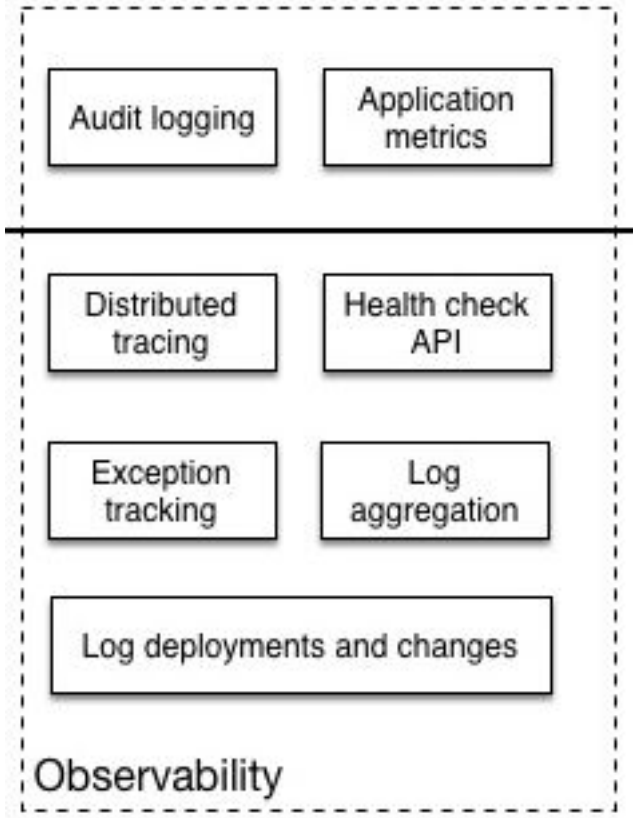
Note:

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

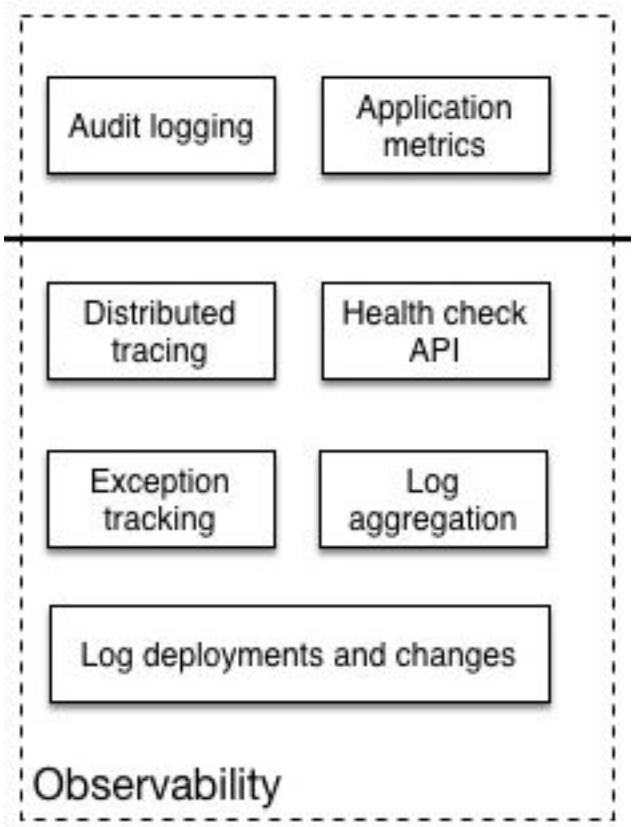
Common Distributed Tracing functionality

- 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

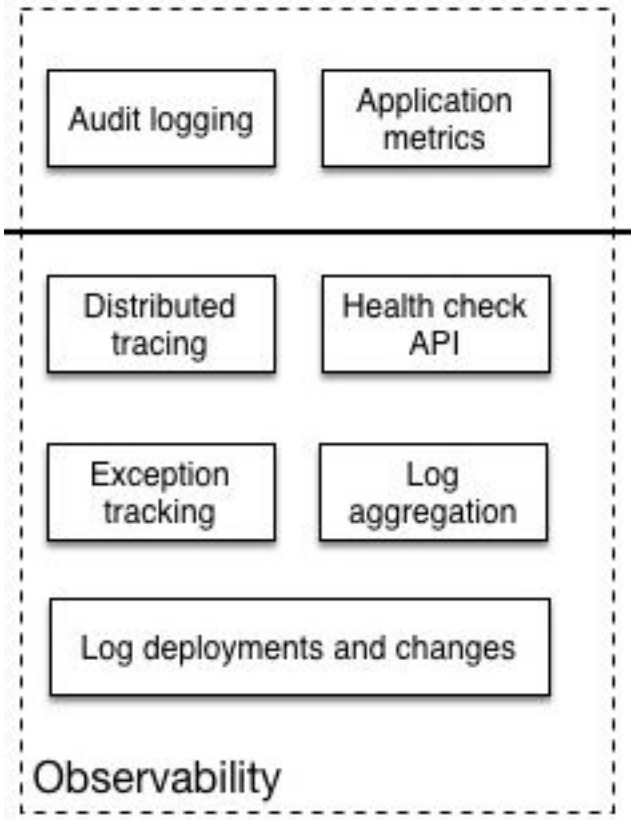
- Track changes as they are opportunities for failure
- For example, tracking deployments and changes so they can be easily correlated with issues later for faster resolution

*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

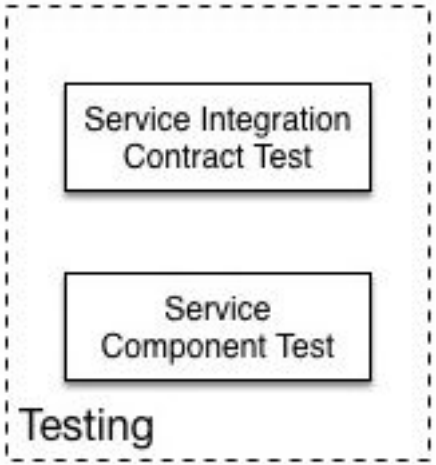
- Data Management
- Communication
- Deployment
- Discovery
- Observability
- Testing

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

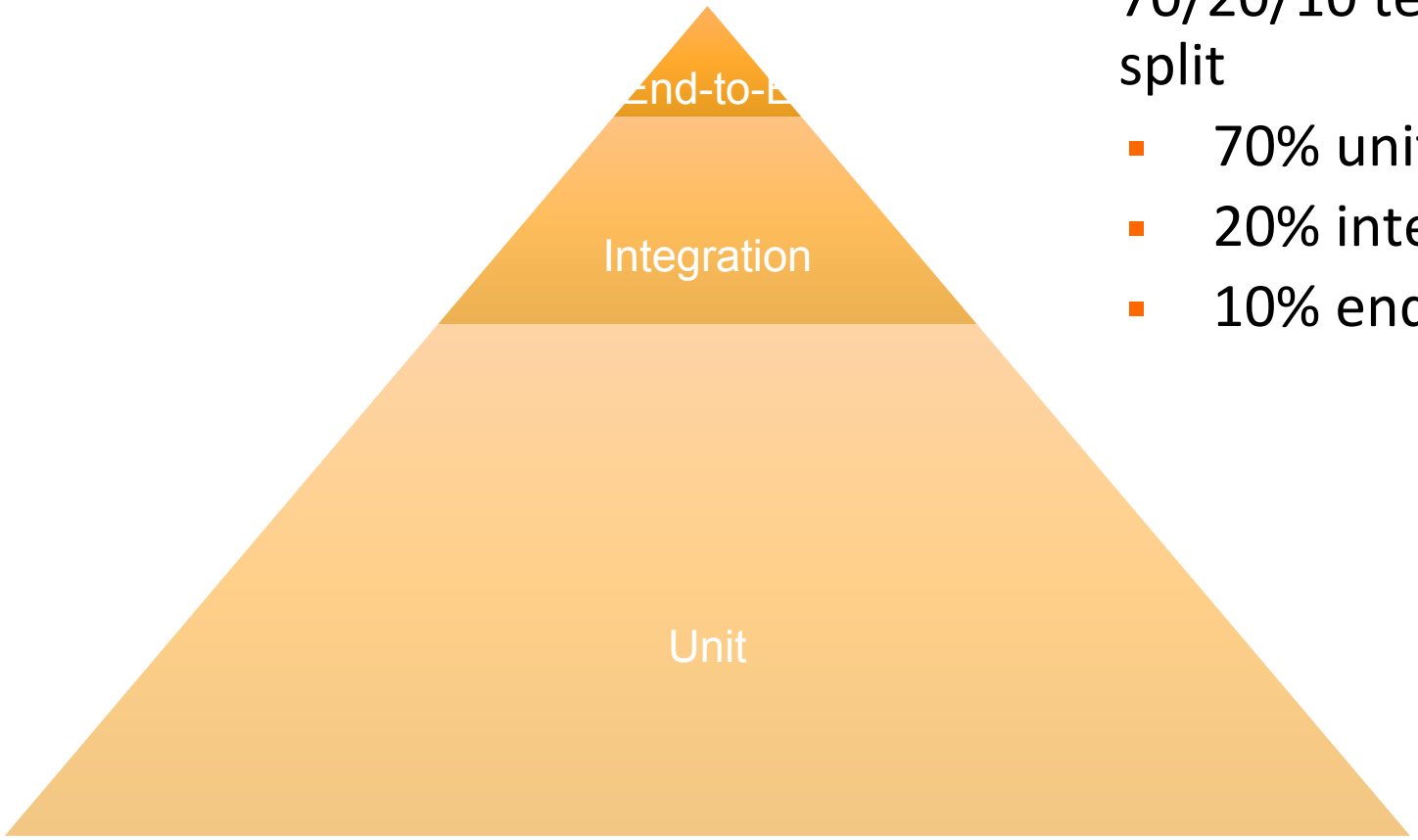
- Slow
- No isolated failures
- Simulates a real user

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



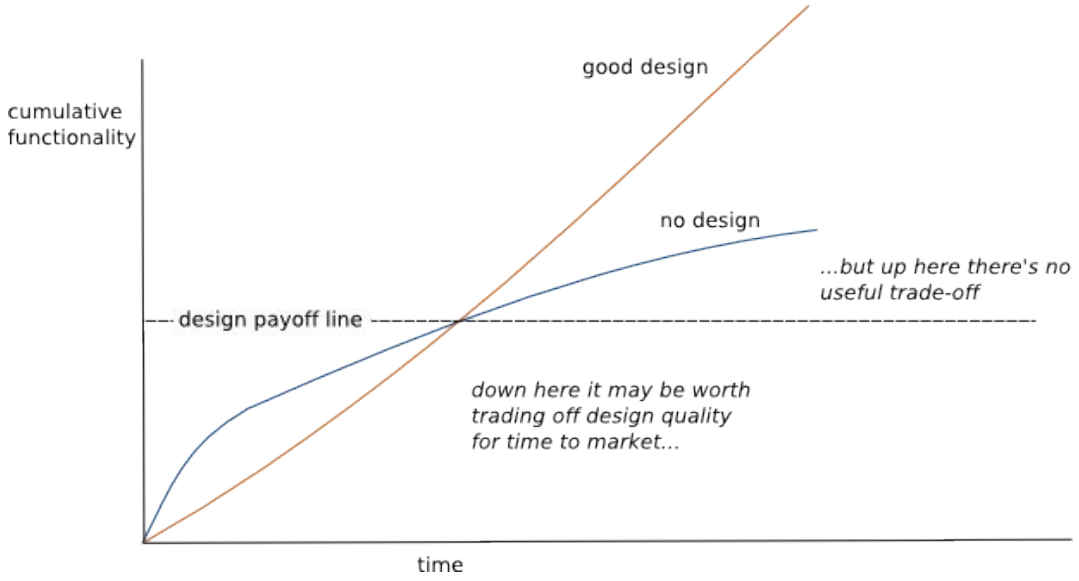
Google recommends a 70/20/10 test composition split

- 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 tracking 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)

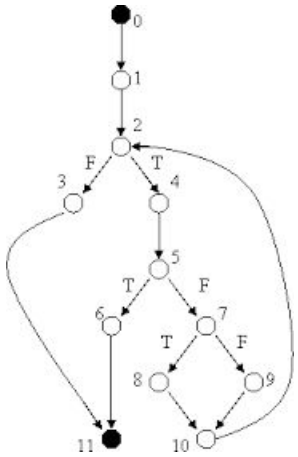


Static Code Metrics

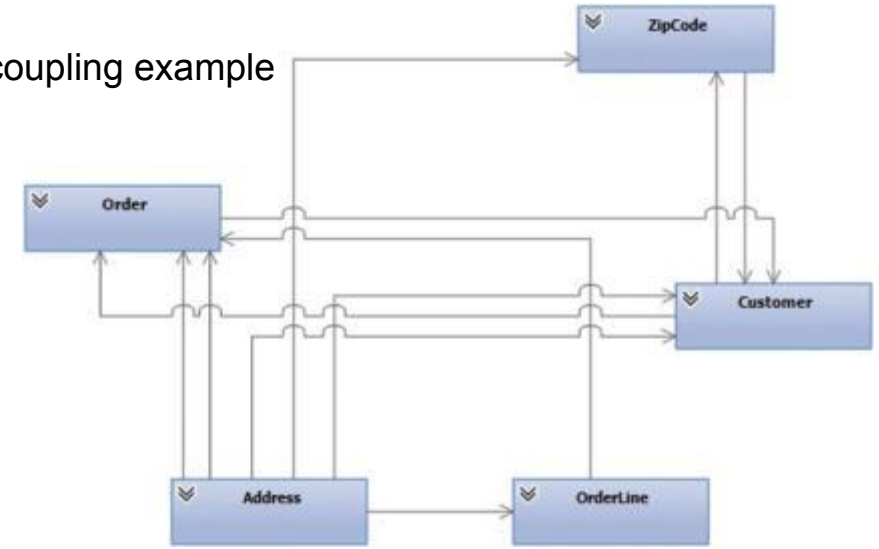
Coding Techniques to Improve Architecture Agility

- Code Complexity – recommended 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.

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

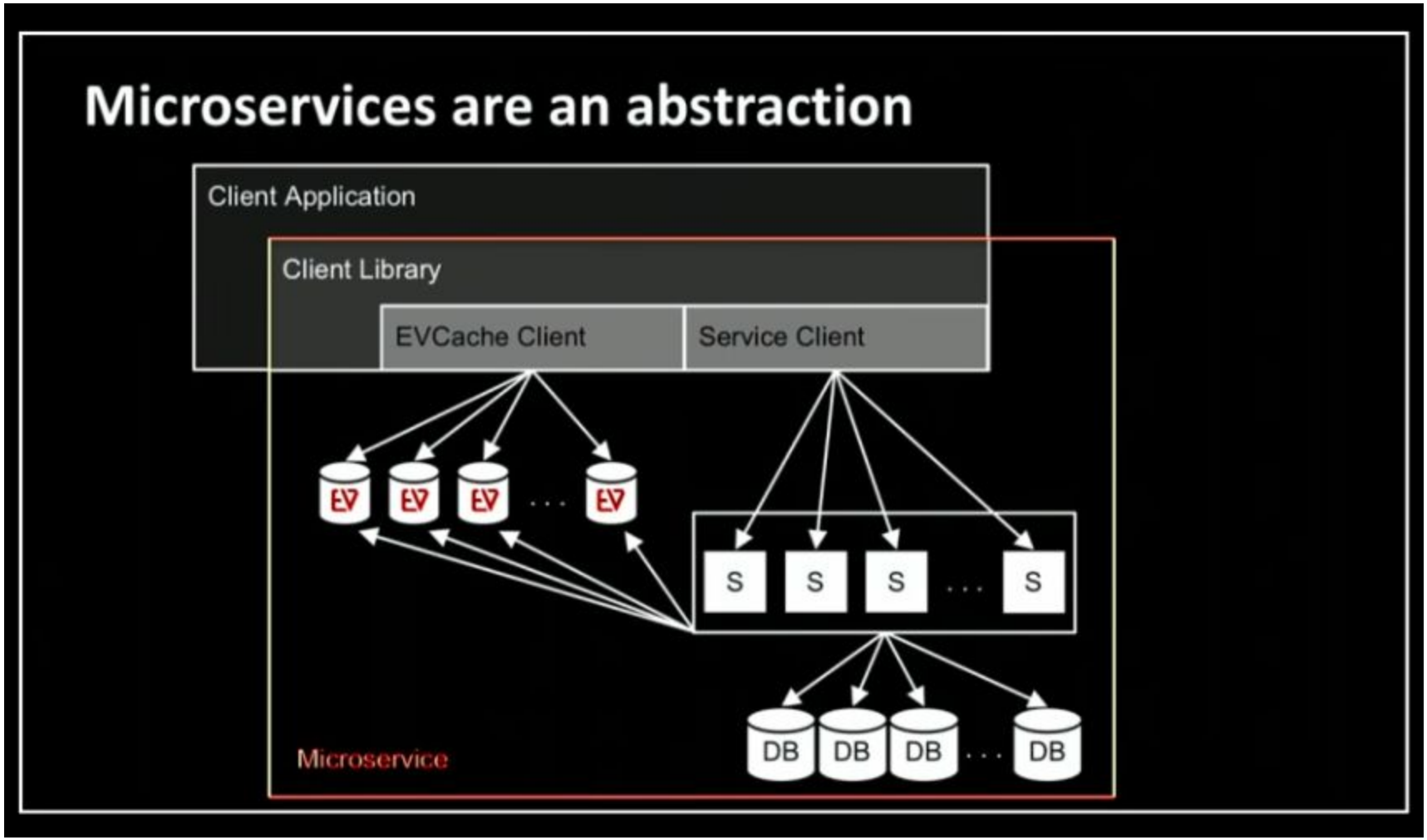
Code Stability
- Uptime >= 99.9 %
- Mean Time To Recovery <= 4 hours

