Current Landscape of Container Virtualization Technology and Trends

2019 May 10th

Byungchul Tak

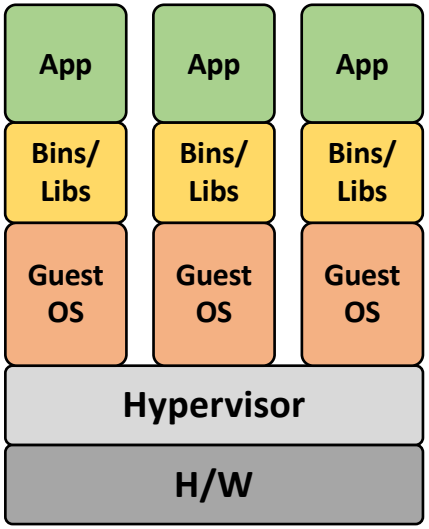
School of Computer Science and Engineering

Kyungpook National University

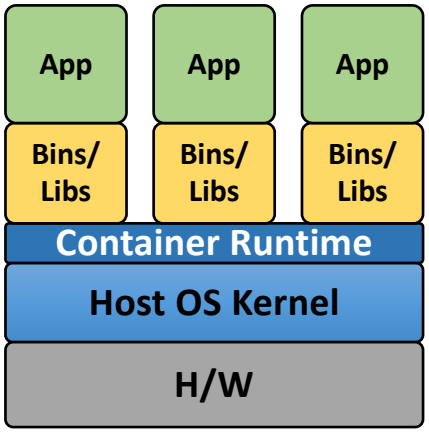
Table of Contents

Container

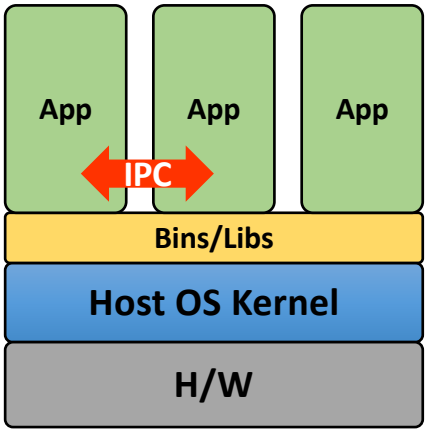
- Definition
  - Most similar to: **Process**
  - But differs in that it has: **Isolation property**



(Type-1) Virtualization



Container



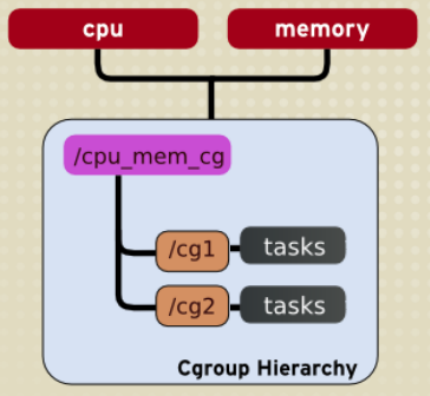
Bare-metal

Container Building Blocks

❑ Cgroups (Control Groups)

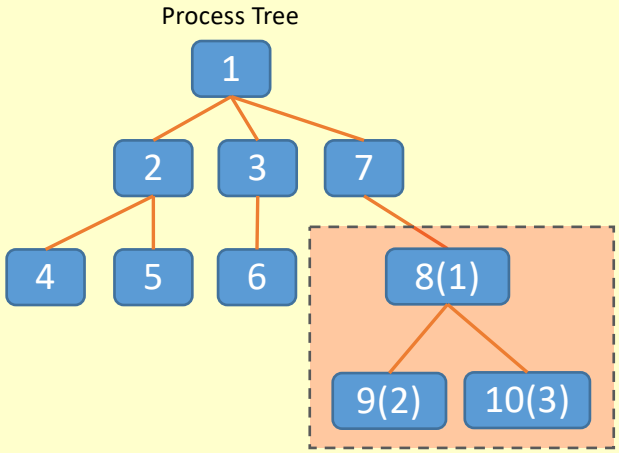
- Kernel mechanisms for **resource allocation (limiting)** and **metering**

- Processes are divided into *hierarchical* groups (subsystems)
  - can migrate between them
- Total 12 controllers
  - blkio, cpu, memory, netcls, netprio, devices, pids …
  - ex) pid controller: limit # of procs that can be forked in the group → counter the fork-bomb



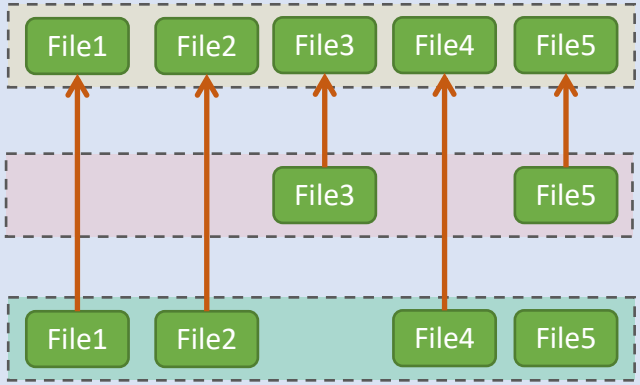
❑ Namespaces

- Custom view of the ID space
- Limit what a process see and access
- 7 namespaces: mnt, pid, net, uts, ipc, user, cgroup



