Orchestration: Containers

Gabriel Scalosub

Borrowed extensively from:

Elisha Rozensweig, Erez Biton

and various other papers/resources (see list at the end)

Recap: Containers

Introduction to Docker

- Software development platform
  - Open Source, written in Go
  - Packaging applications in containers
- Main features
  - Portability, platform-independent
    - Assuming docker support is available
  - Reuse and Sharing
    - Modify any “template”
    - Public/private repositories
  - Simple interfaces
    - CLI, RESTful API
- DevOps-oriented
  - Simplify process / reduce time from development to deployment

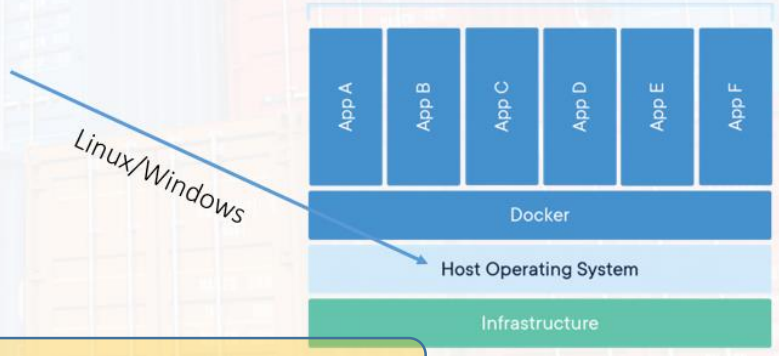


Image: Docker

- What is DevOps?

Recap: Containers

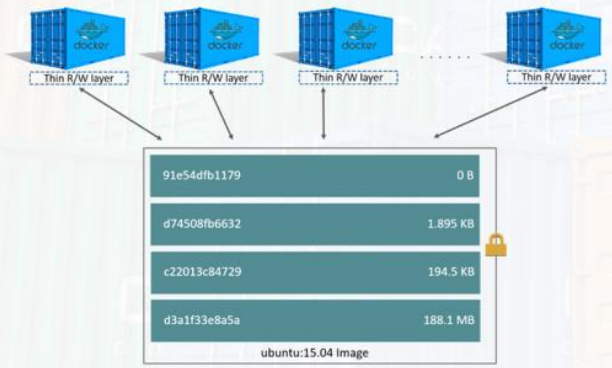
- What is DevOps?
- How can we spawn multiple copies of containers?

Introduction to Docker

Images and Containers

- Layers
  - Sequence of files generated while docker builds an image using Dockerfile
  - RO Layers can be shared by various containers

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```



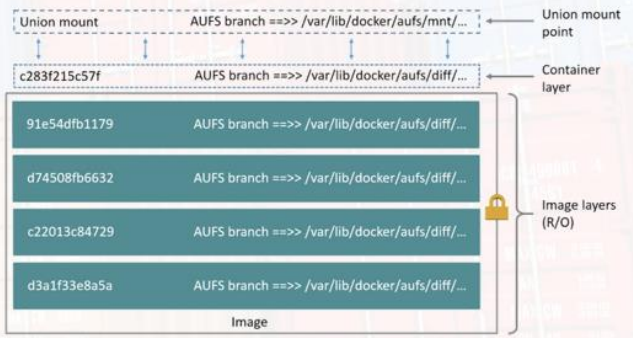


Image: Docker

Recap: Containers

- What is DevOps?
- How can we spawn multiple copies of containers?
- How can we deploy systems on multiple hosts? Manage?

Introduction to Docker

Images and Containers

Docker Compose

- Deploying an application using containers
  - Each container provides a distinct service
- Structure
  - Multiple containers, single host
  - `docker-compose.yml` file
- Example:
  - Web server with a database
    - Define images to use per container
    - Define networking

build web server using Dockerfile in current workdir

define container-host port mapping

```
version: '3'
services:
  web:
    build: .
    ports:
     - "5000:5000"
  redis:
    image: "redis:alpine"
```

Outline

- (Ultra quick) introduction to DevOps
- The task of Orchestration
  - (Some) contending suites
- Kubernetes (in some more depth)
  - Architecture
  - Specific components
- Load Balancing
- Miscellany

Outline

- (Ultra quick) introduction to DevOps
- The task of Orchestration
  - (Some) contending suites
- Kubernetes (in some more depth)
  - Architecture
  - Specific components
- Load Balancing
- Miscellany

Software Engineering

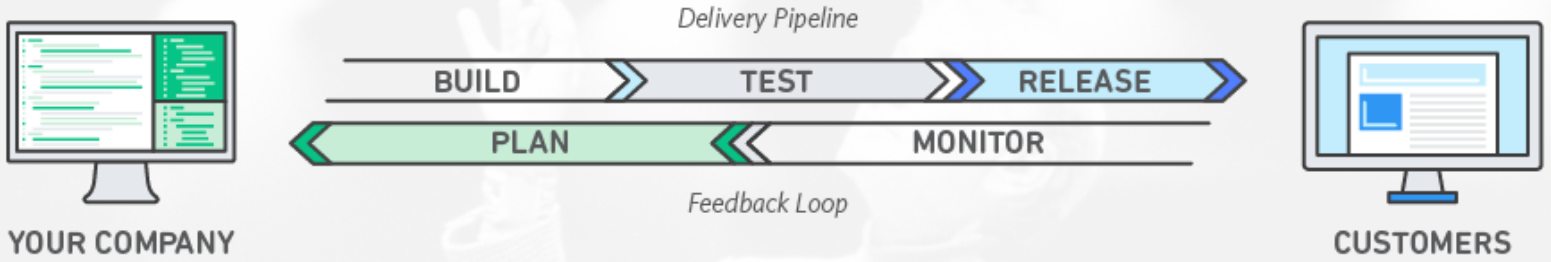
"The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software."

Bourque & Fairley (2014)

- Requirements (collection, analysis, specification)
- SW Design (architecture & components, interfaces)
- Implementation (writing code…)
- Testing (e.g., unit-testing of functions/classes/modules)
  - Existence, input-output behavior
- Operation
- Maintenance

Development

DevOps: Quick Introduction



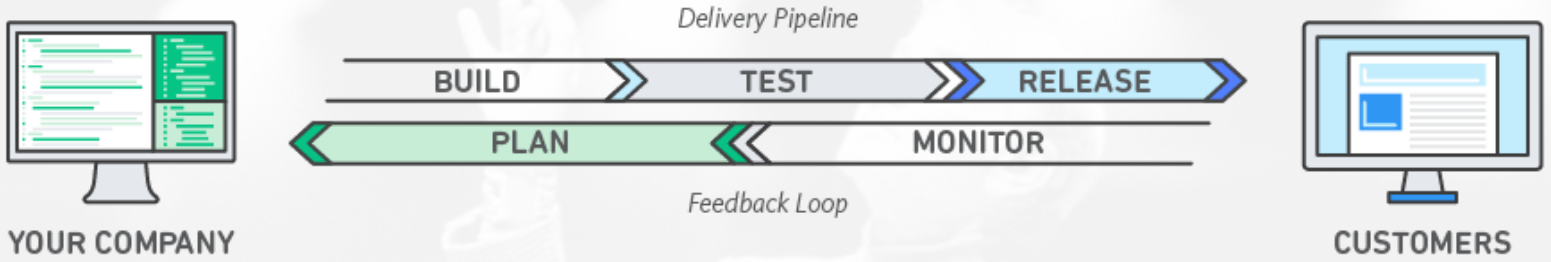
Source: Amazon

- DevOps = Development + Operations
- Development goal:
  - Produce innovation
  - Innovation reaches users fast
- Operations (IT) goal:
  - Users have good access to system
    - Stable, fast, responsive

"Come on! Let's add this feature! Get a move on!"

"Wait! It might not work! I have to check!"

DevOps: Quick Introduction

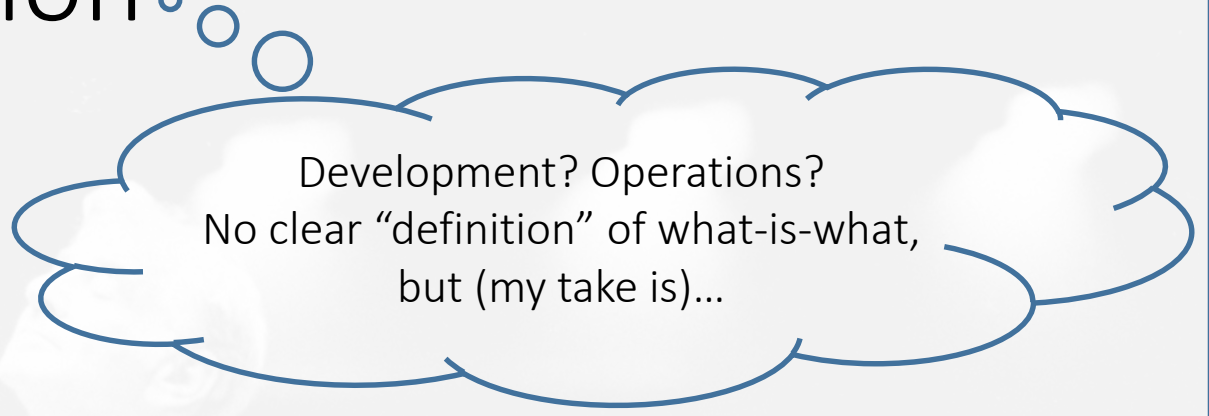


Source: Amazon

- DevOps = Development + Operations
- Development goal:
  - Produce innovation
  - Innovation reaches users fast
- Operations (IT) goal:
  - Users have good access to system
    - Stable, fast, responsive

Source: West Side Story (1961)

DevOps: Quick Introduction

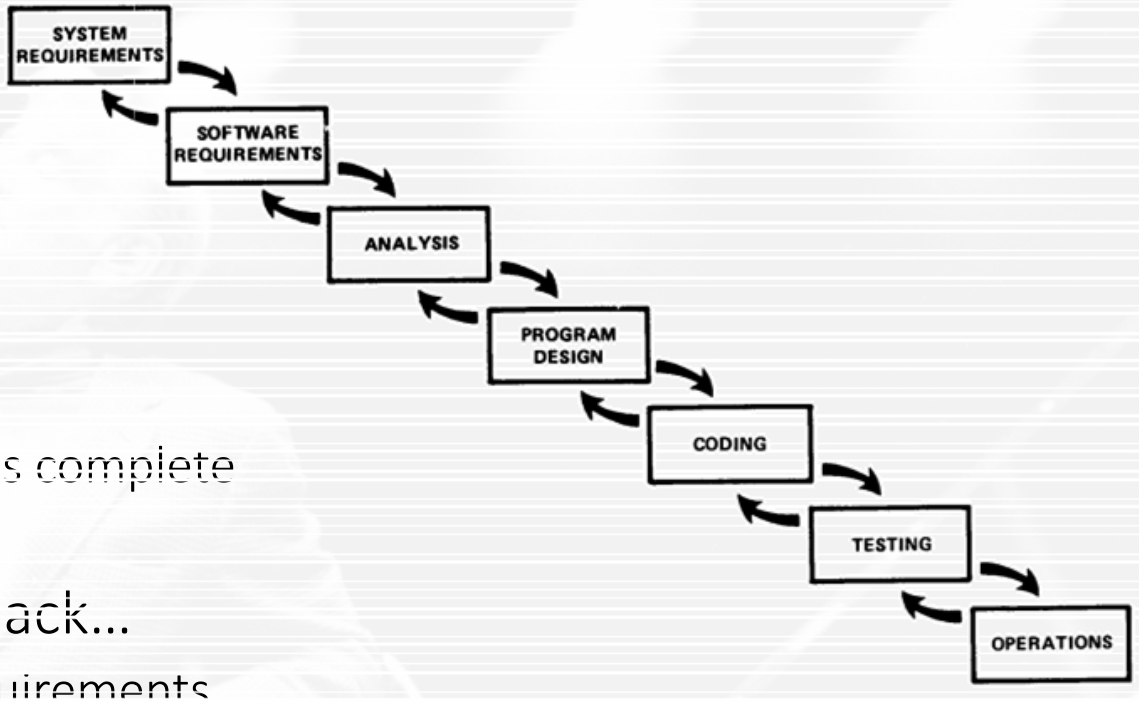


- Development
  - Main concern: business logic
  - New customer/product requirements → new features
    - Implementation through new code
  - Testing business logic
- Operations
  - Main concern: system logic / performance
  - Make sure system (predominantly production) works well at all times
    - Scaling, failures, updates, patching, costs…
    - New builds!!
  - Monitoring & Testing