Tests
- Unit Coverage >= 80%

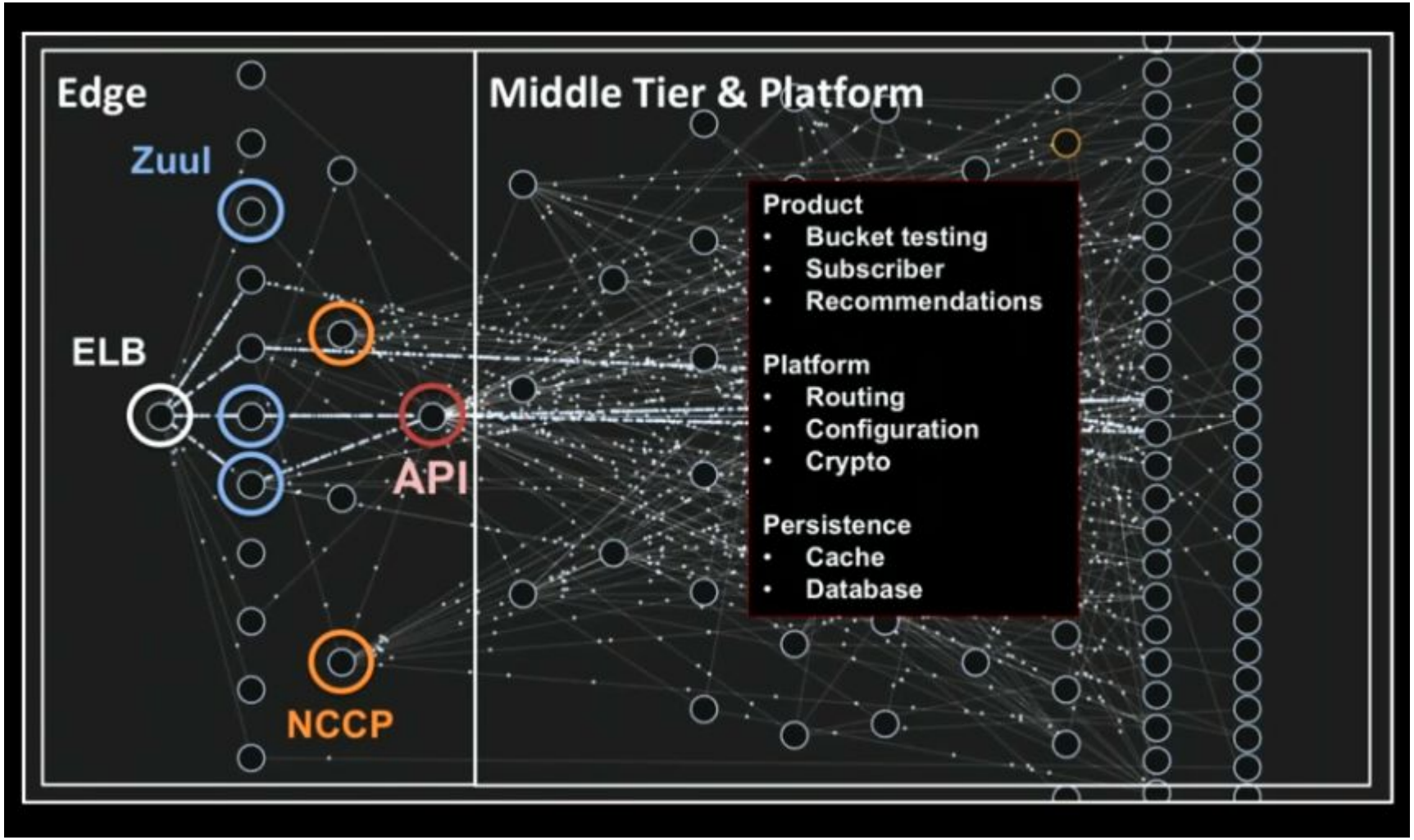
Microservices - Definition of Done

- 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



Microservices in Netflix



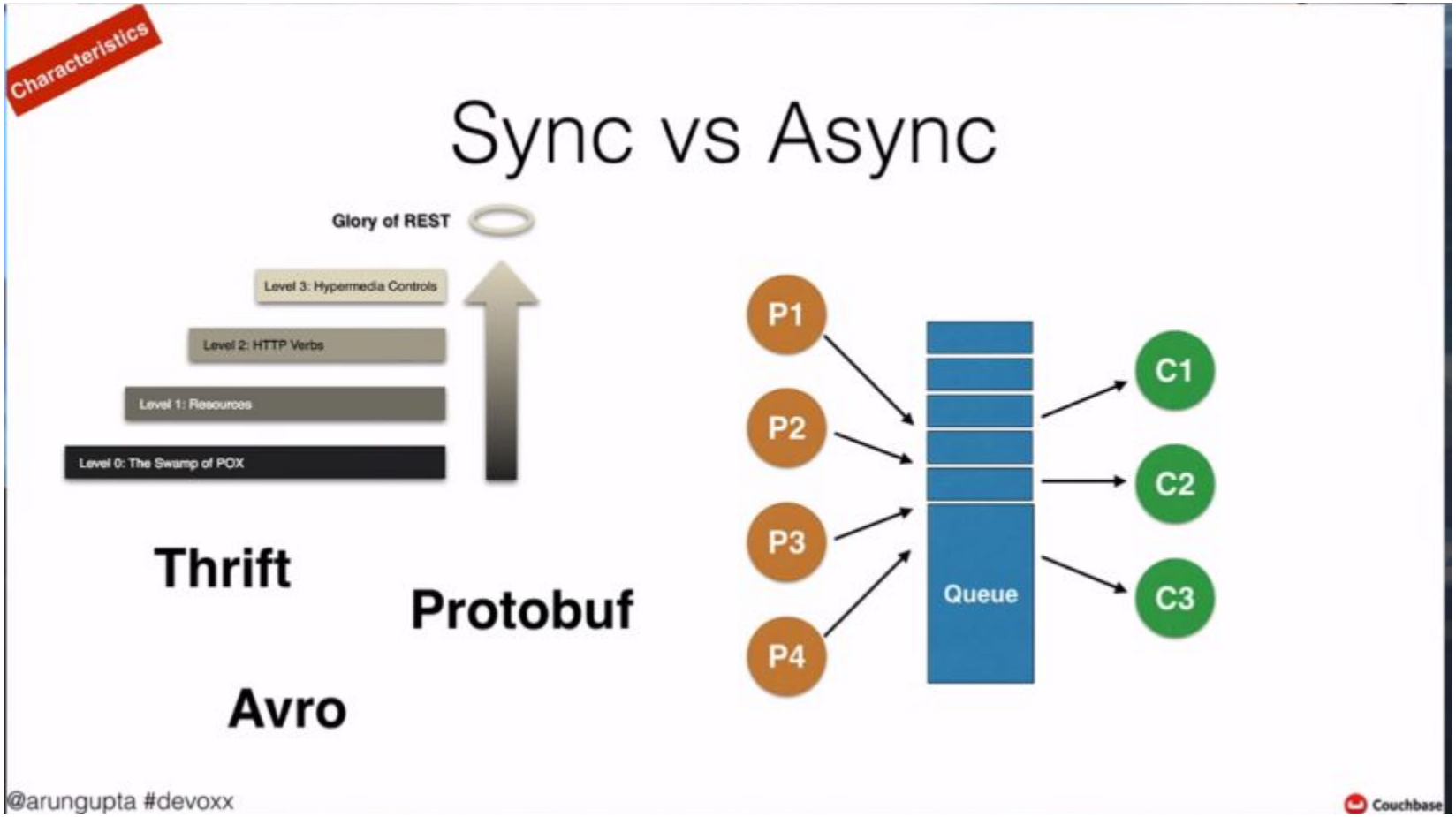
- **Microservices in Netflix**
  - **By Josh Evans (Director of Operational Engg.)**
  - **At Qcon, San Francisco, 2016**

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

Microservices in Netflix

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

Microservices in Couchbase

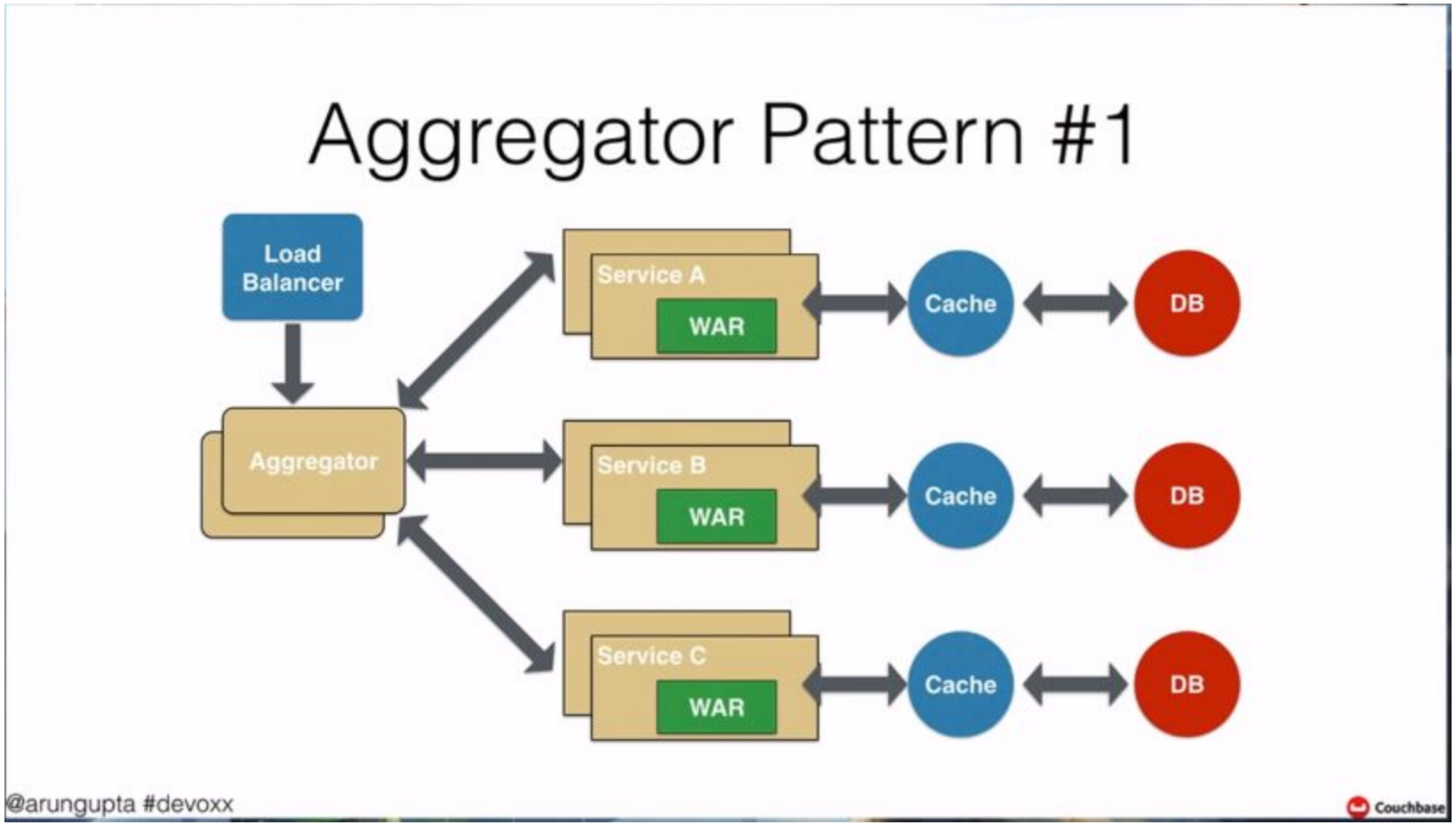


Microservices in Couchbase

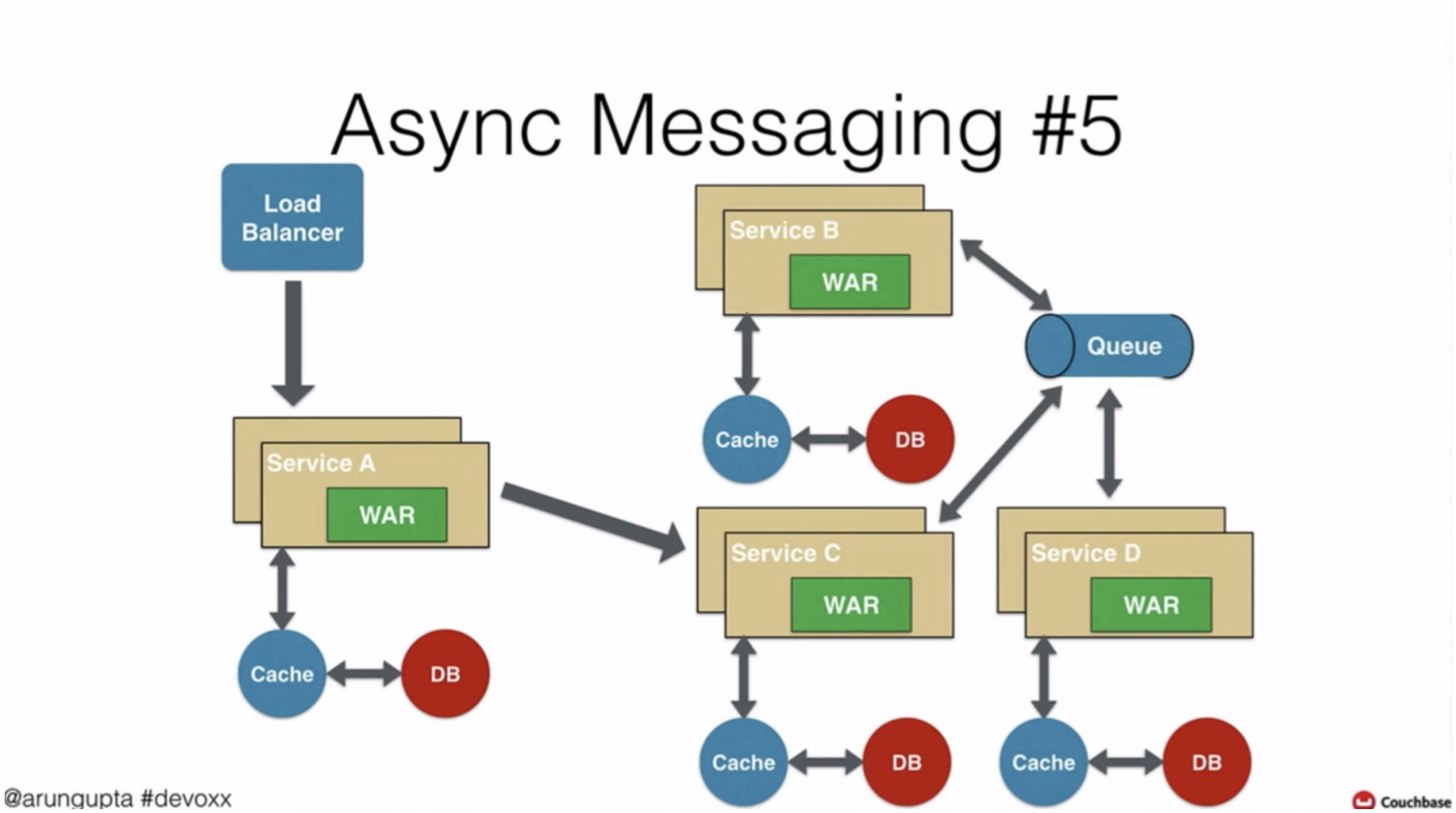
Strategies for decomposing

- 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 Couchbase (first half hour)**
  - **By Arun Gupta (VP Engg.)**
  - **At Devoxx, Belgium, 2015**

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

Part 2:
Microservices in Development

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
```

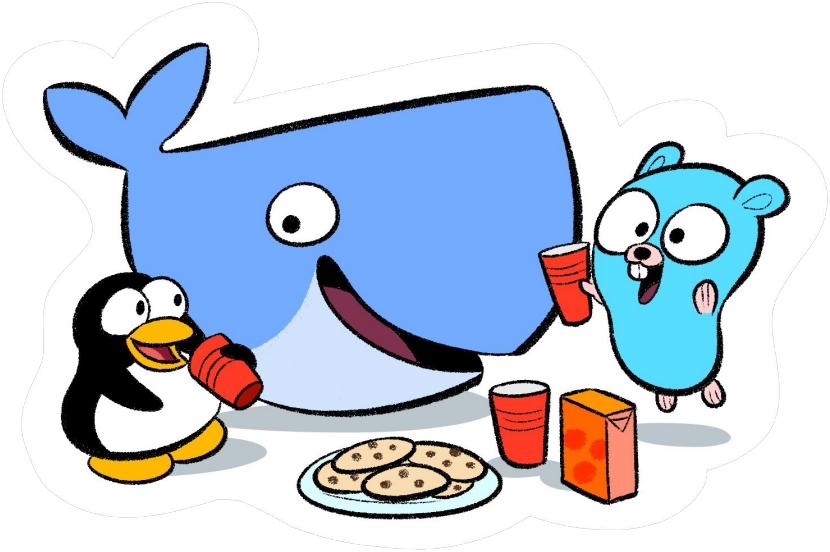
add sudo access to the user

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

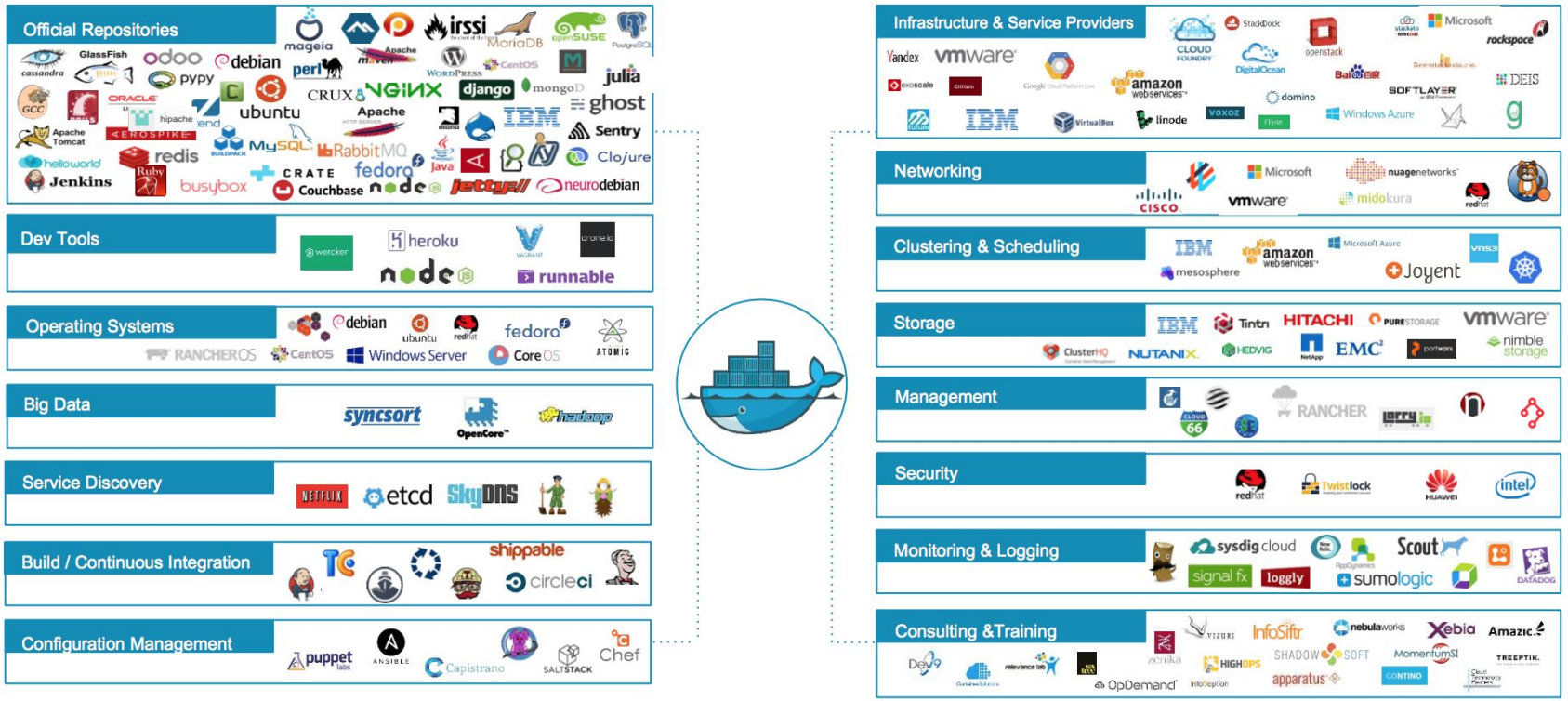
install docker-compose