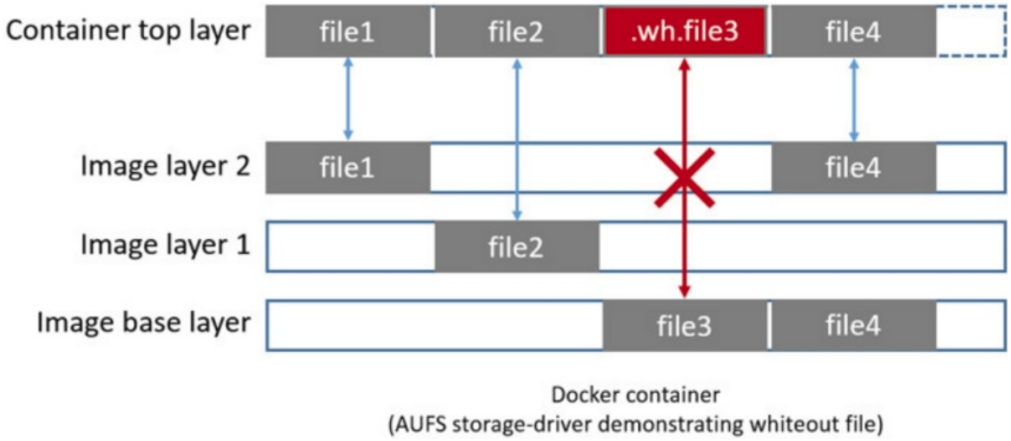
❑ Union mount

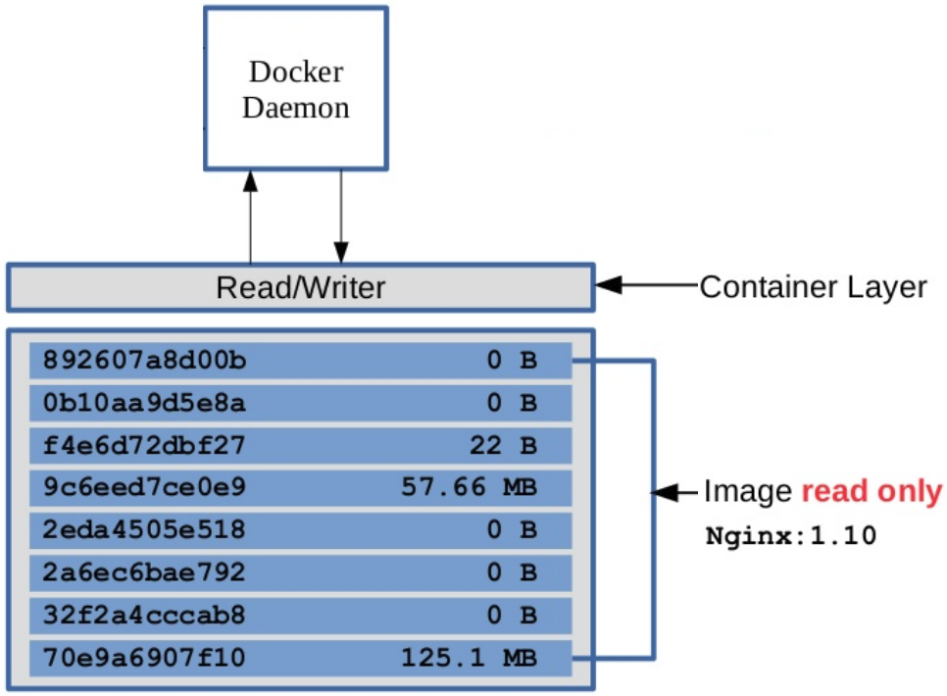
- Combining multiple directories into one combined view
- copy-on-write policy
- Overlaying of file system
  - aufs, overlayfs



Union Mount in Docker using Aufs

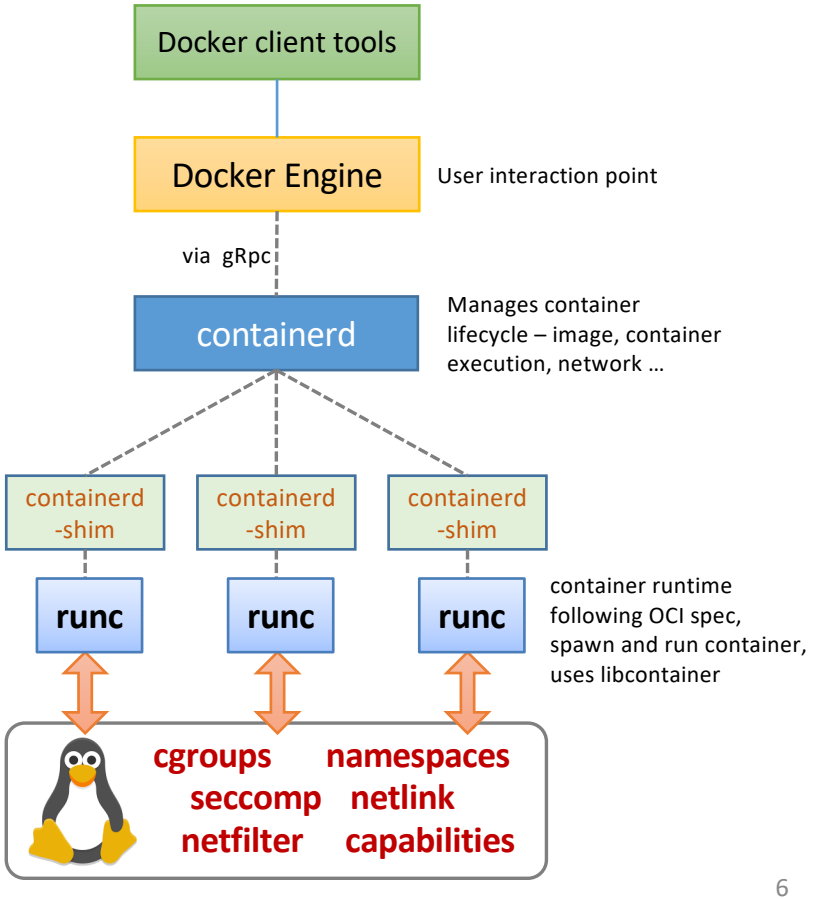
- AUFS: Another Union File System





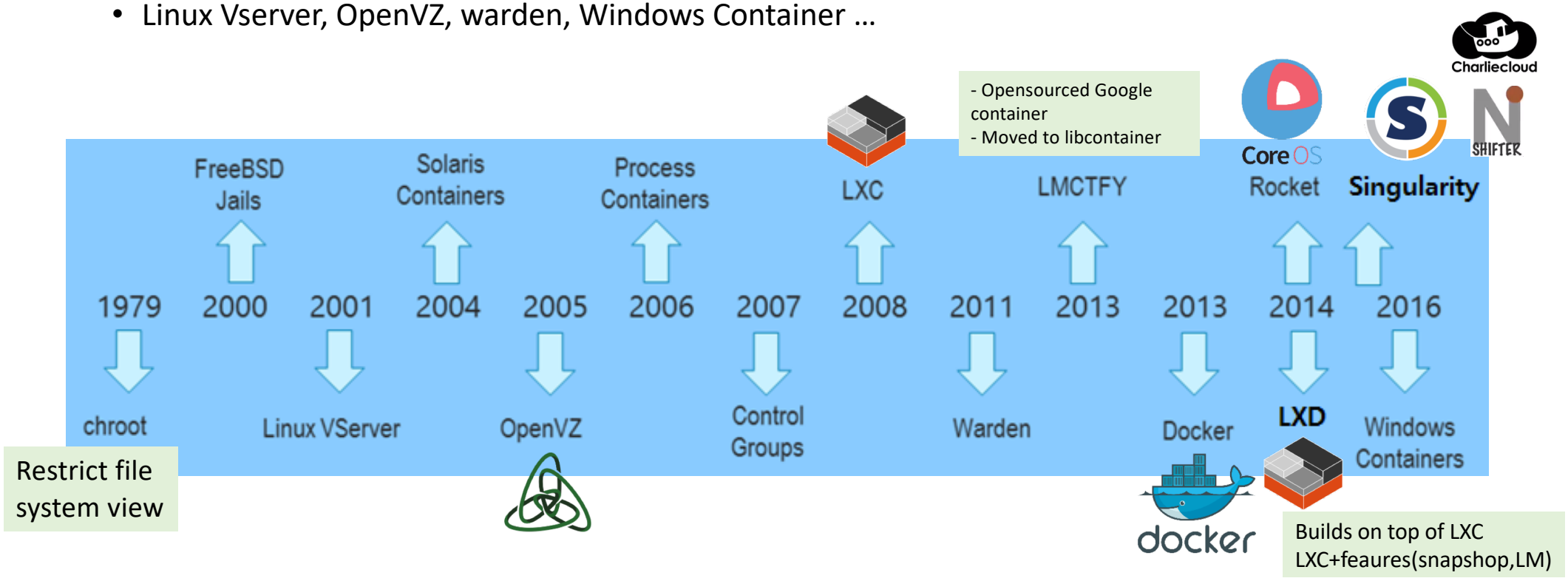
Container Platforms

- Docker
  - Most popular container platform
  - Started as open-source project that automates deployment of applications inside containers
  - Provides wrapper around a software package
    → ***Build, Ship and Run Any App Anywhere***
  - Easy creation, update and distribution of container images
  - Public DockerHub
  - Previously based on LXC, now uses libcontainer
    - libcontainer: library for container execution driver, interface component to use Linux features



Container Platforms

- Other platforms
  - rkt (Rocket): developed by CoreOS, no daemon
  - LXC/LXD: system container rather than application container
  - Linux Vserver, OpenVZ, warden, Windows Container …



