Introduction to Containers, Kubernetes, and Cloud Technology

Patrick Ladd
Poughkeepsie ACM
November 2021

pmladd@gmail.com / pladd@redhat.com /
people.redhat.com/pladd

Containers

What is a container?

A. A file format
B. A runtime environment
C. A process

Containers

What is a container?

A. A file format
B. A runtime environment
C. A process
✓ All of the above

Container Technology

Containers

"Linux Containers" is a Linux kernel feature to contain a group of processes in an independent execution environment

The kernel provides an independent application execution environment for each container including:

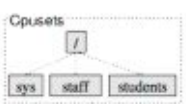
- Independent filesystem
- Independent network interface and IP address
- Usage limit for resources - memory / CPU time / etc.

Linux containers are realized with integrating many existing Linux features. There are multiple container management tools such as lxctools, libvirt and docker. They may use different parts of these features.

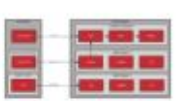
Container History

2000:
JAILS ADDED
TO FREEBSD

2003:
SELINUX ADDED TO
LINUX MAINLINE



2006:
GENERIC PROCESS
CONTAINERS



2008:
KERNEL AND USER
NAMESPACES

2013:
RED HAT
ENTERPRISE LINUX

OPENSHIFT
RED HAT
ATOMIC ENTERPRISE
PLATFORM

2015:
RHT CONTAINER
PLATFORMS

Linux
VServer

2001:
LINUX -VSERVER
PROJECT

2005:
FULL RELEASE OF
SOLARIS ZONES

2007:
GPC RENAMED
CONTROL GROUPS

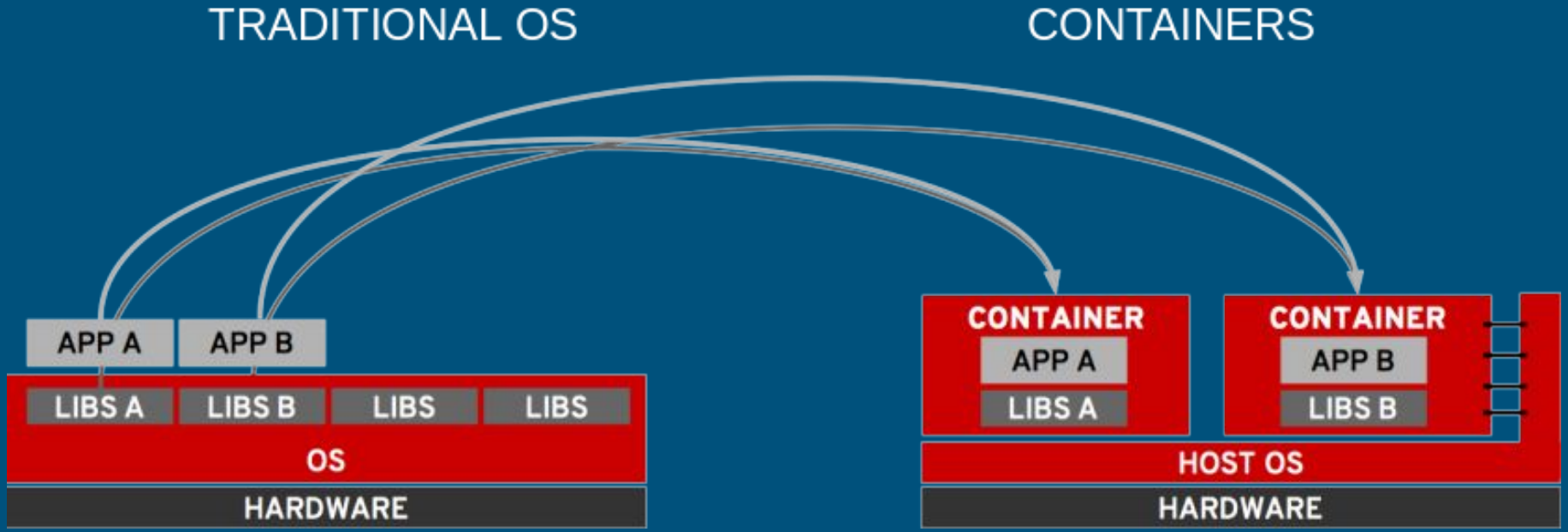
2013:
DOT CLOUD PYCON
LIGHTNING TALK

2014:
GOOGLE
KUBERNETES

2015:
STANDARDS VIA OCI
AND CNCF

2000
2005
2010
2015

Containers
≠
Virtualization


TRADITIONAL OS
CONTAINERS
APP A
APP B
LIBS A
LIBS B
LIBS
LIBS
OS
HARDWARE
CONTAINER
APP A
LIBS A
CONTAINER
APP B
LIBS B
HOST OS
HARDWARE

Underlying Technology

Enabling Technology in Linux has been present for many years

- Namespaces
  - Process
  - Network
  - Filesystem
  - User
  - IPC
  - UTS (UNIX Technology Services)
- cgroups - Control Groups
- Union (overlay) Filesystems

Namespces

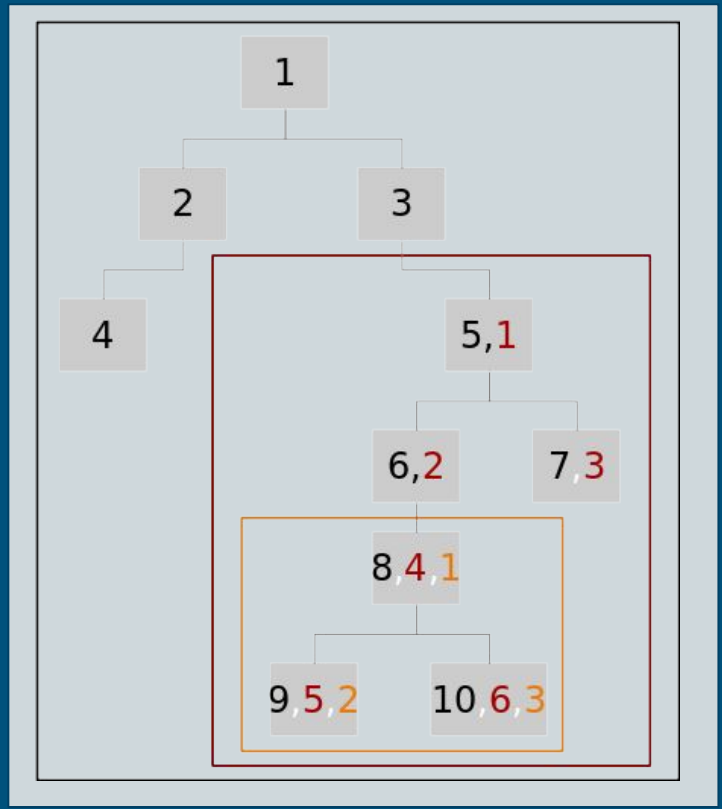
Process Namespaces

Original UNIX Process Tree

- First process is PID 1
- Process tree rooted at PID 1
- PIDs with appropriate privilege may inspect or kill other processes in the tree