```
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
```

Introduction to Docker



Docker ecosystem



Docker Basics

Docker Image

- The basis of a Docker container

Docker Container

- The standard unit in which the application service resides

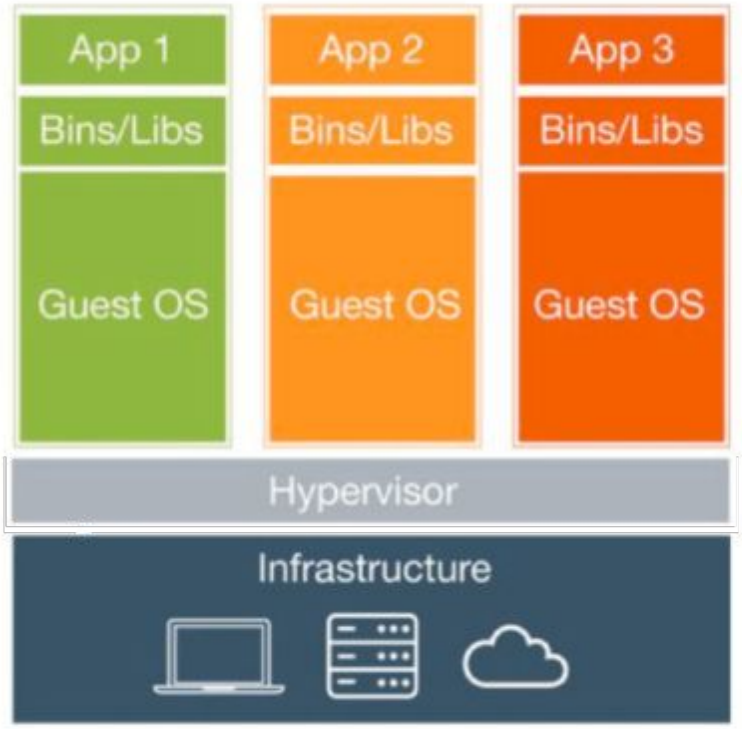
Docker Engine

- Creates, ships and runs Docker containers deployable on physical or virtual host locally, in a datacenter or cloud service provider

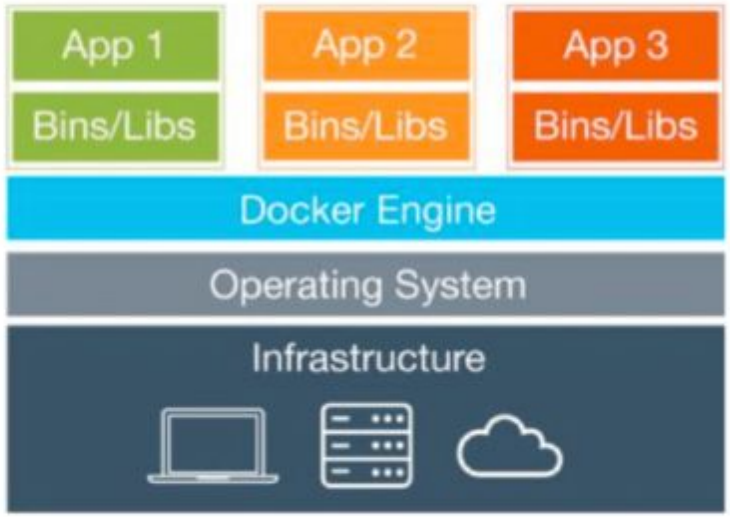
Docker Registry

- On-premises registry for image storing and collaboration

VMs vs. Containers



Virtual Machines

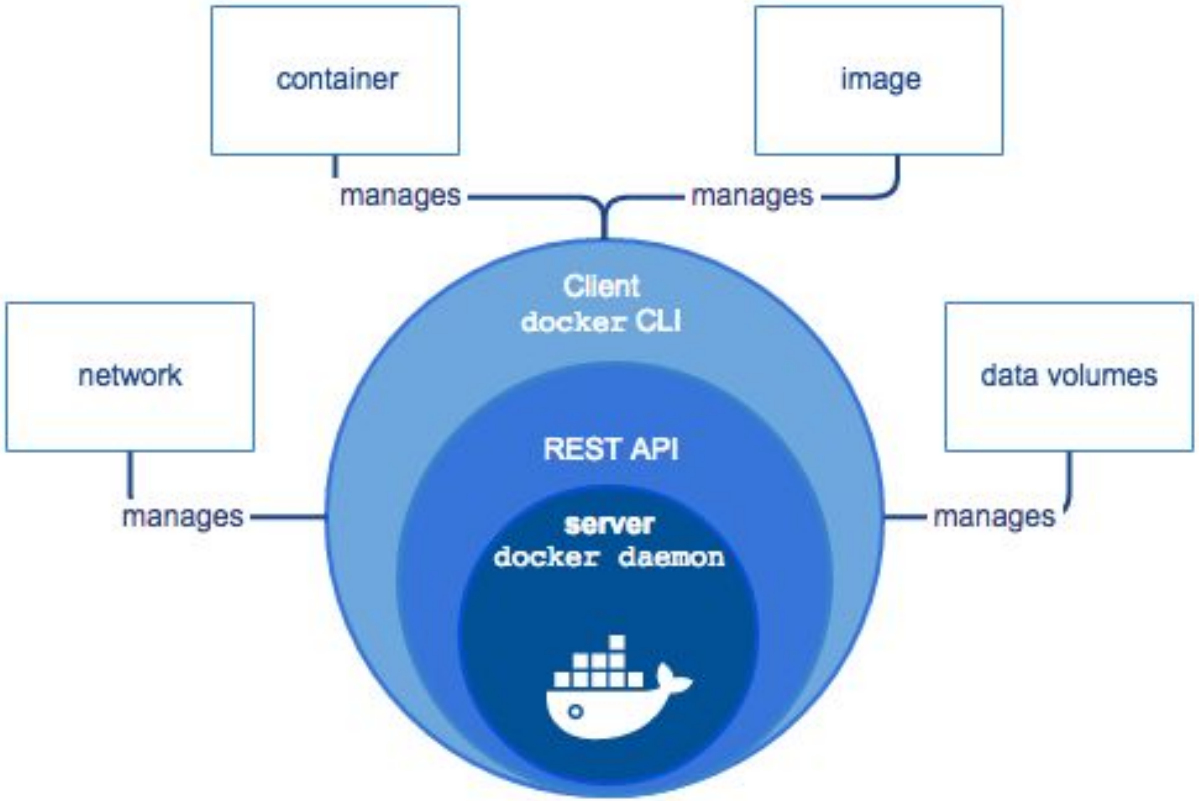


Containers

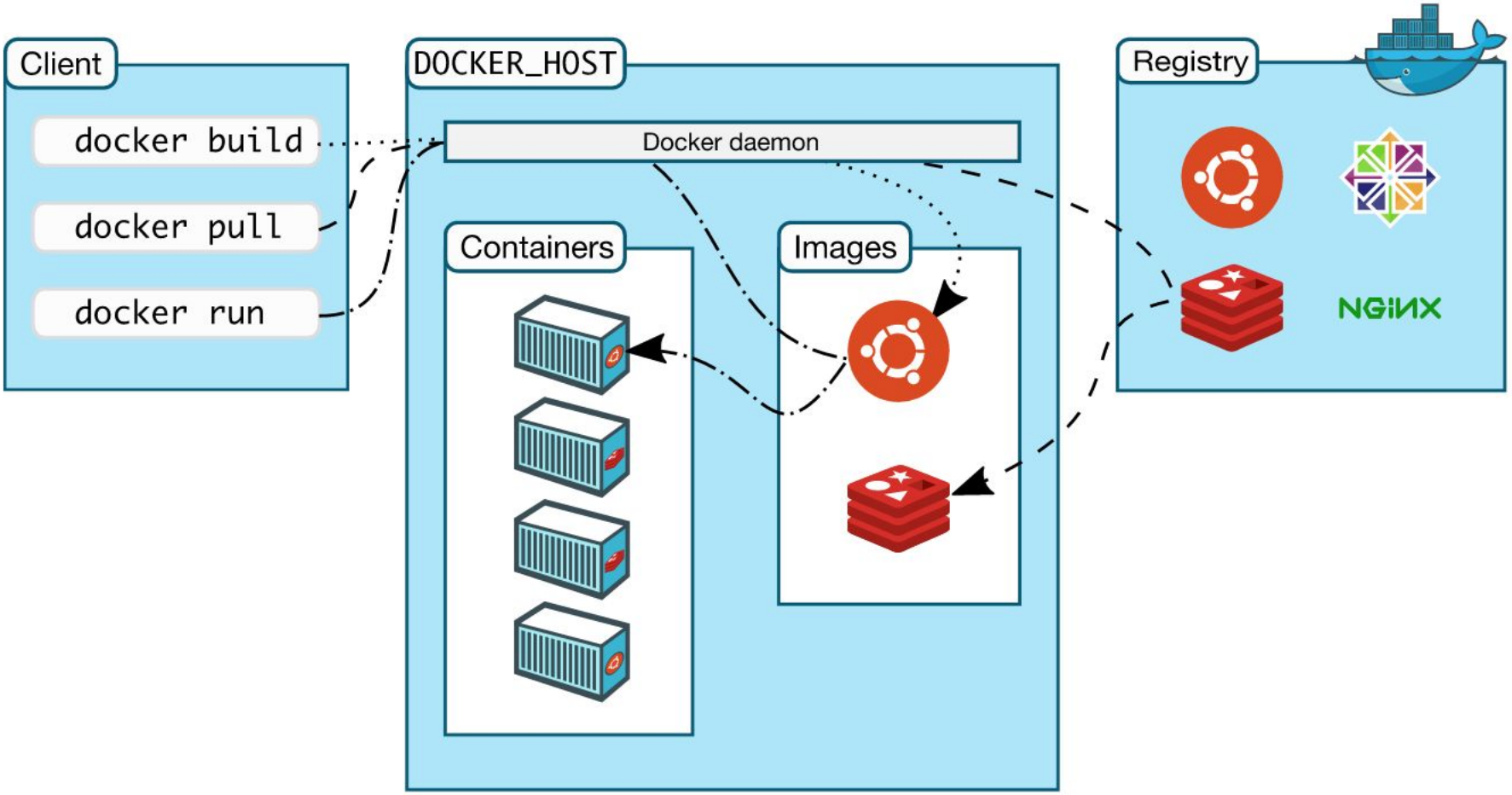
VMs vs. Containers

Virtual Machines (VMs)	Containers
Represents hardware-level virtualization	Represents operating system virtualization
Heavyweight	Lightweight
Slow provisioning	Real-time provisioning and scalability
Limited performance	Native performance
Fully isolated and therefore more secure (maybe)	Process-level isolation and therefore less secure (maybe)

What is ‘Docker Engine’?



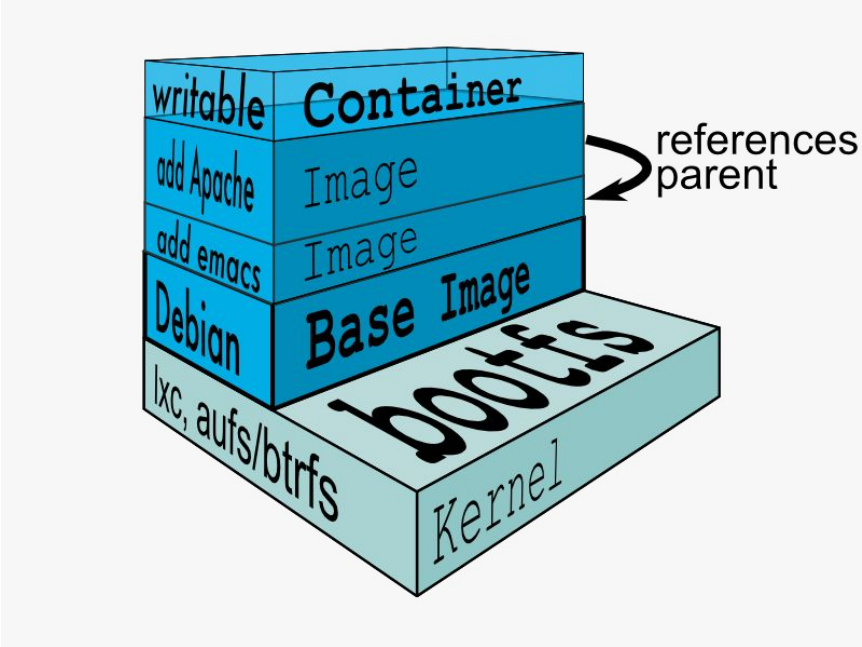
How is Docker Architected?



Docker Union File System

AuFS (AnotherUnionFS) is a multi-layered filesystem that implements union mount

- allows several filesystems or directories to be simultaneously mounted and visible through a single mount point
- appears to be one filesystem to the end user



Example Docker Layers

Docker Layer – Each Docker image is composed of a series of layers. Docker uses Union File Systems to combine these into a single image. The combination of these layers gives the illusion of a traditional file system. Only the top layer is writeable.

Where Containers are used?

- DevOps (Mostly in all types of Automation)
- Automated testing (Unit, acceptance and stress testing)
- CICD (if not tested on production-like environment, CICD will be risky)
- Microservices
- Less risky deployment architectures
- Chef kitchen test
- Not only used for testing but is used in production (along with container orchestration) in B2C and B2B organizations

Advantage of using Containers

- Developers testing their code in Prod environments
- Lesser hand-offs and drastically lesser waiting time (to get new environment setup)
  - –Reduce technical debt
- Overall system resilience - Service failure does not kill the application & may be invisible to users
- Scalability
  - – Seamless replication
- High Availability
- Less risky patterns for rolling updates available
- Prevent server sprawl and Jenga infrastructure

Introduction to Kubernetes

Part 1:

Kubernetes

The name Kubernetes is Greek

- Translates to “helmsman” or “pilot”
- Root of ”Governor” and “Cybernetic”

“K8s” is a shorthand abbreviation derived by replacing the 8 letters “ubernete” with 8.

Cloud Native Computing Foundation (CNCF)

Mission of the Cloud Native Computing Foundation.

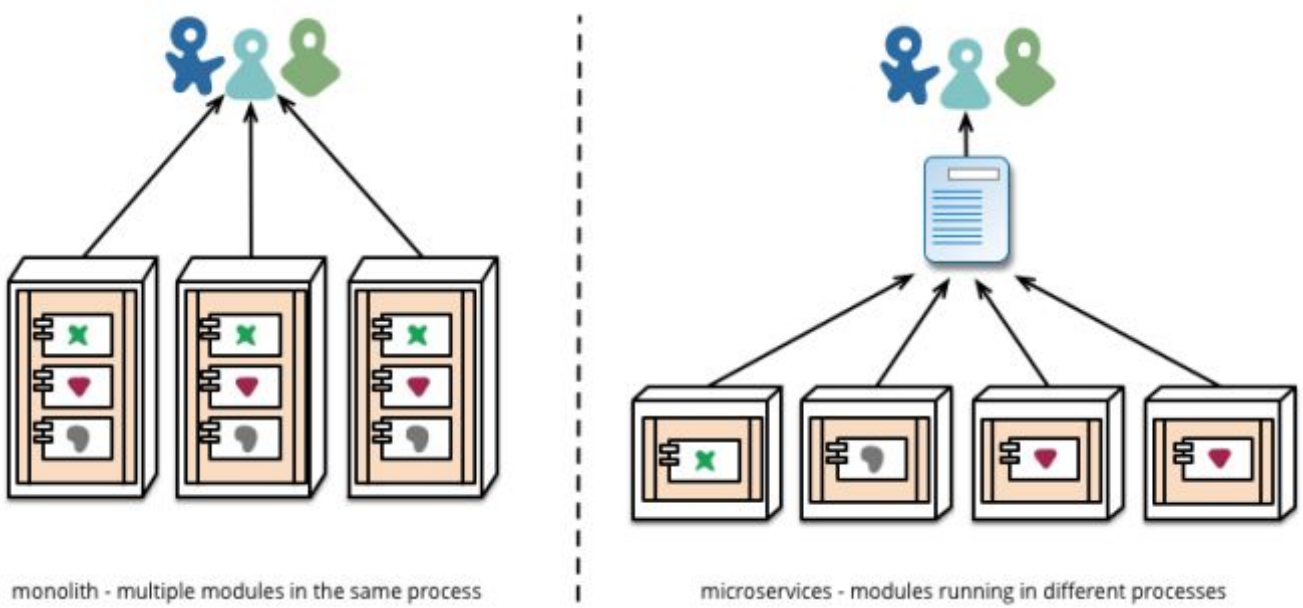
- "The Foundation's mission is to create and drive the adoption of a new computing paradigm that is optimized for modern distributed systems environments capable of scaling to tens of thousands of self healing multi-tenant nodes."

Formed in Nov 6, 2015 with Kubernetes as its first project

- The CNCF is building an ecosystem of complementary projects including Prometheus, Fluentd, Linkerd, Rkt, and others

What is a Microservice?

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



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

Kubernetes Setup

Kubernetes can be setup on multiple clouds

- Google(GCP) & Microsoft offer Kubernetes as a service, with reliable experiences
- AWS has two great open source methods, 'Kops' and Heptio's Cloud Formation Script
- Last year AWS added Kubernetes as a Service
- Kubernetes can readily be setup locally, with 'Minikube' or 'kubeadm'

Kubernetes Setup (AWS) - Kubeadm

We will use AWS for this class

- Each student will log on to a AWS EC2 machine
- 1 node will be used as a master and client
- We will use set up instructions on github
- We will see a demo to use 2 nodes and treat them as a cluster.

Local Kubernetes

Use the instructions to run the 4 instructions

-- will set up master

-- enable networking, untaint the master node

-- should be able to run commands

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_setup_files/Kubernetes_Setup_AWS.md

Example Commands

kubectl get nodes --all-namespaces -o wide

kubectl get services --all-namespaces -o wide

kubectl get pods --all-namespaces -o wide

kubectl get deployment --all-namespaces -o wide