Container Orchestration

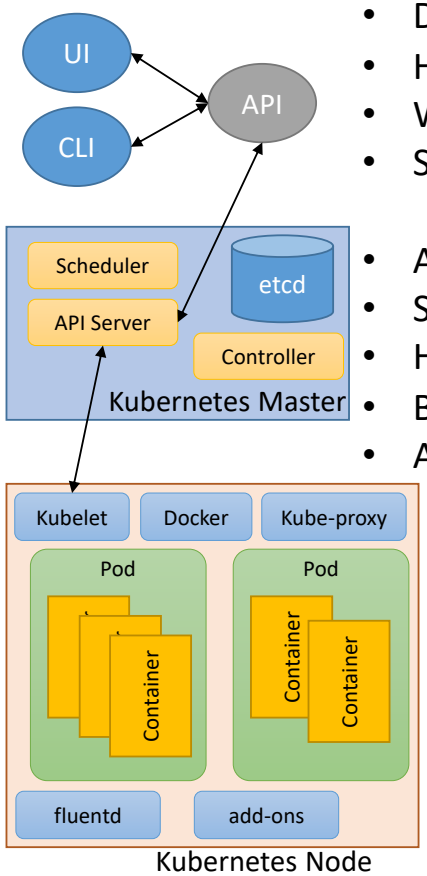
Container Orchestration

- Container is a solutions for:
  - Enterprise workloads
  - Micro-service architecture
  - DevOps, CI/CD (Continuous Integration/Delivery)
  - ***Scalability***

- Issues with Scaling your Application
  - Management Burden increases
    - Communication among them
    - Need to be placed appropriately
      - Container Scheduling
    - Automatic scaling based on workloads
    - Load-balancing
    - Handle failed containers

Container Orchestration Platforms

kubernetes

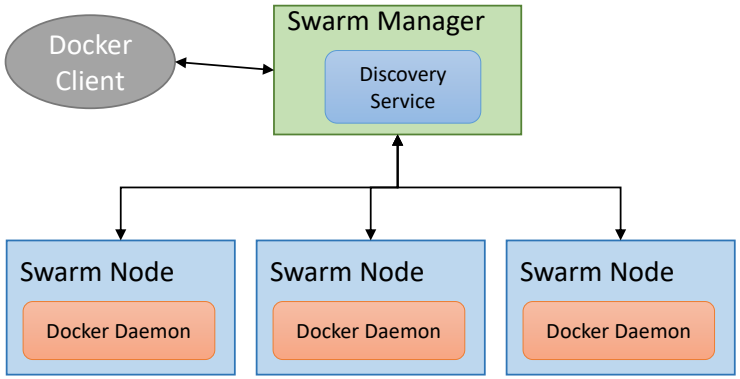


- Developed by Google
- Huge community
- Written in Go
- Solid API

- Automatic binpacking
- Self-healing
- Horizontal Scaling
- Batch execution
- Auto rollback/rollout

docker SWARM

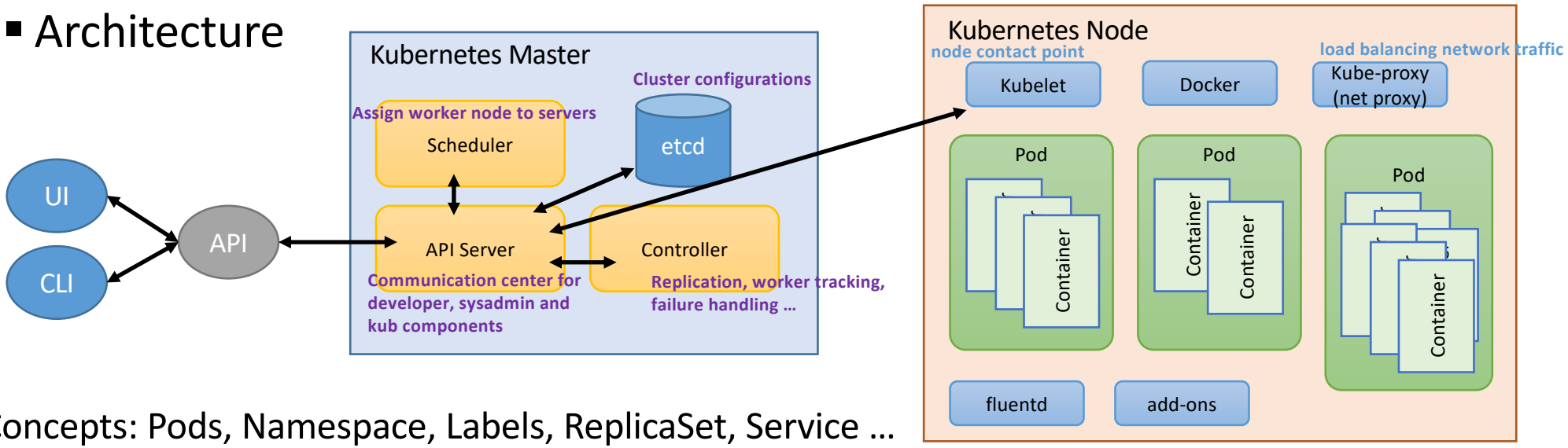
- Integrated runtime and orchestration
- No concept of Pods
- Faster scaling and reaction time than Kubernetes
- Overall, simpler than Kubernetes



MESOS
MESOSPHERE
DC/OS
Amazon ECS
OPENSHIFT by Red Hat
HashiCorp Nomad
KONTENA
RANCHER

Kubernetes Architecture

- Architecture

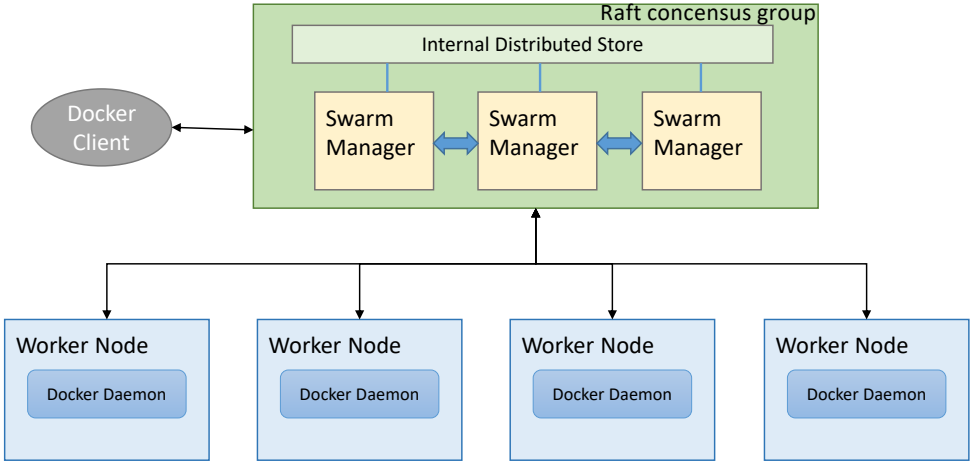


- Concepts: Pods, Namespace, Labels, ReplicaSet, Service …
- Scheduling
  - Filtering: filter out nodes that do not meet the requirement
    - NoDiskConflict, PodFitResoruce, PodFitHostPort …
  - Ranking
    - finalScoreNodeA = (weight1 * priorityFunc1) + (weight2 * priorityFunc2)
    - LeastRequestedPriority, CalculateNodeLabelPriority, BalancedResourceAllocation, CalculateSpreadPriority, CalculateAntiAffinityPriority

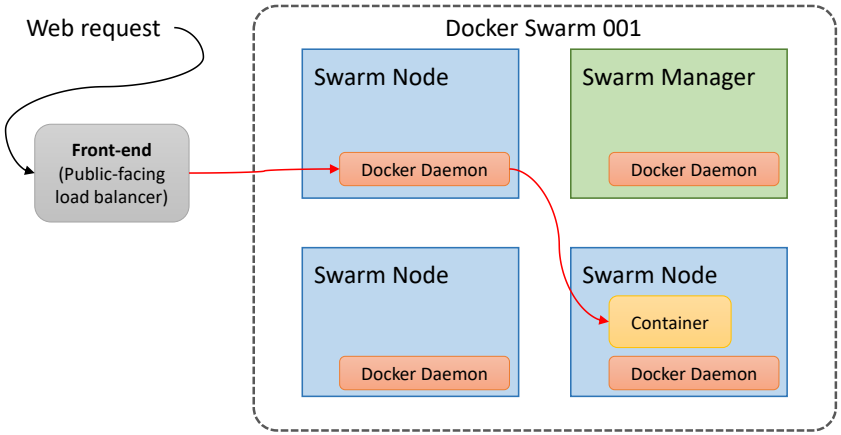
- Pods
  - Basic unit of scheduling and deployment
  - Group of containers + config + shared storage
  - Moves in group
  - Own IP address
  - Stateless