Linux Namespaces

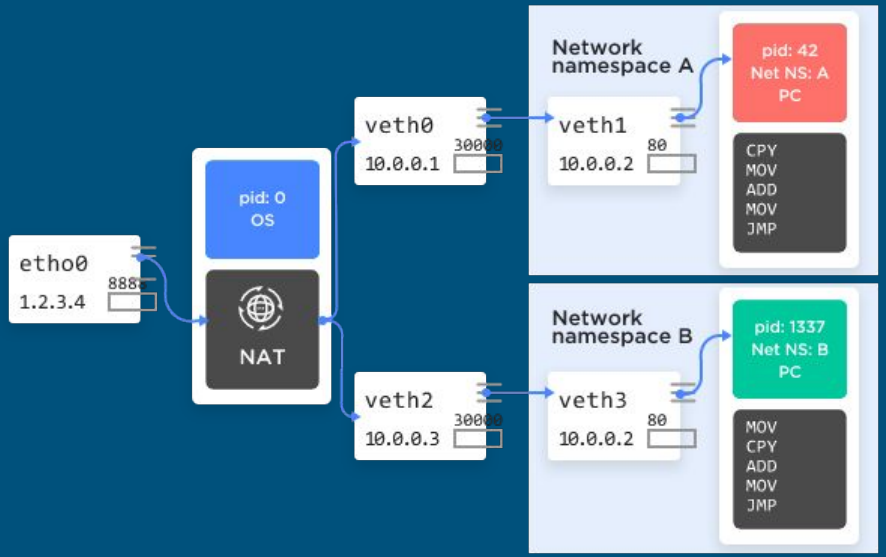
- Multiple, nested process trees
- Nested trees cannot see parent tree
- Process has multiple PIDs
- One for each namespace it is a member of



Network Namespaces

Presents an entirely separate set of network interfaces to each namespace

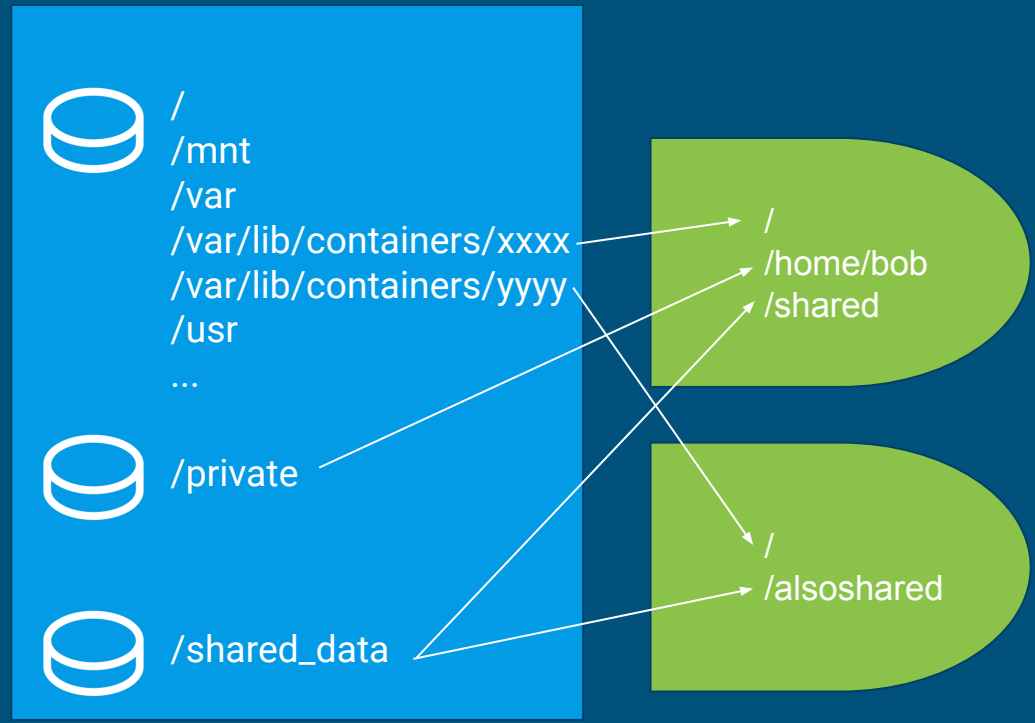
- All interfaces including loopback are virtualized
- Ethernet bridges may be created
  - `ip link add name veth0 type veth peer name veth1 netns <pid>`
- Routing process in global namespace to route packets



Filesystem Namespaces

Clone / Replace list of mounted filesystems

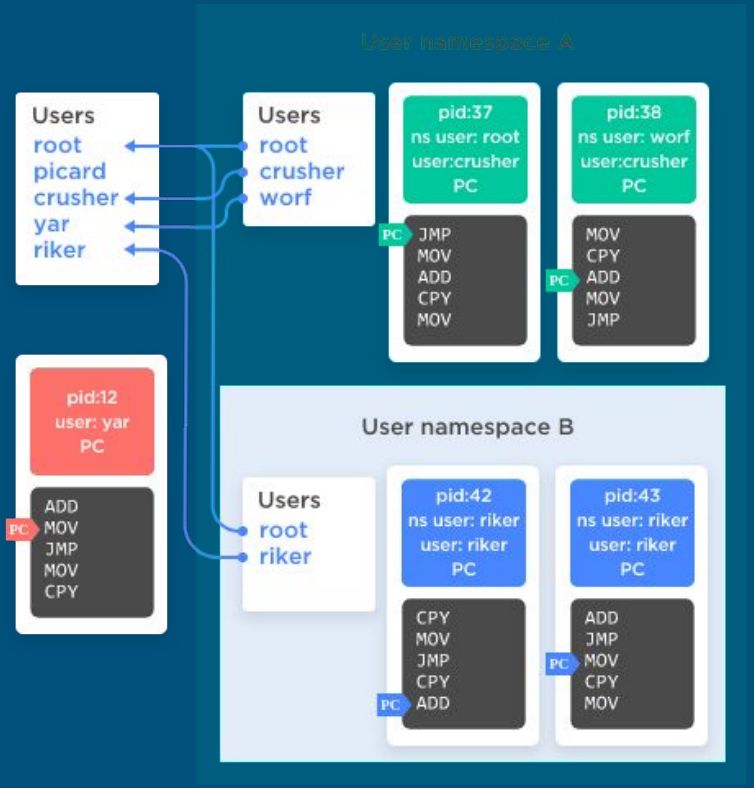
- Similar to chroot
- Allows isolation of all mount points, not just root
- Attributes can be changed between namespaces (read only, for instance)
- Used properly, avoids exposing anything about underlying system