DevOps: Development Models (Philosophy)

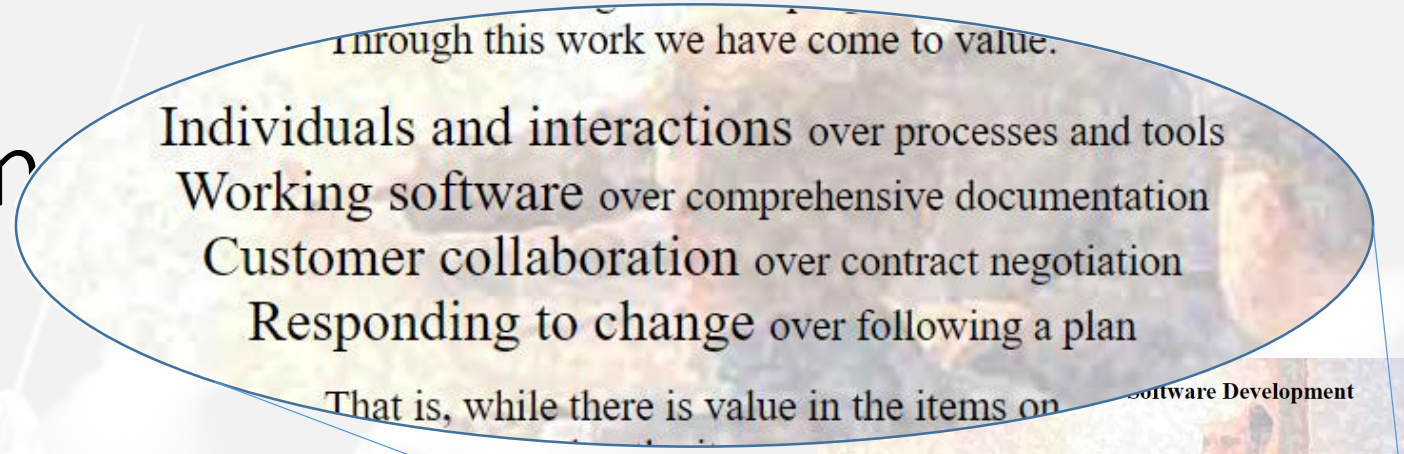
- How should it be done??

- Waterfall (1970)
  - Simple, clear, linear process
    - Move to next phase only after current phase is complete
    - Motivated by HW development process
  - Main handicap: problems → slow(!!) rollback…
    - E.g., failed test might imply reviewing the requirements
  - Rigid, heavy documentation, perfectionist…
    - Based on getting it all right in every step



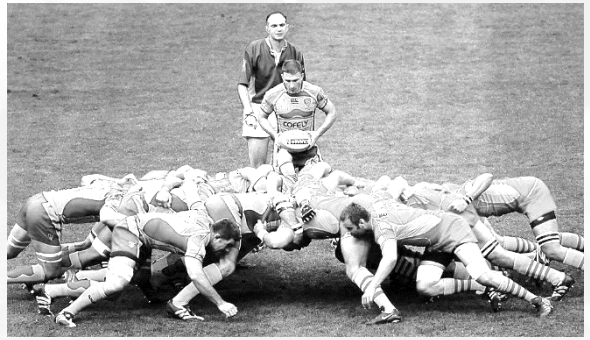
Source: Royce (1970)

DevOps: Developm



- Agile (2001)
  - Focus on flexibility, “embracing” change
    - Requirements, customer satisfaction
    - Unexpected limitations / constraints (incl. money/time)
  - Short “time to market”
    - Small features/improvements deployment
    - Short feedback loops (customer/product)
- Some related terms:
  - Scrum, sprint, standup, retrospective, backlog
    - Short (~2W) team efforts on specific “tasks”
  - CI/CD
    - Coming next…

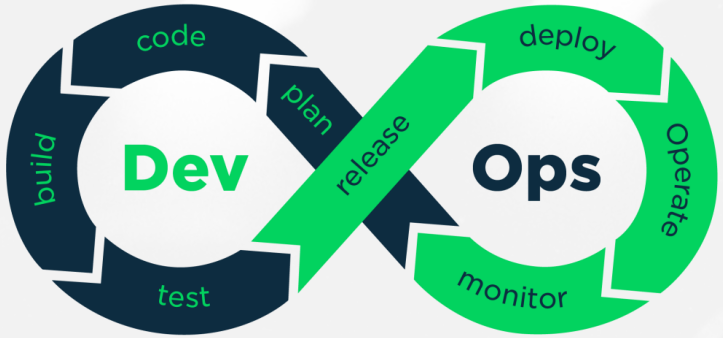
Source: agilemanifesto



Rugby scrum
Source: sportycious.com

DevOps: CI/CD

- Continuous Integration (CI)
  - Targets fresh code view to all developers
  - Constant (e.g., every few hours) merge-test cycles
    - Monitor codebase commits
    - Build
    - Perform various automated (unit, integration) tests on new version
  - Integration of (new) code into the main repository
- Continuous Delivery (CD)
  - Targets having SW always ready for release
    - Small cycles
    - Passing all tests
      - Usually in a **staging** environment (similar to production)
  - Manually authorized release
    - Automating release: Continuous Deployment



Source: testim.io

DevOps: Some Tools

- Integrated Development Environment (IDE)
  - Editor, debugging, build automation, test automation
  - Version control plugins, deployment plugins (VMs, containers), cloud integration
- Git
  - De-facto standard for code repositories version control
  - Distributed, private & public (e.g., GitHub) repositories
- Jenkins
  - Top tool for CI/CD
  - Open source automation server, running in containers
    - Build, test, report
  - Also used for orchestration
    - Infrastructure orchestration (puppet, chef, Ansible, etc.): stay tuned…

Outline

- (Ultra quick) introduction to DevOps
- The task of Orchestration
  - (Some) contending suites
- Kubernetes (in some more depth)
  - Architecture
  - Specific components
- Load Balancing
- Miscellany

What is Orchestration

- “Single task” automation
  - E.g.: launch server, connect to network, `apt-get update`
- Orchestration
  - Managing many(!!) automated tasks
  - Tasks built into pipelines and workflows
  - Related to multiple services that should be combined

Orchestration Tasks

- Provisioning and Deployment
- Configuration
- Fault tolerance and healing
- Scaling
- Load Balancing
- Monitoring
- Management

Automation

Orchestration: Infrastructure

- Resource provisioning
- Configuration management
  - Terraform, Ansible, puppet, chef, …
  - Describe desired state/action configuration
    - Using .yml, .json, or some specific description language
    - State/action:
      - package availability (e.g., ensure apache is installed and available)
      - service state (e.g., ensure sshd service is running, listening for incoming ssh requests)
      - execute commands (e.g., update via apt-get)
      - …
- Performed on any subset of infrastructure
- Infrastructure-as-Code (IaC): SW-based management

Terraform ANSIBLE CHEF puppet

```
case $operatingsystem {
  centos, redhat: { $service_name = 'ntpd' }
  debian, ubuntu: { $service_name = 'ntp' }
}

package { 'ntp':
  ensure => installed,
}

service { 'ntp':
  name      => $service_name,
  ensure    => running,
  enable    => true,
  subscribe => File['ntp.conf'],
}

file { 'ntp.conf':
  path    => '/etc/ntp.conf',
  ensure  => file,
  require => Package['ntp'],
  source  => "puppet:///modules/ntp/ntp.conf",
  # This source file would be located on the Puppet master at
  # /etc/puppetlabs/code/modules/ntp/files/ntp.conf
}
```

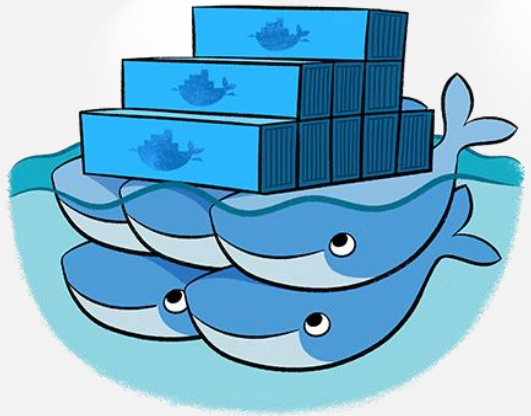
Example Puppet manifest. Source: Puppet

Orchestration: Containers

- CaaS: Container-as-a-Service
- Tasks
  - Deploy services
    - Container instances that may handle requests for service
    - Satisfy compute, storage, networking requirements
    - Replication requirement
  - Monitor and manage
    - Health, recovery, updates, scaling
  - Load balancing
    - Reverse-proxy, explicit load balancers

Containers Orchestration: (Some Contenders)

- Docker Swarm
  - Deploying an application/service using containers on several (virtual?) hosts
  - Part of the Docker framework
  - Fast deployment of containers, limited flexibility



- Apache Mesos Marathon
  - Running containers and varied types of applications
    - Not necessarily containerized, including big-data, analytics
  - Restricted networking capabilities

MESOS

- Kubernetes
  - Coming up next…

kubernetes

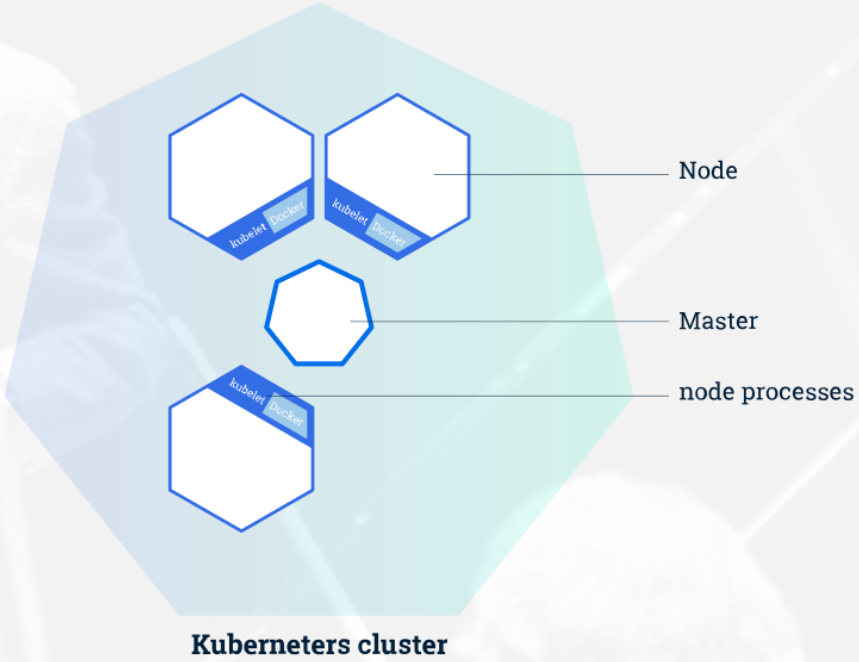
Outline

- (Ultra quick) introduction to DevOps
- The task of Orchestration
  - (Some) contending suites
- Kubernetes (in some more depth)
  - Architecture
  - Specific components
- Load Balancing
- Miscellany

Kubernetes*: Architecture

* Helmsman, pilot (Greek)