Docker Swarm

- Key Concepts
  - Ability to deploy containers across docker hosts
    - Using overlay network for service discovery
    - Built-in load-balancer
  - Features: Cluster management, scheduling, HA, decentralized, scaling, service discovery, load balancing, rolling updates …
- Architecture



- Request Redirection

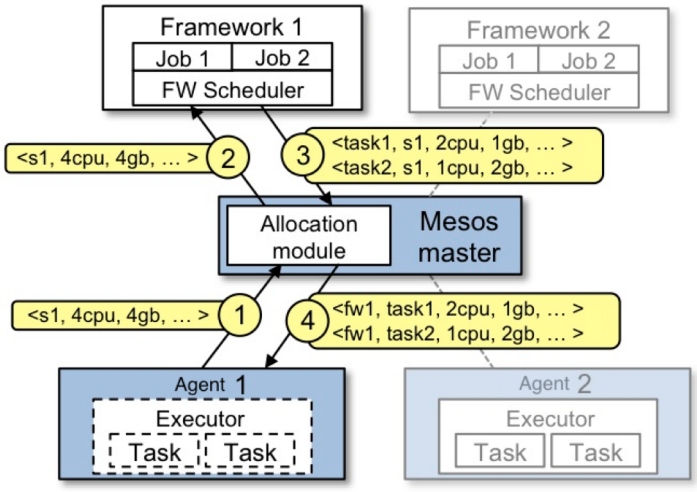


- Service consists of:
  - Image
  - External port
  - Overlay network
  - CPU/mem limits
  - Update policy
  - # of replicas
- Scheduling
  - spread
  - binpack
  - random

Mesos

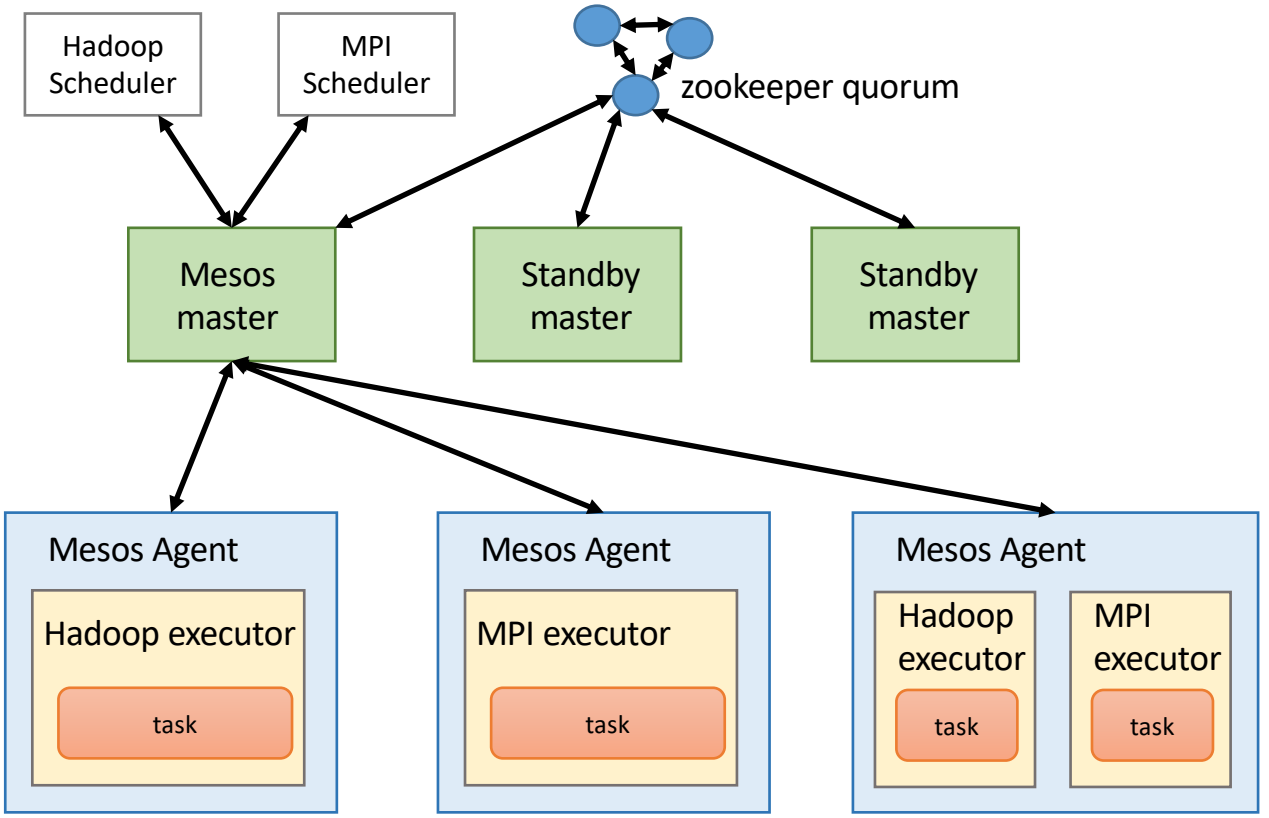
- Key concepts
  - Cluster resource manager
    - Scheduling of VM/containers …
  - Distributed OS
  - Provides single resource image

- Two-level Resource offer mechanism



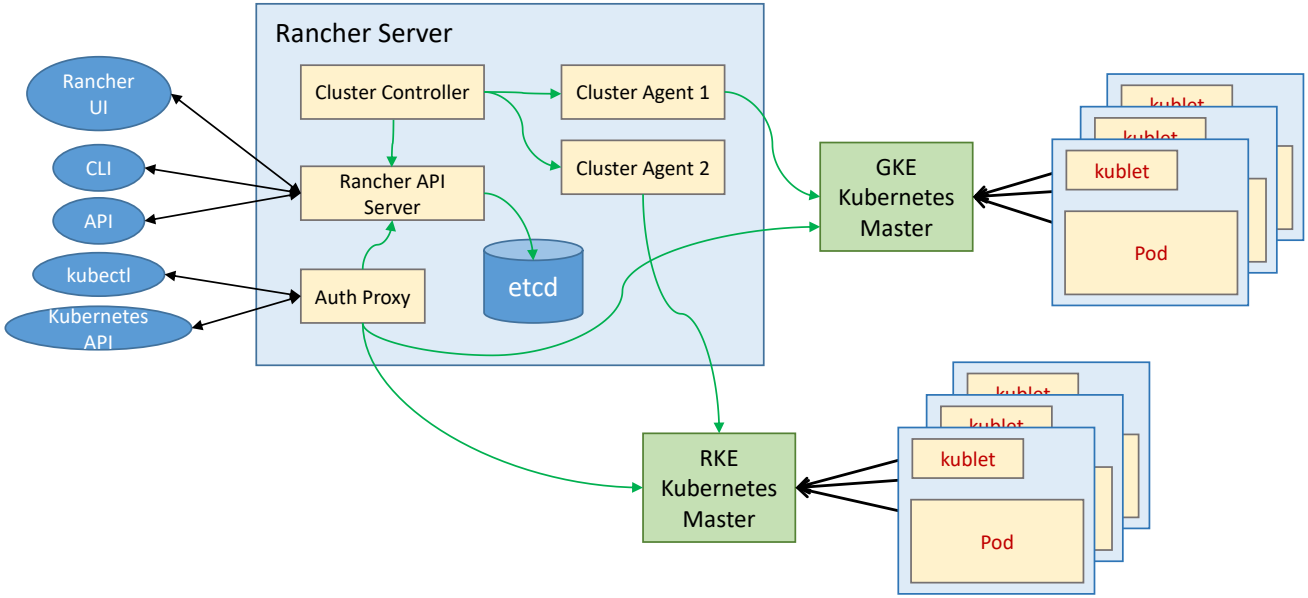
http://mesos.apache.org/documentation/latest/architecture/