User Namespaces

Replace / Extend UID / GID

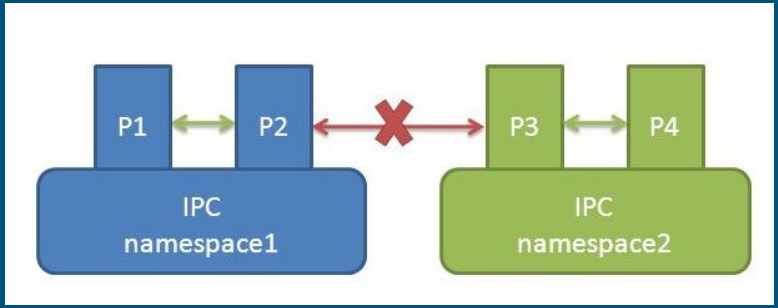
- Delete unneeded UID / GID from container
- Add / change UID / GID map inside container
- Use: root privilege in container, user privilege in base OS



IPC Namespaces

Similar to network namespaces

- Separate interprocess communications resources
- Sys V IPC
- POSIX messaging

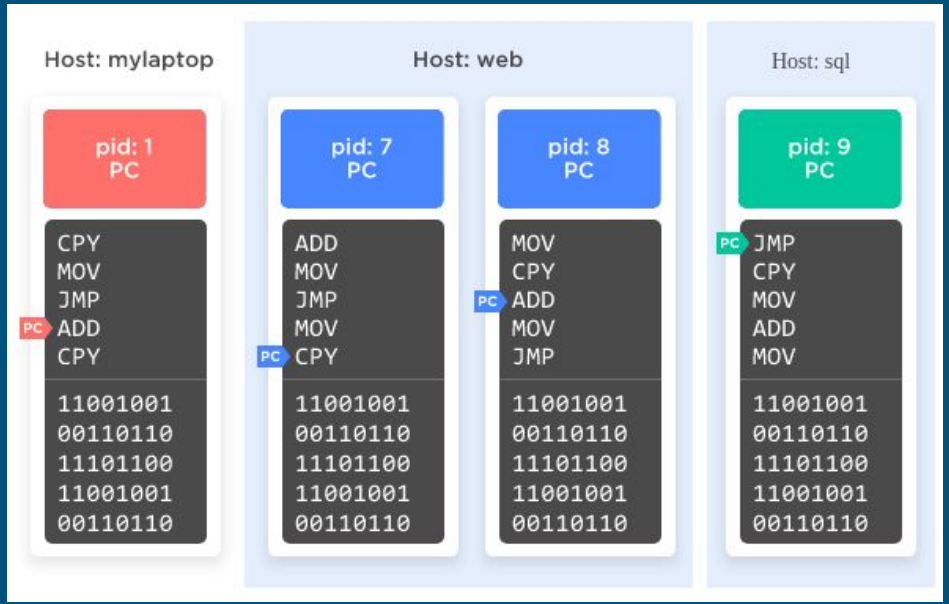


UTS Namespaces

UTS : UNIX Technology Services

Change inside container:

- Hostname
- Domain



Feature availability

Filesystem separation - Mount namespace (kernel 2.4.19)

Hostname separation - UTS namespace (kernel 2.6.19)

IPC separation - IPC namespace (kernel 2.6.19)

User (UID/GID) separation - User namespace (kernel 2.6.23〜kernel 3.8)

Processtable separation - PID namespace (kernel 2.6.24)

Network separation - Network Namespace (kernel 2.6.24)

Usage limit of CPU/Memory - Control groups (kernel 2.6.24)

Namespaces Summary

Isolation / Modification of Container processes from host

- PIDs
- Network
- Filesystems
- UID/GID
- IPC
- Hostname / Domain

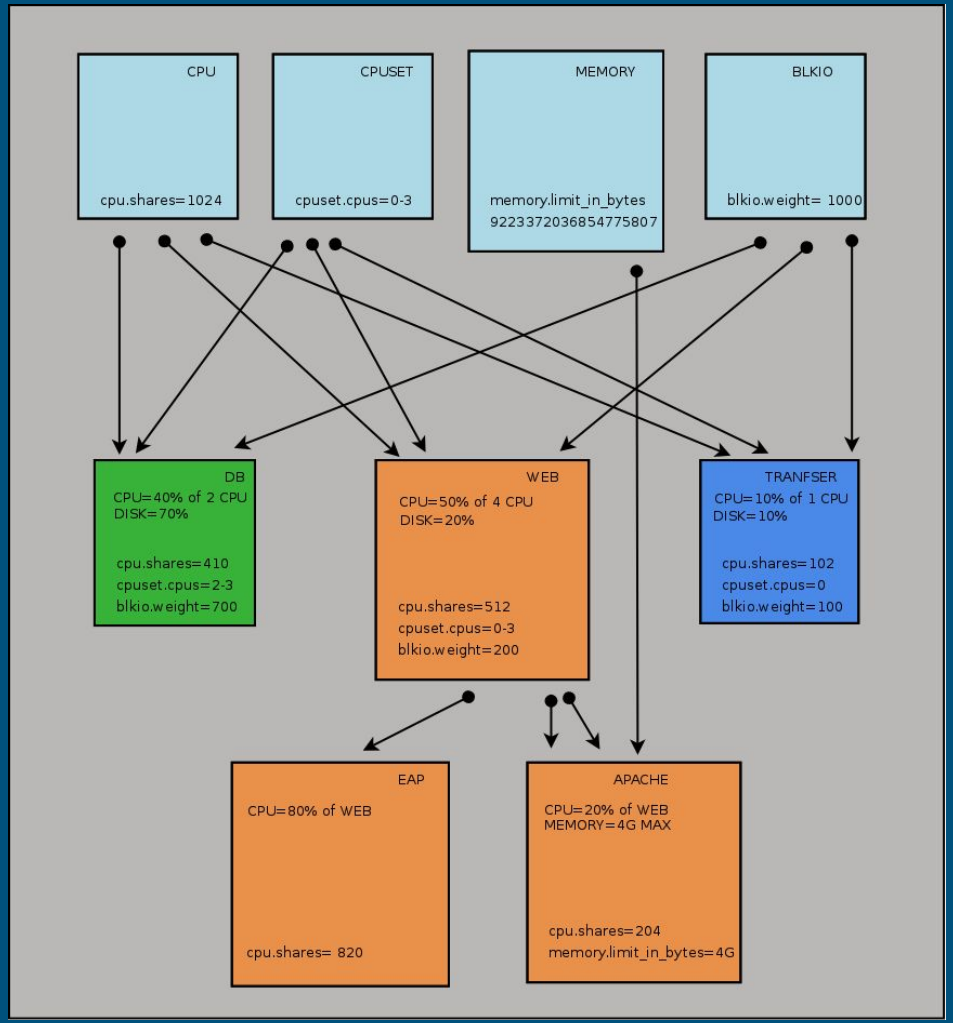
See documentation on clone() system call for more complete details on functionality (Warning: systems programmer jargon territory)