```
Abhiks-MacBook-Pro:Downloads abhikbanerjee$ kubectl get nodes --all-namespaces -o wide
NAME       STATUS    ROLES     AGE       VERSION   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION   CONTAINER-RUNTIME
minikube   Ready     master    157d      v1.10.0   <none>        Buildroot 2017.11    4.9.64           docker://17.12.1-ce
Abhiks-MacBook-Pro:Downloads abhikbanerjee$ kubectl get services --all-namespaces -o wide
NAMESPACE     NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)             AGE       SELECTOR
default       hw                     NodePort    10.99.202.72    <none>        80:30302/TCP        3d        run=hw
default       kubernetes             ClusterIP   10.96.0.1       <none>        443/TCP             157d      <none>
kube-system   heapster               ClusterIP   10.104.214.7    <none>        80/TCP              13d       addonmanager.kubernetes.io/mode=Reconcil
e,k8s-app=heapster
kube-system   kube-dns               ClusterIP   10.96.0.10      <none>        53/UDP,53/TCP       157d      k8s-app=kube-dns
kube-system   kubernetes-dashboard   NodePort    10.108.26.87    <none>        80:30000/TCP        157d      app=kubernetes-dashboard
kube-system   monitoring-grafana     NodePort    10.107.147.86   <none>        80:30002/TCP        13d       addonmanager.kubernetes.io/mode=Reconcil
e,k8s-app=influx-grafana
kube-system   monitoring-influxdb    ClusterIP   10.104.16.247   <none>        8083/TCP,8086/TCP   13d       addonmanager.kubernetes.io/mode=Reconcil
e,k8s-app=influx-grafana
Abhiks-MacBook-Pro:Downloads abhikbanerjee$ kubectl get pods --all-namespaces -o wide
NAMESPACE     NAME                                    READY     STATUS        RESTARTS   AGE       IP             NODE
default       example-daemonset-t24bl                 1/1       Running       193        12d       172.17.0.11    minikube
default       example-daemonset2-zgjsg                1/1       Running       192        12d       172.17.0.6     minikube
default       hw-596b578c58-pq547                     1/1       Running       0          3d        172.17.0.9     minikube
default       hw-596b578c58-rp8hl                     1/1       Running       0          3d        172.17.0.10    minikube
default       hw-596b578c58-vcznz                     1/1       Running       0          3d        172.17.0.8     minikube
default       nginx-deploy-585856bc9-grf7m            1/1       Running       0          3d        172.17.0.4     minikube
default       print-secrets-4mr71                     0/1       Terminating   0          6d        <none>         minikube
default       print-secrets-8qd77                     0/1       Terminating   0          8d        <none>         minikube
default       zk-0                                    1/1       Running       1          12d       172.17.0.12    minikube
kube-system   etcd-minikube                           1/1       Running       0          8d        10.0.2.15      minikube
kube-system   heapster-wlq6k                          1/1       Running       1          13d       172.17.0.5     minikube
kube-system   influxdb-grafana-67rxd                  2/2       Running       2          13d       172.17.0.3     minikube
kube-system   kube-addon-manager-minikube             1/1       Running       4          157d      10.0.2.15      minikube
kube-system   kube-apiserver-minikube                 1/1       Running       0          8d        10.0.2.15      minikube
kube-system   kube-controller-manager-minikube        1/1       Running       0          8d        10.0.2.15      minikube
kube-system   kube-dns-86f4d74b45-6qmtv               3/3       Running       16         157d      172.17.0.2     minikube
kube-system   kube-proxy-xsb52                        1/1       Running       0          8d        10.0.2.15      minikube
kube-system   kube-scheduler-minikube                 1/1       Running       3          149d      10.0.2.15      minikube
kube-system   kubernetes-dashboard-5498ccf677-mdv8v   1/1       Running       12         157d      172.17.0.7     minikube
kube-system   storage-provisioner                     1/1       Running       12         157d      10.0.2.15      minikube
Abhiks-MacBook-Pro:Downloads abhikbanerjee$
```

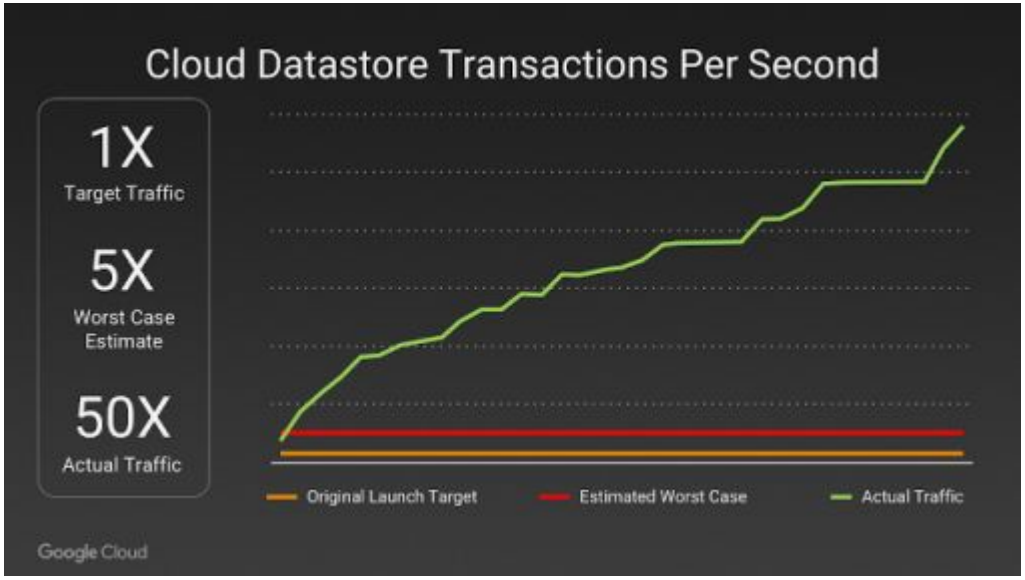
Pokemon Go

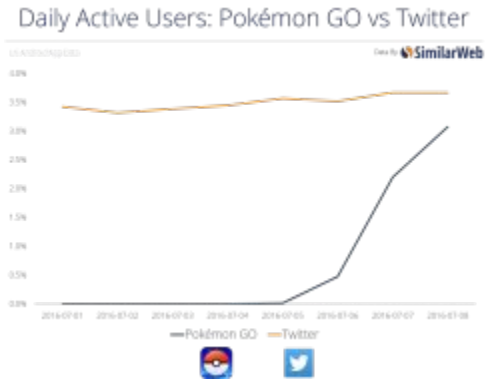
Pokemon Go: Situation

Pokemon Go launched a hit app! The application was a viral, overnight roaring success.

The challenge? It rapidly had 50x the worst case traffic estimate and grew rapidly

- Achieved the same active users in a month that took Twitter years to reach
- Was the most downloaded app in history within a few days





Pokemon Go: Action

Application for the logic ran on Google Container Engine powered by Kubernetes.

- Enabled automation to scale the application to millions of users
- Also enabled upgrading the version of the cluster and load balancer without downtime while millions of players were online

Pokemon Go: Action

Even Google doesn't have unlimited resources and engineers

- Google CRE (Cloud Reliability Engineer team) worked closely with Niantic to review and optimize the architecture.

Pokemon Go used Cloud Datastore, a cloud native NoSQL database.

- This enabled the application to scale without the sharding requirements that would have been needed with MySQL or Postgres for example

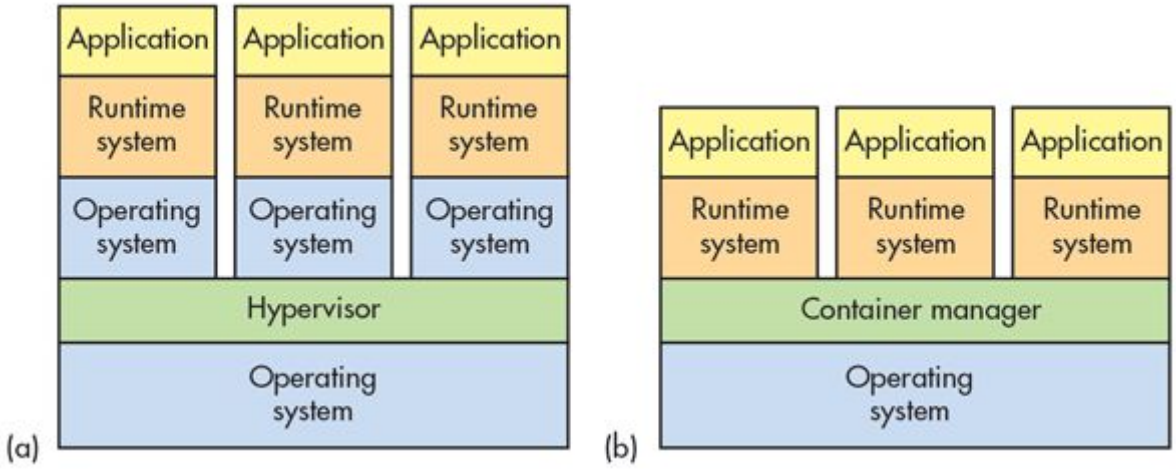
Pokemon Go: Result

Was able to launch in Japan a mere two weeks after the US launch, where they received 3x the US users without incident.

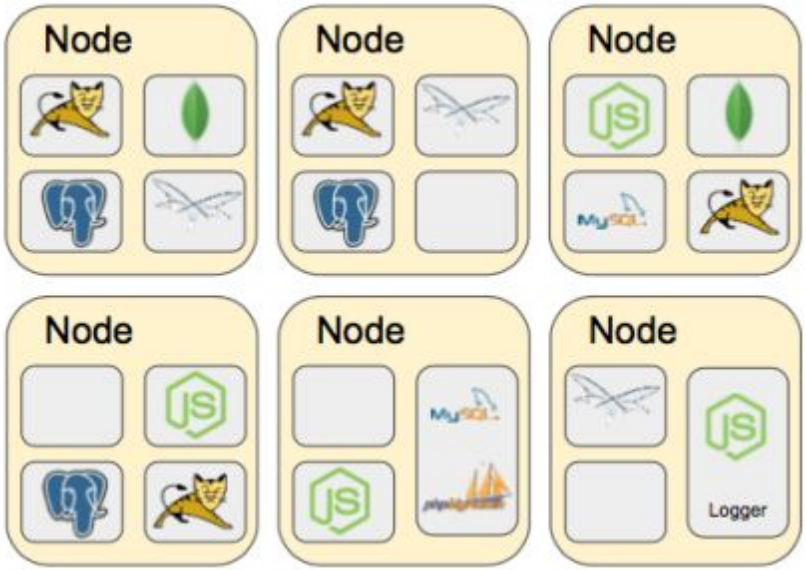
- Was the largest Kubernetes deployment ever, with multitude of bugs identified, fixed and merged into the open source repo
- The cluster ended up utilizing tens of thousands of cores
- Arguably the first successful overnight/viral planet-wide launch
- Was eventually downloaded more than 500 million times and 25 million players at peak
- Inspired users to walk over 5.4 billion miles over the course of a year due to gameplay

Container Advantages

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



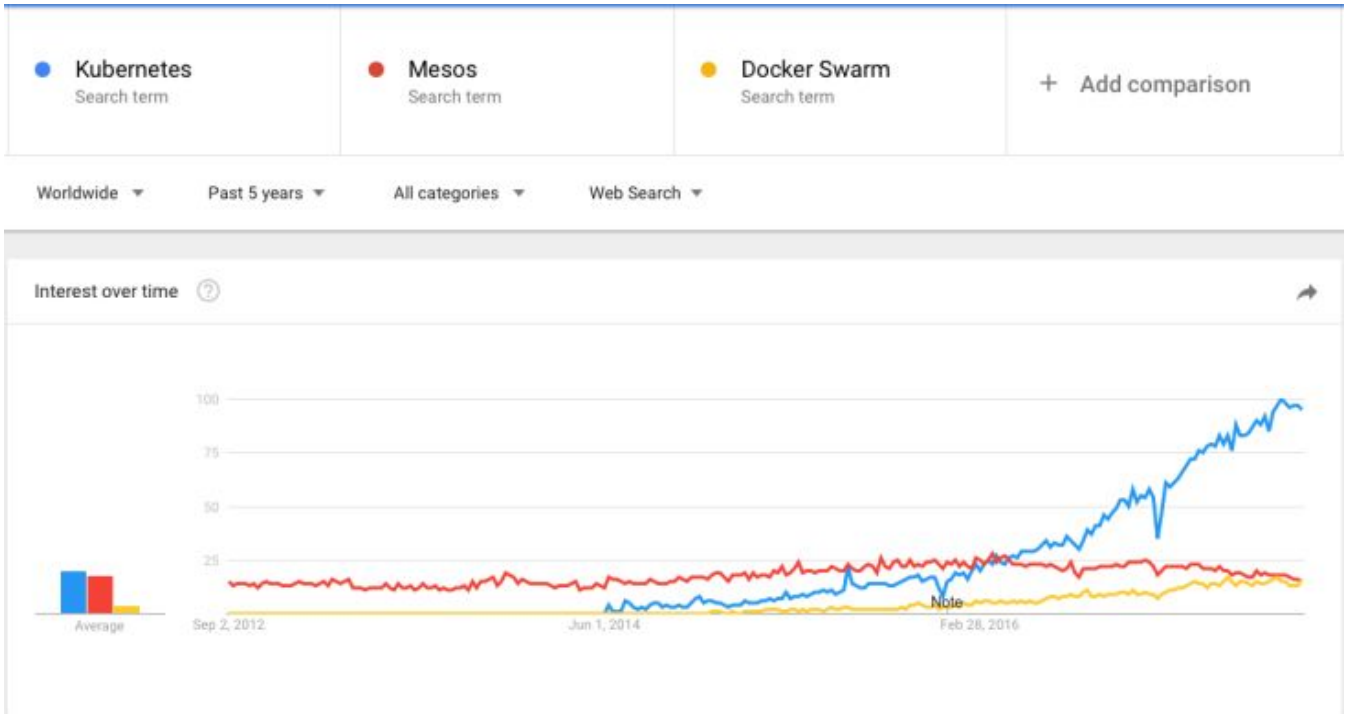
Challenges with multiple containers



- 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?

Container Orchestration Tools

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

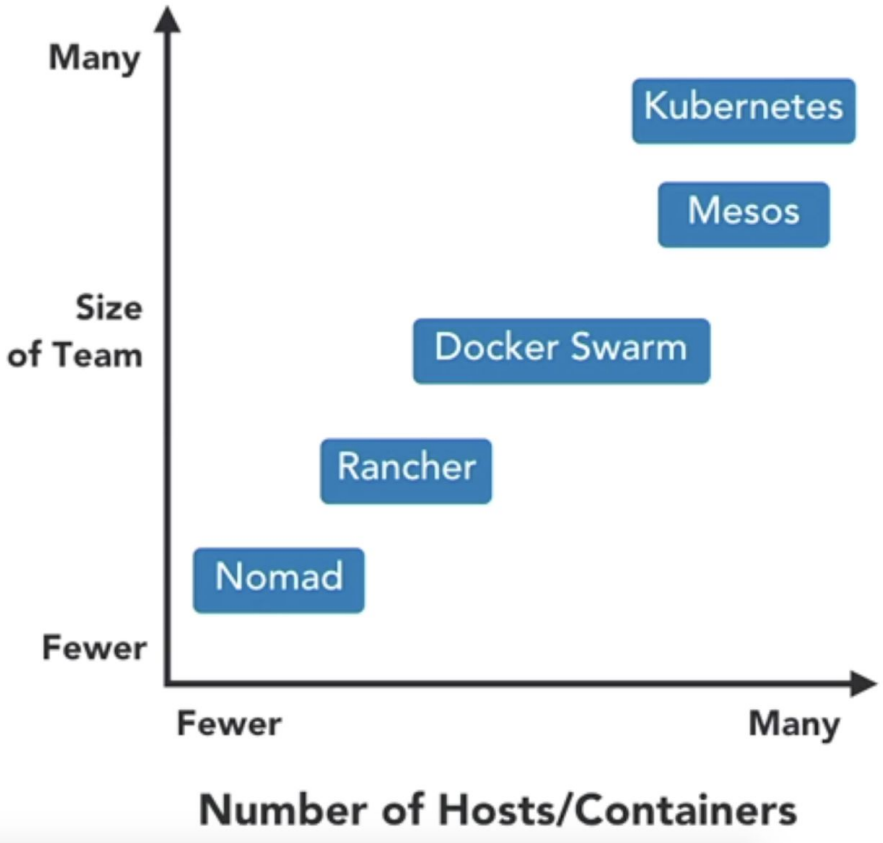


Orchestration Tool Functions

- Provision Hosts
- Instantiate Containers on Hosts
- Restart Failing Containers
- Expose Containers as service outside the cluster
- Scale the cluster up or down

Guidelines for right infra

Guidelines for the Right Orchestrator for the Job

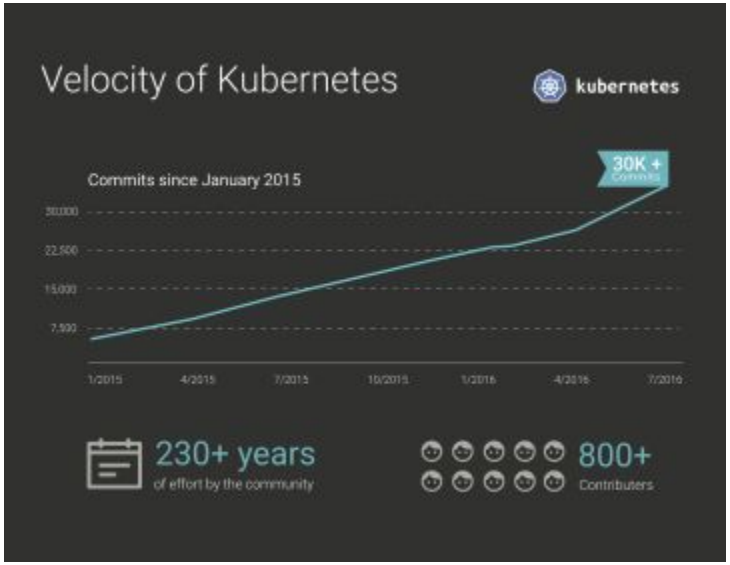


Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

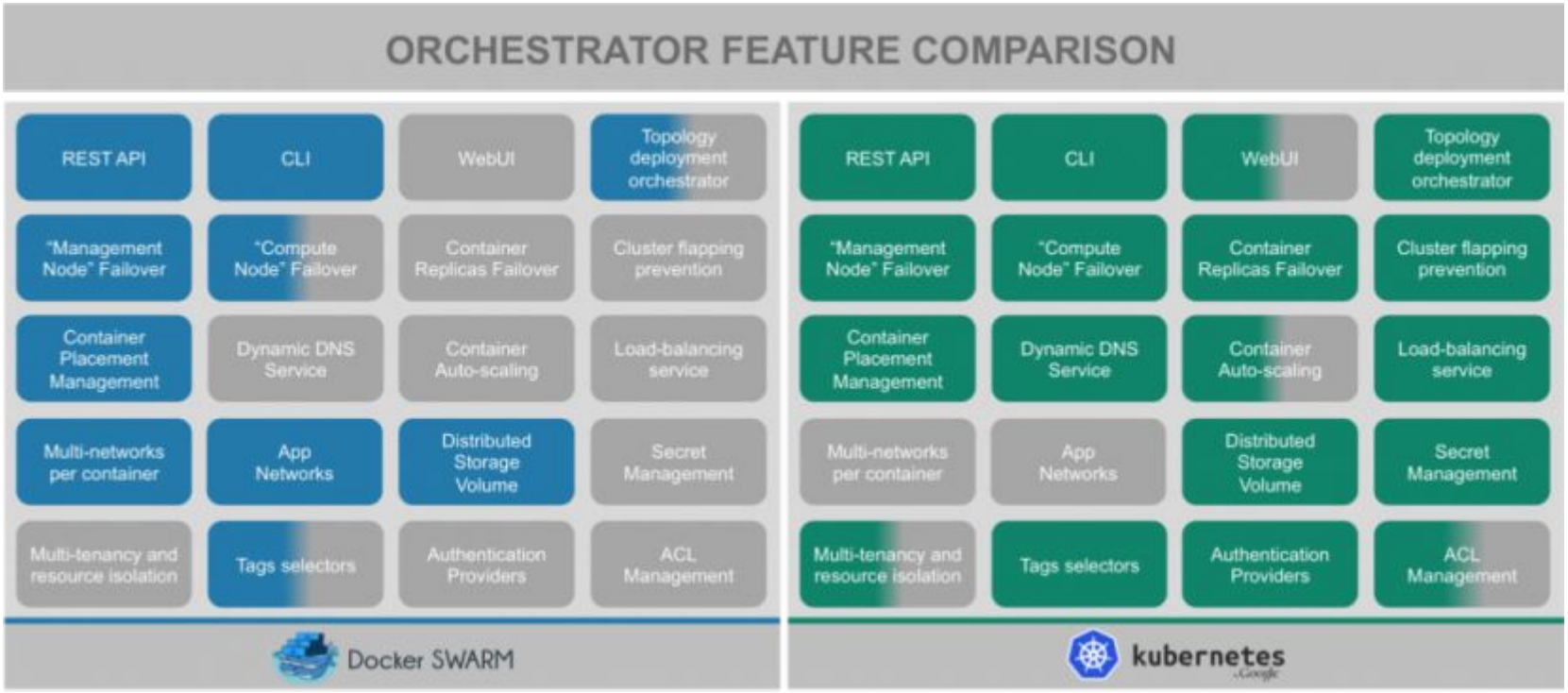
Kubernetes Stats

- 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)



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

Why Kubernetes?

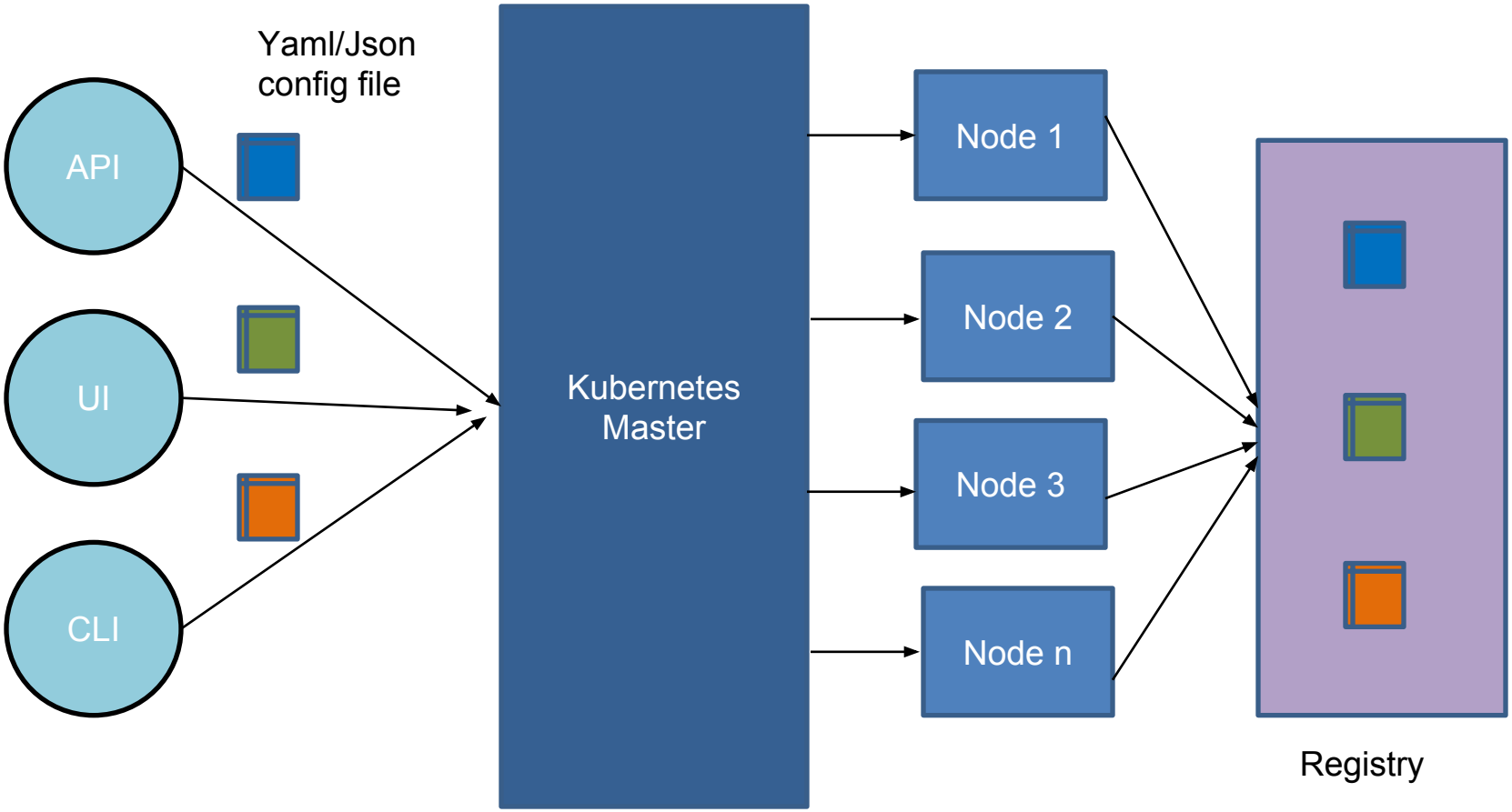
Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.

- Co-locating helper processes, facilitating composite applications and preserving the one-application-per-container model
- Mounting storage systems
- Distributing secrets
- Checking application health
- Replicating application instances
- Using Horizontal Pod Autoscaling
- Balancing loads
- Rolling updates
- Monitoring resources
- Debugging applications
- Providing authentication and authorization

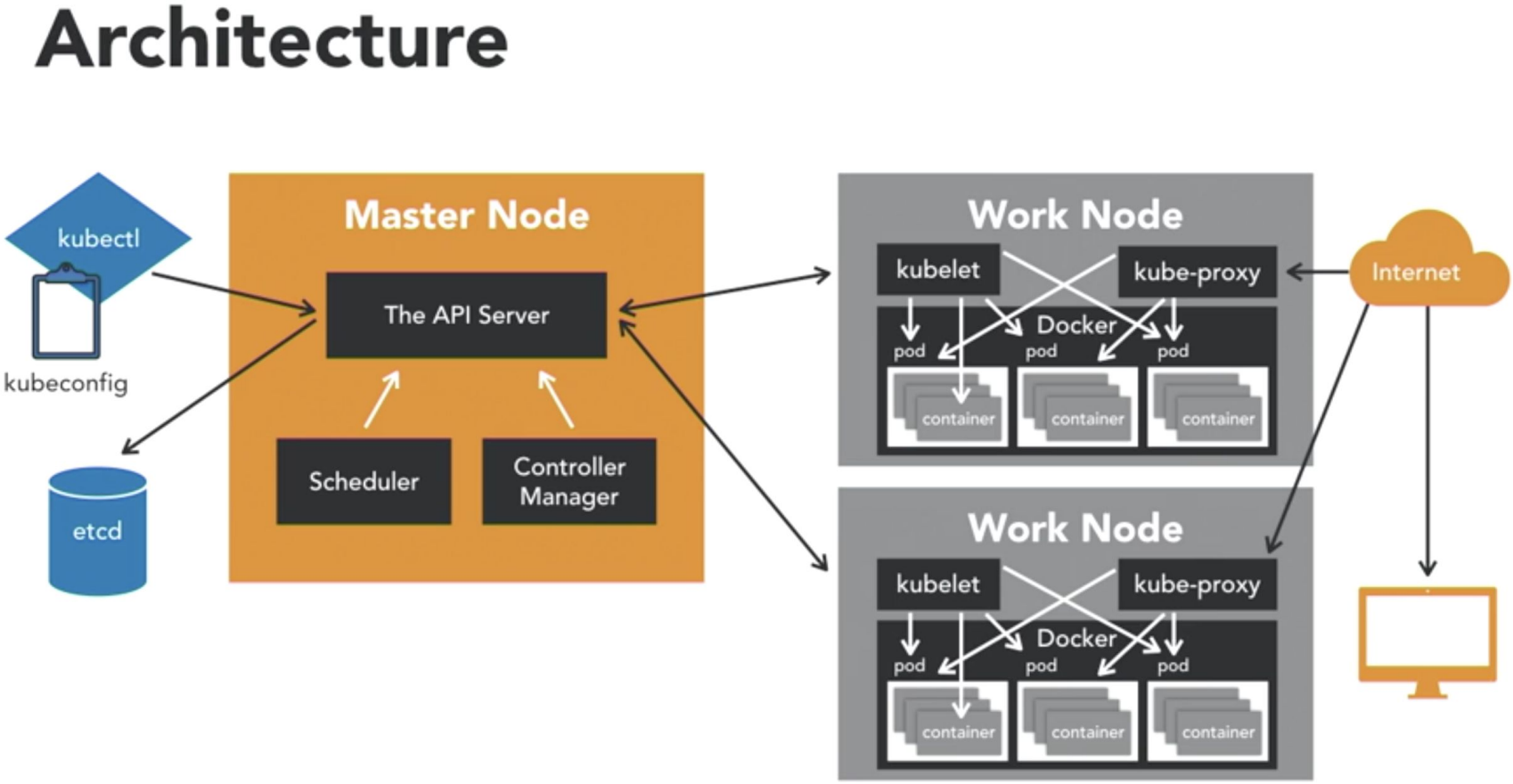
What is Kubernetes?

- Use Kubernetes API object to describe cluster's desired state
- Next the Kubernetes Control Plane works to make the desired state same to the current state
- Kube master is a collection of Kube-apiserver, kube-controller, and kube-scheduler
- Kubernetes Node consists of kubelet, kube-proxy, and container management
- Basic Kubernetes Objects (Pods, Services, Volumes, Namespaces)
- Higher Level abstractions (ReplicsaSets, StatefulSets, DaemonSets, Deployments, Jobs)

Kubernetes Architecture

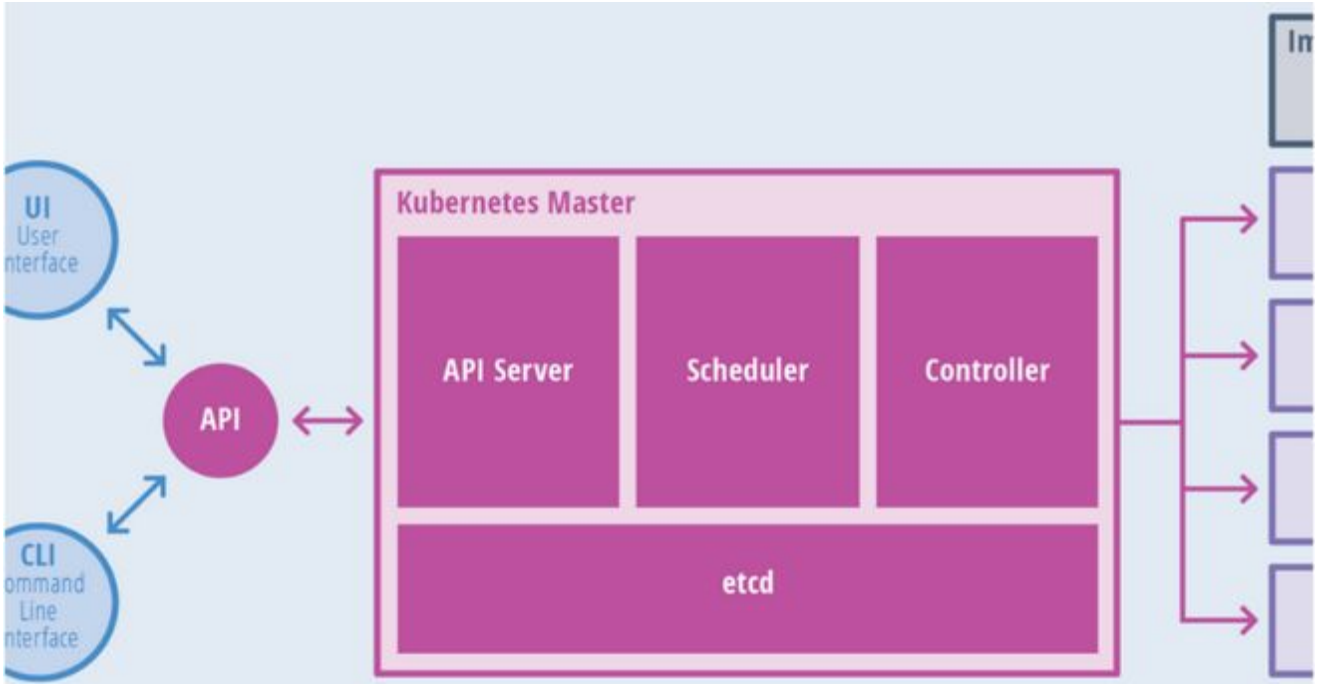


K8s Architecture - detailed



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

Kubernetes Master



Components of master:

- API Server
- Scheduler
- Controller
- etcd

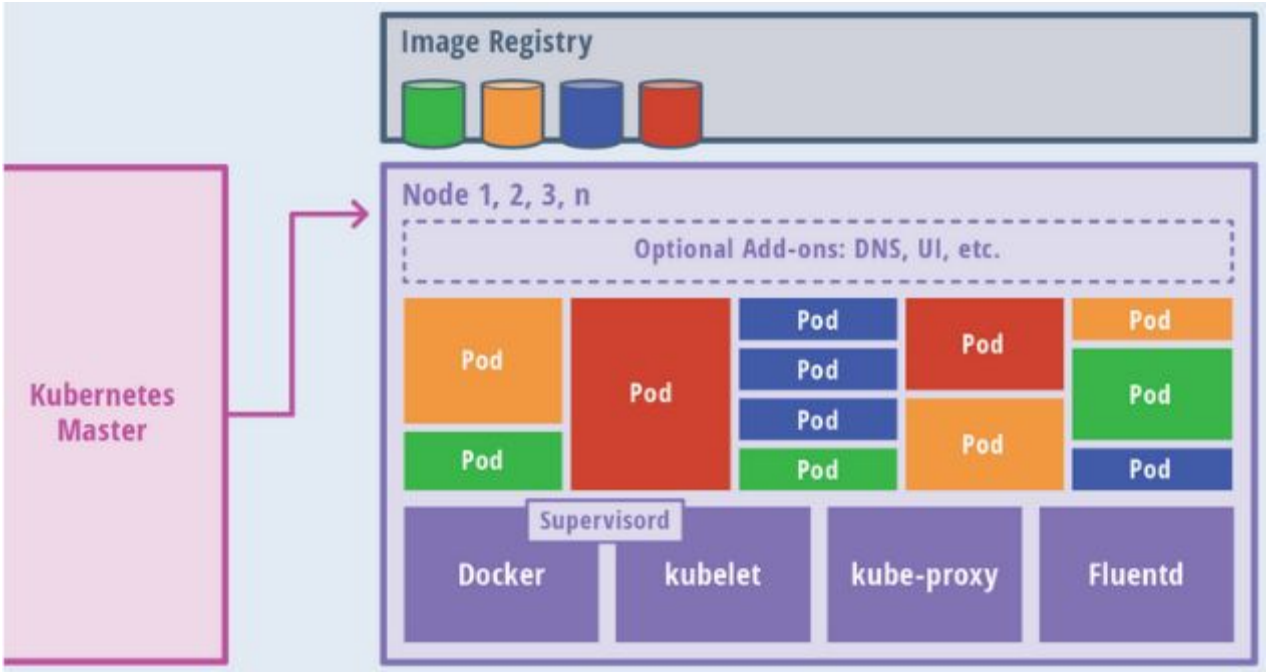
Kubernetes Master

- 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

- **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 non terminating 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**, **node controller**, and **service account & token 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

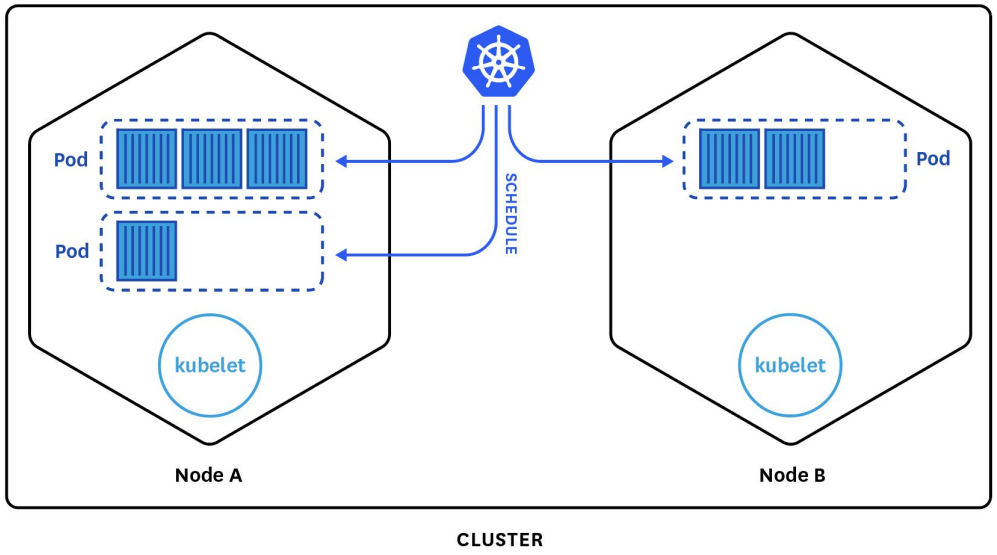


Components of a node:

- kubelet
- Kube-proxy
- Docker
- Fluentd

Kubernetes: Nodes

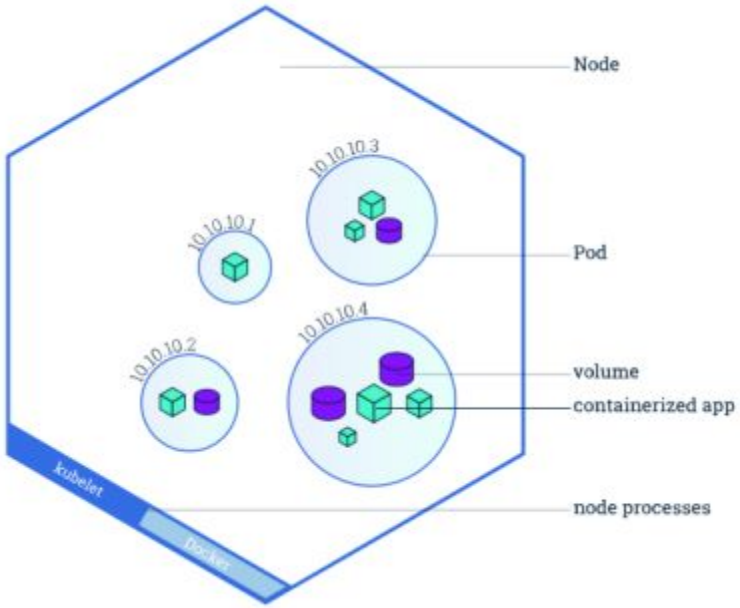
A node is a worker machine in Kubernetes. A node may be a VM or physical machine, depending on the cluster. Each node has the services necessary to run pods and is managed by the master components. The services on a node include Docker, kubelet and kube-proxy. See The Kubernetes Node section in the architecture design doc for more details.



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
- Runs 3 processes - Kubelet, Kube Proxy and Container Runtime



Kubernetes: Nodes

Node Status

A node status contains:

- Addresses
  - Hostname
  - ExternalIP
  - InternalIP
- Condition - describe condition of nodes.
- Capacity - describe the resources available on the node
- Info - general information about the node

$ 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
```

