CSCI-GA.2250-001

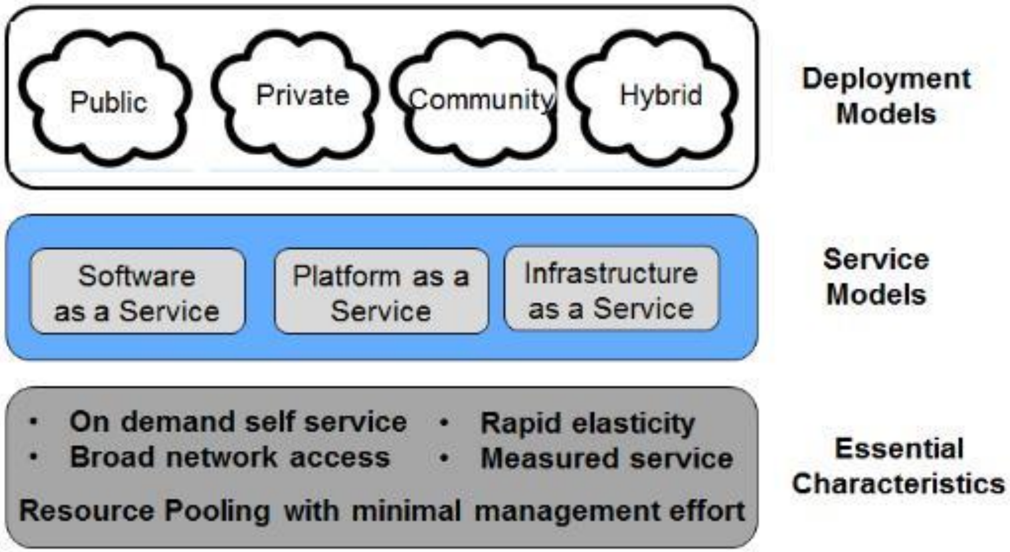
Operating Systems

Cloud and Operating Systems

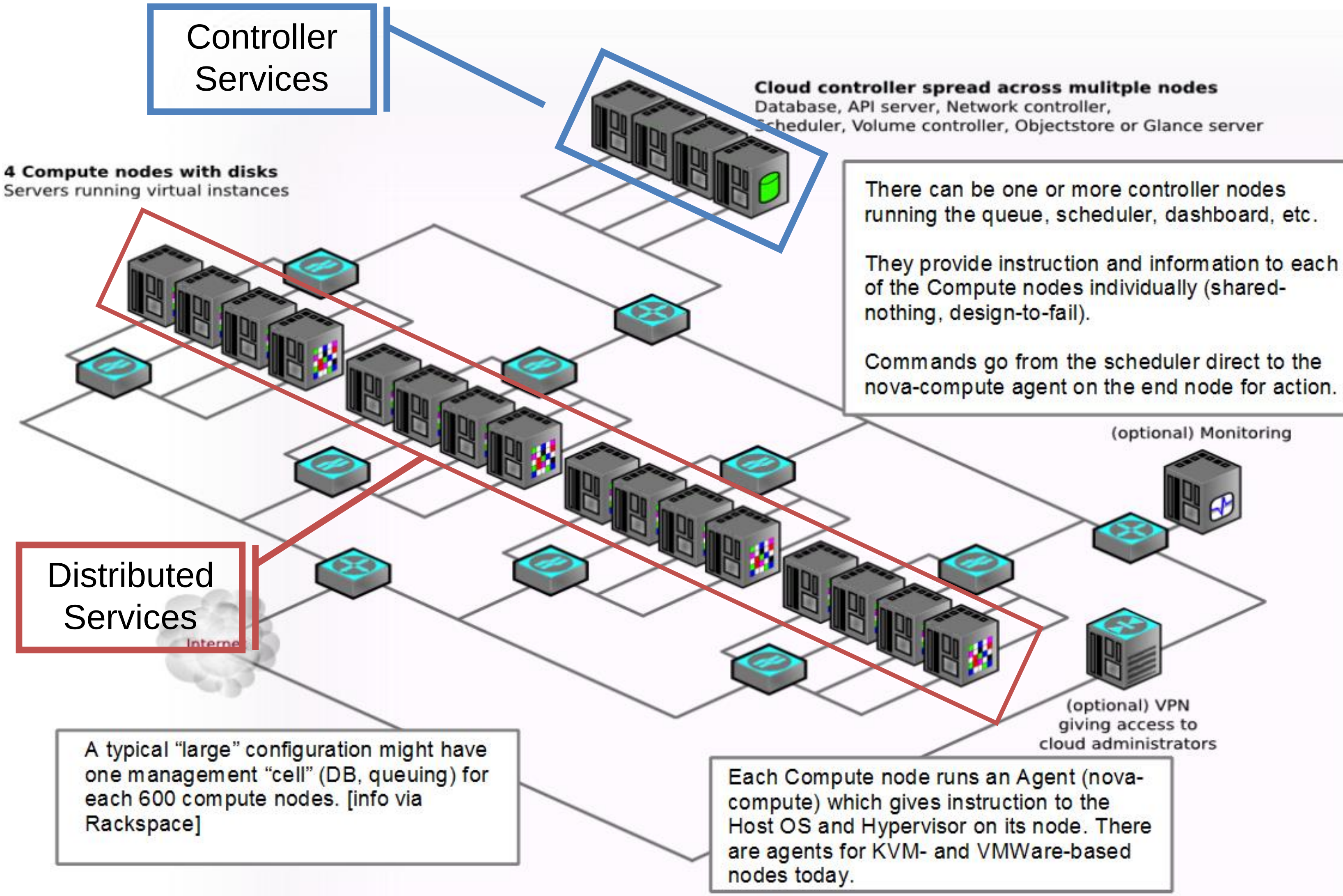
Hubertus Franke
frankeh@cs.nyu.edu

Cloud Computing

- (New) IT delivery model
- Access your services remotely (cloud)
- Pay as you go model
  - Amazon Webservices, Azure, Google, IBM, ...



Cloud implementation modules can be categorized into two groups – controller services and distributed services



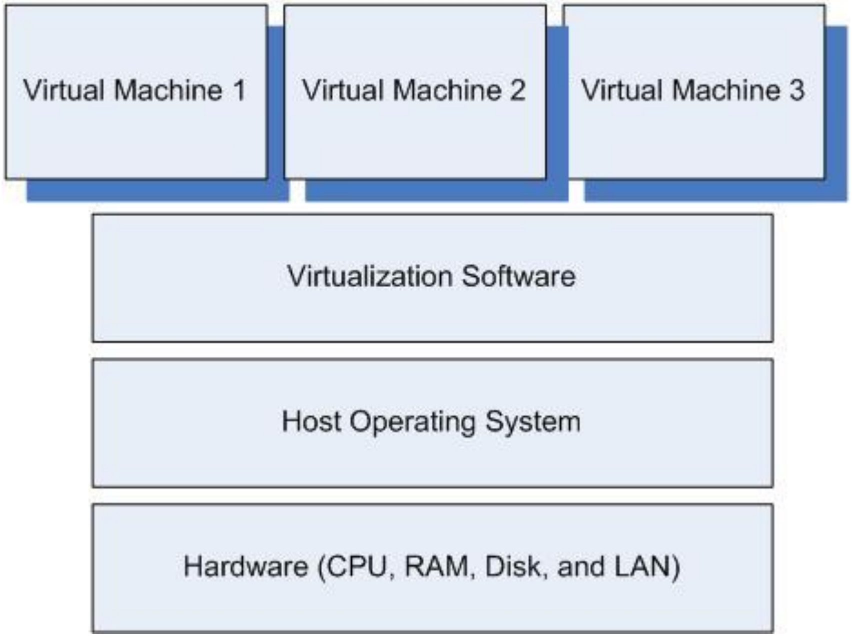
IaaS (Infrastructure as a Service)

- Request "machines" with certain characteristics
- IaaS layer provisions necessary resources
  - Compute
  - Network
  - Storage
- Provides connectivity to the machine
- Heavily relies on virtualization technology

→ Virtual Machine Technology

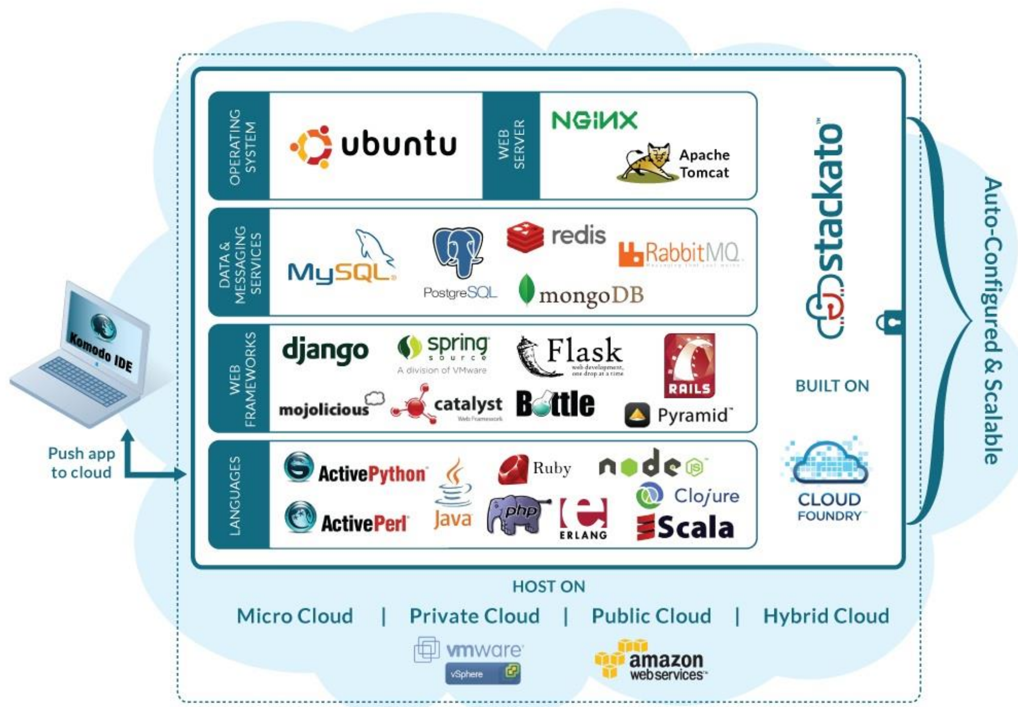
OS and Virtualization

- Allows a single machine to run several *guest OSes* simultaneously
- Each OS is running on a virtual machine (VM)
- If a VM crashes the others are not affected
- Services are
  - Virtual machine
  - Virtual storage (block)
  - Virtual networking (BYOIP)