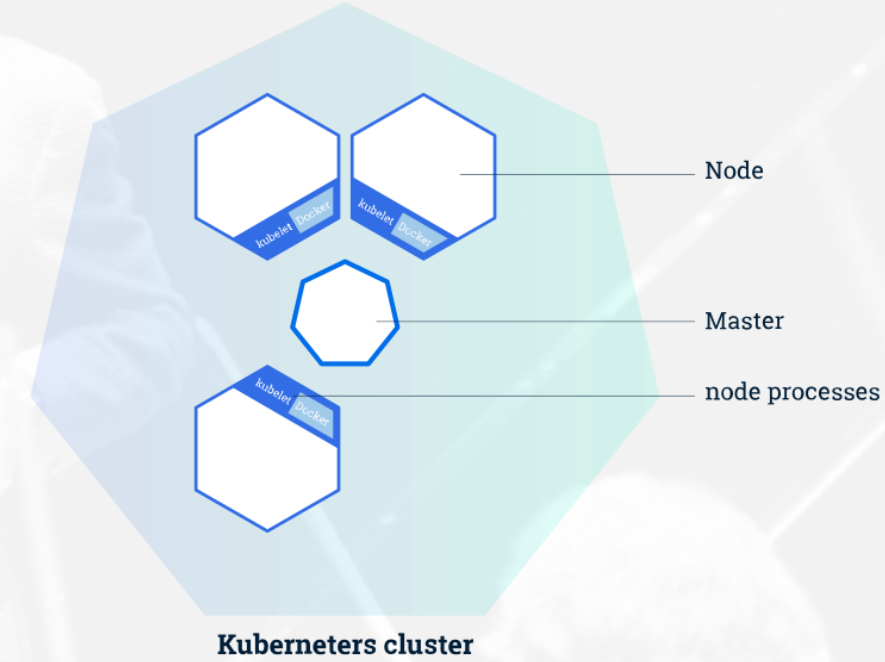
- Open-source platform for managing containerized services
  - Developed by Google, open-sourced at 2014
  - Follow-up on Google's Borg container management framework
  - K8s abbreviation ('k', 8 letters, 's')
- Key terms:
  - Cluster
  - Master
  - Node
  - Pod
  - (container…)
- Main API call:
  - `kubectl [command] [TYPE] [NAME] [flags]`



Source: Kubernetes

Kubernetes: Cluster

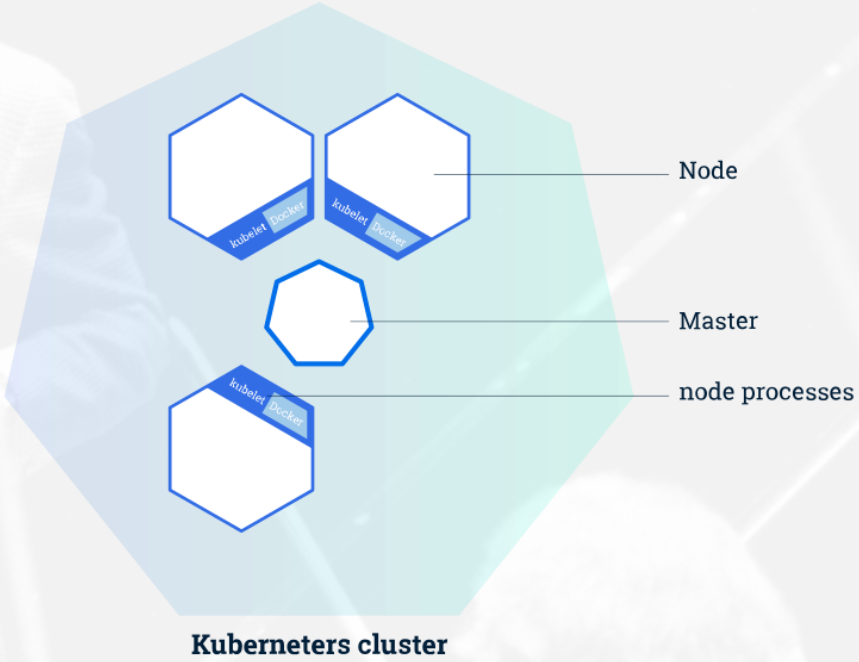
- Master
  - Manages the cluster
    - RESTful API
      - Via `kube-apiserver`
    - Scheduling applications
      - Via `kube-scheduler`
    - Maintaining application desired state
    - Scaling applications
    - Rolling out new updates



Source: Kubernetes

Kubernetes: Cluster

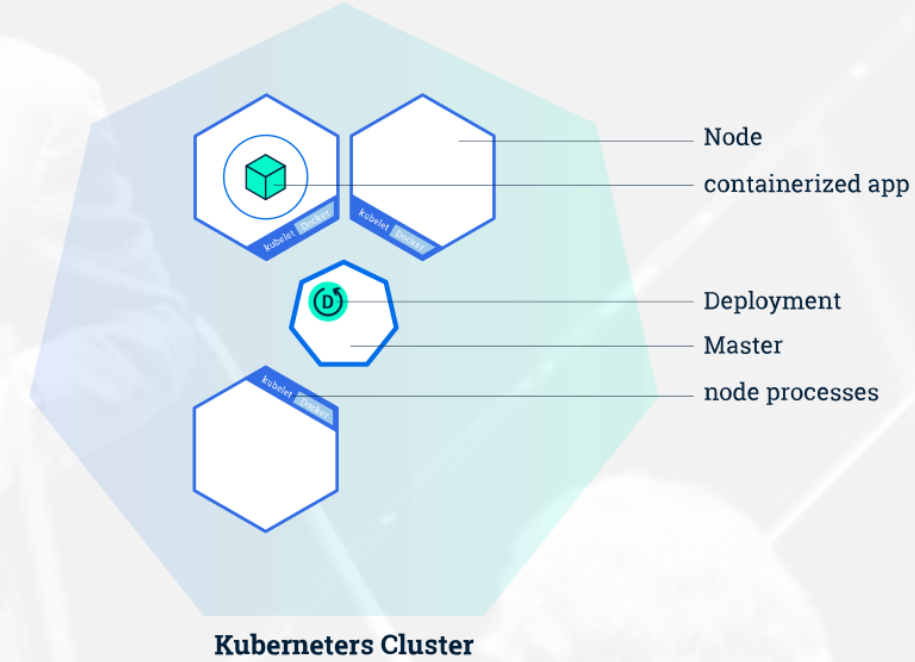
- Nodes
  - VM/physical machine running service(s)/application(s)
  - Components
    - Container management process (e.g., Docker)
    - `Kubelet`
      - Node management agent
      - Interacting with master via API
    - Network proxy
      - Via `kube-proxy`
      - Details to follow…



Source: Kubernetes

Kubernetes: Deployment (Basic)

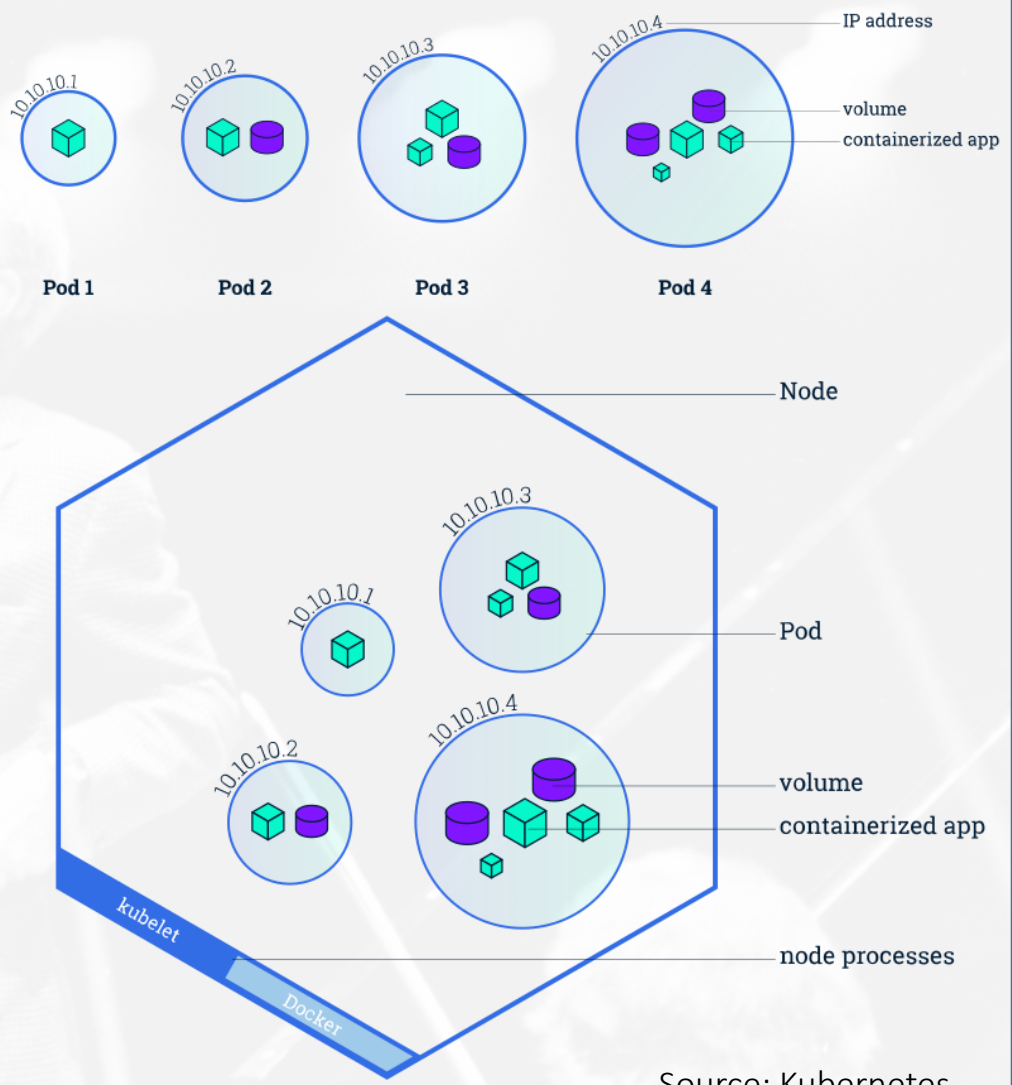
- Deployment
  - Creating and updating containers of service/application