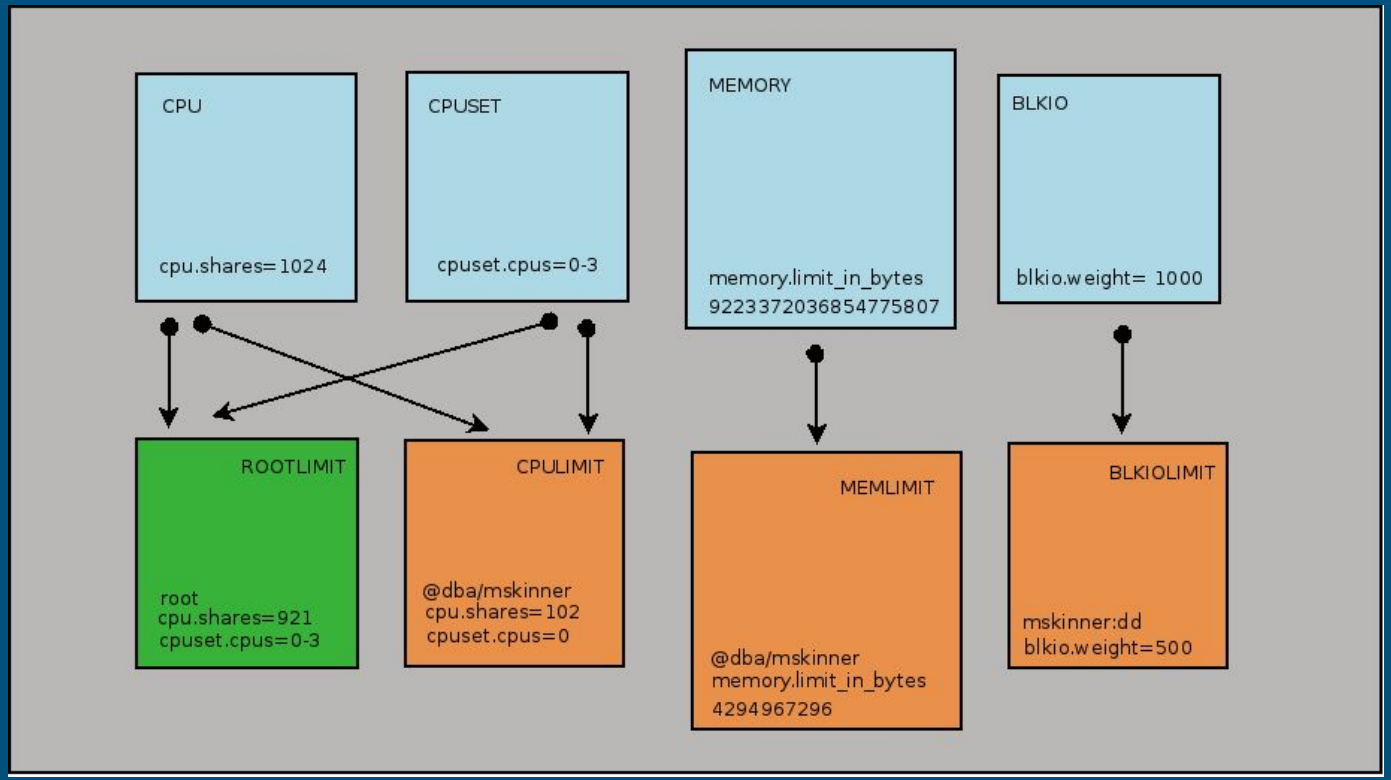
cgroups

Control Groups

- Control allocation of resources to processes running on a system
- Hierarchical and can be dynamically added, changed and removed
- Made up of several subsystems also called Resource Controllers

- cgroups-v1 since kernel 2.6.24
- You must install userspace tools
  - Install libcgroup


CPU
cpu.shares=1024
CPUSET
cpuset.cpus=0-3
MEMORY
memory.limit_in_bytes
9223372036854775807
BLKIO
blkio.weight= 1000
DB
CPU=40% of 2 CPU
DISK=70%
cpu.shares=410
cpuset.cpus=2-3
blkio.weight=700
WEB
CPU=50% of 4 CPU
DISK=20%
cpu.shares=512
cpuset.cpus=0-3
blkio.weight=200
TRANFSER
CPU=10% of 1 CPU
DISK=10%
cpu.shares=102
cpuset.cpus=0
blkio.weight=100
EAP
CPU=80% of WEB
cpu.shares= 820
APACHE
CPU=20% of WEB
MEMORY=4G MAX
cpu.shares=204
memory.limit_in_bytes=4G


CPU
cpu.shares=1024
CPUSET
cpuset.cpus=0-3
MEMORY
memory.limit_in_bytes
9223372036854775807
BLKIO
blkio.weight= 1000
ROOTLIMIT
root
cpu.shares=921
cpuset.cpus=0-3
CPULIMIT
@dba/mskinner
cpu.shares=102
cpuset.cpus=0
MEMLIMIT
@dba/mskinner
memory.limit_in_bytes
4294967296
BLKIOLIMIT
mskinner:dd
blkio.weight=500