PaaS (Platform as a Service)

- Obtain Services (DBs, Queues, ..)
- Customize

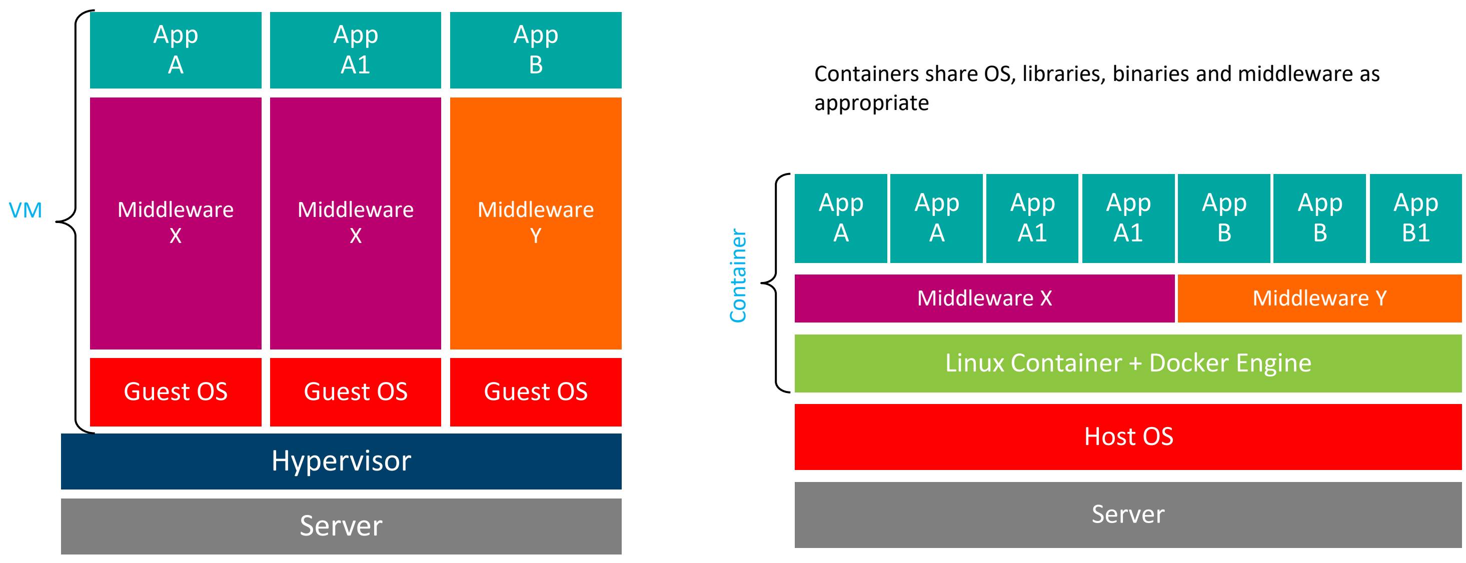


SaaS (Software as a Service)

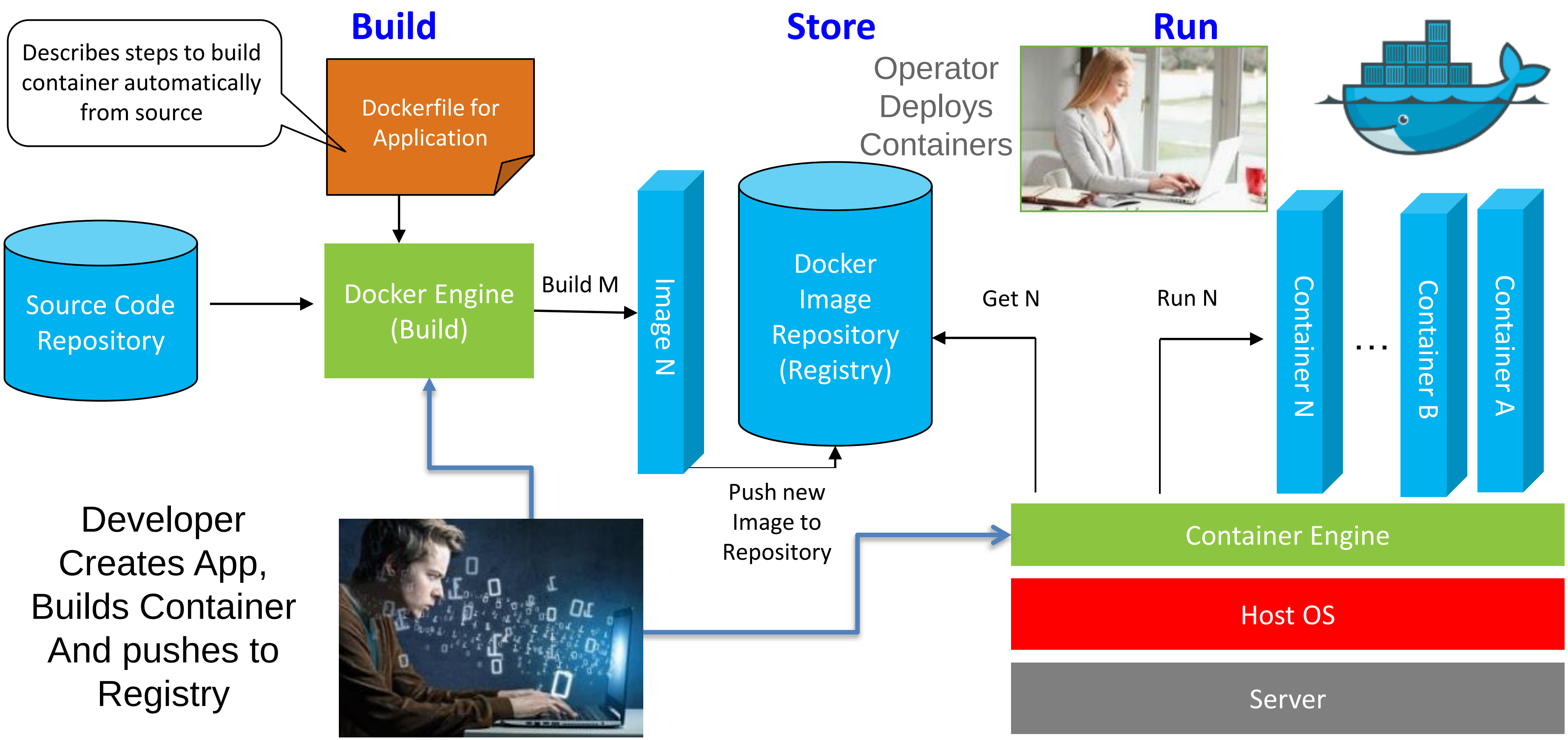
- Company hosts software (e.g. CRM, HRM, ERP)
  - Provides software and datacenter
- Takes care of maintenance, upgrades, support, …
- Pay per use (seats, transactions)
- Example:
  - Gmail
  - Online Tax Filing
  - Salesforce.com

What's a Container and how it differs from a VM?

- The concept of containers emerged 2 decades ago (e.g. Sun Solaris Zones and IBM AIX's WPARs). **Docker** is built on open source container capabilities inside Linux kernel (cgroups, namespaces, selinux, etc)
- A container encapsulates an application and its dependencies which run in an isolated process on the host's operating system (all application share the same OS)
- Traditional hardware virtualization creates an entire virtual machine. Each VM contains not only the application (which may only be 10's of MB) but must include and an entire Guest operating System (which may measure in 1-10s of GB).



What are the Basic Functions of Containers



How is “Containment” achieved

We are still running on a shared kernel !!!
Think: “ls –ls /” and “ps –edf”, you see everything

- Process (group) Isolation

  → namespaces

- Process (group) Performance Isolation

  → cgroups

- Storage Isolation/Virtualization

  → overlay filesystems

Sources: https://www.nginx.com/blog/what-are-namespaces-cgroups-how-do-they-work/
Lectures: NYU Seetharami, Chung, Yu

Namespaces

•A user namespace has its own set of user IDs and group IDs for assignment to processes. In particular, this means that a process can have root privilege within its user namespace without having it in other user namespaces.

•A process ID (PID) namespace assigns a set of PIDs to processes that are independent from the set of PIDs in other namespaces. The first process created in a new namespace has PID 1 and child processes are assigned subsequent PIDs. If a child process is created with its own PID namespace, it has PID 1 in that namespace as well as its PID in the parent process' namespace.