Note: Affinity & Anti-affinity (Beta) allow further constraints on where a pod may run (coupled, de-coupled, only nodes with certain labels, etc)

Kubernetes Node - Kubelet

- **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 **via api-server**. 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 Node - Kube Proxy

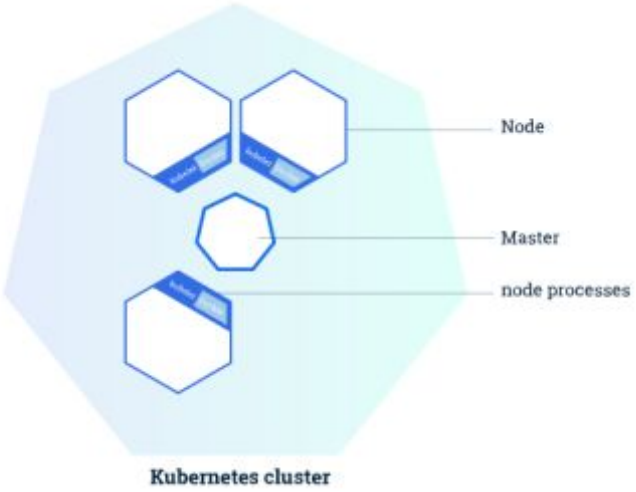
- Kubernetes **Proxy** Service is a proxy service which runs on each node and helps in making services available to the external host. It helps in forwarding the request to correct containers and is capable of performing primitive load balancing. It makes sure that the networking environment is predictable 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.

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)



Addons

- DNS - While the other addons are not strictly required, all Kubernetes clusters should have cluster DNS, as many examples rely on it.
- Web UI (Dashboard) - Dashboard is a general purpose, web-based UI for Kubernetes clusters.
- Container Resource Monitoring - Container Resource Monitoring records generic time-series metrics about containers in a central database, and provides a UI for browsing that data.
- Cluster-level Logging - A Cluster-level logging mechanism is responsible for saving container logs to a central log store with search/browsing interface.

Kubernetes Terms

- Containers
- Nodes
- Clusters
- Pods (Object)
- Deployment (Controllers)
- ReplicaSets (Controllers)
- Services (Object)
- Labels
- Volumes (Object)
- Namespaces (Object)
- StatefulSet (Controllers)
- DaemonSet (Controllers)
- Job (Controllers)

Kubernetes: 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.

- Containers in a pod share
  - IP address and port space
  - Filesystem
  - Storage (Volumnes)
  - Labels
  - Secrets

Kubernetes: Pods

Pods Are...

- Ephemeral, disposable
- Never self-heal, and not restarted by the scheduler by itself
- Never create pods just by themselves
- Always use higher-level constructs

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

Kubernetes: Pod States

- **Pod States**
  - Pending (request accepted, one or more containers are still being created)
  - Running (Pod has been bound to a node, all of the Containers have been created. At least one Container is still running, or is in the process of starting or restarting.)
  - Succeeded (All Containers in the Pod have terminated in success, and will not be restarted)
  - Failed (All Containers in the Pod have terminated, and at least one container has terminated with some failure)
  - CrashLoopBackOff (pod starting, crashing, starting again, and then crashing again.)

pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-apparmor
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

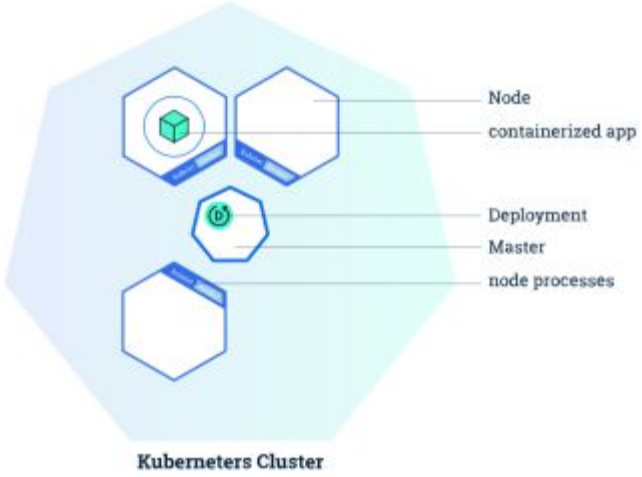
```
root@ip-172-31-68-96:~/code# kubectl create -f pod.yml
pod/nginx-apparmor created
root@ip-172-31-68-96:~/code# kubectl get pods
NAME              READY     STATUS    RESTARTS   AGE
nginx-apparmor    1/1       Running   0          4s
root@ip-172-31-68-96:~/code#
```

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.

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 Replica Sets
  - 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



Kubernetes: Deployment

Abstraction that use Replication controller (Controller manager of kubernetes master) to manage replicas of pods (replaced by replica-sets and deployments)

- If object {pod} is used, and it dies, it will not start again
- If object {deployment} is used to start pods, if the pod dies, deployment use replica-set to start the pods again to make sure desired number of pods are always alive
- Try deleting a pod (deployed using deployment) and check if it comes back?

dep.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
          app: nginx
  template:
    metadata:
          labels:
            app: nginx
    spec:
          containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

```
root@ip-172-31-68-96:~/code# kubectl create -f dep.yml
deployment.apps/nginx-deployment created
root@ip-172-31-68-96:~/code# kubectl get pods,deployments
NAME                                    READY     STATUS    RESTARTS   AGE
pod/nginx-apparmor                      1/1       Running   0          52s
pod/nginx-deployment-67594d6bf6-7jc78   1/1       Running   0          11s
pod/nginx-deployment-67594d6bf6-tf5df   1/1       Running   0          11s
pod/nginx-deployment-67594d6bf6-xzwww   1/1       Running   0          11s

NAME                                   DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/nginx-deployment   3         3         3            3           11s
root@ip-172-31-68-96:~/code#
```

$ kubectl create / apply

- kubectl create -f <yaml_file>
- kubectl create -f <yaml_file_1> -f <yaml_file_2>
- kubectl create -f <json_file>
- kubectl create -f <url_name>
- kubectl create -f <directory_name>

- kubectl create -f <yaml_file> (ok when multiple teams are not working on the same config files)
- kubectl apply -f <json_file> (practically used in teams where multiple teams / members want to maintain versions)

- $ kubectl create -f blue.yaml
- $ kubectl create -f green.yaml -f red.yaml
- $ kubectl create -f pink.yaml ,
- $ kubectl edit -f pink.yaml
- $ kubectl create -f colors/

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

Kubernetes: RS, RC and Deployment

Replication Controllers - A Replication Controller is a structure that enables you to easily create multiple pods, then make sure that that number of pods always exists. If a pod does crash, the Replication Controller replaces it. The major difference is that the rolling-update command works with Replication Controllers, but won't work with a Replica Set. This is because Replica Sets are meant to be used as the backend for Deployments

A **Replica Set** Replica Sets are declared in essentially the same way as Replication Controllers, except that they have more options for the selector.

Deployments - We have seen this before, uses replication sets , and also provide rolling updates.

Ref:- https://www.mirantis.com/blog/kubernetes-replication-controller-replica-set-and-deployments-understanding-replication-options/

Exercise - 1

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

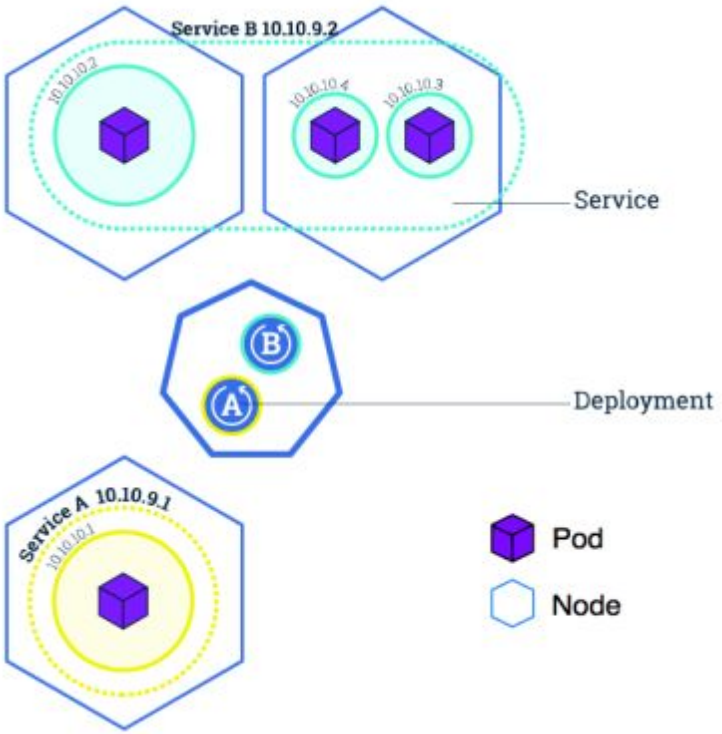
- https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_rc_rs_deploy.md

Ref:- https://www.mirantis.com/blog/kubernetes-replication-controller-replica-set-and-deployments-understanding-replication-options/

Kubernetes: Service

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

- 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.

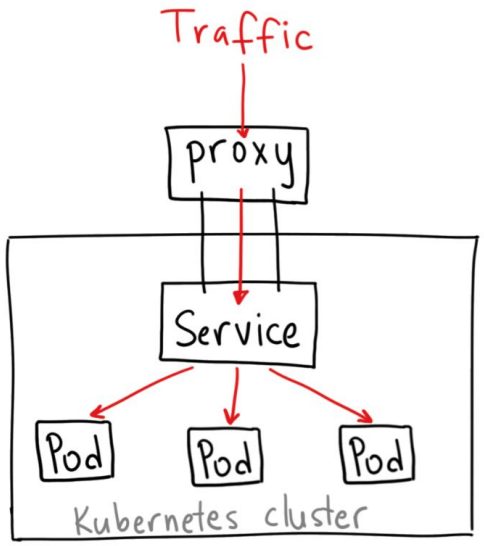


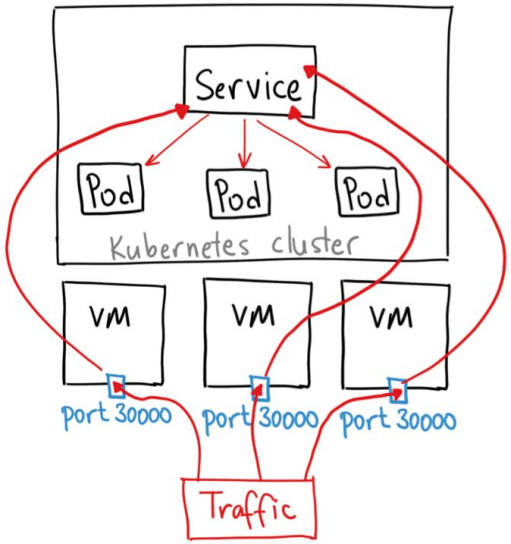
Kubernetes: Types of Services

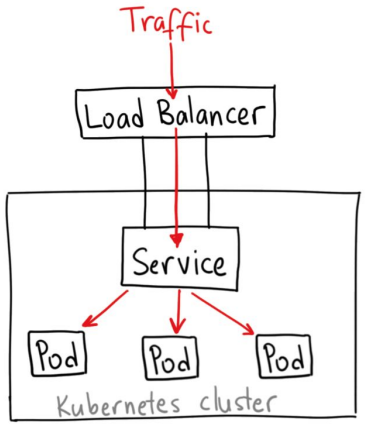
- **Cluster IP**:- Expose the service on a cluster-internal IP. Using this makes the service only reachable from within the cluster (no external access).
- **NodePort** :- Expose the service on each Node's IP at a static port. One service per port, only use ports 30000–32767, have to deal with port changes
- **Load Balancer** :- Expose the service externally using load balancer. The Kubernetes service controller automates the creation of the external load balancer, health checks (if needed), firewall rules (if needed) and retrieves the external IP allocated by the cloud provider and populates it in the service object

Kubernetes: Services

- **Cluster IP**:- $ kubectl proxy --port=8080
- **NodePort** :- check yaml config
- **Load Balancer** :- check yaml config





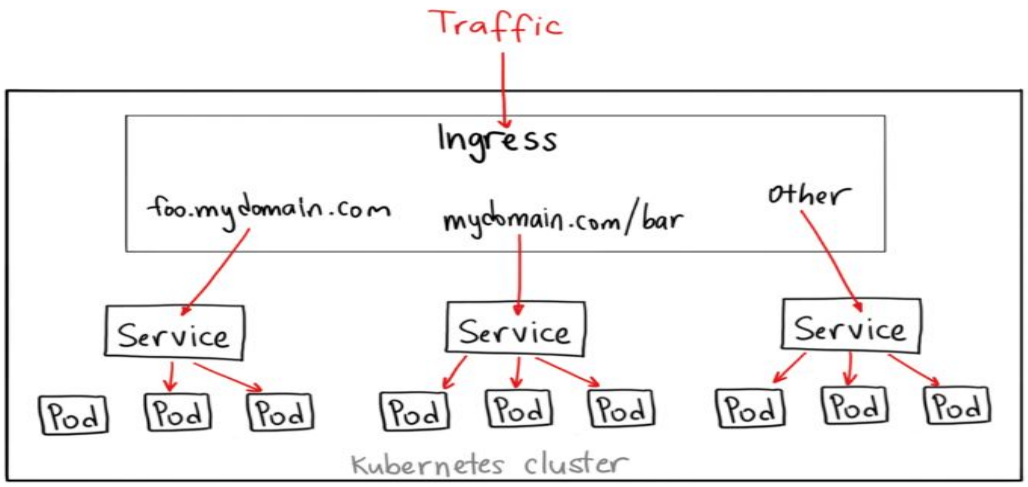


Ref:- https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0

Kubernetes Ingress

Simplified Layer 7 access to Kubernetes services i.e. allows “internet” to reach services (not a type of service)