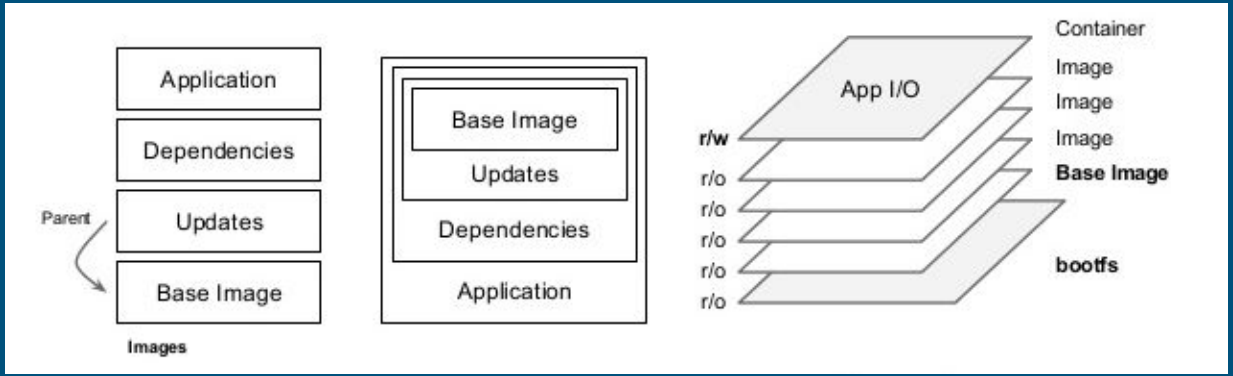
Resource Controllers

- **blkio** — this subsystem sets limits on input/output access to and from block devices such as physical drives (disk, solid state, USB, etc.)
- **cpu** — this subsystem uses the scheduler to provide cgroup tasks access to the CPU
- **cpuacct** — this subsystem generates automatic reports on CPU resources used by tasks in a cgroup
- **cpuset** — this subsystem assigns individual CPUs (on a multicore system) and memory nodes to tasks in a cgroup
- **devices** — this subsystem allows or denies access to devices by tasks in a cgroup
- **freezer** — this subsystem suspends or resumes tasks in a cgroup
- **memory** — this subsystem sets limits on memory use by tasks in a cgroup, and generates automatic reports on memory resources used by those tasks
- **net_cls** — this subsystem tags network packets with a class identifier (classid) that allows the Linux traffic controller (tc) to identify packets originating from a particular cgroup task
- **net_prio** — this subsystem provides a way to dynamically set the priority of network traffic per network interface
- **ns** — the namespace subsystem

Union (overlay) Filesystems

Union Filesystems

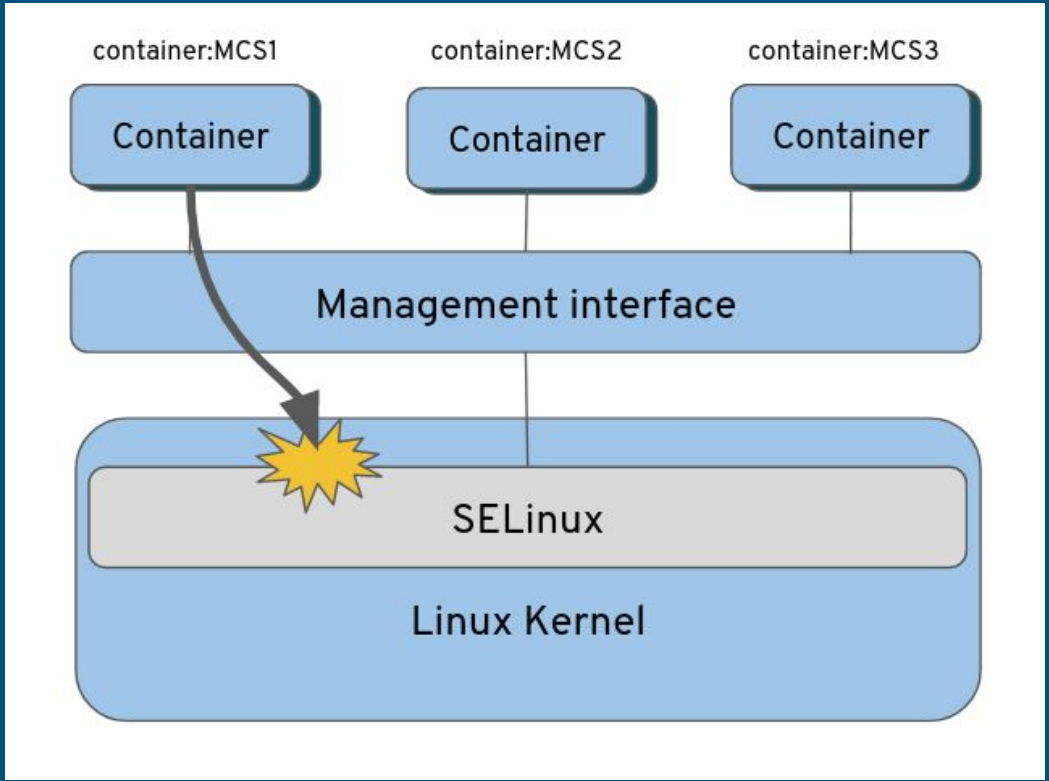
- Stacked / Layered Storage
- Copy on write
- Many available underlying implementations
  - Aufs
  - OverlayFS
  - Btrfs
  - LVM
  - Device mapper



Container Security

CONTAINERS ARE NOT SECURE BY DEFAULT

Container Isolation with SELinux



SELinux - Learn More

Coloring books!

Container Commandos! - sequel to the Selinux Coloring Book

Container Tooling

Tooling

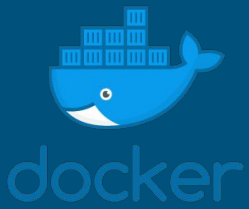
Container Operations

- Build Container Images
  - Originally - dockerfile
  - Now - many options
- Manage Container Images
  - Manage layers / Dependencies
  - Download / Upload - Container registries
  - List / Delete
  - Naming / Tagging
- Run Container Images
  - Image name
  - Namespace data
  - Mountpoints
  - Exposed Ports
  - et. al.

Tooling

Docker vs. docker

- Bundles all operations in daemon running as root



Emergence of OCI (Open Container Initiative) - 2015

- Runtime Specification
- Image Specification

Proliferation of Tools

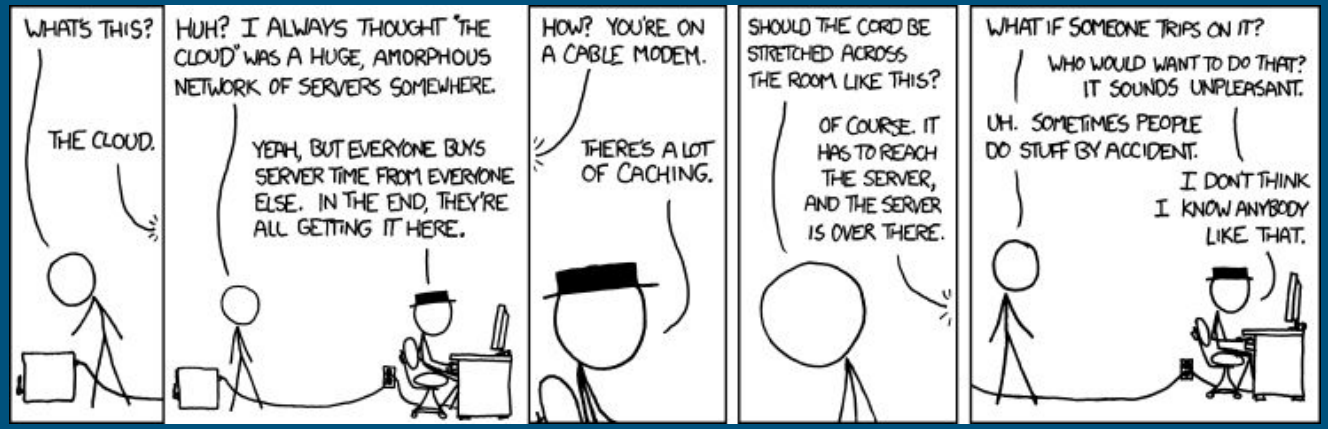
- Podman / Buildah / Skopeo
- cri-o - Container Runtime for OpenShift