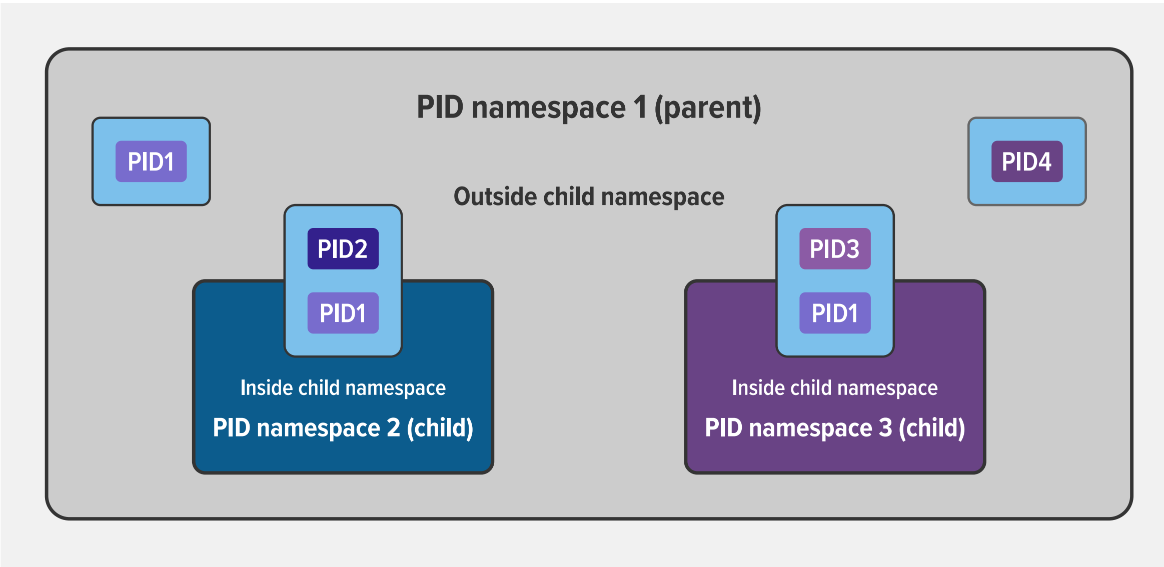
•A network namespace has an independent network stack: its own private routing table, set of IP addresses, socket listing, connection tracking table, firewall, and other network-related resources.

•A mount namespace has an independent list of mount points seen by the processes in the namespace. This means that you can mount and unmount filesystems in a mount namespace without affecting the host filesystem.

•An interprocess communication (IPC) namespace has its own IPC resources, for example POSIX message queues.

•A UNIX Time-Sharing (UTS) namespace allows a single system to appear to have different host and domain names to different processes.

Namespaces (example PID)




```
struct task_struct {
#ifdef CONFIG_THREAD_INFO_IN_TASK
        /*
         * For reasons of header soup (see current_thread_info()), this
         * must be the first element of task_struct.
         */
        struct thread_info            thread_info;
#endif
        unsigned int                  __state;
        :
        /* Namespaces: */
        struct nsproxy                *nsproxy;
        :
}
```

```
struct nsproxy {
        atomic_t count;
        struct uts_namespace *uts_ns;
        struct ipc_namespace *ipc_ns;
        struct mnt_namespace *mnt_ns;
        struct pid_namespace *pid_ns_for_children;
        struct net           *net_ns;
        struct time_namespace *time_ns;
        struct time_namespace *time_ns_for_children;
        struct cgroup_namespace *cgroup_ns;
};
```

Creation of new namespace (1)

- Using `clone()` syscall

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg, ...
          /* pid_t *ptid, struct user_desc *tls, pid_t *ctid */ );
```

```
/*
 * cloning flags:
 */
#define CSIGNAL         0x000000ff      /* signal mask to be sent at exit */
#define CLONE_VM        0x00000100      /* set if VM shared between processes */
#define CLONE_FS        0x00000200      /* set if fs info shared between processes */
#define CLONE_FILES     0x00000400      /* set if open files shared between processes */
#define CLONE_SIGHAND   0x00000800      /* set if signal handlers and blocked signals shared */
#define CLONE_PIDFD     0x00001000      /* set if a pidfd should be placed in parent */
#define CLONE_PTRACE    0x00002000      /* set if we want to let tracing continue on the child too */
#define CLONE_VFORK     0x00004000      /* set if the parent wants the child to wake it up on mm_release */
#define CLONE_PARENT    0x00008000      /* set if we want to have the same parent as the cloner */
#define CLONE_THREAD    0x00010000      /* Same thread group? */
#define CLONE_NEWNS     0x00020000      /* New mount namespace group */
#define CLONE_SYSVSEM   0x00040000      /* share system V SEM_UNDO semantics */
#define CLONE_SETTLS    0x00080000      /* create a new TLS for the child */
#define CLONE_PARENT_SETTID     0x00100000      /* set the TID in the parent */
#define CLONE_CHILD_CLEARTID    0x00200000      /* clear the TID in the child */
#define CLONE_DETACHED          0x00400000      /* Unused, ignored */
#define CLONE_UNTRACED          0x00800000      /* set if the tracing process can't force CLONE_PTRACE on this clone */
#define CLONE_CHILD_SETTID      0x01000000      /* set the TID in the child */
#define CLONE_NEWCGROUP         0x02000000      /* New cgroup namespace */
#define CLONE_NEWUTS            0x04000000      /* New utsname namespace */
#define CLONE_NEWIPC            0x08000000      /* New ipc namespace */
#define CLONE_NEWUSER           0x10000000      /* New user namespace */
#define CLONE_NEWPID            0x20000000      /* New pid namespace */
#define CLONE_NEWNET            0x40000000      /* New network namespace */
#define CLONE_IO                0x80000000      /* Clone io context */
```

Creation of new namespace (2)

- Cmd line: `unshare`

```
frankeh@lnx2:~$ id
uid=1000(frankeh) gid=1000(frankeh) groups=1000(frankeh),4(adm),24(cdrom),27(
sudo),30(dip),46(plugdev),120(lpadmin),999(sambashare)
frankeh@lnx2:~$ unshare --user --map-root-user --fork bash
root@lnx2:~# ps -edf | head -10
UID          PID    PPID  C STIME TTY          TIME CMD
nobody         1       0  0 18:21 ?        00:00:01 /sbin/init splash
nobody         2       0  0 18:21 ?        00:00:00 [kthreadd]
nobody         3       2  0 18:21 ?        00:00:00 [rcu_gp]
nobody         4       2  0 18:21 ?        00:00:00 [rcu_par_gp]
nobody         6       2  0 18:21 ?        00:00:00 [kworker/0:0H-events_high
pri]
nobody         9       2  0 18:21 ?        00:00:00 [mm_percpu_wq]
nobody        10       2  0 18:21 ?        00:00:00 [rcu_tasks_rude_]
nobody        11       2  0 18:21 ?        00:00:00 [rcu_tasks_trace]
nobody        12       2  0 18:21 ?        00:00:00 [ksoftirqd/0]
```

```
frankeh@lnx2:~$ id
uid=1000(frankeh) gid=1000(frankeh) groups=1000(frankeh),4(adm),24(cdrom),27(
sudo),30(dip),46(plugdev),120(lpadmin),999(sambashare)
frankeh@lnx2:~$ unshare --user --pid --map-root-user --mount-proc --fork bash
root@lnx2:~# ps -edf
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 20:28 pts/0    00:00:00 bash
root           7       1  0 20:28 pts/0    00:00:00 ps -edf
root@lnx2:~#
```

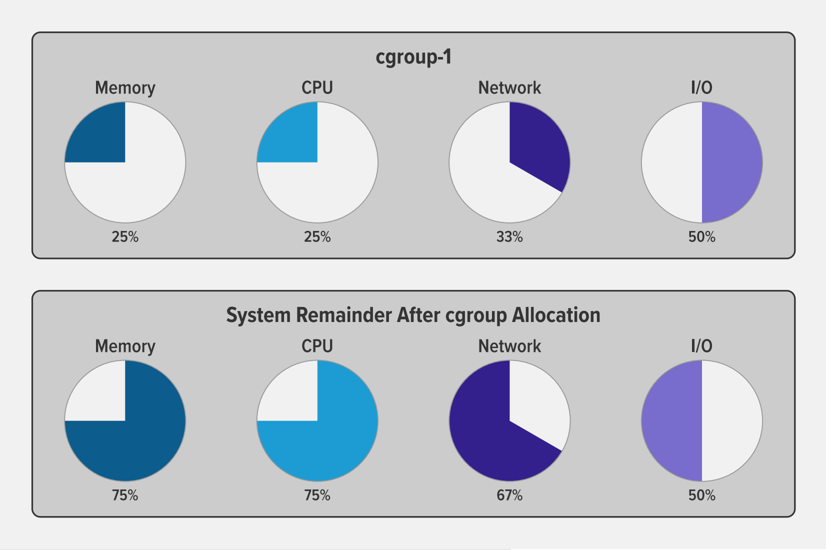
Cgroups

A control group (**cgroup**) is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, and so on) of a collection of processes.

Cgroups provide the following features:

- **Resource limits** – You can configure a cgroup to limit how much of a particular resource (memory or CPU, for example) a process can use.
- **Prioritization** – You can control how much of a resource (CPU, disk, or network) a process can use compared to processes in another cgroup when there is resource contention.
- **Accounting** – Resource limits are monitored and reported at the cgroup level.
- **Control** – You can change the status (frozen, stopped, or restarted) of all processes in a cgroup with a single command.

Example of a cgroup