- Works with load balancers, including the Google Cloud Load Balancer, Nginx, Contour, Istio, and more
- expose multiple services under the same IP address



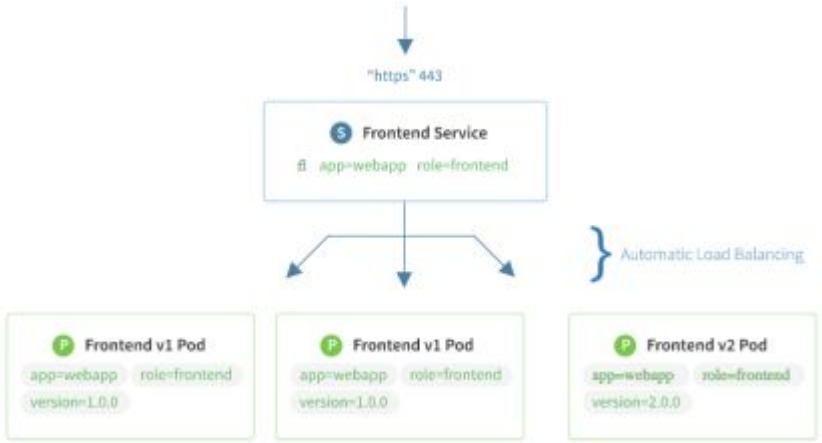
Ref:- https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0

```
root@ip-172-31-68-96:~# kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   37m
root@ip-172-31-68-96:~#
root@ip-172-31-68-96:~# kubectl expose deployment nginx-deployment
--type=LoadBalancer --name=nginx-service
service/nginx-service exposed
root@ip-172-31-68-96:~#
root@ip-172-31-68-96:~# kubectl get svc
NAME            TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP      10.96.0.1      <none>        443/TCP        37m
nginx-service   LoadBalancer   10.102.89.83   <pending>     80:31095/TCP   3s
root@ip-172-31-68-96:~#
```

$ 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’



$ kubectl Demo services

-- Create directly a deployment from CLI without the config file (**Object management using Imperative Commands** - e.g. run , expose, autoscale)

$ kubectl run nginx-deployment --image nginx --port 80

$ kubectl get nodes

$ kubectl get po, deploy, rs

-- **exposing the deployment as a service** LoadBalancer type and access the Nginx

$ kubectl expose deploy/nginx-deployment --type=LoadBalancer --name nginx-service

$ kubectl get all (command shows all)

$ kubectl get deploy/hw -o yaml (returns the yaml that creates the deployment)

-- then you can describe the service and do a curl , on port 80 with internal IP or external IP and the exposed port outside

$ kubectl expose pod pod_name --type NodePort --name pod_name-service --port 30006

$ kubectl Demo services

-- Create a deployment with the config file (Object management using **Imperative configuration** files)

$ kubectl create -f red.yaml

$ kubectl get po, deploy, rs

-- Create Kubernetes Object using **Declarative management** of objects (The **kubectl apply** command calculates the patch request using the configuration file, the live configuration, and the **last-applied-configuration** annotation stored in the live configuration)

$ kubectl apply -f <file_name>

$ kubectl get po, deploy, rs

edit the file , and try the apply again

$ kubectl apply -f <file_name>

$ kubectl get po, deploy, rs

Ref:- https://kubernetes.io/docs/concepts/overview/object-management-kubectl/declarative-config/

Exercise - 2 (Pods, Deploy, Services)

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

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_pod_deploy_service.md

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

Kubernetes help commands

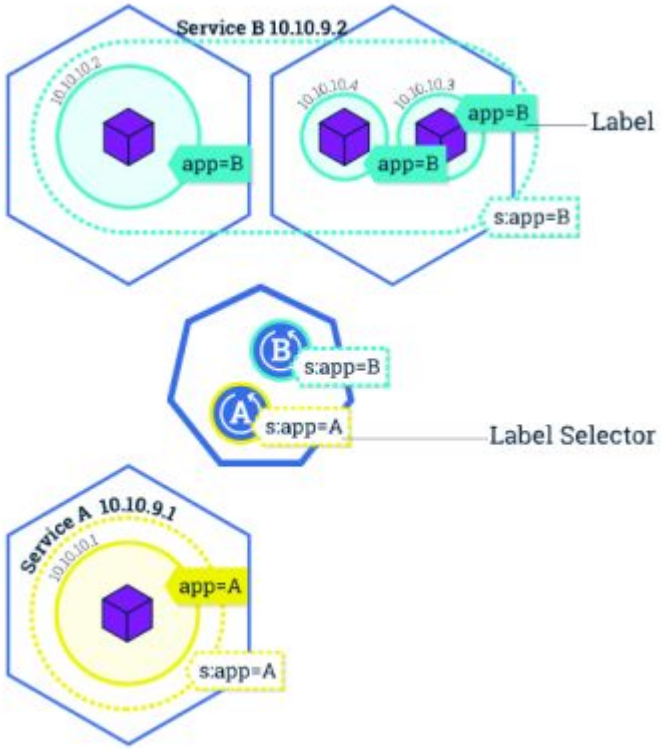
Most common access point for launching workloads on Kubernetes clusters.

- **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: 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

- **labels can be used to provide logical structure**
- **divide by teams, or by environments, or versions**
- **annotations and labels have very subtle difference**
- $ kubectl create -f helloworld-pod-with-labels.yml
- $ kubectl get po --show-labels

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

Exercise - 3 (Kubernetes Labels)

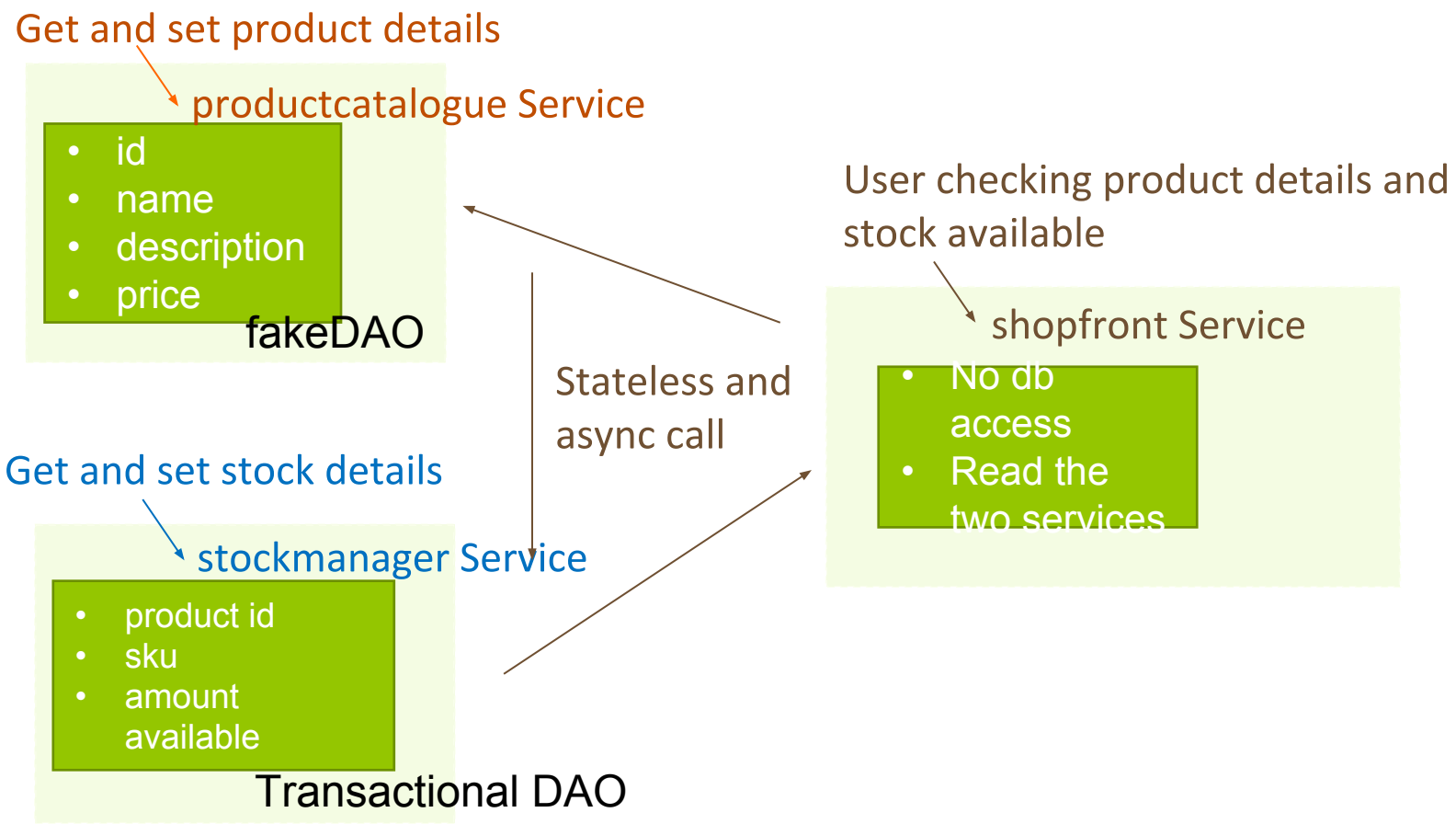
- Create a series of pods with different labels
- Query the pods based on various labels
- Delete pods based on certain selectors

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_labels_usage.md

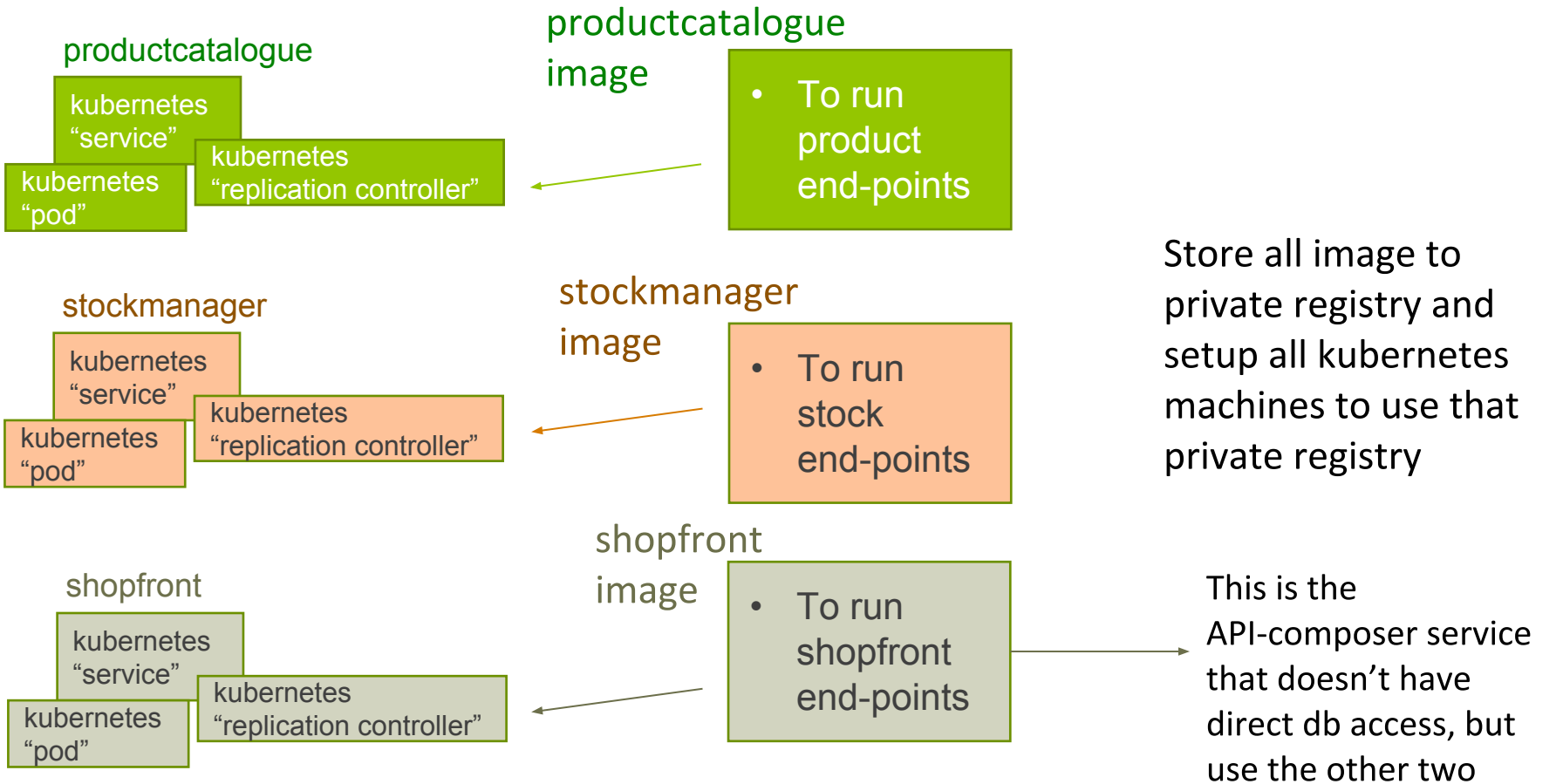
Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

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

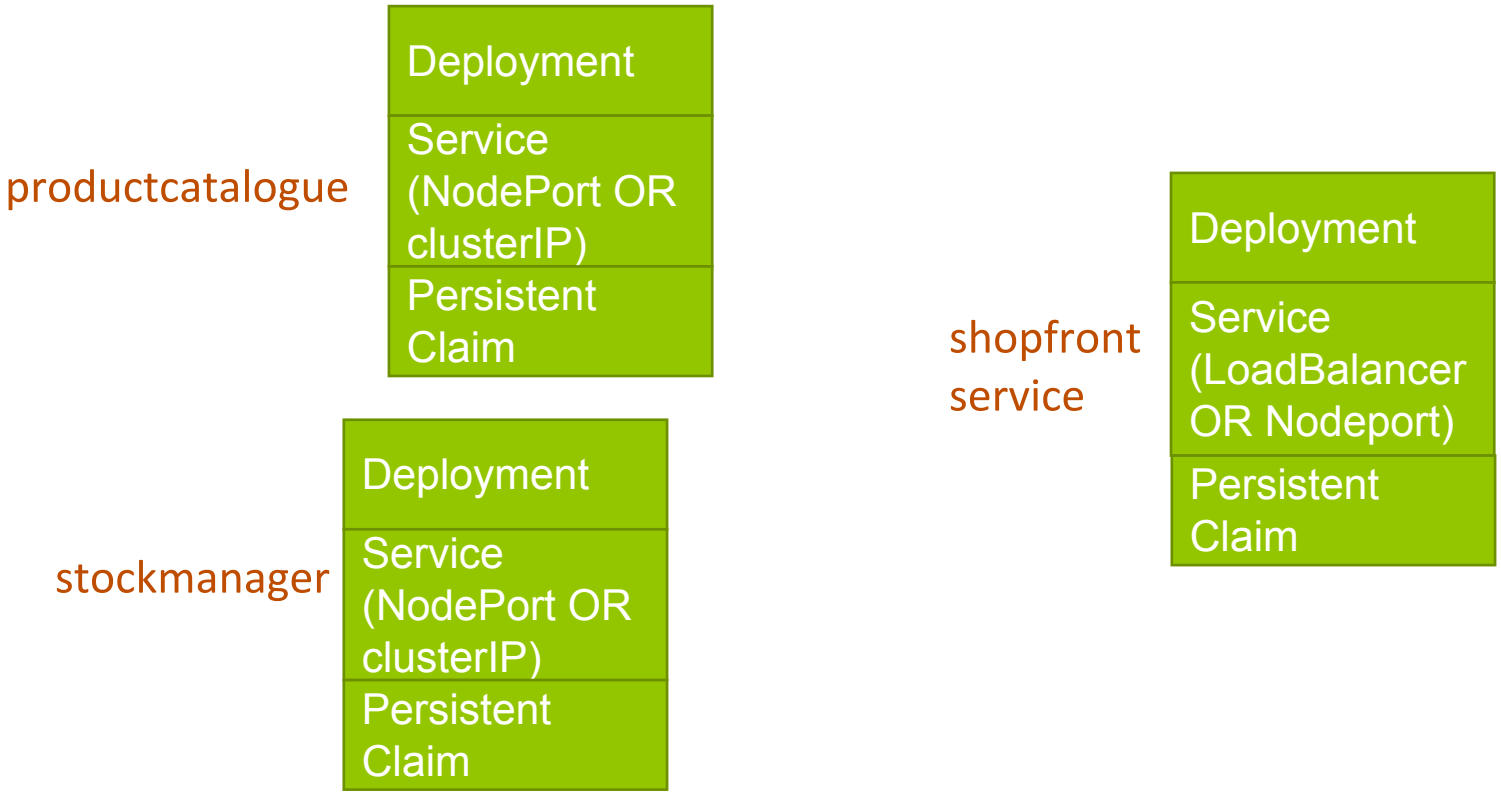
Microservice App in Java
(Taken from Day 1)



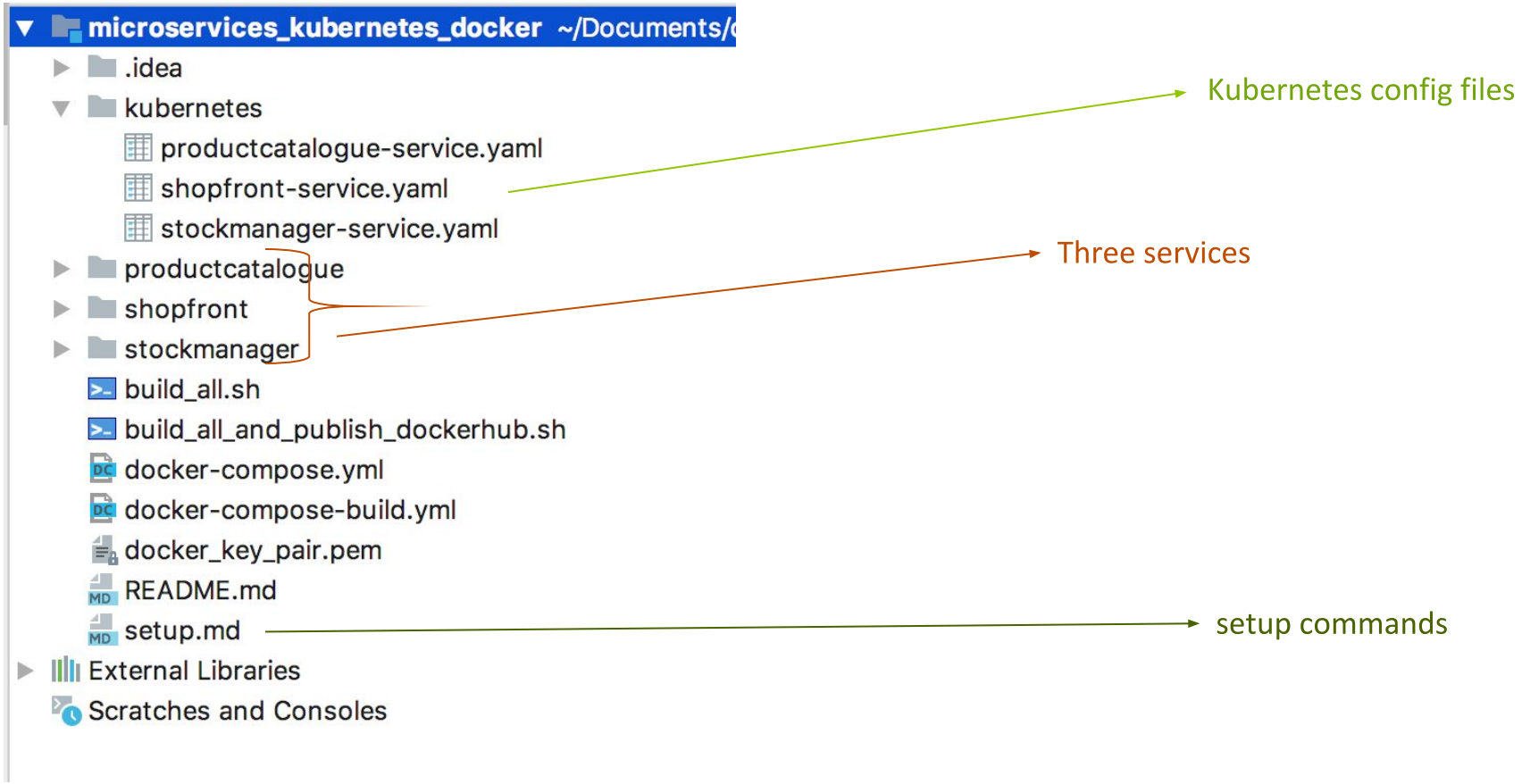
Docker Image & Kubernetes Requirements
(To run the App)



Kubernetes Resources
(To run the App)