Source: Kubernetes

Kubernetes: Pods

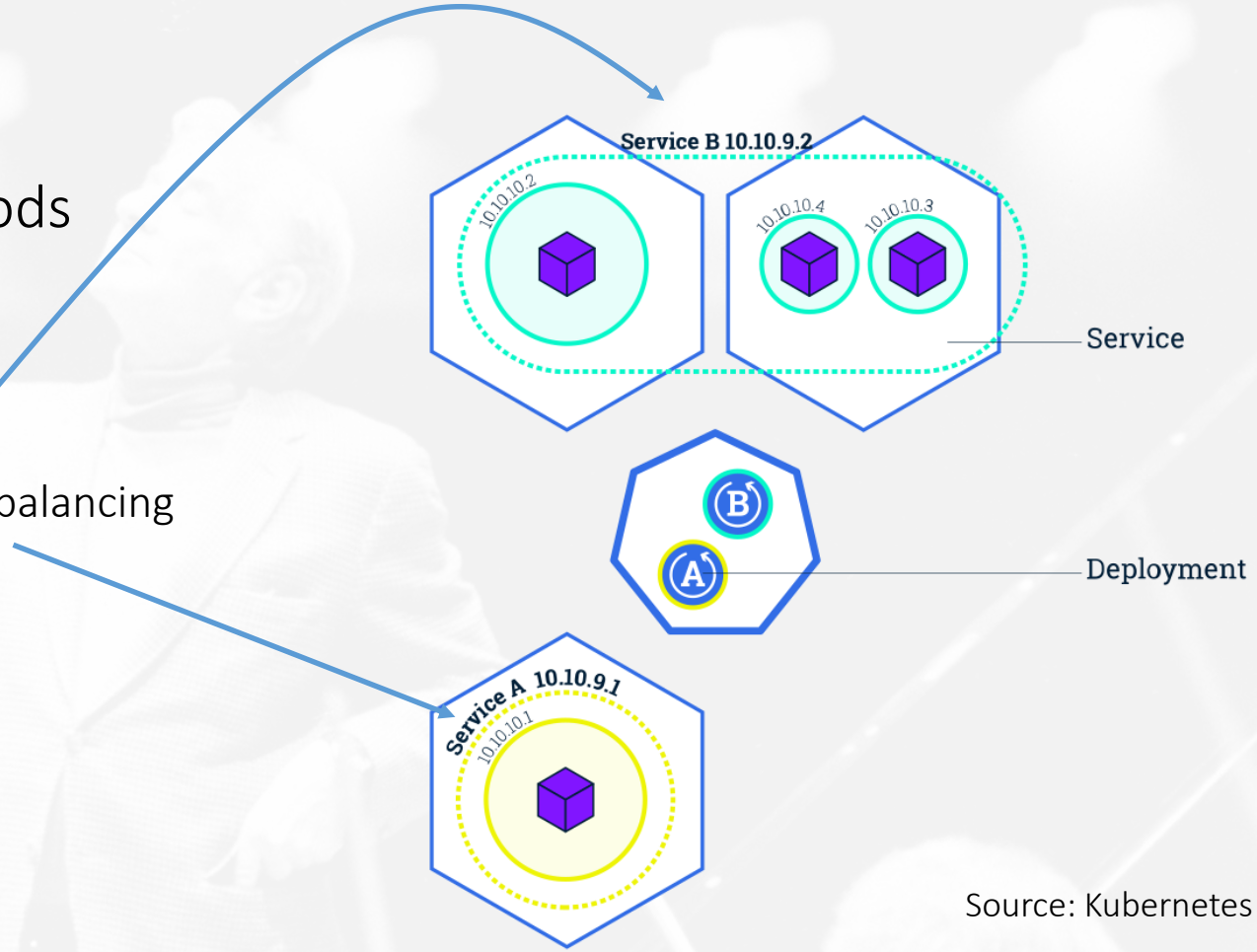
- Pod
  - Group of tightly coupled containers
  - Run on same node
  - A node may run multiple (independent) pods
  - Shared namespaces per pod
    - Unique IP across cluster
    - Possibly shared volume storage
  - Atomic deployment unit in Kubernetes
    - Often includes just a single container
    - Deployment actually spawns pods
      - Not specific containers



Source: Kubernetes

Kubernetes: Services

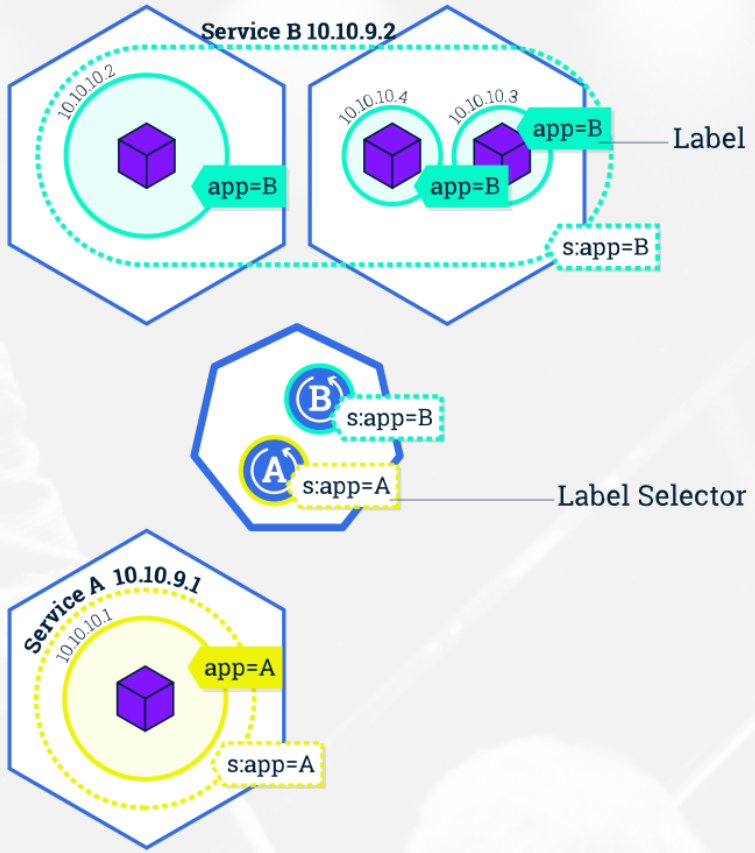
- Service
  - Abstraction layer for logical set of pods
    - Potentially on multiple nodes
  - Enables:
    - External traffic to/from pods
      - Via single exposed IP addresss + load balancing
      - Same port on all nodes (using NAT)
    - Load balancing
    - Service discovery
  - Example use case:
    - Service 1: frontend (e.g., webserver)
    - Service 2: backend (e.g., DB)
    - Services can communicate, abstracting away actual node deployment



Source: Kubernetes

Kubernetes: Labels and ReplicaSet

- Labels can be applied to group pods
  - Usually applied to pods in a service




```
"release" : "stable", "release" : "canary"

"environment" : "dev", "environment" : "qa", "environment" : "production"

"tier" : "frontend", "tier" : "backend", "tier" : "cache"

"partition" : "customerA", "partition" : "customerB"

"track" : "daily", "track" : "weekly"
```

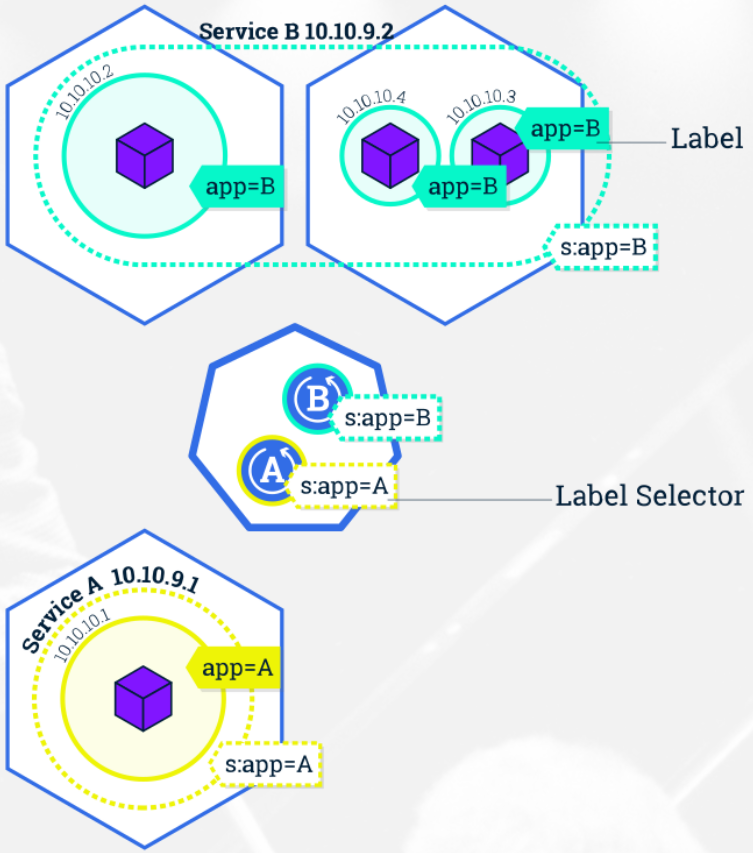
Source: Kubernetes

Kubernetes: Labels and ReplicaSet

- Labels can be applied to group pods
  - Usually applied to pods in a service
- Label selector
  - Enables defining sets of pods/services with certain label values
    - Equality-based
    - Set-based

```
environment = production
tier != frontend
```

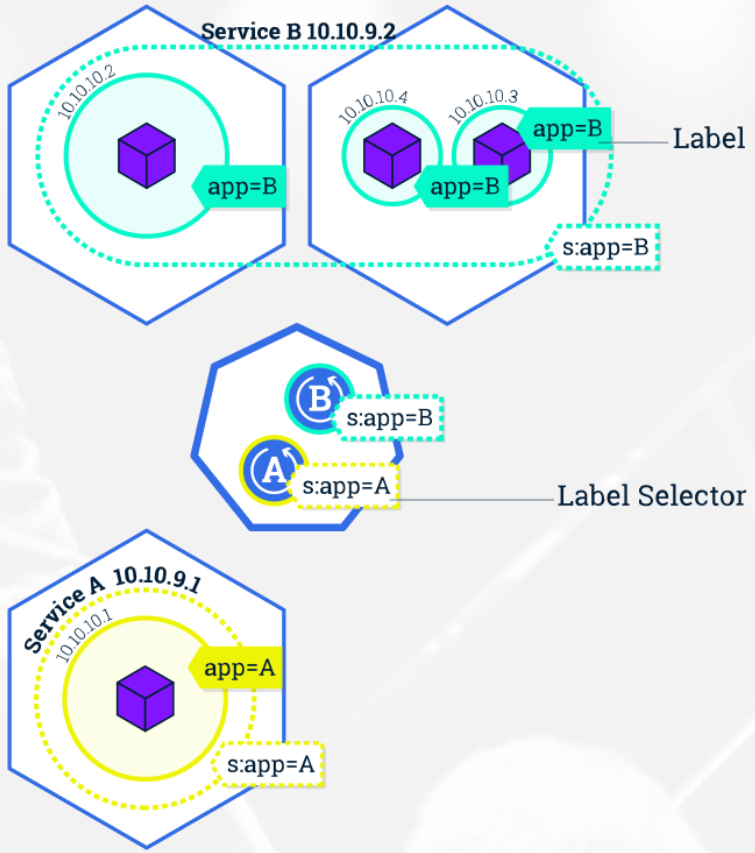
```
environment in (production, qa)
tier notin (frontend, backend)
partition
!partition
```



Source: Kubernetes

Kubernetes: Labels and ReplicaSet

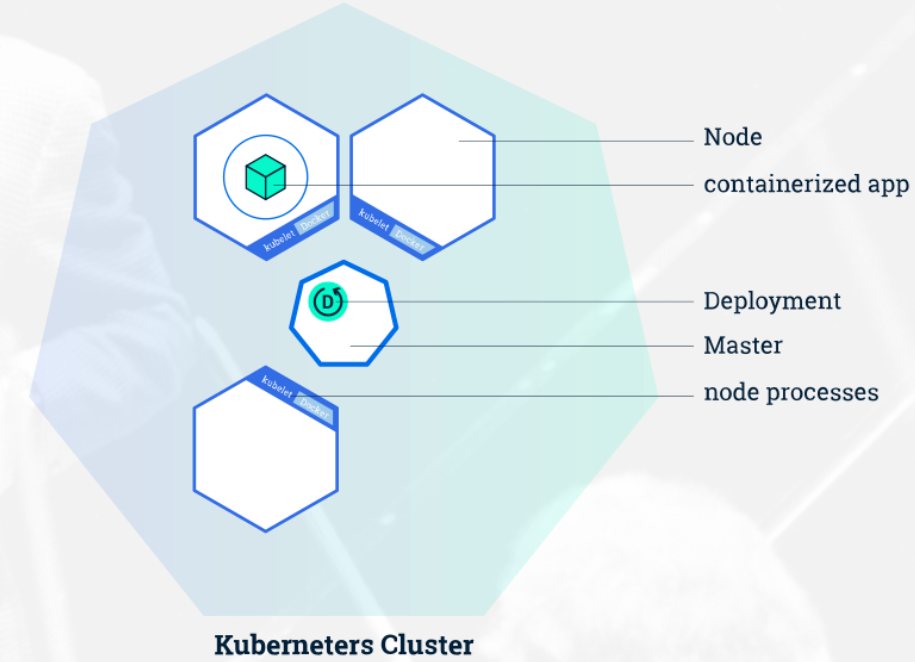
- Labels can be applied to group pods
  - Usually applied to pods in a service
- Label selector
  - Enables defining sets of pods/services with certain label values
    - Equality-based
    - Set-based
- ReplicaSet
  - Maintains a stable set of replica nodes running
  - Used to guarantees availability of service
    - Uses selector to define set
    - Defines number of replicas required