```
root # mkdir -p /sys/fs/cgroup/memory/foo
root # echo 50000000 > /sys/fs/cgroup/memory/foo/memory.limit_in_bytes
```

```
root # ./test.sh &
[1] 2428
root # cgroup testing tool
root # echo 2428 > /sys/fs/cgroup/memory/foo/cgroup.procs
```

```
root # ps -o cgroup 2428
CGROUP
12:pids:/user.slice/user-0.slice/\
session-13.scope,10:devices:/user.slice,6:memory:/foo,...
```

Docker Layered Filesystem

- Docker uses a Copy-On-Write layered filesystem
  - Only changes from the read-only layers are copied
- You can see the layers when you pull or push an image

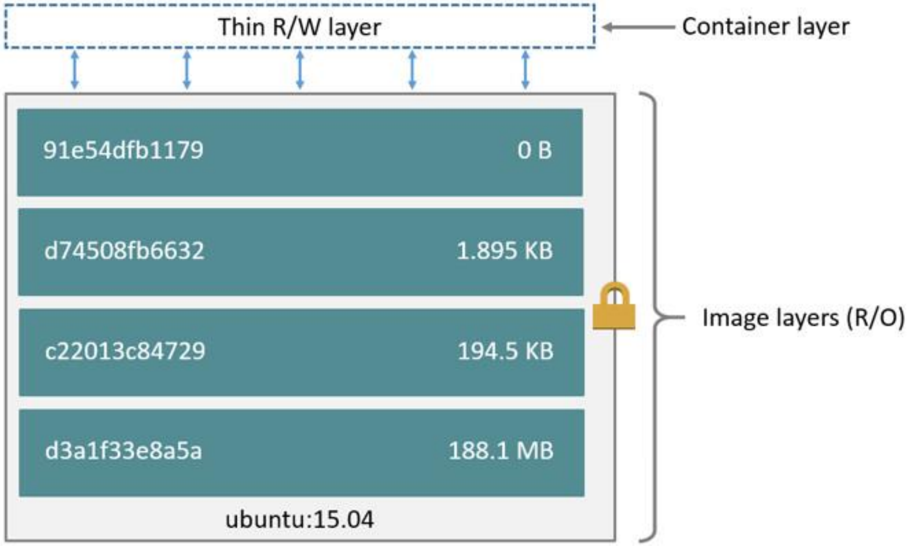
```
$ docker pull ubuntu:15.04