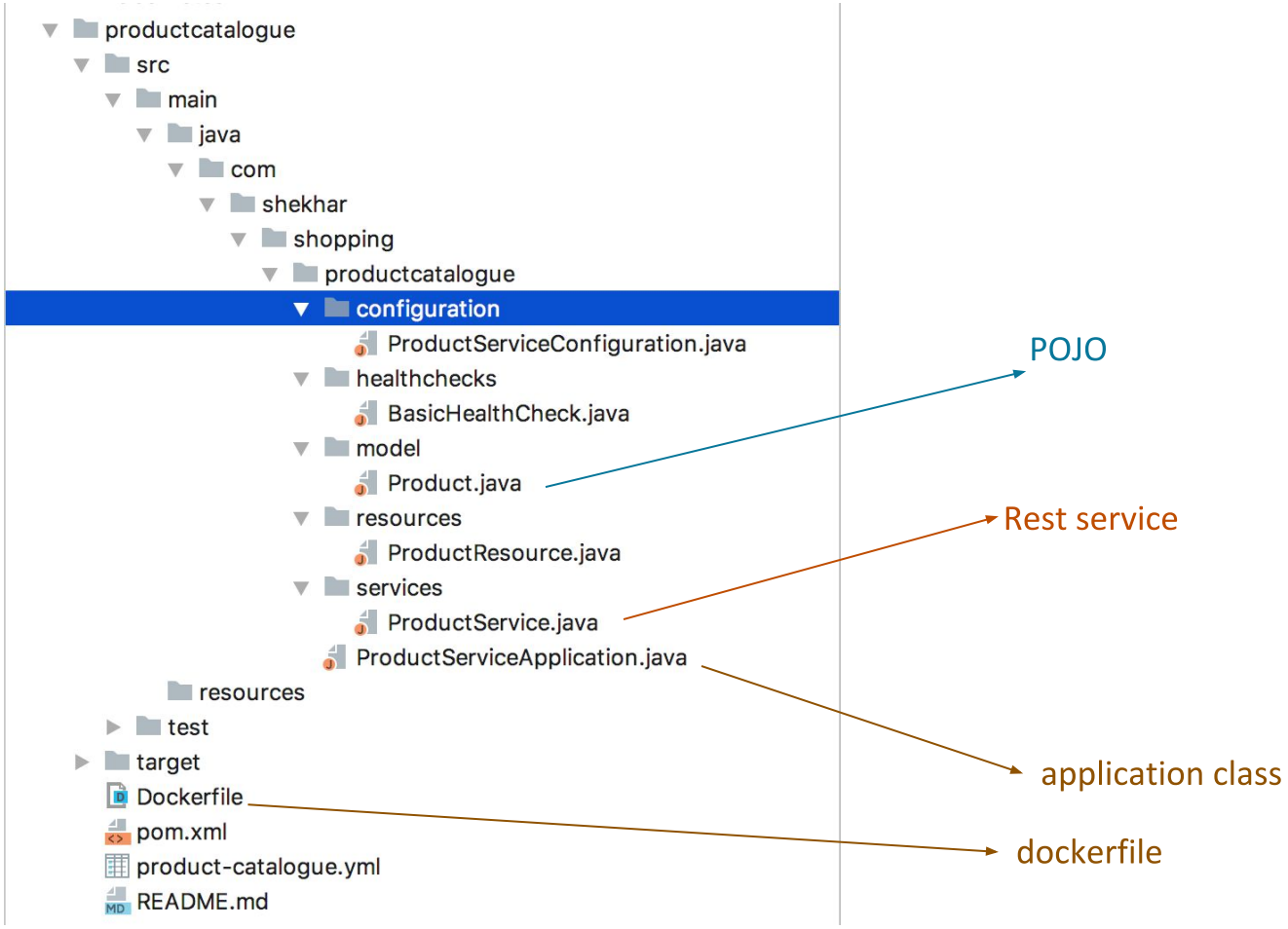


Project Structure – Project and Kubernetes



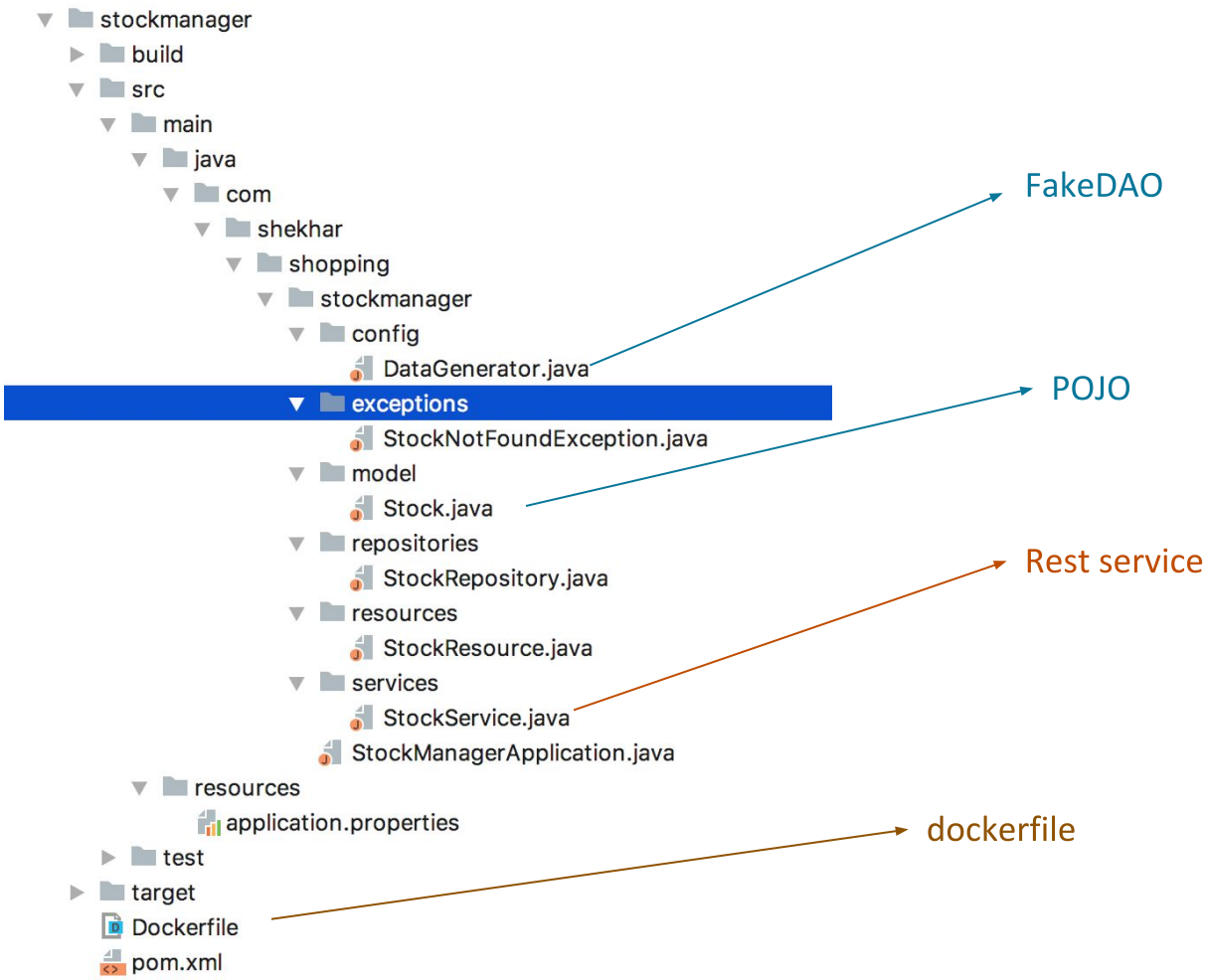
https://github.com/shekhar2010us/microservices_kubernetes_docker

Project Structure – productcatalogue Service



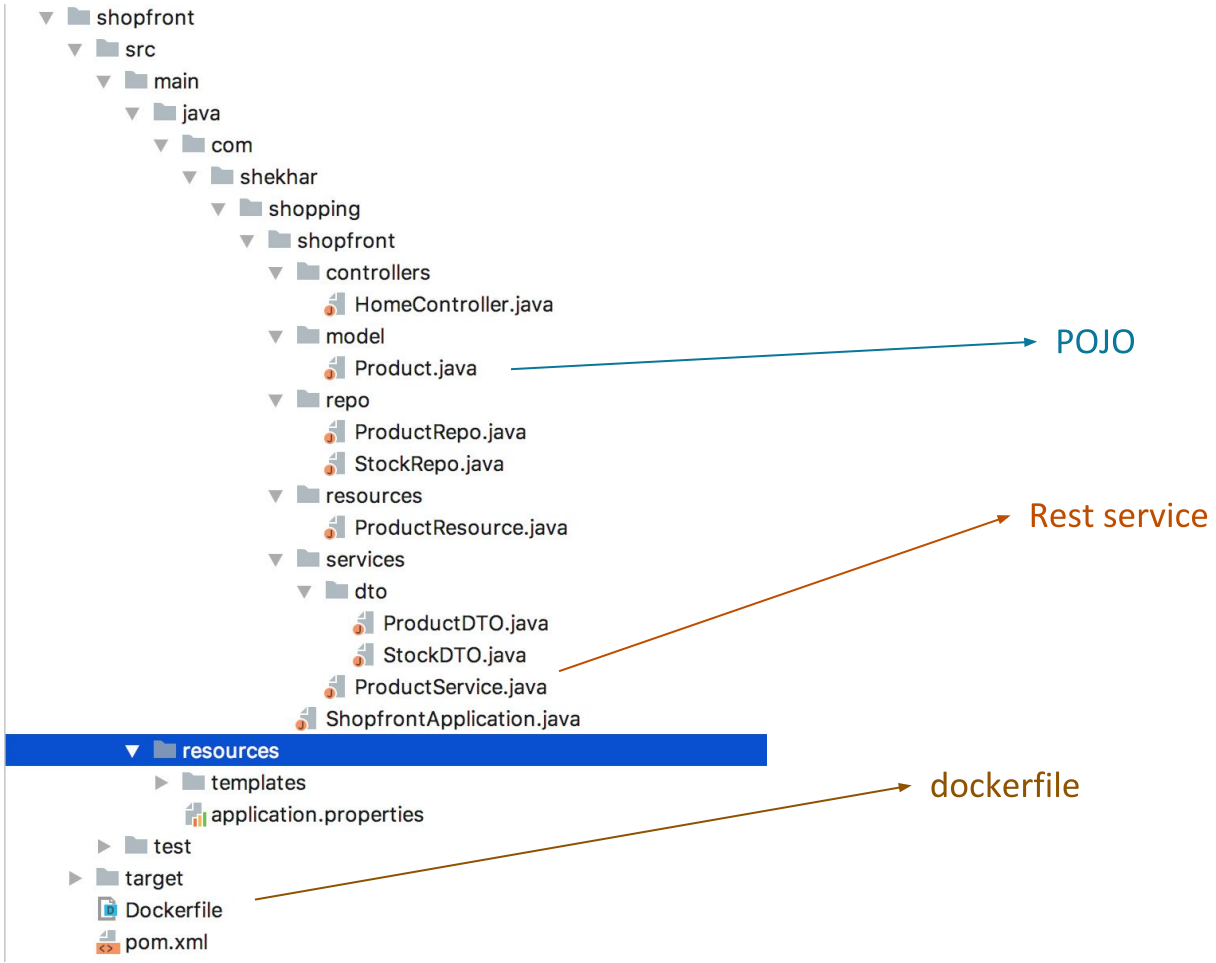
https://github.com/shekhar2010us/microservices_kubernetes_docker/tree/master/productcatalogue

Project Structure – stockmanager Service



https://github.com/shekhar2010us/microservices_kubernetes_docker/tree/master/stockmanager

Project Structure – shopfront Service



https://github.com/shekhar2010us/microservices_kubernetes_docker/tree/master/shopfront

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
- stock manager 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/shekhar2010us/microservices_kubernetes_docker/blob/master/setup.md

Converting Monolithic to Microservices - Java

CLASSROOM WORK 103 (75 minutes)

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

- https://github.com/shekhar2010us/microservices_kubernetes_docker/blob/master/setup.md

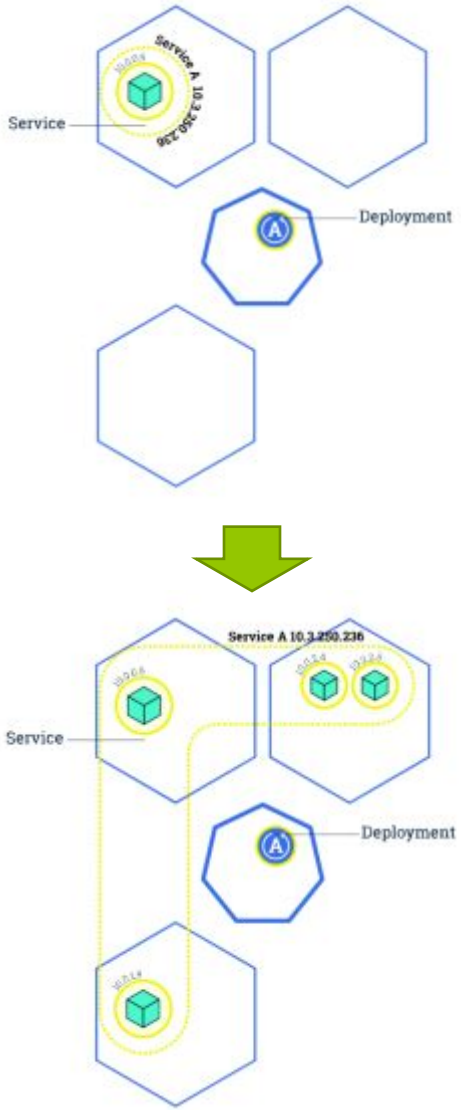
Successfully converted a Monolith to Microservices and deployed in Kubernetes

Part 3:
Microservices in Production

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>

$ kubectl scale deploy/nginx --replicas=4

```
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. Many a new user has repeated deleted pods and gotten frustrated when they respawn (as the deployments/replicasets are programed to do).

```
root@ip-172-31-0-112:~/code# kubectl run nginx --image nginx --port 80
deployment.apps/nginx created
root@ip-172-31-0-112:~/code#
root@ip-172-31-0-112:~/code# kubectl get deployments
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx              1         1         1            1           8s
nginx-deployment   3         3         3            3           17m
redis              1         1         1            1           33m
root@ip-172-31-0-112:~/code#
root@ip-172-31-0-112:~/code# kubectl scale deployments/nginx --replicas=3
deployment.extensions/nginx scaled
root@ip-172-31-0-112:~/code# kubectl get deployments
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx              3         3         3            3           34s
nginx-deployment   3         3         3            3           18m
redis              1         1         1            1           33m
root@ip-172-31-0-112:~/code# kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
nginx-6f858d4d45-b4cv2              1/1       Running   0          42s
nginx-6f858d4d45-g42mg              1/1       Running   0          16s
nginx-6f858d4d45-l69jv              1/1       Running   0          16s
nginx-apparmor                      1/1       Running   0          30m
nginx-deployment-67594d6bf6-7v5lk   1/1       Running   0          12m
nginx-deployment-67594d6bf6-fdqx2   1/1       Running   0          18m
nginx-deployment-67594d6bf6-sx62p   1/1       Running   0          18m
redis-7869f8966-sq8xm               1/1       Running   0          33m
```

Kubernetes: AutoScaling

HPA – Horizontal Pod Autoscaler:

Based on memory and CPU utilization by a pod, autoscaling scales up or down the number of pods

$ kubectl autoscale deployment nginx --min=2 --max=10

$ kubectl autoscale deployment nginx-deployment --max=5 --cpu-percent=80

$ kubectl get hpa

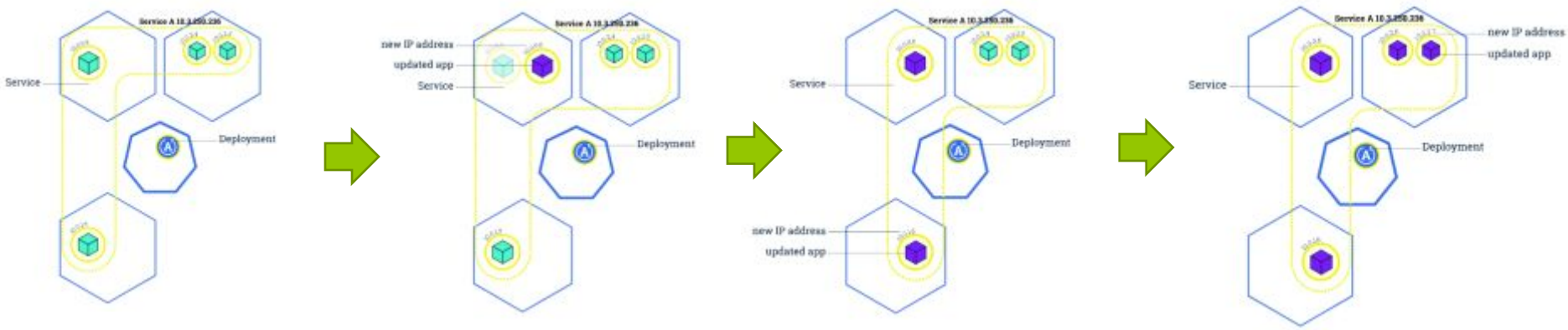
$ kubectl get hpa

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
nginx	Deployment/nginx	<unknown>/80%	1	4	3	12m
nginx-deployment	Deployment/nginx-deployment	<unknown>/80%	2	5	0	5s

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



$ kubectl set image <deployment> <new-image>

```
peter@Azure:~$ kubectl run nginx --image nginx:1.12.1 --port 80
deployment "nginx" created
peter@Azure:~$ kubectl set image deployments/nginx nginx=nginx:latest
deployment "nginx" image updated
peter@Azure:~$ kubectl describe deployment nginx
Name:                   nginx
Namespace:              default
CreationTimestamp:      Fri, 15 Sep 2017 13:08:50 +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:  1 max unavailable, 1 max surge
Pod Template:
  Labels:       run=nginx
  Containers:
   nginx:
    Image:              nginx:latest
    Port:               80/TCP
    Environment:        <none>
    Mounts:             <none>
  Volumes:              <none>
Conditions:
  Type          Status  Reason
  ----          ------  ------
  Available     True    MinimumReplicasAvailable
OldReplicaSets: <none>
NewReplicaSet:  nginx-2688028062 (1/1 replicas created)
Events:
  FirstSeen     LastSeen        Count   From                    SubObjectPath   Type            Reason                  Message
  ---------     --------        -----   ----                    -------------   --------        ------                  -------
  32s           32s             1       deployment-controller                   Normal          ScalingReplicaSet       Scaled up replica set nginx-1295584306 to 1
  10s           10s             1       deployment-controller                   Normal          ScalingReplicaSet       Scaled up replica set nginx-2688028062 to 1
  10s           10s             1       deployment-controller                   Normal          ScalingReplicaSet       Scaled down replica set nginx-1295584306 to 0
```

Note: Rolling updates can also be undone (see next slide)

$ kubectl rollout undo <deployment>

```
peter@Azure:~$ kubectl rollout undo deployments/nginx
deployment "nginx" rolled back
peter@Azure:~$ kubectl describe deployments/nginx
Name:                   nginx
Namespace:              default
CreationTimestamp:      Fri, 15 Sep 2017 13:36:49 +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:  1 max unavailable, 1 max surge
Pod Template:
  Labels:       run=nginx
  Containers:
   nginx:
    Image:              nginx:1.12.1
    Port:               80/TCP
    Environment:        <none>
    Mounts:             <none>
  Volumes:              <none>
Conditions:
  Type          Status  Reason
  ----          ------  ------
  Available     True    MinimumReplicasAvailable
OldReplicaSets: <none>
NewReplicaSet:  nginx-1295584306 (1/1 replicas created)
Events:
  FirstSeen     LastSeen        Count   From                    SubObjectPath   Type            Reason              Message
  ---------     --------        -----   ----                    -------------   --------        ------              -------
  1m            1m              1       deployment-controller                   Normal          ScalingReplicaSet   Scaled up replica set nginx-2688028062 to 1
  1m            1m              1       deployment-controller                   Normal          ScalingReplicaSet   Scaled down replica set nginx-1295584306 to 0
  1m            10s             2       deployment-controller                   Normal          ScalingReplicaSet   Scaled up replica set nginx-1295584306 to 1
  10s           10s             1       deployment-controller                   Normal          DeploymentRollback  Rolled back deployment "nginx" to revision 1
  10s           10s             1       deployment-controller                   Normal          ScalingReplicaSet   Scaled down replica set nginx-2688028062 to 0
```

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

```
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>

Exercise 4 - (scaling, Auto-scaling, roll out)

- For a previously created deployment scale the number of replicas
- see the difference in replicasets, pods , and see with a describe
- Rolling update, undo and see the changes in replica sets

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_scaling_roll_out.md

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps
https://container-solutions.com/kubernetes-deployment-strategies/
https://github.com/ContainerSolutions/k8s-deployment-strategies
https://container-solutions.com/deployment-strategies/

Deployment Strategies

- Recreate
- Ramped (Similar to Rolling Updates)
- Blue - Green Deployment
- Canary Deployment
- A/B Testing
- Shadow Deployments

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps
https://container-solutions.com/kubernetes-deployment-strategies/
https://github.com/ContainerSolutions/k8s-deployment-strategies
https://container-solutions.com/deployment-strategies/

Kubernetes: Edit

Edit the configuration of a running entity

```
[root@ip-172-31-68-96:~# kubectl get deploy
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx              4         4         4            4           28m
nginx-deployment   2         2         2            2           37m
[root@ip-172-31-68-96:~# kubectl edit deploy/nginx
deployment.extensions/nginx edited
[root@ip-172-31-68-96:~# kubectl get deploy
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx              2         2         2            2           28m
nginx-deployment   2         2         2            2           38m
root@ip-172-31-68-96:~#
```

How a real world deployment looks like we can do a demo -

$ kubectl create -f guestbook.yaml

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

Kubernetes: Pod Priority

Set the priority for pod scheduling, cannot ask never to kill a pod

Example PriorityClass

```yaml
apiVersion: scheduling.k8s.io/v1alpha1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000
globalDefault: false
description: "This priority class should be used for XYZ service pods only."
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  priorityClassName: high-priority
```

Basic Troubleshooting

- Describe is your friend , start from deployment,
- check for logs of the specific pods using kubectl logs pod_name
- kubectl exec -it pod_name /bin/bash - if there are multiple containers in a single pod, we may need to use -c container_name (like helloworld)

```
service/kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP        158d
Abhiks-MacBook-Pro:helper_yaml_files abhikbanerjee$ kubectl describe deploy blue
Name:                   blue
Namespace:              default
CreationTimestamp:      Sun, 16 Sep 2018 21:39:06 -0700
Labels:                 app=blue
Annotations:            deployment.kubernetes.io/revision=1
                        kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"apps/v1beta1","kind":"Deployment","metadata":{"annotations":
{},"name":"blue","namespace":"default"},"spec":{"replicas":3,"selector":{"ma...
Selector:               app=blue
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=blue
  Containers:
   blue:
    Image:        karthequian/helloworld:latest
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Progressing    True    NewReplicaSetAvailable
  Available      True    MinimumReplicasAvailable
OldReplicaSets:  <none>
NewReplicaSet:   blue-79c5c746b7 (3/3 replicas created)
Events:          <none>
Abhiks-MacBook-Pro:helper_yaml_files abhikbanerjee$
```

Basic Troubleshooting - An example

- $ kubectl create -f https://raw.githubusercontent.com/abhikbanerjee/Kubernetes_Exercise_Files/master/helper_yaml_files/Ex_common_debugging/helloworld-with-bad-pod.yaml
- $ kubectl get po,deploy,svc
- $ kubectl describe po/<pod_name>
- $ kubectl logs <pod_name>
- $ kubectl exec -it <pod_name> /bin/bash - won't run
- $ ps -ef
- $ kubectl exec -it <pod_name> -c helloworld /bin/bash
- $ kubectl get <pod_name> -o wide, json, yaml (can use -l option)

  Use sample_retail_infra.yaml
- $ kubectl get pods --sort-by=.metadata.name -o wide --all-namespaces
- $ kubectl get pods -o=jsonpath"{..images}" -l env=staging -n cart
- $ kubectl delete pods -n cart --all (delete all the pods in cart namespace, using --all we want to be sure we want to delete everything)
- $ kubectl get pods -n cart -w (watch the deletion process for a while)
- we can add **--grace-period=0 --force** (doesn't wait for the pod to do the clean up , and force deletes the pods)

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

Kubernetes: Probes

Kubernetes Probe - The kubelet uses liveness probes and readiness probe to check status of containers.