Source: Kubernetes

Kubernetes: Deployment (Revisited)

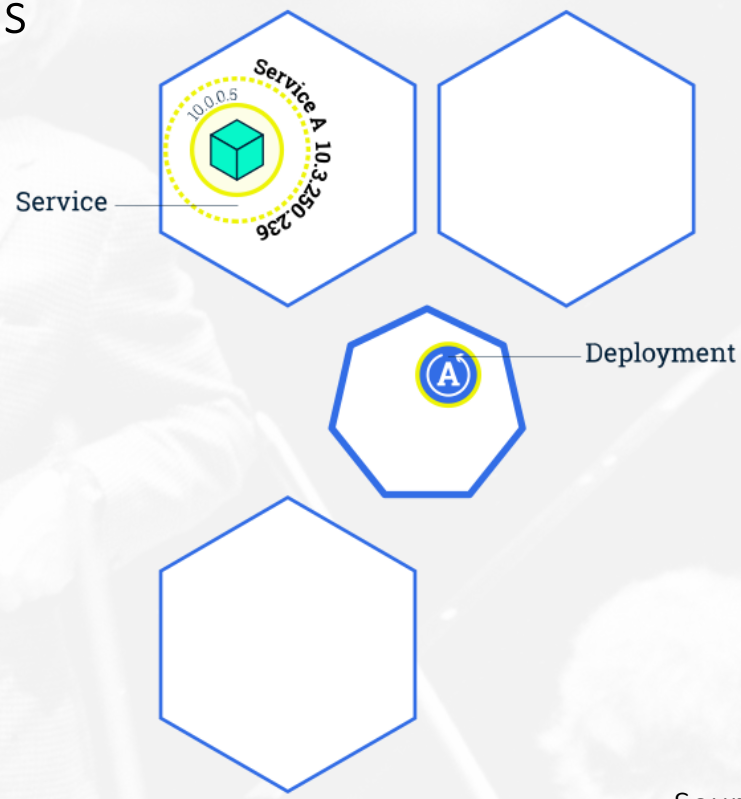
- Deployment
  - Creating and updating ~~containers~~ pods of service(s)/application(s)



Source: Kubernetes

Kubernetes: Deployment (Revisited)

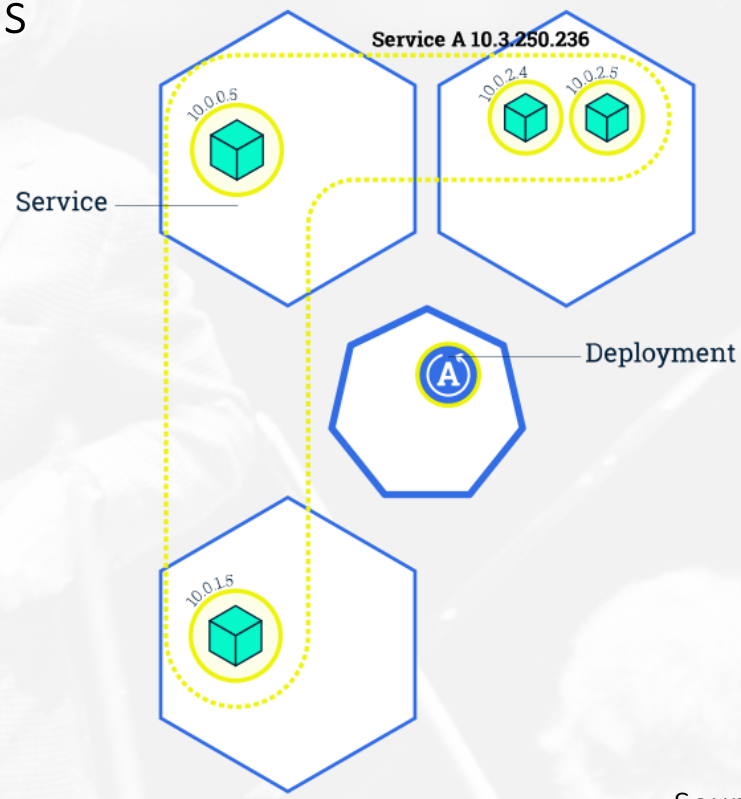
- Deployment
  - Creating and updating ~~containers~~ pods of service(s)/application(s)
  - Controller monitors/manages running instances
    - Examples
      - Scaling up/down a service
        - Increase/reduce number of replicas of pods
        - Also automatic:
          - [min,max], by load criteria (e.g., cpu)
      - Replace instances on failed node (self-healing)
    - Provides high-level ReplicaSet functionality
  - Load balancing only to available pods
  - Update deployment



Source: Kubernetes

Kubernetes: Deployment (Revisited)

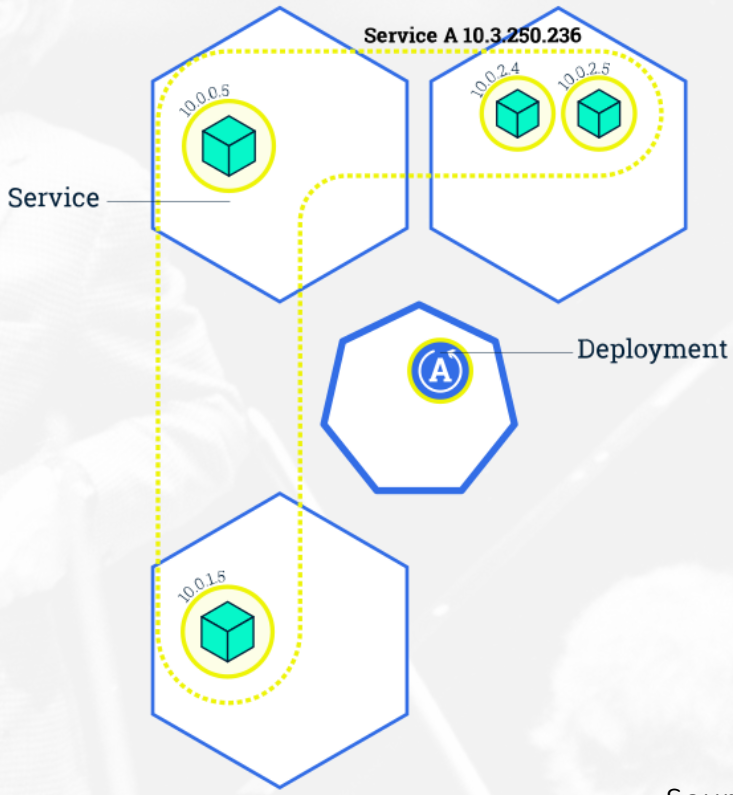
- Deployment
  - Creating and updating ~~containers~~ pods of service(s)/application(s)
  - Controller monitors/manages running instances
    - Examples
      - Scaling up/down a service
        - Increase/reduce number of replicas of pods
        - Also automatic:
          - [min,max], by load criteria (e.g., cpu)
      - Replace instances on failed node (self-healing)
    - Provides high-level ReplicaSet functionality
  - Load balancing only to available pods
  - Update deployment



Source: Kubernetes

Kubernetes: Rolling Updates

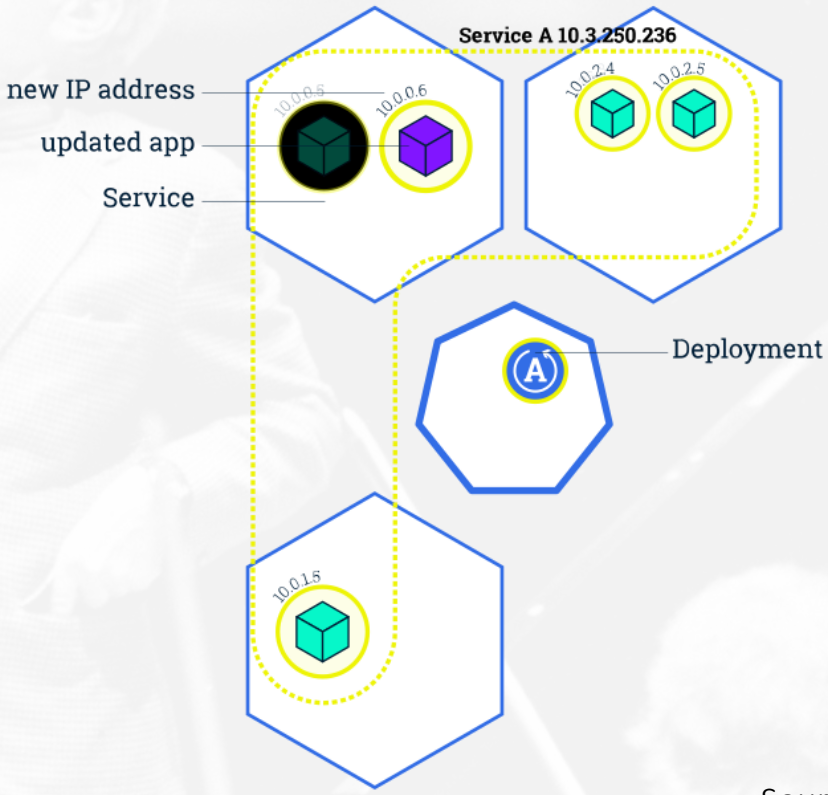
- Update pods
  - New application version (CI/CD)
  - Rollback to previous version
- Pods replaced incrementally
  - Default
    - one-by-one
    - Ensure at least 75% of replicas are available
      - “Make-before-break (too much)”
    - Ensure at most 125% of replicas are available
      - “Don’t-make-before-break (enough)”
- Load balancing only to available pods



Source: Kubernetes

Kubernetes: Rolling Updates

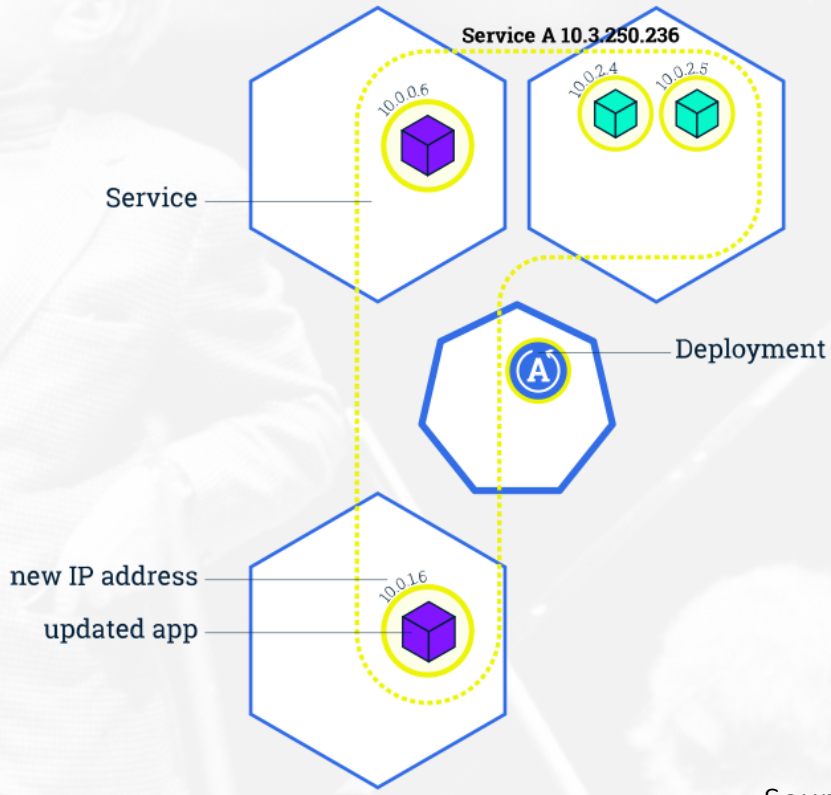
- Update pods
  - New application version (CI/CD)
  - Rollback to previous version
- Pods replaced incrementally
  - Default
    - one-by-one
    - Ensure at least 75% of replicas are available
      - “Make-before-break (too much)”
    - Ensure at most 125% of replicas are available
      - “Don’t-make-before-break (enough)”
- Load balancing only to available pods