Podman / Buildah / Skopeo

Red Hat implementation of OCI Standards

- Podman - Container Runtime
  - New / in progress - rootless Podman
- Buildah - Build Container Images
  - Traditional dockerfile
  - Layer from result of container run
  - bash script
  - Direct mount to local machine
- Skopeo
  - Copy to/from multiple different registries
  - No local copy required

Container Orchestration

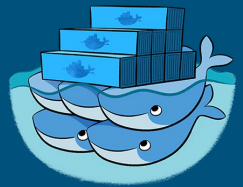

WHAT'S THIS?
THE CLOUD.
HUH? I ALWAYS THOUGHT "THE CLOUD" WAS A HUGE, AMORPHOUS NETWORK OF SERVERS SOMEWHERE.
YEAH, BUT EVERYONE BUYS SERVER TIME FROM EVERYONE ELSE. IN THE END, THEY'RE ALL GETTING IT HERE.
HOW? YOU'RE ON A CABLE MODEM.
THERE'S A LOT OF CACHING.
SHOULD THE CORD BE STRETCHED ACROSS THE ROOM LIKE THIS?
OF COURSE. IT HAS TO REACH THE SERVER, AND THE SERVER IS OVER THERE.
WHAT IF SOMEONE TRIPS ON IT?
WHO WOULD WANT TO DO THAT? IT SOUNDS UNPLEASANT.
UH. SOMETIMES PEOPLE DO STUFF BY ACCIDENT.
I DON'T THINK I KNOW ANYBODY LIKE THAT.


THERE
IS NO
CLOUD
ITS JUST SOMEONE
ELSE'S COMPUTER

More Containers - More Problems

- “Container Orchestration”
  - Managing containers at scale, even on a single machine, is a problem
  - Deployment
  - Networking
  - Management

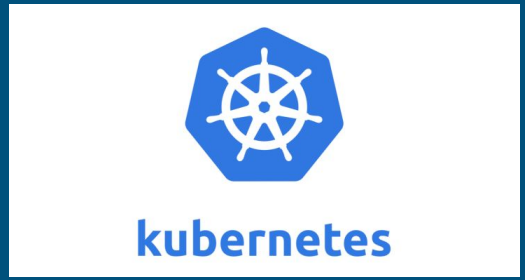
- Docker Swarm - 2016
  - Simplicity as the goal
  - Lacking many features - suitable for some single-host orchestration



Kubernetes

Enter Kubernetes

- Kubernetes - Summer 2014
  - Google project
  - Kubernetes (κυβερνήτης, Greek for "helmsman" or "pilot" or "governor")
  - Abbreviated “k8s”
  - Based on “Project Borg” - original name was “Project 7” for “Seven of Nine” from Star Trek
  - Now run by CNCF (Cloud Native Computing Foundation)
  - Many other companion projects now part of CNCF - see trail map



Kubernetes Principles

“Pets vs Cattle”

- Pods / Nodes are subject to being terminated at any time for any reason
- Clusters / Applications should be architected for this
  - No dependencies on particular nodes / containers running continuously
  - Use k8s features to build application redundancy
- Pod IP addresses are ephemeral - should be grouped in services

Redundancy

Multi-tenancy

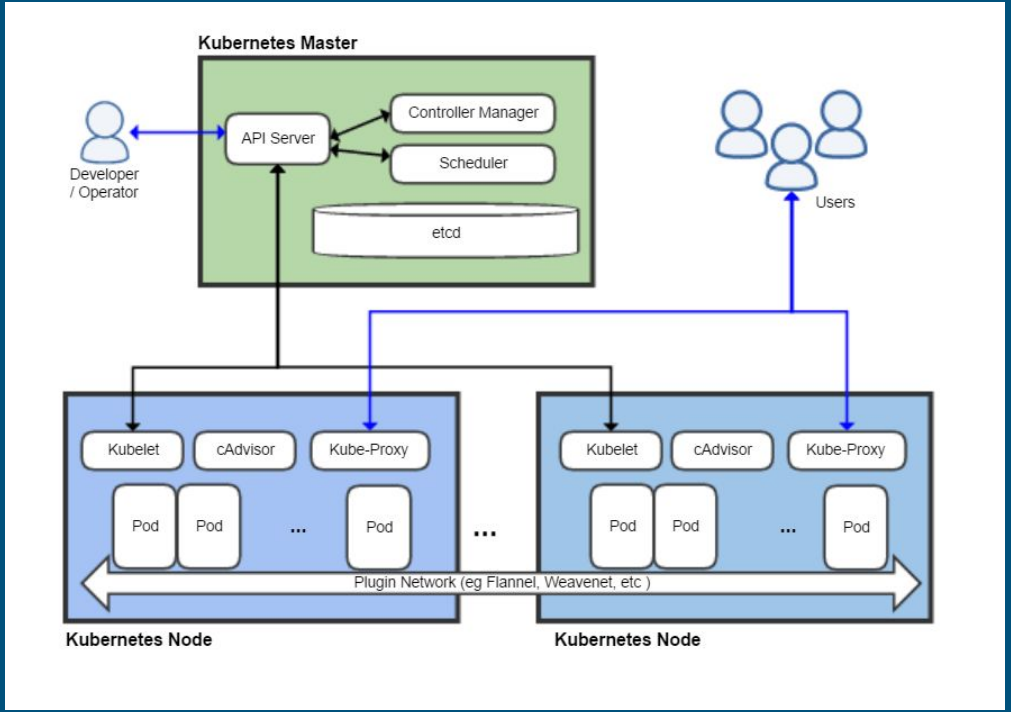
Kubernetes Architecture

Node Types:

- Master(s): Machine(s) for managing the cluster a.k.a. control plane
- Node(s): Workers for scheduling pods on

Master software components:

- API server
- etcd - key/value pair storage of cluster state
- Scheduler - manages placement of pods
- Controller Manager - works to maintain cluster in currently specified state