Kubernetes uses 2 different Probes :-

- **Liveness Probe** : uses liveness probes to know when to restart a Container. Liveness probe could catch a deadlock, where an application is running, but unable to make progress. Restarting a Container in such a state can help to make the application more available despite bugs.
- **Readiness Probe** : The kubelet uses readiness probes to know when a Container is ready to start accepting traffic. A Pod is considered ready when all of its Containers are ready. One use of this signal is to control which Pods are used as backends for Services. When a Pod is not ready, it is removed from Service load balancers

Exercise 5 - (Liveness and Readiness Probes)

- learn the concepts of Liveness and Readiness Probes.
- See the example of CrashLoopBackOff

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_liveness_readiness_probe.md

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

Kubernetes: Volumes

On-disk files in a container are the simplest place for an application to write
data, but this approach has drawbacks. The files are lost when the container crashes or stops for any other reason. Furthermore, files within a container are inaccessible to other containers running in the same Pod. The Kubernetes Volume abstraction addresses both of these issues.

- A directory accessible to all containers in a pod
- Volumes out live containers that run within a pod
- May be passed between pods
- Data is preserved across container restarts

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 Volume

Persistent storage in a cluster

- Separate from **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

Kubernetes: Persistent Volumes

- PV: Storage in the cluster that has been provisioned by an administrator (static or dynamic) and is independent of any individual pod that uses the PV.

Following volumes are supported by Kubernetes:

GCEPersistentDisk ; AWSElasticBlockStore

AzureFile ; AzureDisk

FC (Fibre Channel) ; FlexVolume

Flocker ; NFS

iSCSI ; RBD (Ceph Block Device)

CephFS ; Cinder (OpenStack block storage)

Glusterfs ; VsphereVolume

Portworx Volumes ; ScaleIO Volumes

StorageOS

Kubernetes: Persistent Volumes

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
```

Kubernetes: Persistent Volume Claims

- PVC: Request for a storage by a user, it is similar to pod and it can request specific size and access modes (e.g., can be mounted once read/write or many times read-only).

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/helper_yaml_files/pvc.yaml

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/helper_yaml_files/deploy_pvc.yaml

Exercise 6 - (Volume, PV, PVC)

- create pods,PV and PVC
- Experiment with creating end to end application, mount volume, read from shared pod
- Fix a bug

6.1 Exercise of Volume

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_Creating_Volume.md

6.2 Exercise on PV, PVC

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_Creating_Persistent_Volume.md

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.

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

Exercise 7 - Kubernetes Logging

- Create a logging pd
- View the logs
- Create one-shot pod

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_Kubernetes_Logging.md

Exercise 8 - ElasticSearch example

- Real world example for hosting ElasticSearch (ES)
- Deploy , scale the ES application
- Create Index and Search

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_ElasticSearch_Kubernetes.md

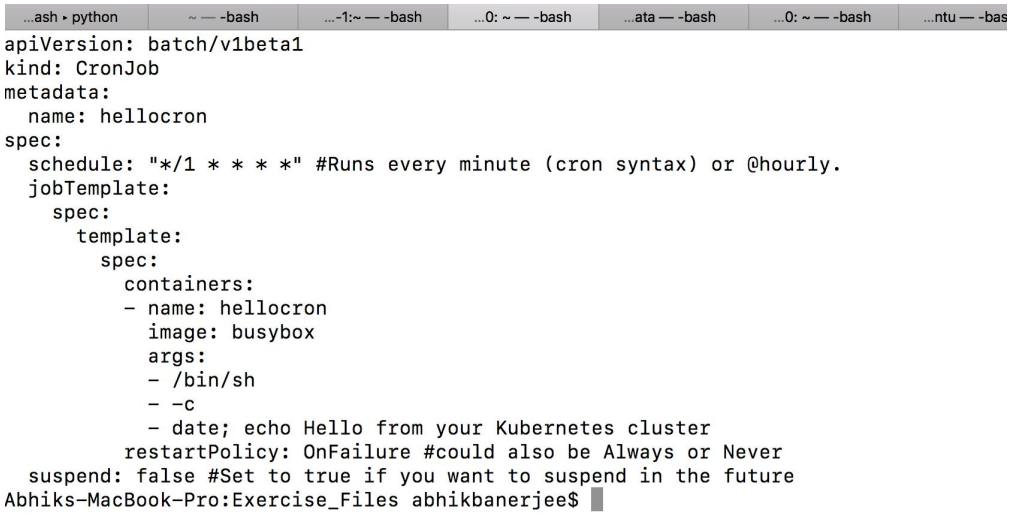
Kubernetes: Jobs and Cron Jobs

Jobs - Creates one or more pods and ensures that a specified number of them successfully terminate. As pods successfully complete, the *job* tracks the successful completions. When a specified number of successful completions is reached, the job itself is complete. Deleting a Job will cleanup the pods it created.

Cron Job - A *Cron Job* creates Jobs on a time-based schedule. One Cron Job object is like one line of a *crontab* (cron table) file. It runs a job periodically on a given schedule

Kubernetes: Jobs and Cron Jobs

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi-with-timeout
spec:
  backoffLimit: 5
  activeDeadlineSeconds: 100
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
```



Exercise 9 - Jobs and Cron jobs

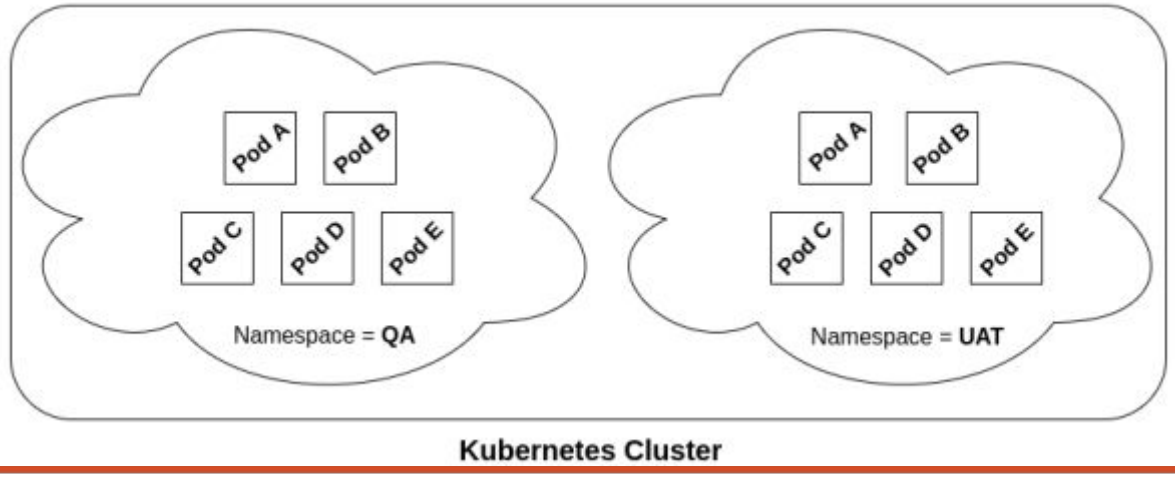
- Create Job and check the logs
- Create a Cron Job,
- Edit the Cron Job

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_job_cronjob.md

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

Kubernetes: Namespace

- Resources are segmented within namespaces
  - Sandbox per developer for example
- Pods will route dns to default (own) namespace
- The same app could run independently in different namespaces
- Namespaces create an excellent mechanism to subdivide resources and provide isolation. This enables using a cluster for multiple projects or multiple teams



Kubernetes: Namespace

- $ kubectl get po,deploy,svc,rs --all-namespaces
- $ kubectl create -f simple_pod.yaml -n abhik_namespace
- $ kubectl get po -o wide
- $ kubectl get po -o wide -n abhik_namespace
- $ kubectl delete po simple_pod
- $ kubectl delete po simple_pod -n abhik_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
```

Exercise 12 - Namespace

- Create a namespace
- check various objects in different namespaces
- Can create or delete objects from a specific namespace

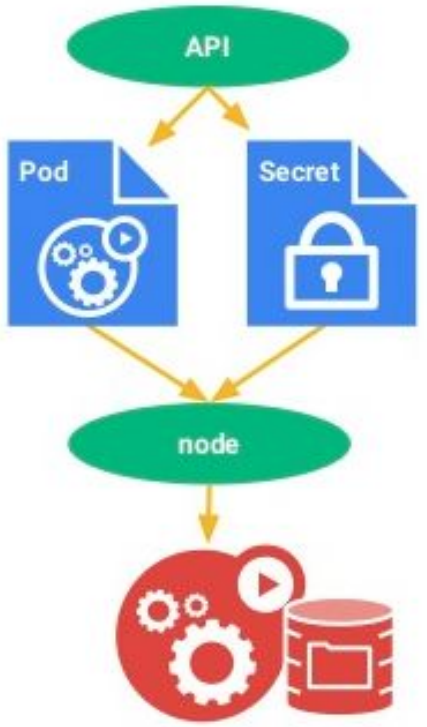
Can use previous examples

- $ kubectl get po,deploy,svc,rs --all-namespaces
- $ kubectl create -f simple_pod.yaml -n abhik_namespace
- $ kubectl get po -o wide
- $ kubectl get po -o wide -n abhik_namespace
- $ kubectl delete po simple_pod
- $ kubectl delete po simple_pod -n abhik_namespace
- $ kubectl api-resources --namespaced=false (shows resources which don't fall under a namespace)

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

$ kubectl create **secret** generic apikey --from-literal=api_key=1234567 (create secrets for passwords)

$ kubectl get secrets

$ **kubectl get secret/apikey -o yaml** and kubectl get secret/apikey lets us inspect the secret key

$ kubectl create -f secretreader-deployment.yaml, check if the deployment and pods are running

$ kubectl logs pod/secretreader-pod-name

$ echo -n <secret_base_64_key >| base64 --decode (the one we get from the -o yaml option)

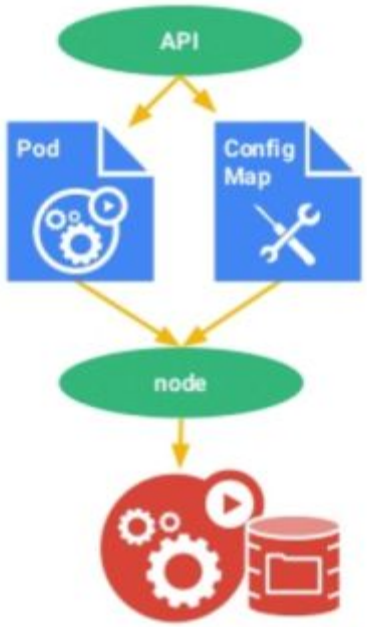
Exercise 13 - Secrets

- Create a Secret
- Check the secret from the object and from the logs
- Also check the base 64 decoder for the secret

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_Application_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
```

Kubernetes: Config Map

```
root@ip-172-31-68-96:~# kubectl create configmap test-config --from-literal=key1=config1 --from-literal=key2=config2
configmap/test-config created
root@ip-172-31-68-96:~# kubectl get configmap
NAME          DATA      AGE
my-config     2         1m
test-config   2         7s
root@ip-172-31-68-96:~#
root@ip-172-31-68-96:~# kubectl describe configmap/test-config
Name:         test-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
key1:
----
config1
key2:
----
config2
Events:  <none>
root@ip-172-31-68-96:~#
```

Kubernetes: Config Map

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-cm
  labels:
    name: test-cm
data:
  key1: value1
  key2: value2
```

```
apiVersion: v1
  kind: Pod
  metadata:
    name: pod-with-cm
  spec:
    containers:
      - name: test-container
        image: k8s.gcr.io/busybox
        command: [ "/bin/sh", "-c", "env" ]
        env:
          - name: SPECIAL_LEVEL_KEY
            valueFrom:
              configMapKeyRef:
                name: test-cm
                key: key1
    restartPolicy: Never
```

Exercise 14 - Config Maps

- Create a deployment which uses hard coded value for log-level
- Create a Config Map
- Use the Config map inside a deployment

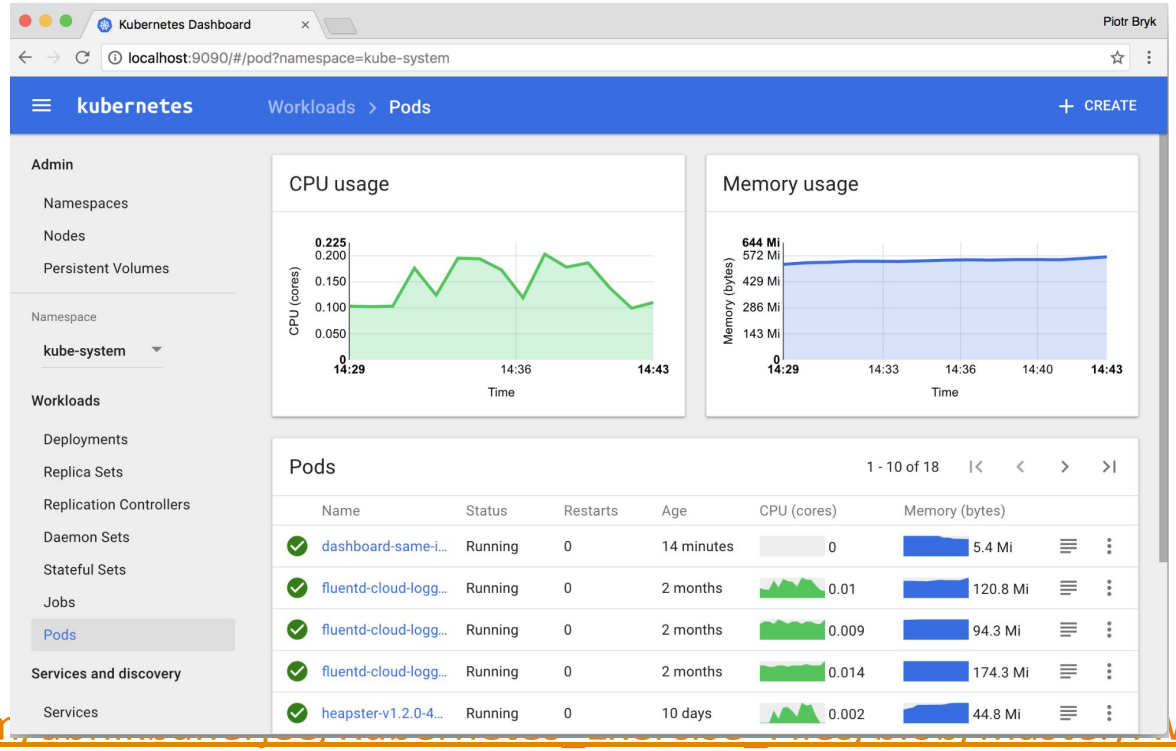
https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_configmap.md

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

Kubernetes Tools : Dashboard

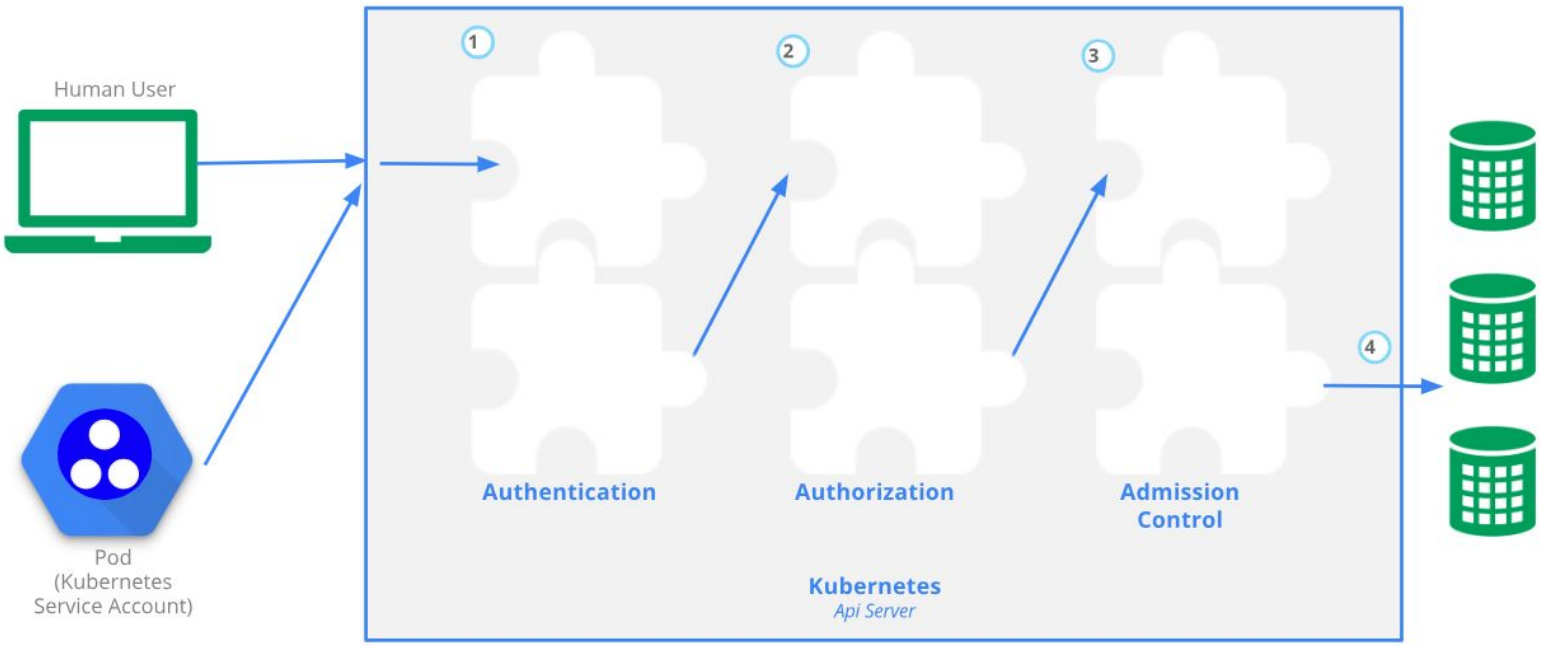
Kubernetes Dashboard - default add on provided by kubernetes

Easy to check deployments, edit deployments, scale deployments, and check cpu, memory usages



https://github.com/[illegible]/Kubernetes_Setup_AWS_setup_files/Kubernetes_Setup_AWS.md

Kubernetes Security



Ref:- https://kubernetes.io/docs/concepts/overview/object-management-kubectl/declarative-config/

Kubernetes Do's and Don'ts

- Decide what is the best setup for your kubernetes environment
- For easy provisioning and testing Minikube comes handy, so does Kubeadm
- Use namespaces and Labels for efficient usage of defining your objects
- Always use config files, or Declarative syntax depending on the teams updating configs
- Use Resource quotas , for better management of hardware resources - https://kubernetes.io/docs/concepts/policy/resource-quotas/
- For quick testing Node Ports are good, but Load Balancers and Ingress are better options for production level settings
- Logs should always go to PV's and for better management use PVCs

Kubernetes Do's and Don'ts

- Microservices Architecture follow the 12-step rule
- Always use controllers , rather than pods
- Always version control your code and config files
- if in confusion use fluentd for logging
- Cluster Federation helps to sync resources into multiple clusters , cross cluster discovery- **Kubefed**
- **Kompose** converts docker compose file into kubernetes objects like deployments, and services,
- **Helm** - streamlines installation and management of kubernetes cluster, its like the package manager for kubernetes, contains chart.yaml, template and values.yaml, provides 3 most important things - helps in creating **templates**, **versioning** and rolled back and tracked, **application packaging.**
- Kubernetes Documentation are the best resource whenever in doubt. - https://kubernetes.io/docs/reference/ and https://kubernetes.io/docs/concepts/

Course Evaluation

Please take a moment to complete your course evaluation which gives you immediate access to your certificate of completion.

All responses are confidential even though an email address is requested.

- **Go to http://www.metricsthatmatter.com/ASPE**
- **From the drop down menu choose Microservices Engineering Boot Camp, site, instructor name, class start date**
- **Fill out and submit the form**
- **It is available for 10 calendar days**

Thank You!