Source: Kubernetes

Kubernetes: Rolling Updates

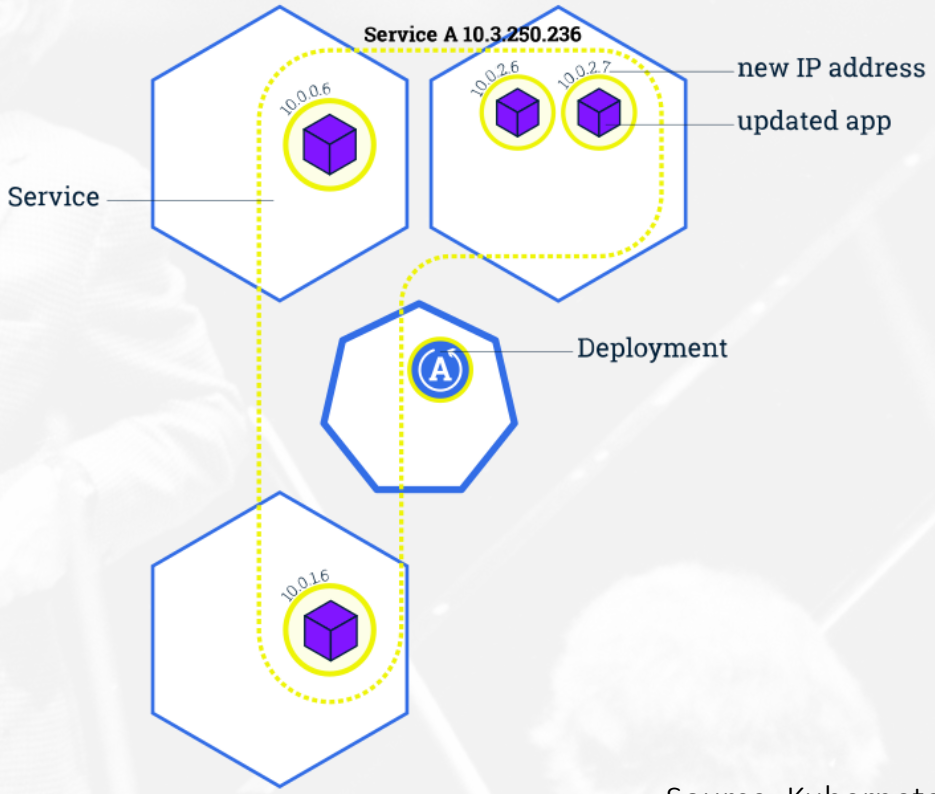
- Update pods
  - New application version (CI/CD)
  - Rollback to previous version
- Pods replaced incrementally
  - Default
    - one-by-one
    - Ensure at least 75% of replicas are available
      - “Make-before-break (too much)”
    - Ensure at most 125% of replicas are available
      - “Don’t-make-before-break (enough)”
- Load balancing only to available pods



Source: Kubernetes

Kubernetes: Rolling Updates

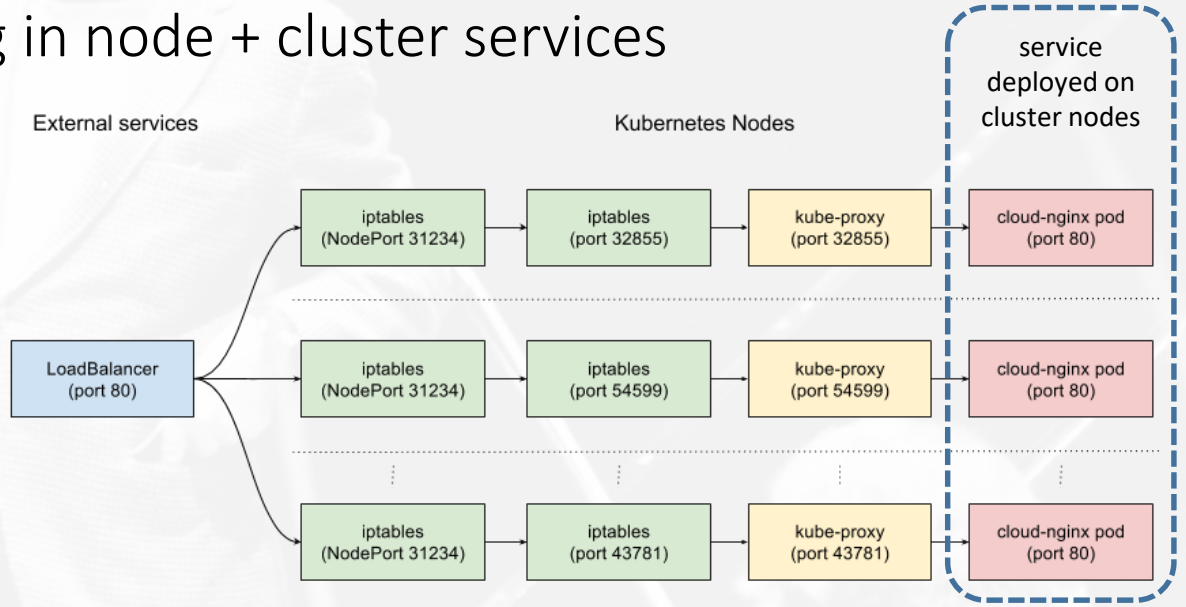
- Update pods
  - New application version (CI/CD)
  - Rollback to previous version
- Pods replaced incrementally
  - Default
    - one-by-one
    - Ensure at least 75% of replicas are available
      - “Make-before-break (too much)”
    - Ensure at most 125% of replicas are available
      - “Don’t-make-before-break (enough)”
- Load balancing only to available pods



Source: Kubernetes

Kubernetes: Load Balancer and `kube-proxy`

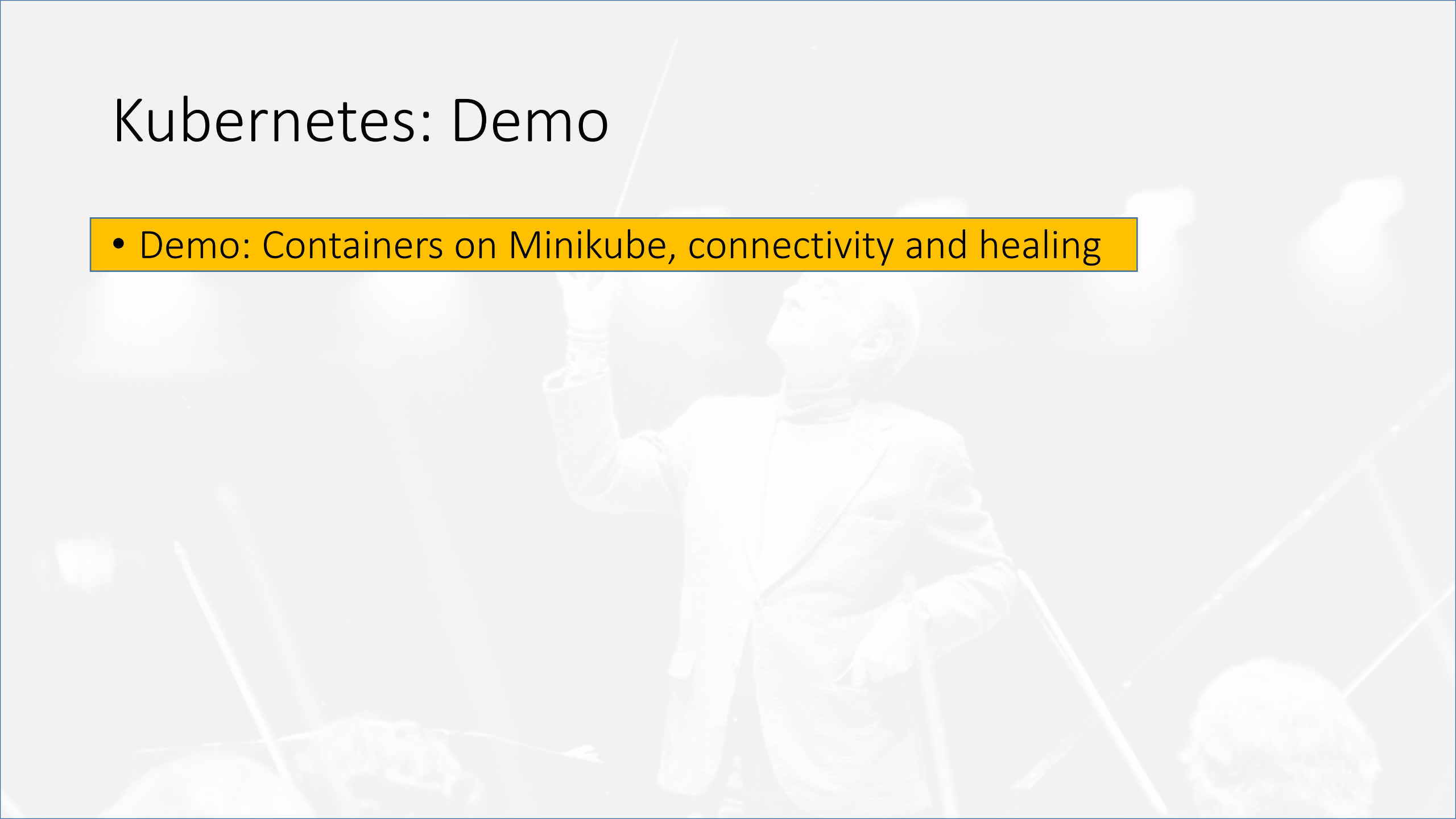
- Load balancing requests to a service:
  - Distribute connections/requests to various nodes running pods of a service
    - E.g., shared `NodePort`
- `kube-proxy` (in every node)
  - Maintains IP addresses of pods running in node + cluster services
    - Regularly updated by master node
  - Reverse-proxy (load balancer)
    - Default: random (but also RR)
  - Manages port-forwarding
    - Maintains and updates `iptables` (user)
      - Sets various `netfilter` (kernel) rules
        - Efficient (heavy-lifting in kernel)



Source: Flaqué (2017)

Kubernetes: Demo

- Demo: Containers on Minikube, connectivity and healing

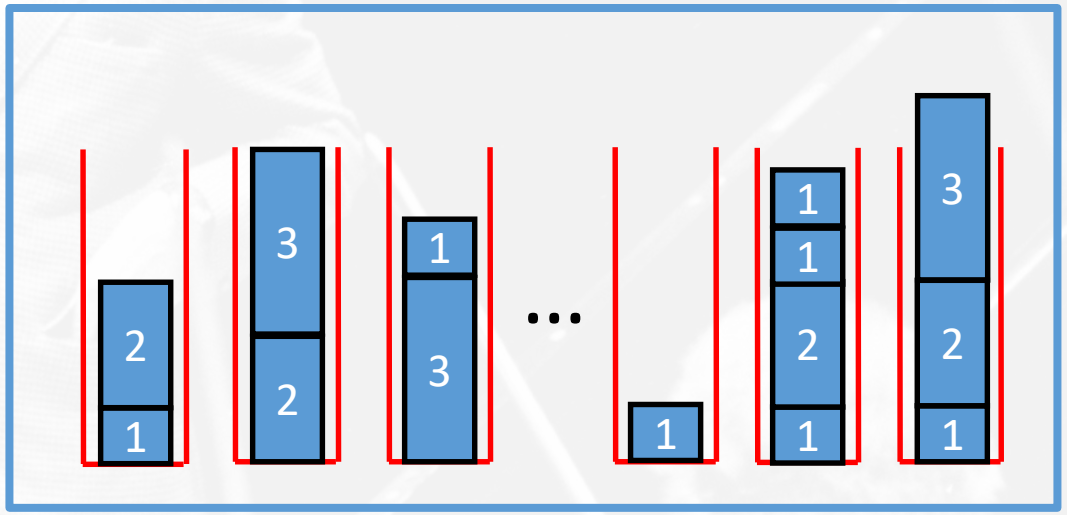


Outline

- (Ultra quick) introduction to DevOps
- The task of Orchestration
  - (Some) contending suites
- Kubernetes (in some more depth)
  - Architecture
  - Specific components
- Load Balancing
- Miscellany