15.04: Pulling from library/ubuntu
1ba8ac955b97: Pull complete
f157c4e5ede7: Pull complete
0b7e98f84c4c: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:5e279a9df07990286cce22e1b0f5b0490629ca6d187698746ae5e28e604a640e
Status: Downloaded newer image for ubuntu:15.04
```

- Why generate so many layers?
  - Think about prime factorization: $9216 = 2^{10}.3^{2}$
  - If one developer is creating many containers, it's likely that one'd reuse a lot of the software modules in a relatively fixed way.
  - #developers >> #reused-software-modules-or-bundles.

Images and Layers

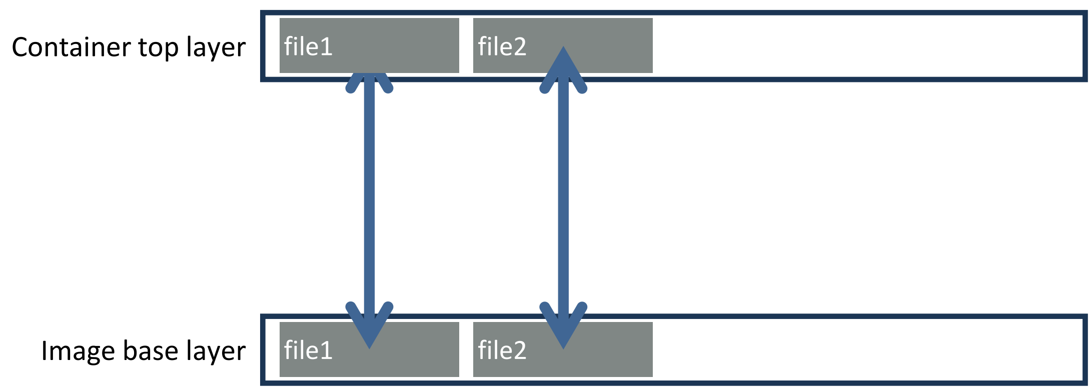
- Each Docker image references a list of read-only layers that represent filesystem differences
- Layers are stacked on top of each other to form a base for a container's root filesystem
- When you create a new container, you add a new, thin, writable layer on top of the underlying stack
- All changes made to the running container - such as writing new files, modifying existing files, and deleting files - are written to this thin writable container layer



Container
(based on ubuntu:15.04 image)

Layers are exposed to Top

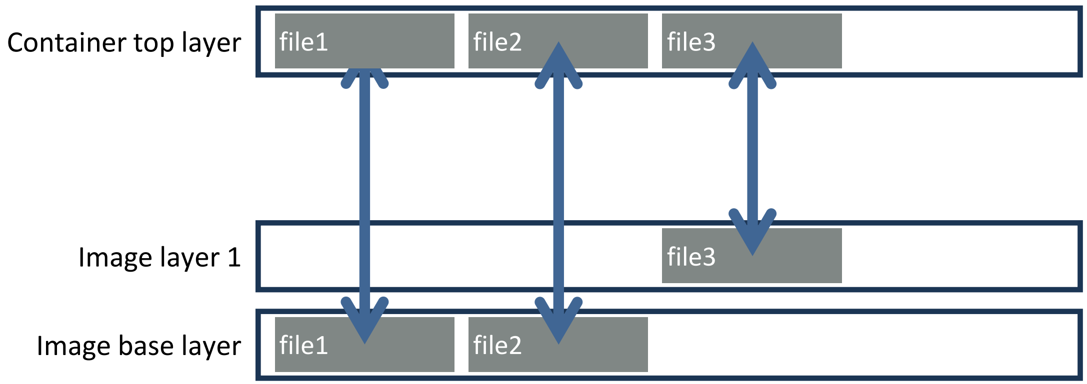
- Files from the read-only layers below are visible to the top layer



Docker container
(AUFS storage-driver demonstrating whiteout file)

Add Layers with more Files

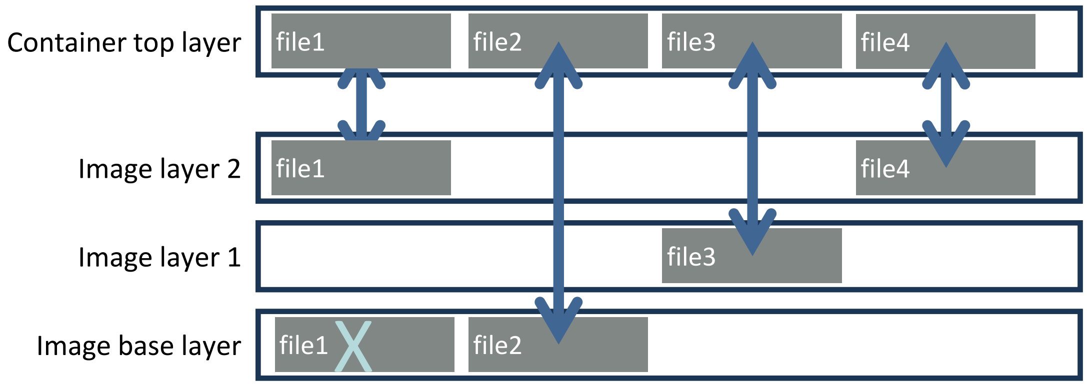
- As you add layers more files become visible at the top layer



Docker container
(AUFS storage-driver demonstrating whiteout file)

New Versions

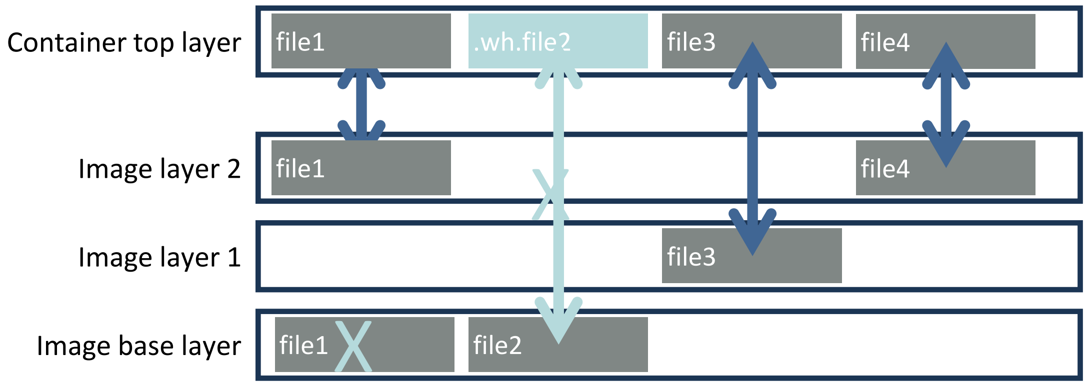
- A new version of `file1` is added and it hides the old `file1` version



Docker container
(AUFS storage-driver demonstrating whiteout file)

White-out Files

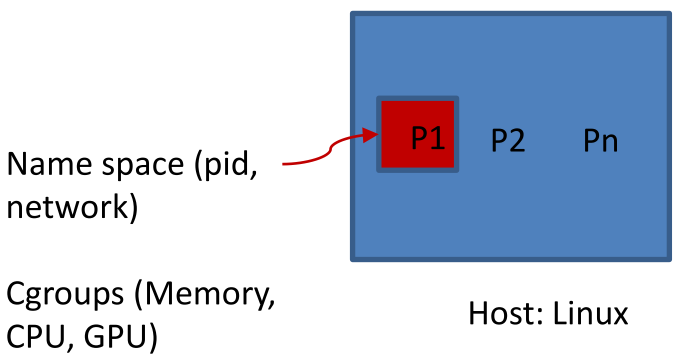
- Special files called "white-out" files are used to make files appear to be deleted



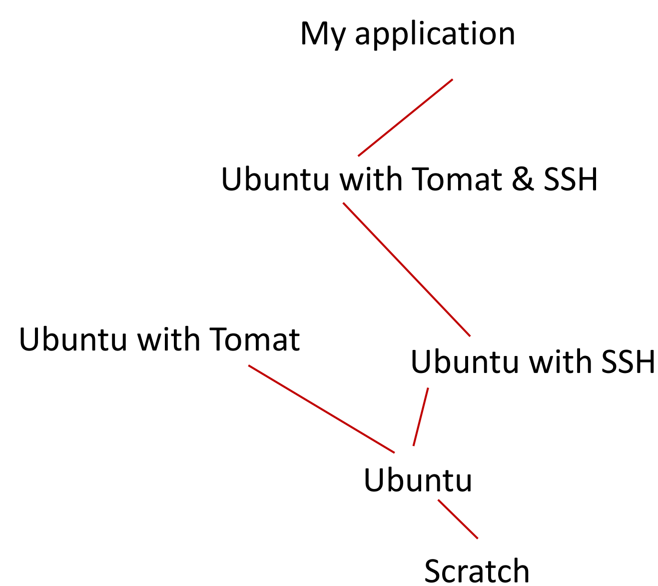
Docker container
(AUFS storage-driver demonstrating whiteout file)

What is a container?

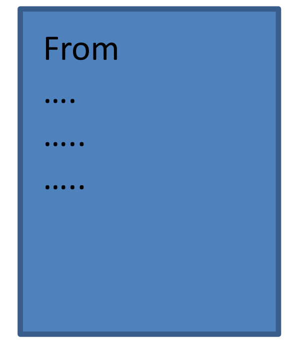
Runtime: A sandbox for a process



Image



Dockerfile



Where are we now?

- So far we talked single machine
  - (namespaces, cgroups, docker)

- Need to move to a scalable cloud deployment /runtime for containers

- → kubernetes

What is Kubernetes?

- “Kubernetes is a portable, extensible open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.“
  - From: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
- Kubernetes was built to run distributed systems over a cluster of machines

Brief History behind Kubernetes

- 2003-2004: Birth of the Borg System
- 2013: From Borg to Omega
- 2014: Google Introduces Kubernetes
- 2015: The year of Kube v1.0 & Cloud Native Computing Foundation
- 2016: The Year Kubernetes Goes Mainstream!
- 2017: The Year of Enterprise Adoption & Support
- 2018: Various cloud providers announce K8S as service

Providers of Kubernetes based Container orchestration

- Amazon Elastic Service for Kubernetes (EKS)
- Google Kubernetes Engine (GKE)
- Azure Kubernetes Service (AKS)
- IBM Kubernetes Service (IKS)
- IBM Cloud Private with Kubernetes (ICP)
- RedHat OpenShift (commercial extension of k8s)
- … at least 30 others (VMWare, Digital Ocean, Oracle, Docker, etc)

Source
https://blog.codeship.com/a-roundup-of-managed-kubernetes-platforms/

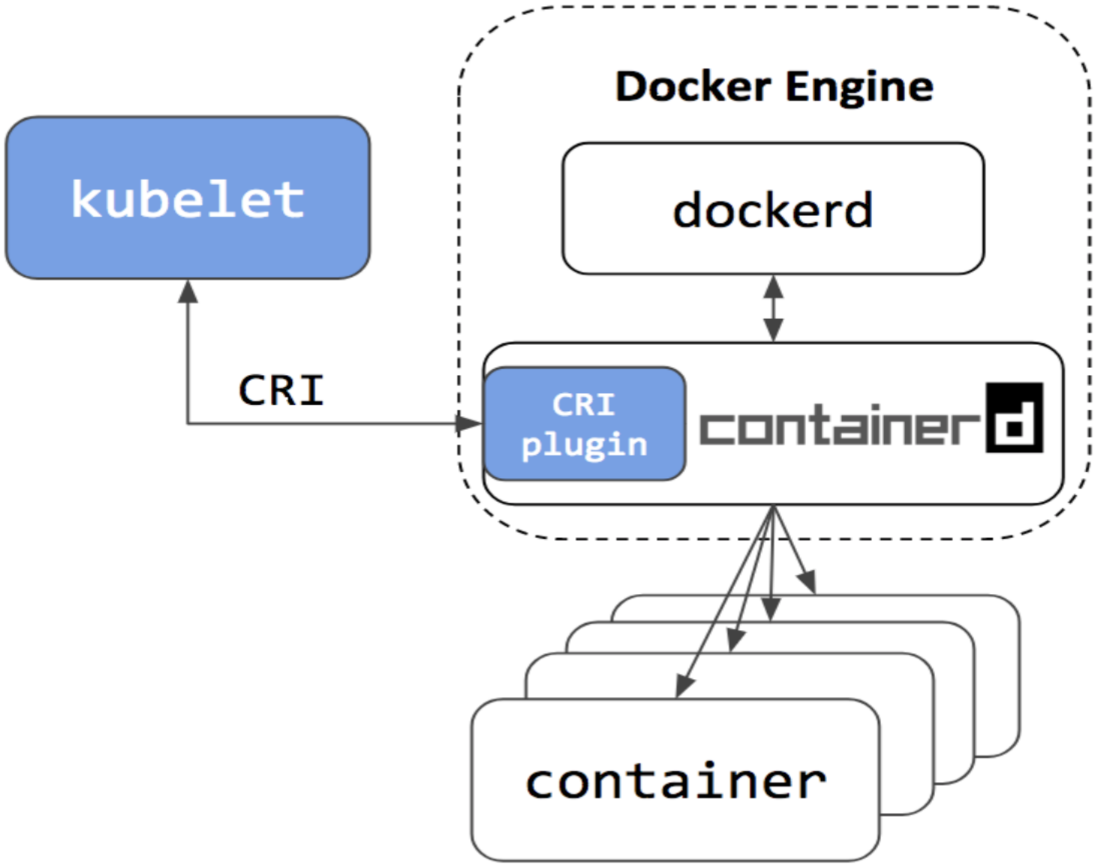
Kubernetes by numbers (by April 2021)

- $4.3 billion market for containers in 2022… 30% CAGR
- 87% surveyed running with containers
- 90% of those with container are in production
- 8K attend KubeCon
- 40% enterprises running with Kubernetes
- ZipRecruiter pegs that national average salary for Kubernetes-related roles at more than $144,000 → $203,000

https://enterprisersproject.com/article/2020/6/kubernetes-statistics-2020