- Architecture

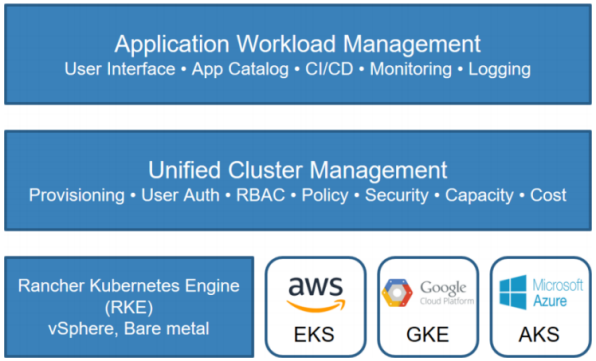


Rancher

- Architecture



- Software Stack



Comparison of Container Orchestration Tools

	Kubernetes	Docker Swarm	Mesos
Auto-scaling	**User specify the # of Pods, CPU utilization per Pods**	**No autoscaling. User manually specifies the # of instance and update the config**	**Need to use Marathon framework (marathon-autoscale.py)**
Load balancing	Pods exposed as Service. Ingress is used.	Auto-forwarding between nodes	Need to use add-on 'Marathon' framework
Service Discovery	Yes, Embedded DNS server	Yes, Embedded DNS server	Yes, Embedded DNS server
Self-healing	Yes, Pod state is defined and liveness, readiness check is performed to find any failures	Yes	Marathon framework needed for self-healing functionality
High Availability	Yes, Master replicated	Yes, multiple manager node supported	Yes, multiple master with zookeeper coordination
Secret (Pwd/token/key) management	Yes, Secret objects created at apiserver	Yes, Docker Secrets manager provided	No
Scheduler	Two step algorithm of Filtering, Ranking	Spread, Binpack, Random Algorithm Support	Two level algorithm: master offers resource amount and framework accepts it
Licence	Apache License 2.0	Apache License 2.0	Apache License 2.0
Networking	Flat network model	Support bridge, overlay, macvlan or 3rd party plugin driver	less focus on networking, 3rd party plugin driver
Rolling Update	Yes	Yes	No
Application Definition	Pod, Deployment, Service definitions in Yaml format	Docker compose	Json format definition managed by Marathon
Health Check	Liveness check, Readiness check for application pods	Dockerfile can specify HEALTHCHECK directive	Native health check and plugin (HTTP) check by Marathon or Aurora

Container for HPC

Container for HPC

- Pain points of HPC that lasted decades
  - Dependency hell
  - Reproducibility → distribute and validate
  - Mobility
- Mismatch of Docker (or Containers) use case
  - Containers target enterprise workload
    - micro-services and massive/fast scale-out
    - fast continuous deployment cycle
  - HPC does not need massive scale-out

Docker provides:

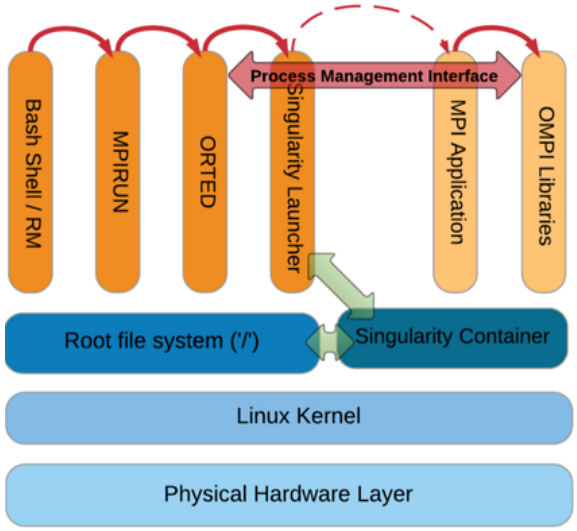
- ✓ Solution to dependency hell
- ✓ Reproducibility
- ✓ Mobility (partially)

But, Docker introduces new problems:

- Dangerous to install to HPC center
  ex) kernel version upgrade maybe needed
- Security issue
  - docker deamon runs in root
  - container root can have root access of host
- Performance issue (use of spec. HW)
  - Lack of support for HPC S/W stacks
    - MPI, Slurm, torque, GPU libs …
- Integration to WLM

Singularity

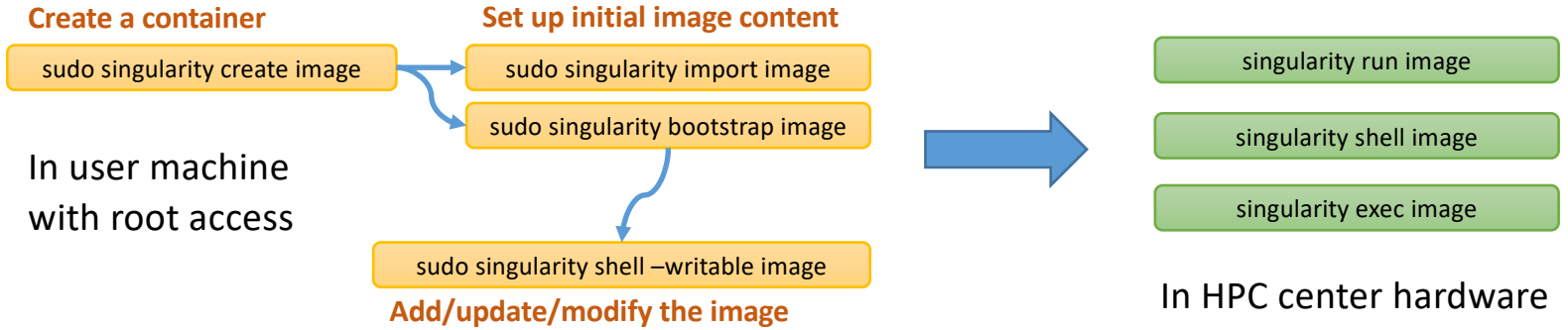
- Design goal
  - Mobility of compute, BYOE, UDSS
  - Single file has everything
    - Docker uses layers
  - Limit user privileges
    - Must be root outside to be root inside
  - No root-owned daemon (like Docker)
  - Integration with HPC S/W stacks and infrastructure
    - Resource manager(Slurm), GPU lib, MPI, IB …
  - Docker Hub compatible
    - Can pull images from docker hub to build an image