Introduction to Load Balancing (and scheduling)

- m machines: identical, related (different speeds), unrelated
- n jobs (or unbounded sequence, if departures are allowed)
  - p_{ij}: job j processing time on machine i
  - Permanent, release times, deadlines, precedence, preemption allowed, …
- Optimization objectives
  - Minimize max completion time
    - Makespan
  - Maximize jobs completed by deadline
  - Minimize max lateness
    - (completion – deadline)
  - …



Makespan

- Place jobs $J_1, \ldots, J_n$ of sizes $p_1, \ldots, p_n$ onto m identical machines
- Objective: minimize max load (equivalently, minimize max completion time)

- Single machine: trivial…
- NP-hard (for $m \geq 2$)
  - Equivalent to SubsetSum for $m = 2$

- c-approximation algorithm (minimization problem): ALG $\leq c \cdot$ OPT $\quad (c > 1)$
  - Offline
- c-competitive algorithm (online)

Makespan

- Place jobs $J_1, \ldots, J_n$ of sizes $p_1, \ldots, p_n$ onto m identical machines
- Objective: minimize max load (equivalently, minimize max completion time)

List Scheduling (LS) Algorithm

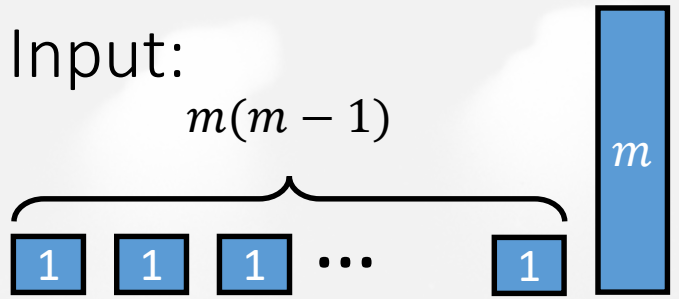
1. sort jobs arbitrarily, $J_1, J_2, \ldots, J_n$
2. for $j = 1, \ldots, n$ do
3. assign J_j to currently least loaded machine
4. end for

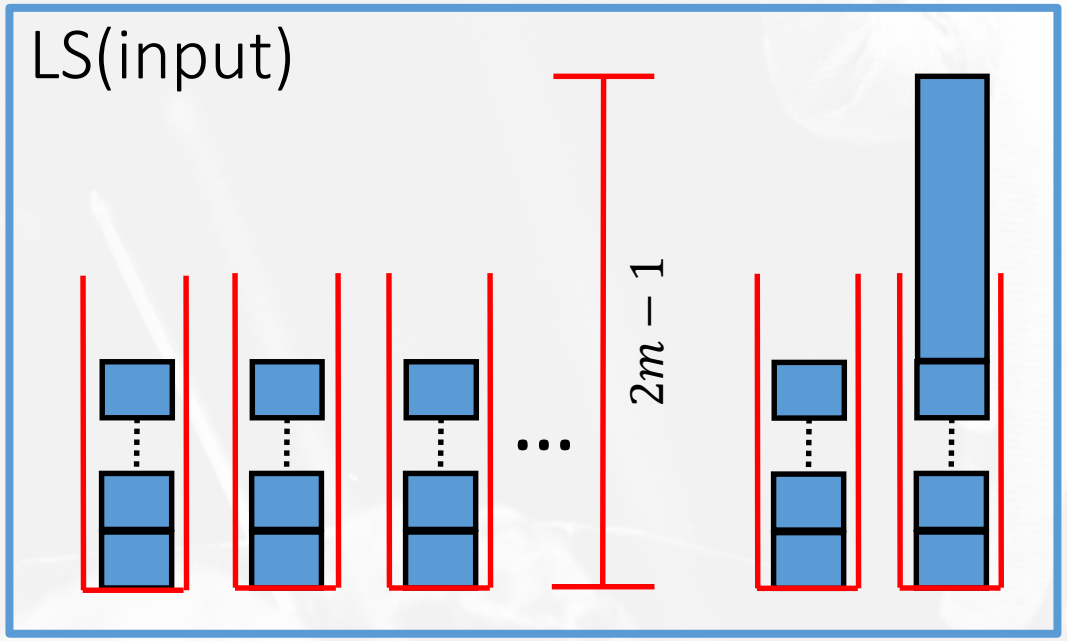
- Offline algorithm?
- Probably the first approximation/competitive algorithm ever
  - Ronald Graham, ’66

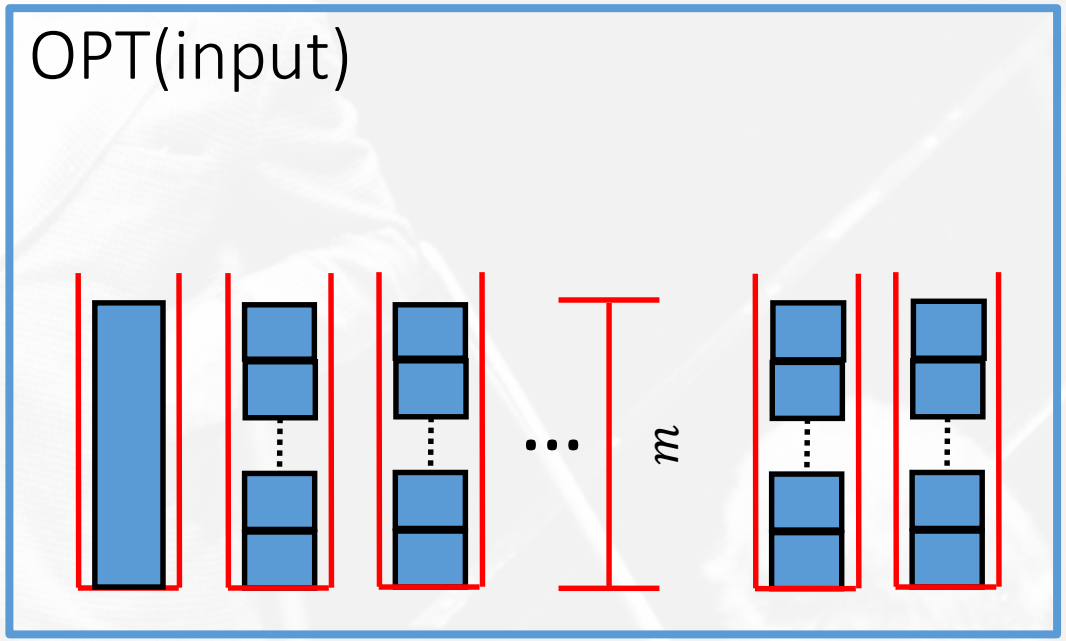
Makespan: LS Lower Bound

- How bad can LS be?
  - LS $\geq \left(2 - \frac{1}{m}\right)$OPT
  - Is this tight?

- Input:







Makespan: LS Upper Bound

- **Theorem**: LS is a $\left(2-\frac{1}{m}\right)$-approximation (competitive) algorithm
- **Proof:**
  - Observation: OPT $\geq \max\left\{\frac{\sum_j p_j}{m}, \max_j p_j\right\}$
  - All machines in LS have load $\geq L - p_{j^*}$

    $\Rightarrow \left(\sum_j p_j\right) - p_{j^*} \geq m \cdot (L - p_{j^*})$

  - It follows that

$$\begin{aligned} L &\leq \frac{\left(\sum_j p_j\right) - p_{j^*}}{m} + p_{j^*} \\ &= \frac{\sum_j p_j}{m} + \left(1 - \frac{1}{m}\right) p_{j^*} \\ &\leq \left(2 - \frac{1}{m}\right) \text{OPT} \end{aligned}$$

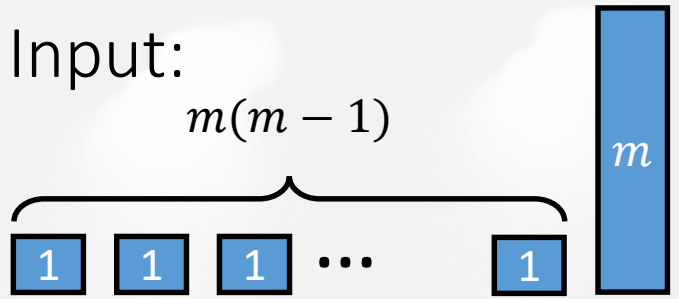
■

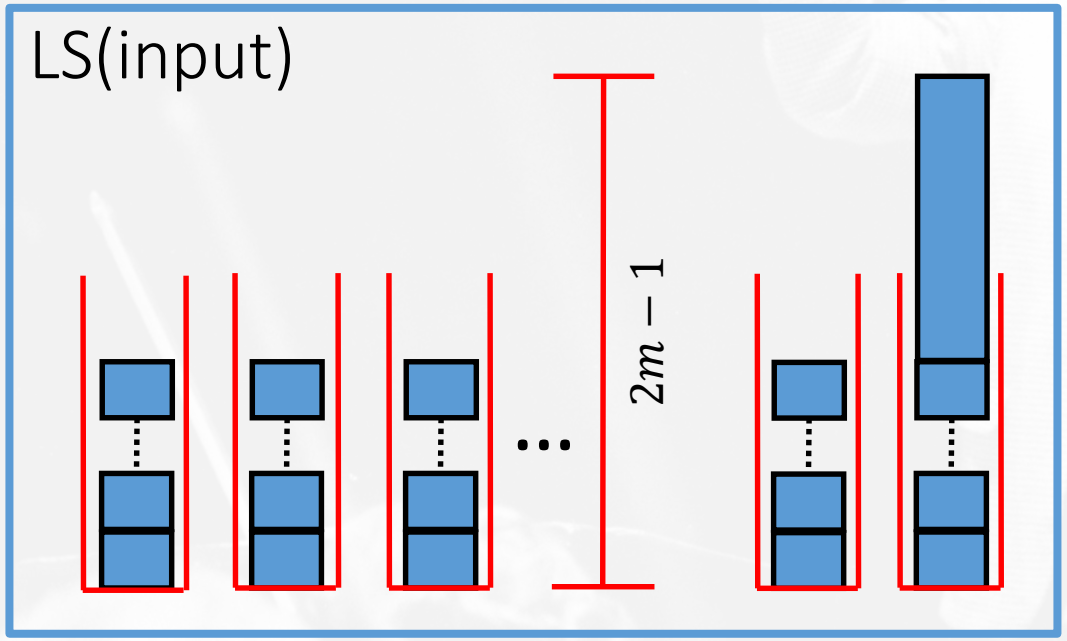
- Notation:
  - L: maximum load in LS
  - i^*: most loaded machine in LS
  - j^*: last job assigned to i^*

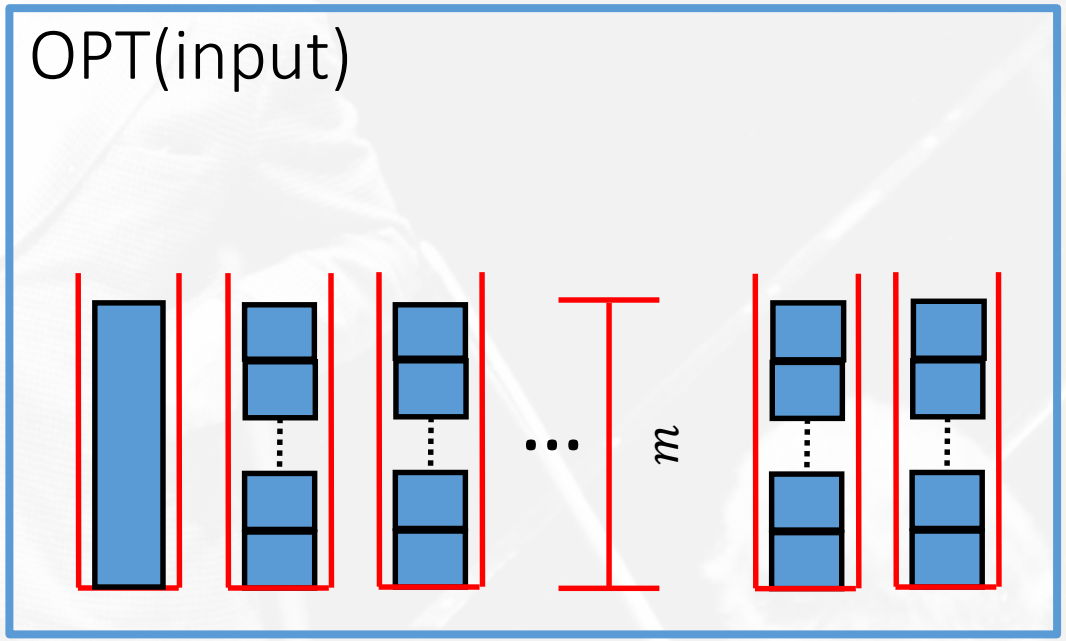
Makespan: Better (and Other) Upper Bounds

- Why does the lower bound "work"?
  - We get the elephant after everything is nicely balanced
- What if we could order the jobs?