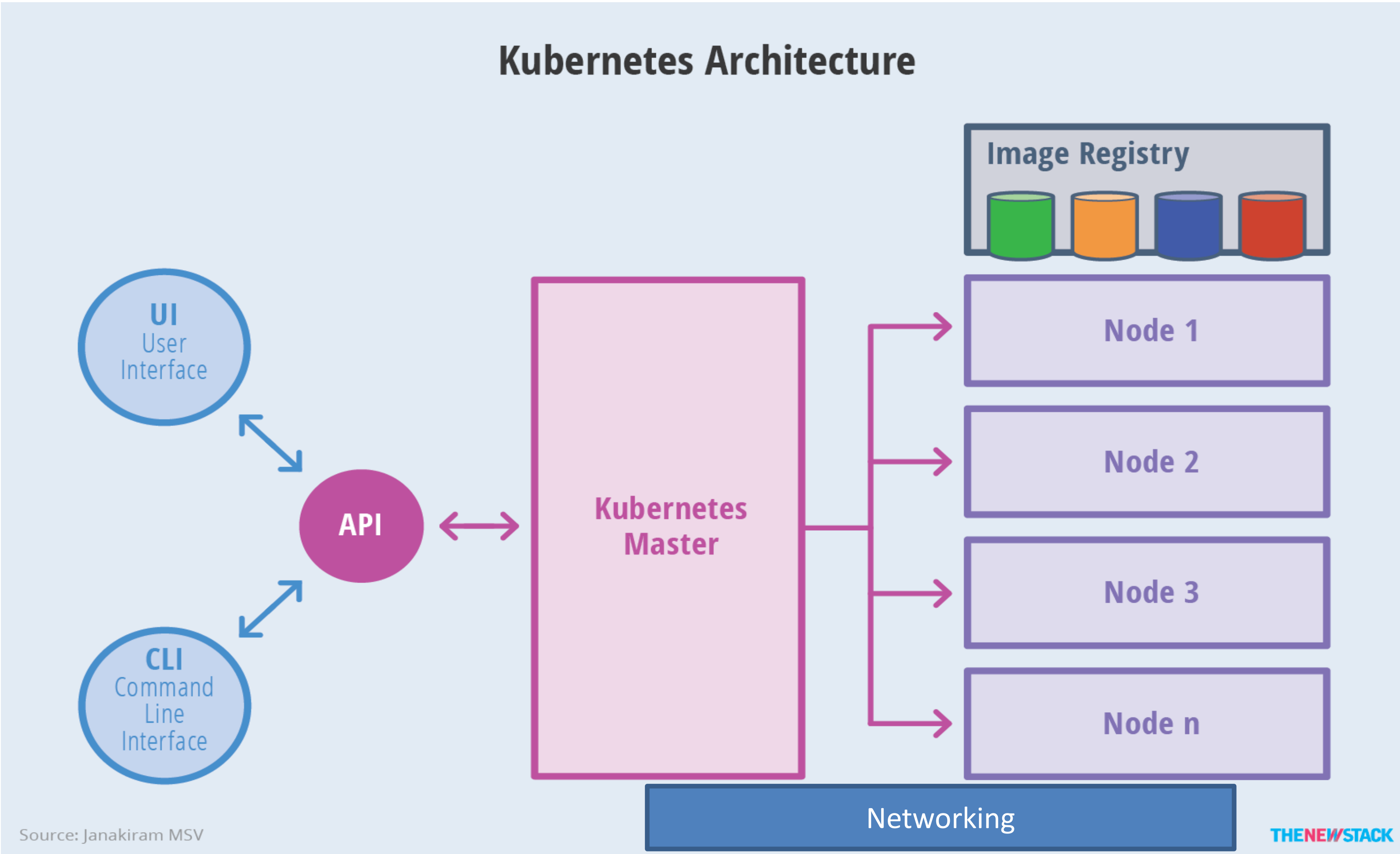
Key attributes of Kubernetes

- Declarative deployment model: desired state vs prescriptive state
  - Think about the states in a state transition diagram, e.g. scheduling, scheduled, running, completed, failed, #replicates, etc.
- Built-in replication and auto-scaling support
- Container centric networking model (key capability vs scheduler):
  - Networking among Containers, Pods, Services, Internet.
- Built-in application health checks
- Extensible: (e.g. custom resource definition)
- Support for broad set of application types:
  (e.g. early GPU support)
- Several services for cloud native applications
  (a collection of small, independent, and loosely coupled services.)



- containerd
- CRI-O
- Docker

https://kubernetes.io/blog/2018/05/24/kubernetes-containerd-integration-goes-ga/


Kubernetes Architecture
Image Registry
UI
User
Interface
API
CLI
Command
Line
Interface
Kubernetes
Master
Node 1
Node 2
Node 3
Node n
Networking
Source: Janakiram MSV
THENEWSTACK

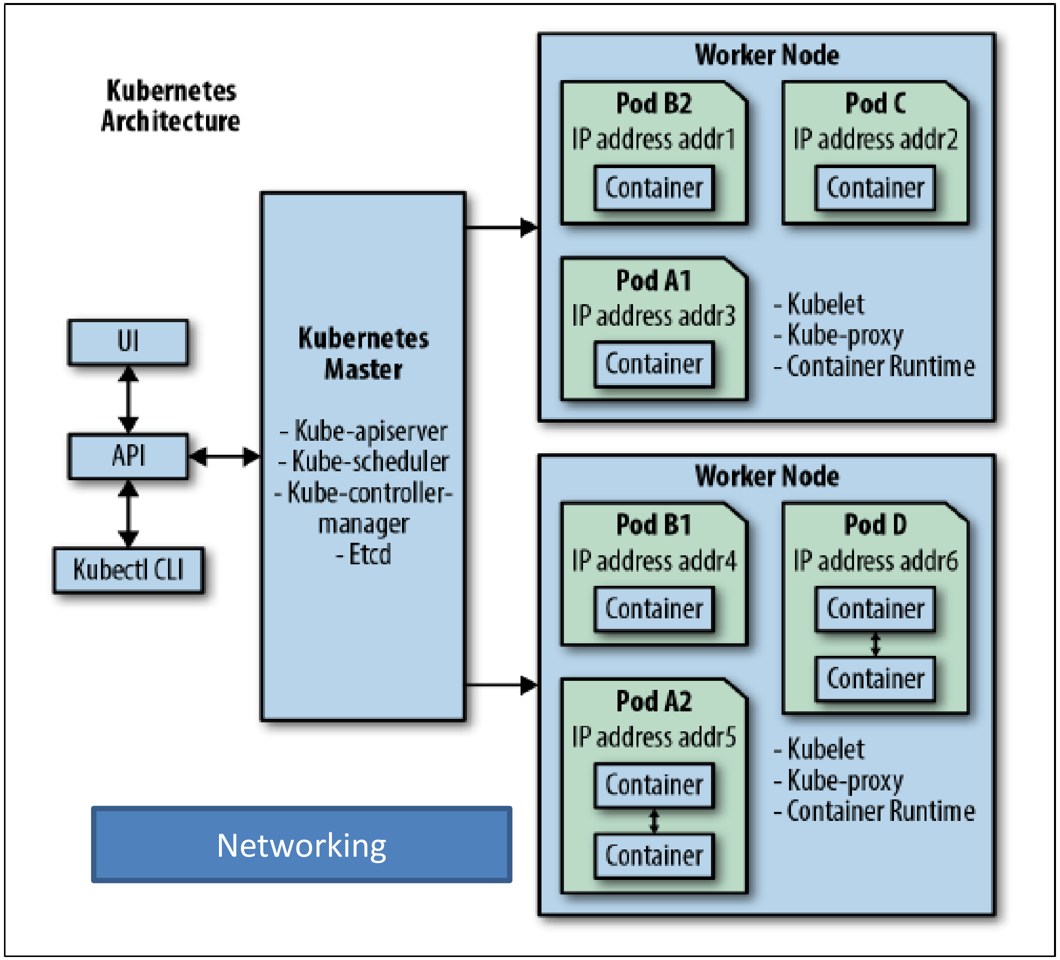


Figure 2-1. Graphical representation of the Kubernetes architecture

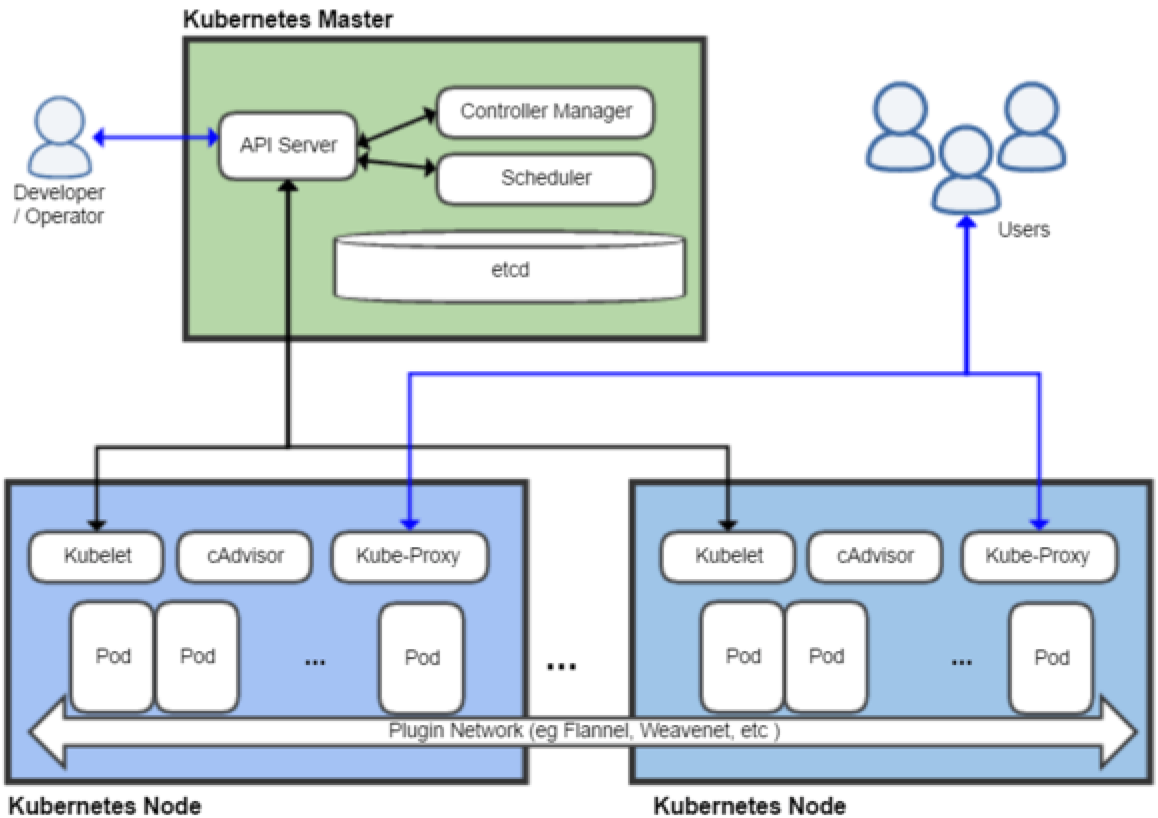
- Master Nodes
  - Api servers
  - Schedulers
  - Controllers
  - Etcd KV Store
- Worker Nodes
  - Kubelet
  - Proxy
  - Container runtime
  - Pods
  - Containers, Containers, containers

From O'Reilly K8S in the enterprise

Bring up Kubernetes in your laptop

- Follow steps to install Minikube here:
  - https://kubernetes.io/docs/tasks/tools/install-minikube/

- See here for other options:
  - https://opensource.com/article/20/11/run-kubernetes-locally
  - Kind
  - CRC
  - MiniShift


Kubernetes Master
API Server
Controller Manager
Scheduler
etcd
Developer
/ Operator
Users
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
...
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
Plugin Network (eg Flannel, Weavenet, etc)
Kubernetes Node
Kubernetes Node

Kubernetes basic objects and controllers

- Basic Kubernetes objects
  - **Pod**
  - **Service**
  - **Volume**
  - Namespace

- Kubernetes Machinery
  - **Kube API Server**
  - **Kube Scheduler**
  - **Controller-manager**
  - **Kubelet**
  - **Kube-proxy**

- Controllers
  - **ReplicaSet**
  - **Deployment**
  - StatefulSet
  - DaemonSet
  - Job

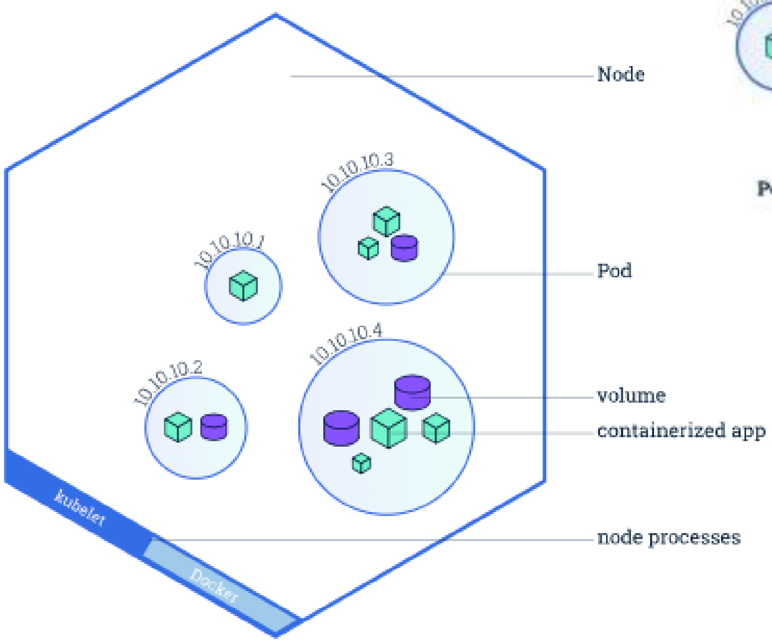
Pods

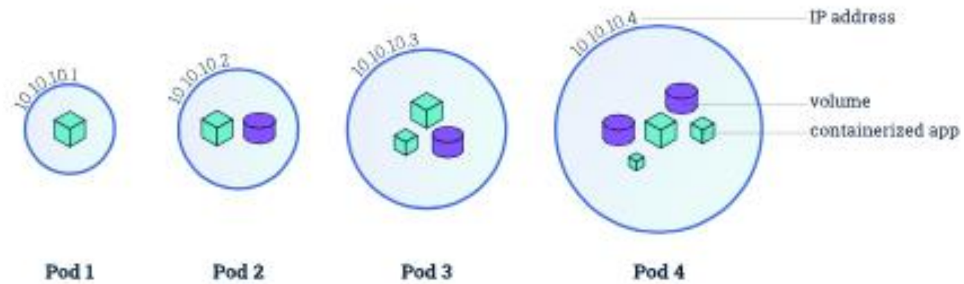
- *Pods* are the smallest deployable units of computing that you can create and manage in Kubernetes.

- *A Pod* is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers.

- A Pod's contents are always co-located and co-scheduled, and run in a shared context. A Pod models an application-specific "logical host": it contains one or more application containers which are relatively tightly coupled. In non-cloud contexts, applications executed on the same physical or virtual machine are analogous to cloud applications executed on the same logical host.

Nodes & Pods





Pod:

1. Containers run in Pods
2. Shared storage
3. Shared network
4. Container behavior (ports, health, resources)
5. All containers of a pod run on the same machine

Node:

1. Kubelet (communication between master → Node)
2. Container runtime (docker, rkt)

"Hello world" pod spec

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']
```

```
-bash-4.2$ kubectl create -f hello.yaml
pod/myapp-pod created
-bash-4.2$ kubectl get pods
NAME                                        READY     STATUS              RESTARTS   AGE
ibm-cert-manager-cert-manager-768b66977-qp6db   1/1       Running             0          12d
myapp-pod                                   0/1       ContainerCreating   0          5s
test-pd                                     1/1       Running             0          2d10h
-bash-4.2$
```