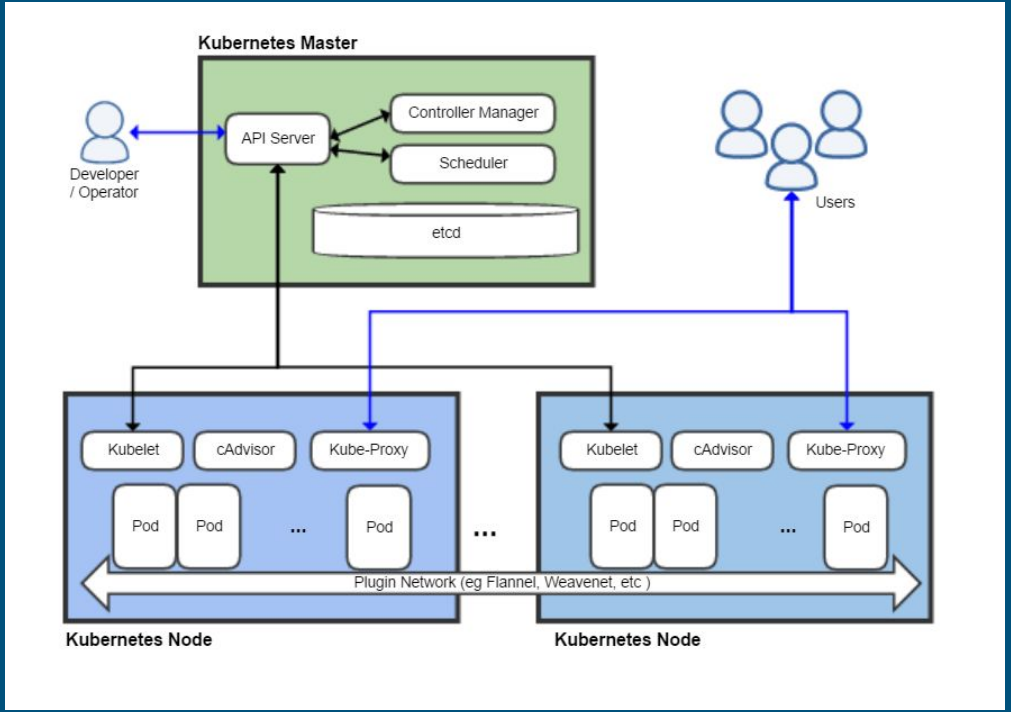


https://kubernetes.io/docs/concepts/overview/components/

Kubernetes Architecture

Major Node software:

- Kubelet: manages node state via API
- Kube-proxy: Cluster network plugin
- Plugin Network: manages actual implementation of cluster network
- cAdvisor: basic metrics gathering



https://kubernetes.io/docs/concepts/overview/components/

Kubernetes Objects

- Stored in etcd
- Accessed by API
- State information for cluster
- YAML Syntax

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Kubernetes Terms / Concepts

- Pod: Smallest schedulable entity - runtime grouping of 1 or more containers
  - All scheduled on same node
  - Dedicated unique IP address
- Sidecar: Secondary / helper containers in a pod other than the “primary” container
- Namespace: Project-based mechanism to separate applications from each other - allows for multiple users / teams / applications to be deployed on a single cluster
- Label: A way to identify characteristics of particular pods or other k8s objects
- Selector: A way to pick k8s objects based on common characteristics, especially labels
- ConfigMaps and Secrets: Storage location for common configuration data used by pods

Kubernetes Scheduling

- Bare pods - not recommended
- ReplicaSet: group of pods with defined number of identical deployed pods
- StatefulSet: group of pods with defined requirements for ordering / scaling - used to manage stateful applications
- DaemonSet: group of pods with on pod deployed to each node with specified labels. Most often used for utility services

Deployments: Managed rollout of pods / ReplicaSets

In newer versions:

- Jobs: Runs pods until a specific number are successfully executed
- CronJob: Runs jobs at specific intervals

Scheduling controls:

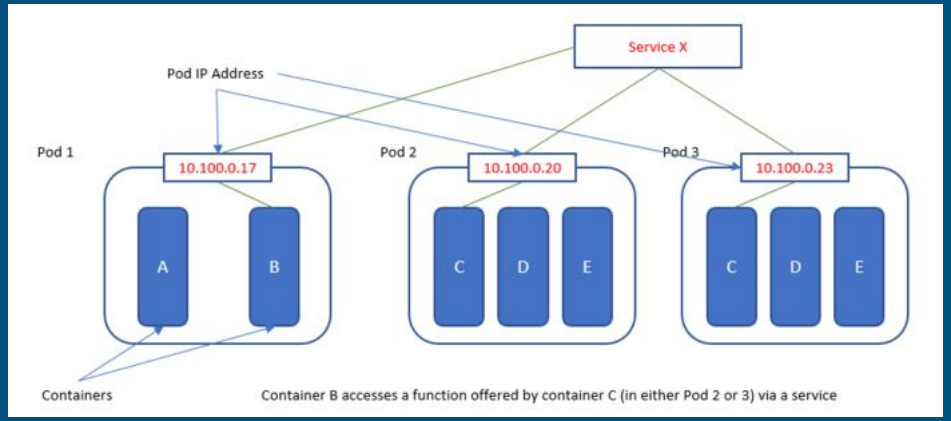
Filtering & Scoring

- Resource requests (CPU / memory / Network / other & custom)
- Taints & Tolerations
- nodeSelector

Kubernetes Networking

- Pods do not generally operate with Node IPs - they use a cluster virtual network
- Pods are assigned unique ephemeral IP addresses on the cluster's virtual network
- Accessing pods this way is discouraged Instead use one of:

- Service: A set of pods addressable through a common IP / DNS name
- Ingress: External access to Services



Kubernetes Storage

Ephemeral Storage

- Many built-in types supported, only guaranteed for life of container

Persistent Storage

- Static - pre-allocated by cluster
- Dynamic - managed by a StorageClass
- PersistentVolume (PV):
  - Provide storage to mount into a pod
  - Characteristics:
    - Read / Write or Read Only
    - Once or Many
- PersistentVolumeClaim (PVC):
  - Mapping of a PV onto a particular pod or pods

Kubernetes Storage - Continued

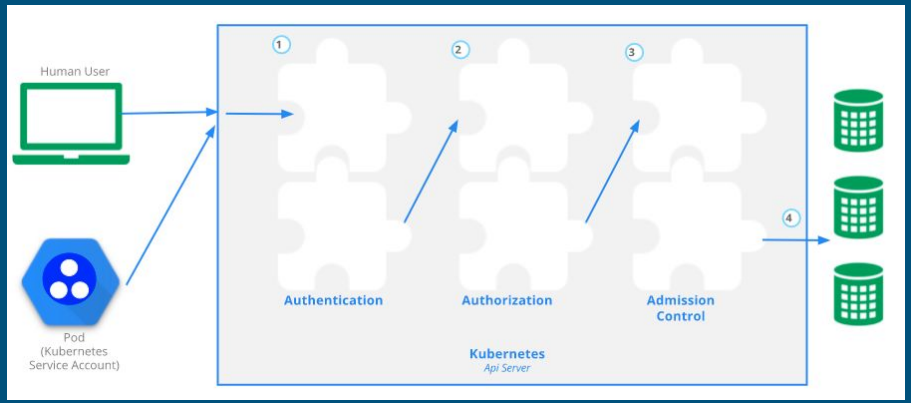
Snapshots - lifecycle controlled by VolumeSnapshotClass