MPI Support of Singularity

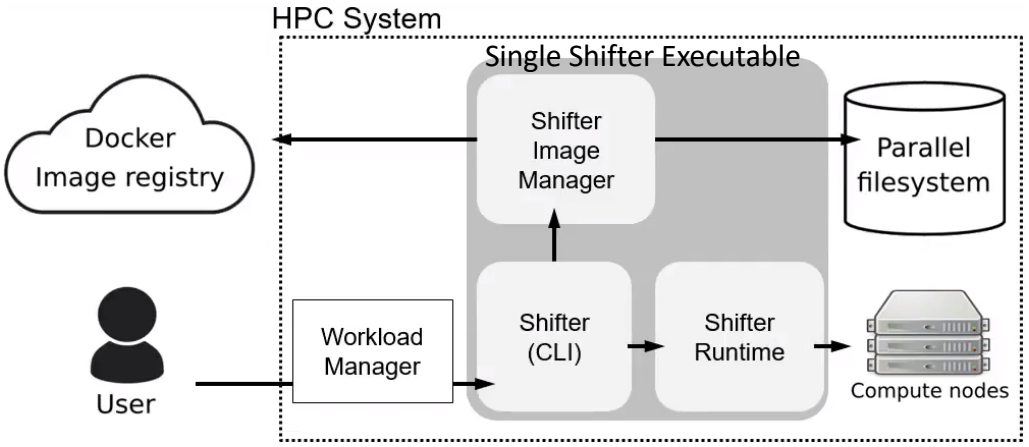
- *mpirun* invoked
- orted created
- singularity 'exec'ed
- mpi program within the container run
- mpi tries to contact orted via PMIx
- singularity connects this PMIx link to orted

Singularity Workflow



Shifter

- What is Shifter?
  - Solution for running Docker container in HPC environment
- Characteristics
  - Provides Docker-compatible container runtime
  - Native GPU support
    - Automatic import of CUDA driver and dev
  - Native MPI support
    - Swap container MPI with host MPI lib at run time
      - For utilizing vendor-specific features (IB)
  - Single executable
  - Image manager component (image conversion)
  - Docker-like CLI
  - Flattened file system for performance



- ❑ Image manager
  - Written in C++
  - Can import tar files
  - Parallel and robust download
- ❑ Shifter CLI
  - Similar interface to Docker
  - Support for 3rd-party registry

Workflow

Create Docker Image → Push to Docker Hub → Pull the image to HPC center → Run at scale in HPC system

CharlieCloud

- Design goal
  - Simplicity
    - Principle of least privilege
    - Make it do one thing well
- Characteristics
  - All processes are unprivileged
  - cgroups not used
  - PID namespace not used
  - UTS namespace not used
  - MNT namespace is used
- Real problems with using Docker
  - Root-owned daemon of docker is not a real issue
  - Performance is the problem – overlayfs
  - Associativity – docker cli and container association to resource manager
    - WLM integration issue

- Workflow
  - Preparing an image (need privilege here)
    - Pull from Docker, or
    - Use *ch-build* command

```
ch-build –t hello ~/container_src
ch-docker2tar hello /var/tmp
```

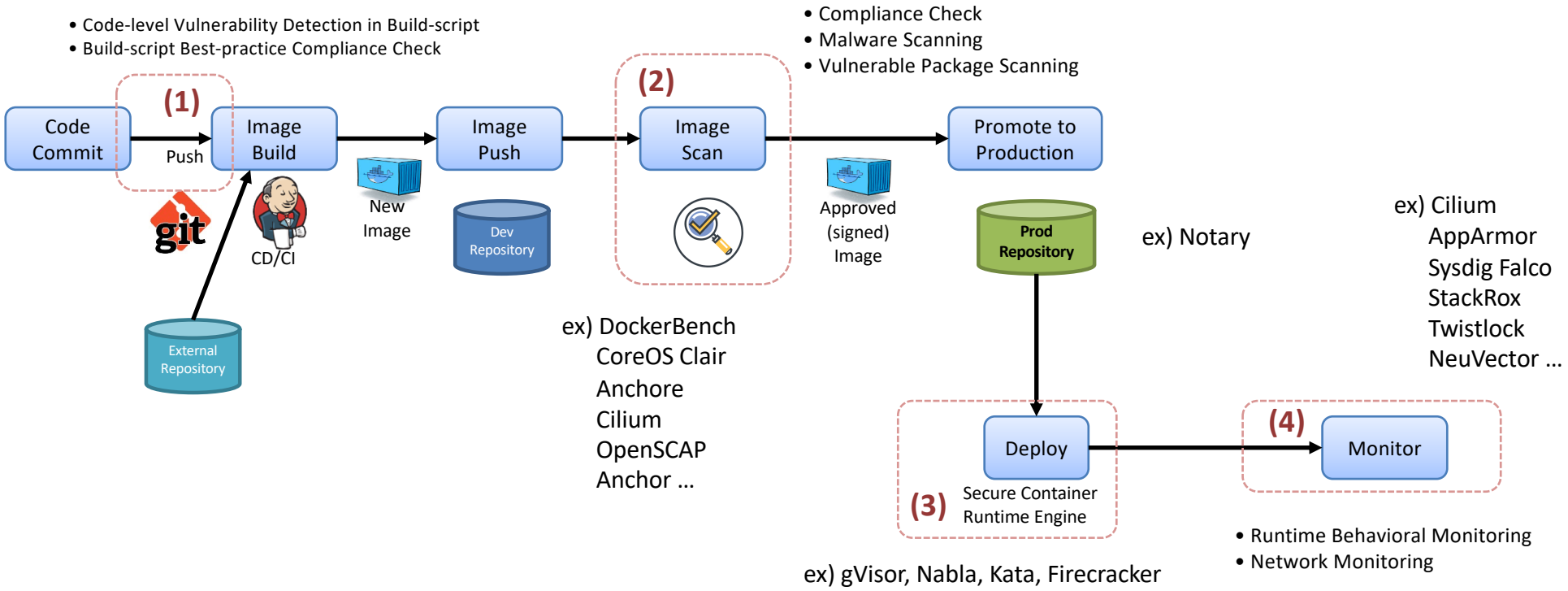
  - Running a container (unprivileged)

```
ch-tar2dir /var/tmp/hello.tar.gz /var/tmp/hello
ch-run /var/tmp/hello – cat /etc/debian_version
```

    - ch-run performs:
      - Set up namespace
      - bind-mount host directories
      - change container root directory via *pivot_root*
      - perform *execvp*

Container Security

Container Security Domains



Domain of Container Security

- Topics
  - Container image scan
    - Vulnerabilities using CVE data
    - Compliance conformance, best practice rules
  - Image signing
  - Docker engine, Docker daemon security, Host security
  - Network security
  - Runtime protection
  - Multi-functional tools
    - Compliance, Image scan, Vulnerabilities, Runtime protection
    - CI/CD integration
    - Machine learning for behavioral patterns

Container Security Tools

Name	Functions	Lincense	Notes
Anchor	Image scan (Vulnerabilities), Compliance	Proprietary	
AppArmor	Runtime protection	Opensource	Integrated to docker
AquaSec	Image scan, Runtime protection	Proprietary	
Black Duck Docker Security	Image scan	Proprietary	
Cilium	Network security	Opensource	Uses BPF. Good community.
Cavirin	Image scan, Runtime protection, Compliance	Proprietary	
CoreOS Clair	Image scan	Opensource	Static analysis
Docker-bench for Security	Compliance	Opensource	Based on CIS benchmark
Dockscan	Compliance	Opensource	Simple ruby scripts for docker and running containers
Sysdig Falco	Runtime alert, behavioral monitoring	Opensource	Auditing tools, monitor container without instrumentation,
NeuVector	Compliance, Runtime protection	Proprietary	Enforcer container with full access to docker daemon
Notary	Trusted image repository	Opensource	Docker image signing framework, By Docker, Owned by CNCF.
OpenSCAP	Compliance	Opensource	oscap-docker for image and running containers
Seccomp	System call filtering rules	Opensource	Integrated to docker
StackRox	ML Runtime protection	Proprietary	
Sysdig	debugging, forensics	Both	Syscall recording
Tenable Flawcheck	Image scan	Proprietary	
Twistlock	Image scan, Runtime protection, Compliance	Proprietary	Vulnerability explorer
Drydock	Security audit tool	Opensource	In Python. Inspired by Docker-bench-security.
Actuary	Compliance	Opensource	Inspired by Docker-bench-security.
Dagda	Image scan for vulnerabilities and running containers	Opensource	Static analysis, antivirus scan using ClamAV, in Python
Grafaes	Metadata API for enforcing policies	Opensource	

Docker Security using Kernel Features

- Seccomp
  - Syscall filtering mechanisms of Linux kernel
  - List of syscalls and actions

```
{
   "defaultAction":"SCMP_ACT_KILL",
   "syscalls":[
      {
         "name":"chmod",
         "action":"SCMP_ACT_ERRNO"
      }
   ]
}
```

Set errno without executing the system call

  - docker integration

docker run **–security-opt seccomp=./default.json** alpine sh

  - Problem
    - Which syscall to filter?