Differences between Docker and K8S

busybox /bin/sh -c 'echo hello && sleep 10'

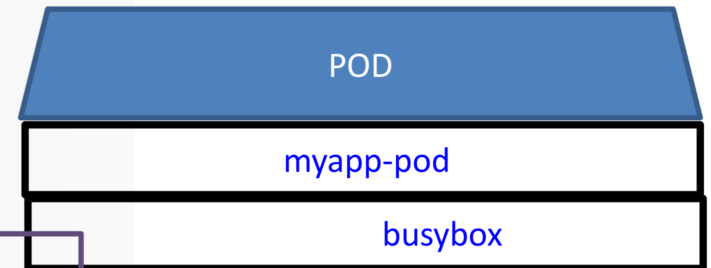
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']
```

Pod spec to Pod

Pod labels itself

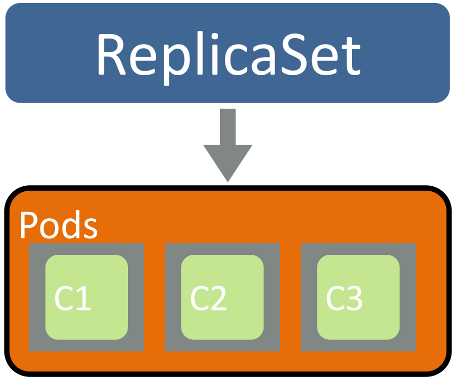
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']
```

Container def



Kubernetes ReplicaSets

- ReplicaSets allow multiple copies of pods to be deployed
- One or more containers in a pod
- If a container dies, the ReplicaSet will spawn a new one
- Mostly motivated from web services:
  - Stateless.
  - Scale-up and down to match the volume of service threads.
  - Microservice architecture.
  - Easy to meet availability requirements.



ReplicaSet

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-rs
  labels:
    app: myapp-rs
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: myapp-rs
  template:
    metadata:
      labels:
        tier: myapp-rs
    spec:
      containers:
      - name: myapp-container
        image: busybox
        command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']
```

ReplicaSet (RS) labels itself

To identify a member pod

Pod labels itself

Container def

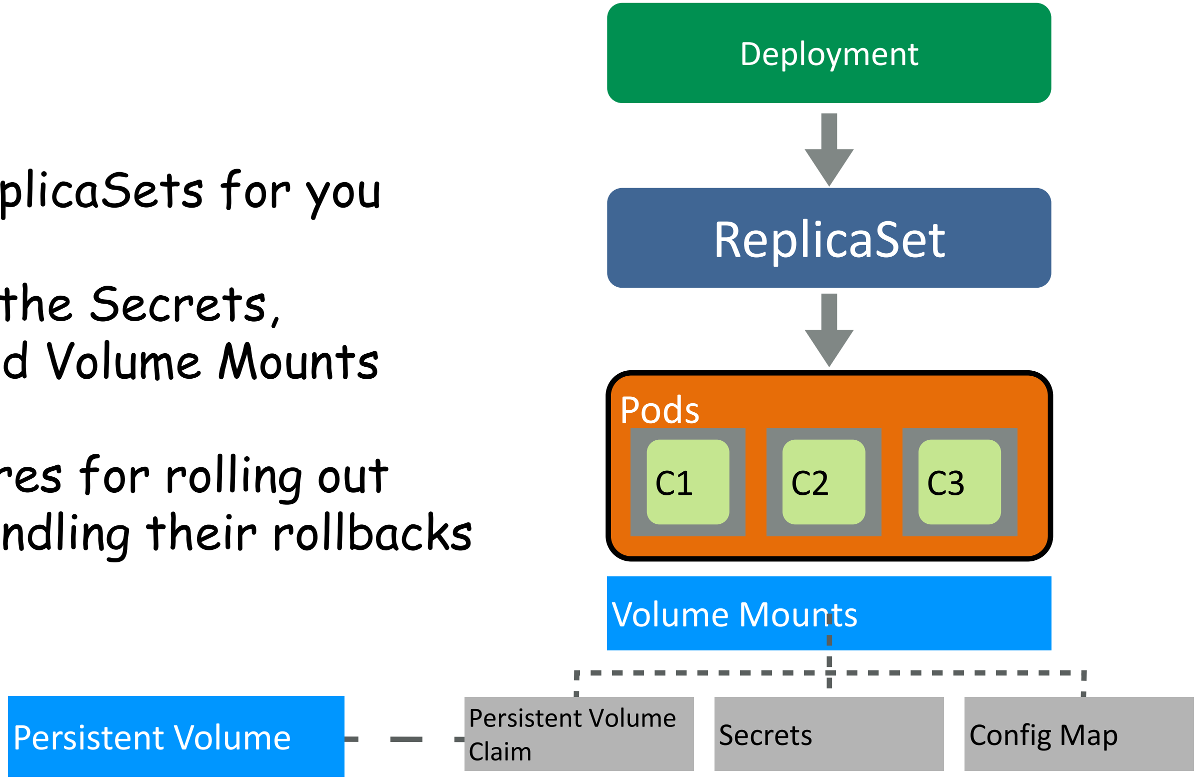
ReplicaSet vs Pod

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-rs
  labels:
    app: myapp-rs
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: myapp-rs
  template:
    metadata:
      labels:
        tier: myapp-rs
    spec:
      containers:
      - name: myapp-container
        image: busybox
        command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 36']
```