Many storage providers available:

- awsElasticBlockStore - AWS Elastic Block Store (EBS)
- azureDisk - Azure Disk
- azureFile - Azure File
- cephfs - CephFS volume
- csi - Container Storage Interface (CSI)
- fc - Fibre Channel (FC) storage
- flexVolume - FlexVolume
- gcePersistentDisk - GCE Persistent Disk
- glusterfs - Glusterfs volume
- hostPath - HostPath volume (for single node testing only; WILL NOT WORK in a multi-node cluster; consider using local volume instead)
- iscsi - iSCSI (SCSI over IP) storage
- local - local storage devices mounted on nodes.
- nfs - Network File System (NFS) storage
- portworxVolume - Portworx volume
- rbd - Rados Block Device (RBD) volume
- vsphereVolume - vSphere VMDK volume

Kubernetes Security

- Namespaces
- Security Profiles
  - Privileged
  - Baseline
  - Restricted
- Authentication / Authorization



Supporting Infrastructure

Kubernetes is Incomplete

- Running bare kubernetes is hard!
- Distributions of Kubernetes fill out the experience for consumption

Things to look for in a bundle

- Container Registry
- Networking Plugin
- Operators
- CI/CD Pipelines
- API Extensions
- Management Tools
- Logging & Metrics
- Security Tooling
  - Authentication / Authorization / Encryption
  - Image Scanning
  - Runtime Scanning
  - Policy Enforcement
- Service Mesh
- Cross Cluster Connectivity
- Multi-Cluster Management

CLOUD NATIVE COMPUTING FOUNDATION

CLOUD NATIVE TRAIL MAP

The Cloud Native Landscape (*cncf.io*) has a large number of options. This Cloud Native Trail Map is a recommended process for leveraging open source, cloud native technologies. At each step, you can choose a vendor-supported offering or do it yourself, and everything after step #3 is optional based on your circumstances.

HELP ALONG THE WAY

A. Training and Certification

Consider training offerings from CNCF and then take the exam to become a Certified Kubernetes Administrator or a Certified Kubernetes Application Developer

cncf.io/training

B. Consulting Help

If you want assistance with Kubernetes and the surrounding ecosystem, consider leveraging a Kubernetes Certified Service Provider

cncf.io/kcsp

C. Join CNCF's End User Community

For companies that don't offer cloud native services externally

cncf.io/enduser

WHAT IS CLOUD NATIVE?

Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

The Cloud Native Computing Foundation seeks to drive adoption of this paradigm by fostering and sustaining an ecosystem of open source, vendor-neutral projects. We democratize state-of-the-art patterns to make these innovations accessible for everyone.

l.cncf.io

v20200501

1. CONTAINERIZATION

- Commonly done with Docker containers
- Any size application and dependencies (even PDP-11 code running on an emulator) can be containerized
- Over time, you should aspire towards splitting suitable applications and writing future functionality as microservices

2. CI/CD

- Setup Continuous Integration/Continuous Delivery (CI/CD) so that changes to your source code automatically result in a new container being built, tested, and deployed to staging and eventually, perhaps, to production
- Setup automated rollouts, roll backs and testing
- Argo is a set of Kubernetes-native tools for deploying and running jobs, applications, workflows, and events using GitOps paradigms such as continuous and progressive delivery and MLops

argo — CNCF Incubating

3. ORCHESTRATION & APPLICATION DEFINITION

- Kubernetes is the market-leading orchestration solution
- You should select a Certified Kubernetes Distribution, Hosted Platform, or Installer: cncf.io/ck
- Helm Charts help you define, install, and upgrade even the most complex Kubernetes application

kubernetes — CNCF Graduated
HELM — CNCF Graduated

4. OBSERVABILITY & ANALYSIS

- Pick solutions for monitoring, logging and tracing
- Consider CNCF projects Prometheus for monitoring, Fluentd for logging and Jaeger for Tracing
- For tracing, look for an OpenTracing-compatible implementation like Jaeger

Prometheus — CNCF Graduated
fluentd — CNCF Graduated
Jaeger — CNCF Graduated
OPENTRACING — CNCF Incubating

5. SERVICE PROXY, DISCOVERY, & MESH

- CoreDNS is a fast and flexible tool that is useful for service discovery
- Envoy and Linkerd each enable service mesh architectures
- They offer health checking, routing, and load balancing

envoy — CNCF Graduated
CoreDNS — CNCF Graduated
LINKERD — CNCF Incubating

6. NETWORKING, POLICY, & SECURITY

To enable more flexible networking, use a CNI-compliant network project like Calico, Flannel, or Weave Net. Open Policy Agent (OPA) is a general-purpose policy engine with uses ranging from authorization and admission control to data filtering. Falco is an anomaly detection engine for cloud native.

CNI — CNCF Incubating
Open Policy Agent — CNCF Incubating
falco — CNCF Incubating

7. DISTRIBUTED DATABASE & STORAGE

When you need more resiliency and scalability than you can get from a single database, Vitess is a good option for running MySQL at scale through sharding. Rook is a storage orchestrator that integrates a diverse set of storage solutions into Kubernetes. Serving as the "brain" of Kubernetes, etcd provides a reliable way to store data across a cluster of machines. TiKV is a high performant distributed transactional key-value store written in Rust.

Vitess — CNCF Graduated
ROOK — CNCF Incubating
etcd — CNCF Incubating
TiKV — CNCF Incubating

8. STREAMING & MESSAGING

When you need higher performance than JSON-REST, consider using gRPC or NATS. gRPC is a universal RPC framework. NATS is a multi-modal messaging system that includes request/reply, pub/sub and load balanced queues. CloudEvents is a specification for describing event data in common ways.

gRPC — CNCF Incubating
NATS — CNCF Incubating
cloudevents — CNCF Incubating

9. CONTAINER REGISTRY & RUNTIME

Harbor is a registry that stores, signs, and scans content. You can use alternative container runtimes. The most common, both of which are OCI-compliant, are containerd and CRI-O.

containerd — CNCF Graduated
HARBOR — CNCF Incubating
cri-o — CNCF Incubating

10. SOFTWARE DISTRIBUTION

If you need to do secure software distribution, evaluate Notary, an implementation of The Update Framework.

TUF — CNCF Graduated
Notary — CNCF Incubating

Shameless Plug

I work for Red Hat - our Kubernetes distribution is Red Hat OpenShift

I think it's the best (even though I'm biased)

https://www.redhat.com/en/technologies/cloud-computing/openshift

Slides available at: https://people.redhat.com/pladd/

Thank you!

Acknowledgements:
Some figures copied from:
https://platform.sh/blog/2020/the-container-is-a-lie/
https://wikipedia.org/
https://kubernetes.io/