- Input:







Makespan: Better (and Other) Upper Bounds

- Offline/online?

- OrderedLS
  - Order jobs in non-increasing order of processing: $p_1 \geq p_2 \geq \cdots \geq p_n$
- **Lemma**: If $n > m$ then OPT $\geq p_m + p_{m+1}$
  - Pigeonhole principle: 2 jobs out of $J_1, \ldots, J_{m+1}$ are on the same machine…
- **Theorem**: OrderedLS is a $\left(\frac{2}{3} - \frac{1}{2m}\right)$-approximation algorithm
- **Proof**:
  - If $j^* \leq m$: OrderedLS is optimal
  - If $j^* > n$: $p_{j^*} \leq \frac{p_m + p_{m+1}}{2} \leq \frac{OPT}{2}$
  - Similarly to previous proof:

$$L \leq \frac{(\sum_j p_j) - p_{j^*}}{m} + p_{j^*} = \frac{\sum_j p_j}{m} + \left(1 - \frac{1}{m}\right) p_{j^*} \leq \left(\frac{2}{3} - \frac{1}{2m}\right) \text{OPT}$$

- Same notation:
  - L: maximum load in LS
  - i^*: most loaded machine in LS
  - j^*: last job assigned to i^*

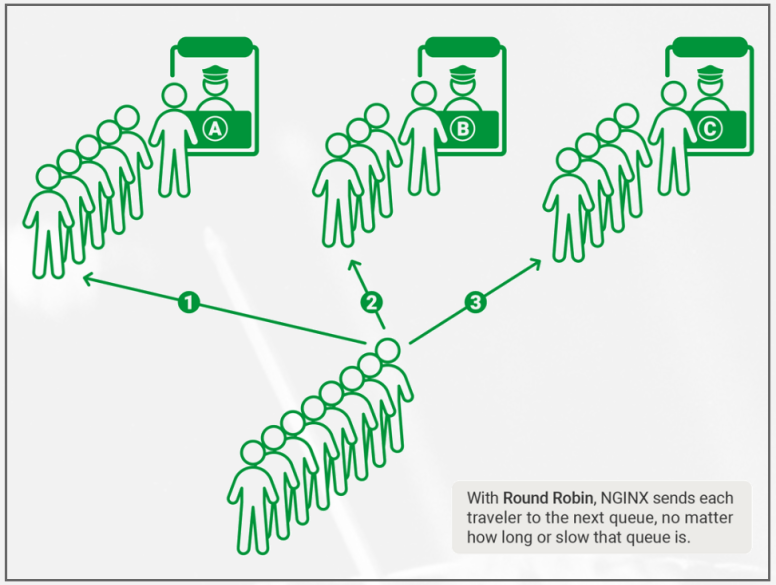
■

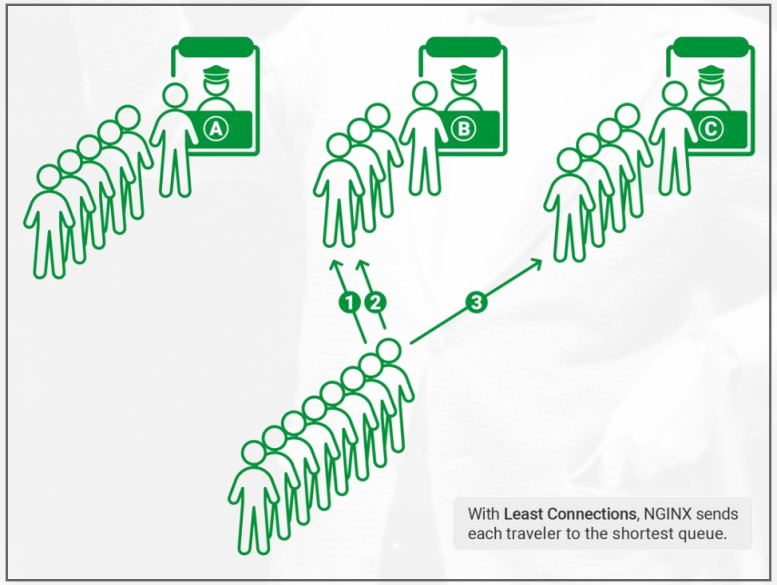
Makespan: Better (and Other) Upper Bounds

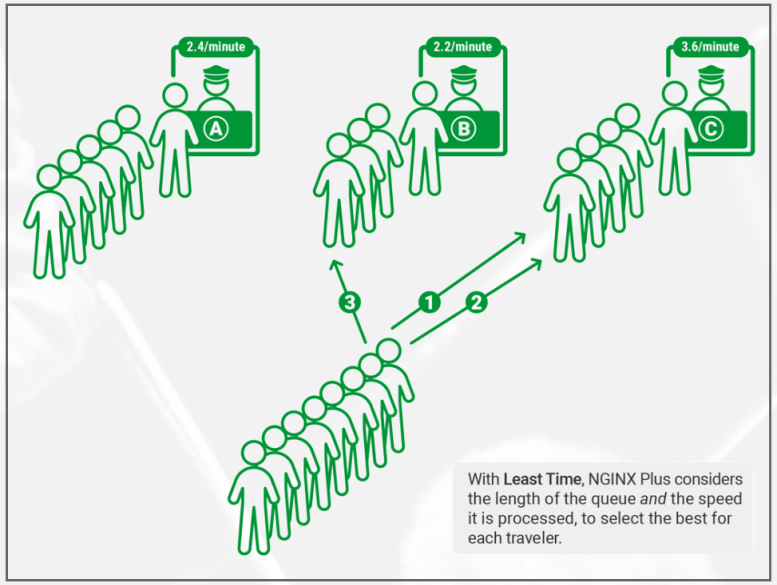
- Online:
  - Create “imbalance” on purpose (not too much…)
- Greedy (and other) algorithms
  - Related machines: constant-competitive
  - Unrelated machines: mostly $\Theta(\log n)$-competitive
- Limited duration jobs (with/without migrations)
  - Known duration: more complex algorithms
  - Unknown duration: much more complex!
- “Know thy workload…”

More Approaches to Load Balancing

- **RoundRobin**
  - Simple, stateless
  - “weighted”
    - E.g., heterogeneous servers
- **LeastConnections**
  - LS…
  - Stateful
    - Maintain state
    - Probe servers
- **LeastTime**
  - Take server processing speed into account
    - “LS in related machines”







Source: nginx.com

More Approaches to Load Balancing

- **Hash**
  - Server chosen by hash(job “ID”)
  - ID: source IP, flow identifier, …
  - stateless
- **Random**
  - Randomly choose server
  - Simple, stateless
- **Power of two choices**
  - Pick two random servers
  - Place job on least loaded of the two
  - Stateless

- Especially useful for parallel/distributed load balancers
- Recently incorporated into various reverse proxies
  - HAProxy, Nginx, …

Outline

- (Ultra quick) introduction to DevOps
- The task of Orchestration
  - (Some) contending suites
- Kubernetes (in some more depth)
  - Architecture
  - Specific components
- Load Balancing
- Miscellany

Monitoring: Prometheus

https://prometheus.io/

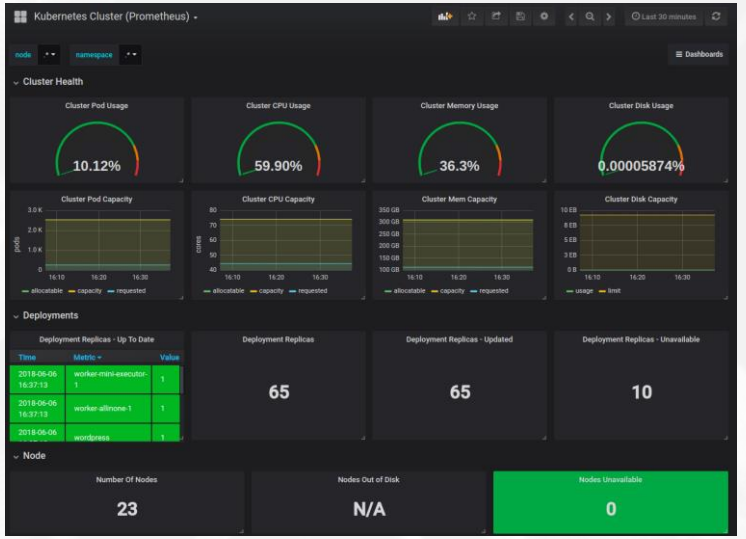
- Open-source monitoring toolkit
- Collecting time-series metrics
  - Machine-centric
  - Service-centric
- Supports rich querying
- Monitoring Kubernetes
  - Create Prometheus deployment in k8s
    - from image, using .yml file,
  - Expose as a service
- Integrates with Grafana dashboard

matching `job` and `handler` labels

```
http_requests_total{job="apiserver", handler="/api/comments"}[5m]
```

all time series with metric `http_request_total` defined

recent 5 min



Source: Grafana

Closing Note: Cloud Native

https://www.cncf.io/

- Open-source software foundation
  - Part of the Linux Foundation
    - https://www.linuxfoundation.org/projects/
  - Sponsors open-source projects related to cloud computing
    - Kubernetes, Prometheus, Envoy Proxy, etc.
- Moto: Cloud portability without vendor lock-in
- A few unknown contributing members…
- Various others foundations
  - Open Container Initiative, Cloud Foundry, …

Source: CNCF

(Partial) Bibliography

- Bourque & Fairley, “Software Engineering Body of Knowledge”, IEEE (2014)
- Sharma, “The DevOps Adoption Playbook”, Wiley (2017)
- Sharma, “A Brief History of DevOps”, https://circleci.com/blog (2018)
- https://aws.amazon.com/devops/
- Royce, “Managing the Development of Large Software Systems", WESCON (1970)
- Beck et al., “Agile Manifesto”, http://agilemanifesto.org/ (2001)
- https://docs.docker.com/
- https://puppet.com/
- Verma et al., “Large-scale cluster management at Google with Borg”, EuroSys (2015)
- https://kubernetes.io/
- Klein, “Introduction to Modern Network Load Balancing and Proxying”, https://blog.envoyproxy.io (2017)
- Flaqué, “Kubernetes: from load balancer to pod”, https://medium.com (2017)
- Betz, “Understanding kubernetes networking”, https://medium.com (2017)
- Mitzenmacher, “The Power of Two Choices in Randomized Load Balancing”. IEEE TPDS. 12(10): 1094-1104 (2001)
- Garret, “NGINX and the “Power of Two Choices” Load-Balancing Algorithm”, https://www.nginx.com/blog/ (2018)
- Tarreau, “Test Driving “Power of Two Random Choices” Load Balancing”, https://www.haproxy.com/blog, (2019)