Kubernetes Deployments

- Deployment
  - Sets up the ReplicaSets for you
  - Also specified the Secrets, ConfigMaps, and Volume Mounts
  - Provides features for rolling out updates and handling their rollbacks



Deployment (better than ReplicaSet) : roll-out

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-deployment
  labels:
    app: hello-d
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello-d
  template:
    metadata:
      labels:
        app: hello-d
    spec:
    spec:
      containers:
      - name: myapp-container
        image: busybox
        command: ['sh', '-c', 'echo Hello Kubernetes! I am updated!! && sleep 3600']
```

Deployment labels itself

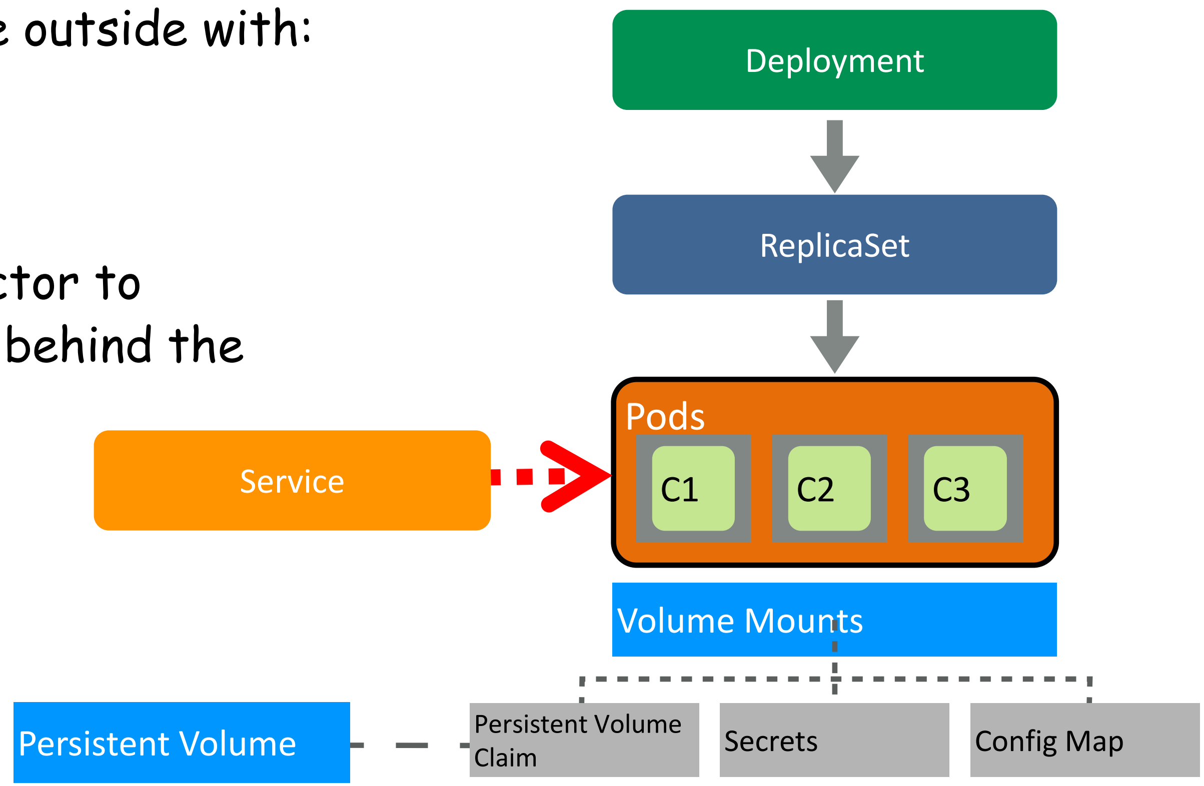
To identify a member pod

Pod labels itself

Container def

Kubernetes Service

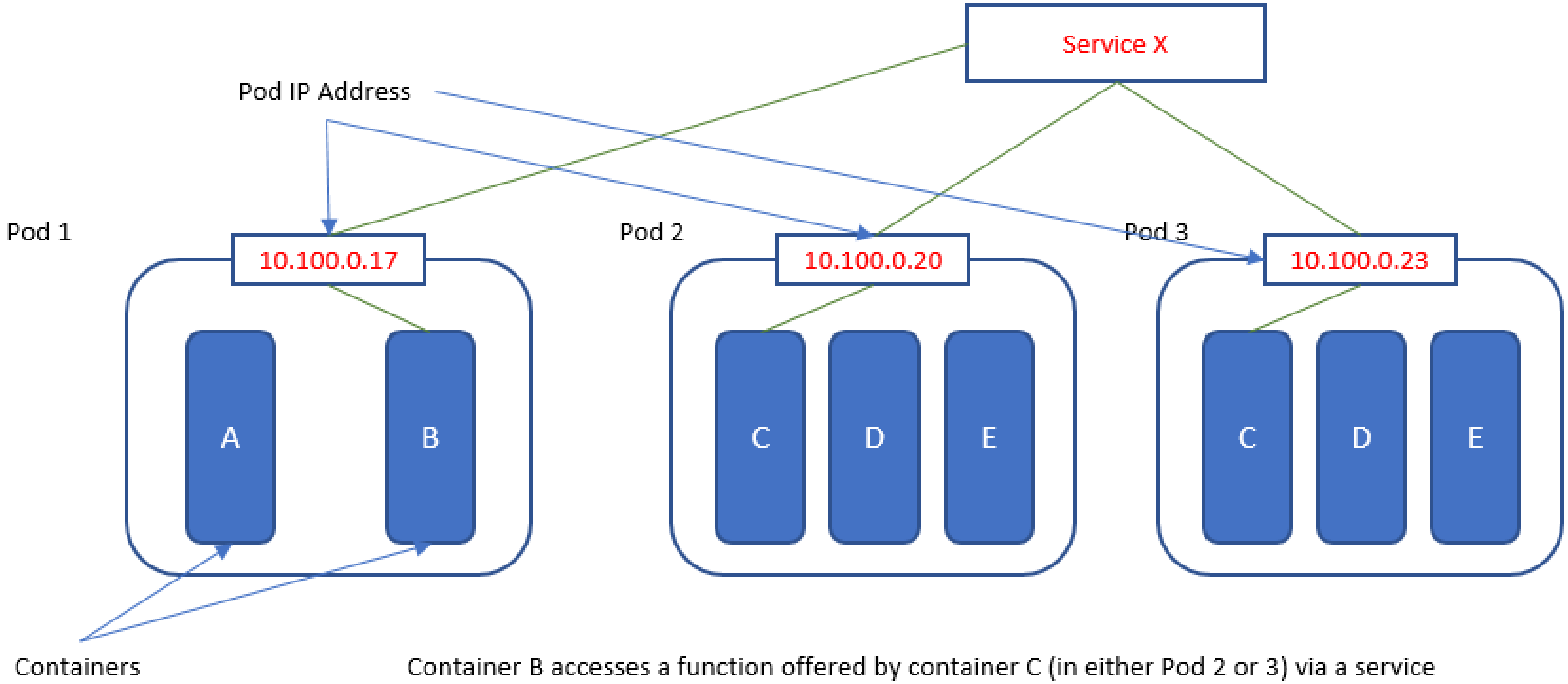
- Abstract way to expose apps in pods to web
  - Exposes Pods to the outside with:
    - ClusterIP
    - NodePort
    - Load Balancer
  - Normally use a selector to determine the pods behind the service



Service Example

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: hello-d
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Kubernetes Service

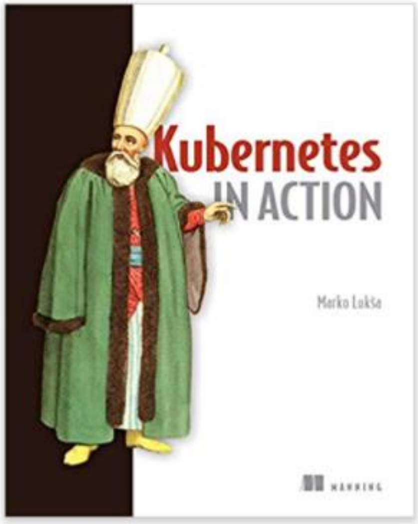


Container B accesses a function offered by container C (in either Pod 2 or 3) via a service

https://en.wikipedia.org/wiki/Kubernetes

Suggested Study Material

- Introduction to Kubernetes:
  - https://www.digitalocean.com/community/tutorials/an-introduction-to-kubernetes
- Facebook Tupperware:
  - https://engineering.fb.com/data-center-engineering/tupperware/
- Kubernetes deconstructed:
  - https://www.youtube.com/watch?v=90kZRyPcRZw
  - http://kube-decon.carson-anderson.com/Layers/0-Intro.sozi.html#frame5378
- The History of Kubernetes on a Timeline:
  - https://blog.risingstack.com/the-history-of-kubernetes/
- The State of the Kubernetes Ecosystem [eBook]:
  - https://thenewstack.io/ebooks/kubernetes/state-of-kubernetes-ecosystem/
- Kubernetes In Action by Marko Luksa [Book]



The final word (before the final)

- Thanks for joining the class
(I know it is a mandatory class, but …)
- Its been a fun class, hope for you as well
- I hope you learned how operating systems at a high level function and how they interact with the platform/system
- Good Luck with the Final Exam