- Linux Capabilities
  - Layer above seccomp
  - More fine-grained control of permissions
    - slicing of root power
  - Integrated to docker
    - default set of capabilities in docker container

chown, dac_override, fowner, fsetid, kill, setgid, setuid, setpcap, net_bind_service, net_raw, sys_chroot, mknod, audit_write, setfcap

docker run –it **–cap-drop=DAC_OVERRIDE** alpine sh

→ disallow root of container to see certain files owned by others

–cap-drop=NET_RAW

→ disallow a container to spy on network packets

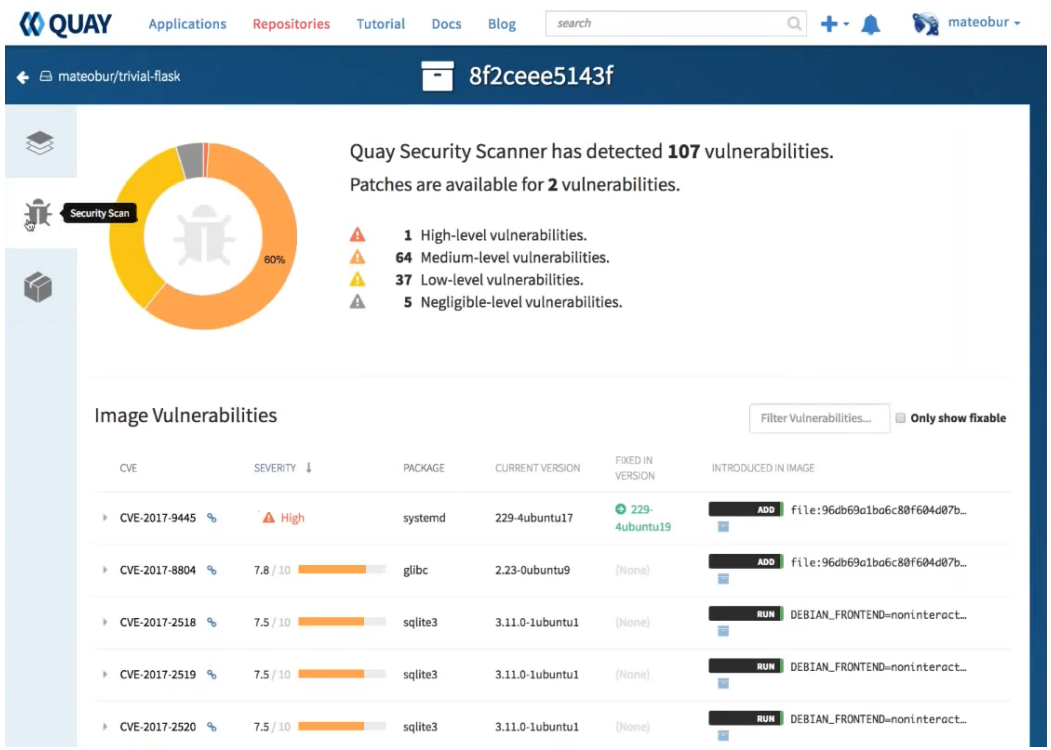
- AppArmor
  - Comprehensive security feature of Linux kernel
  - Per-program profile
  - Fine-grained access to files

```
network inet tcp
network inet udp
network inet icmp

deny network raw
deny network packet

file,
umount,

deny /bin/** wl,
deny /etc/** wl,
deny /home/** wl,
…
capabilities chown,
capabilities dac_override,
capabilities setuid,
capabilities net_bind_service,

deny @{PROC}/* w,
deny @{PROC}/{[^1-9], [^1-9][^0-9], [^1-9][^0-9][^0-9], [^1-9][^0-9][^0-9][^0-9]*}/** w,
deny /sys/[^f]*/** wklx,
```

docker run --rm -it **--security-opt apparmor=docker-default** hello-world

Other Security Tools

- CoreOS Quay
  - Image scanning and analysis
  - Log in, do 'docker push' to quay registry.
  - Check the results on the web page
    - Vulnerabilities, CVE info



CVE-2017-9445

Priority
High

Description
In systemd through 233, certain sizes passed to dns_packet_new in systemd-resolved can cause it to allocate a buffer that's too small. A malicious DNS server can exploit this via a response with a specially crafted TCP payload to trick systemd-resolved into allocating a buffer that's too small, and subsequently write arbitrary data beyond the end of it.

References
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-9445
http://www.ubuntu.com/usn/usn-3341-1

Bugs
https://launchpad.net/bugs/1695546

Notes
```
chrisccoulson> I believe this was introduced in v223 by
 https://github.com/systemd/systemd/commit/a0166609f782da91710dea9183d1bf138538db37
chrisccoulson> systemd-resolved is not used by default in Xenial. It is
 spawned if a user execs the systemd-resolve utility, but that shouldn't
 impact the system.
```

Assigned-to
chrisccoulson

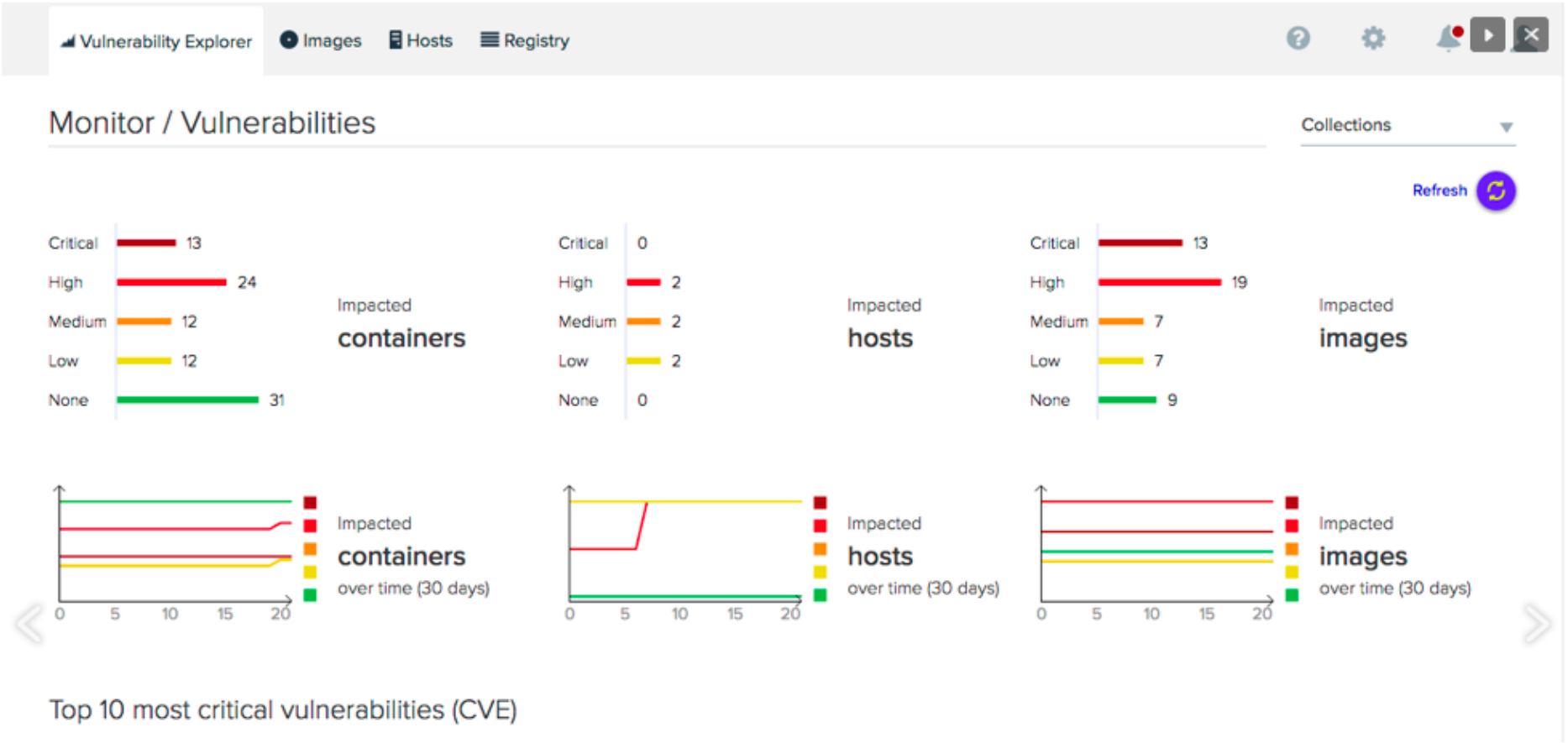
Package
Source: systemd (LP Ubuntu Debian)

Upstream:	needed
Ubuntu 17.10 (Artful Aardvark):	released (233-8ubuntu2)
Ubuntu 12.04 ESM (Precise Pangolin):	DNE
Ubuntu 14.04 LTS (Trusty Tahr):	not-affected (204-5ubuntu20.24)
Ubuntu Core 15.04:	not-affected (219-7ubuntu6)
Ubuntu 16.04 LTS (Xenial Xerus):	released (229-4ubuntu19)
Ubuntu 17.04 (Zesty Zapus):	released (232-21ubuntu5)

More Information
Mitre
NVD
Launchpad
Debian
Updated: 2017-08-11 23:56:02 UTC (commit 13081)

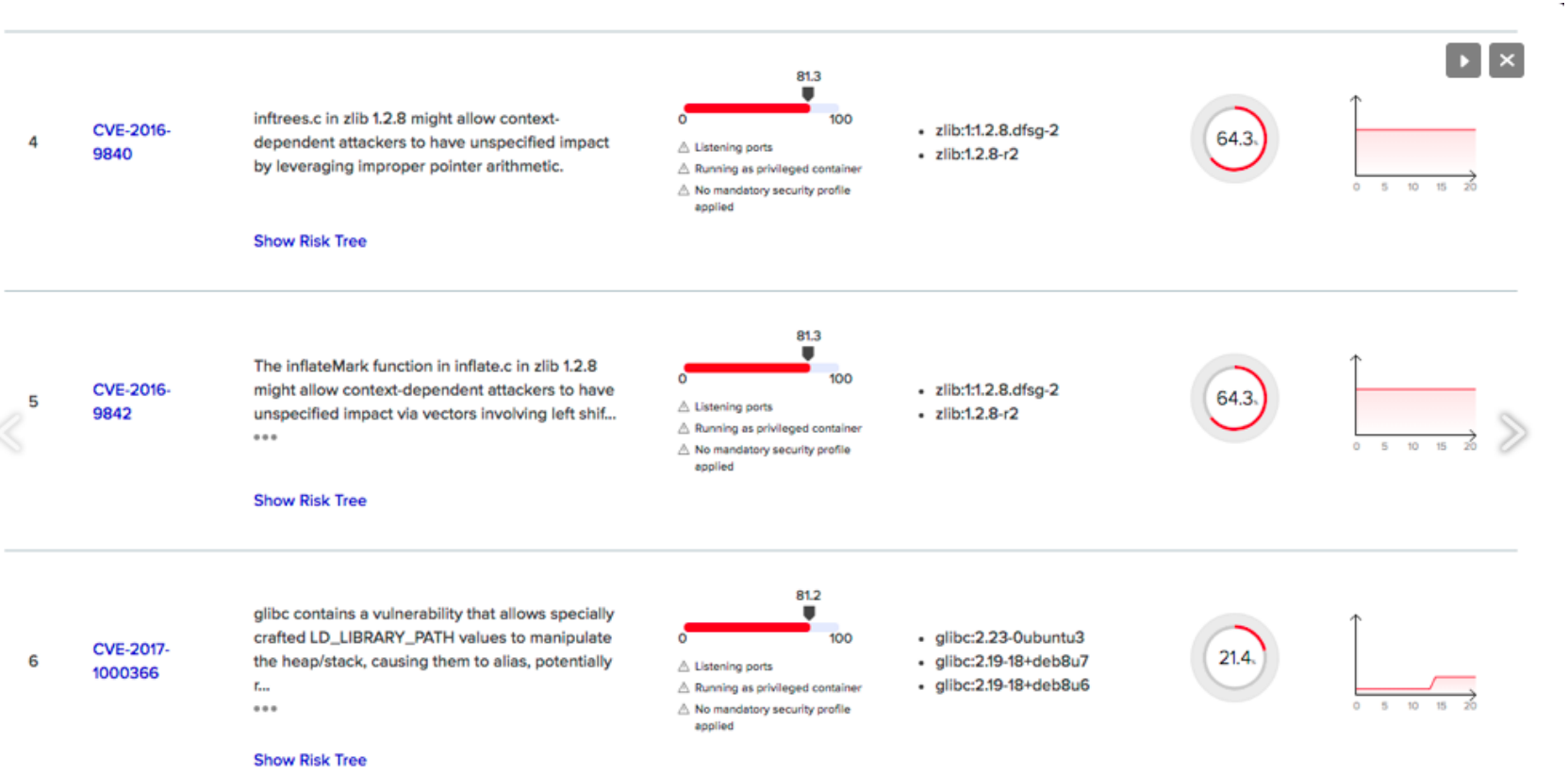
Other Security Tools

- Twistlock



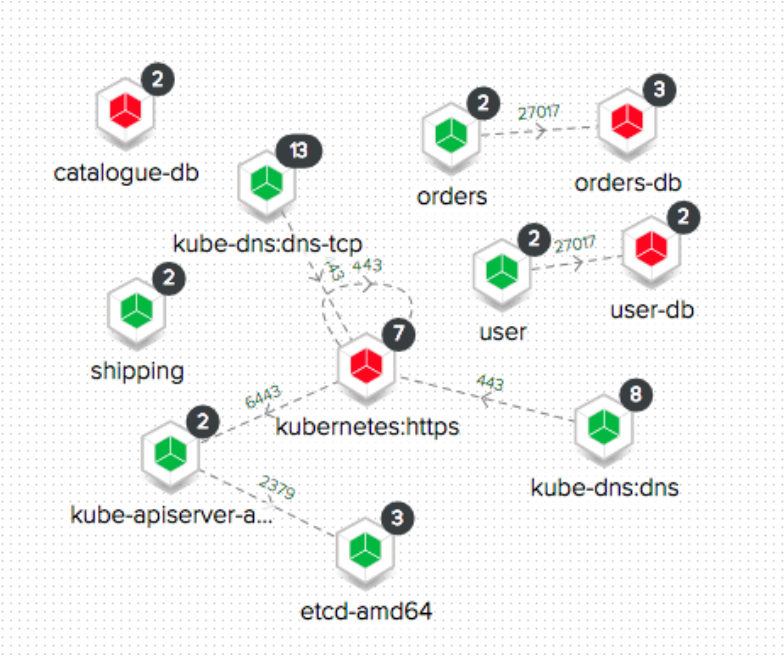
Other Security Tools

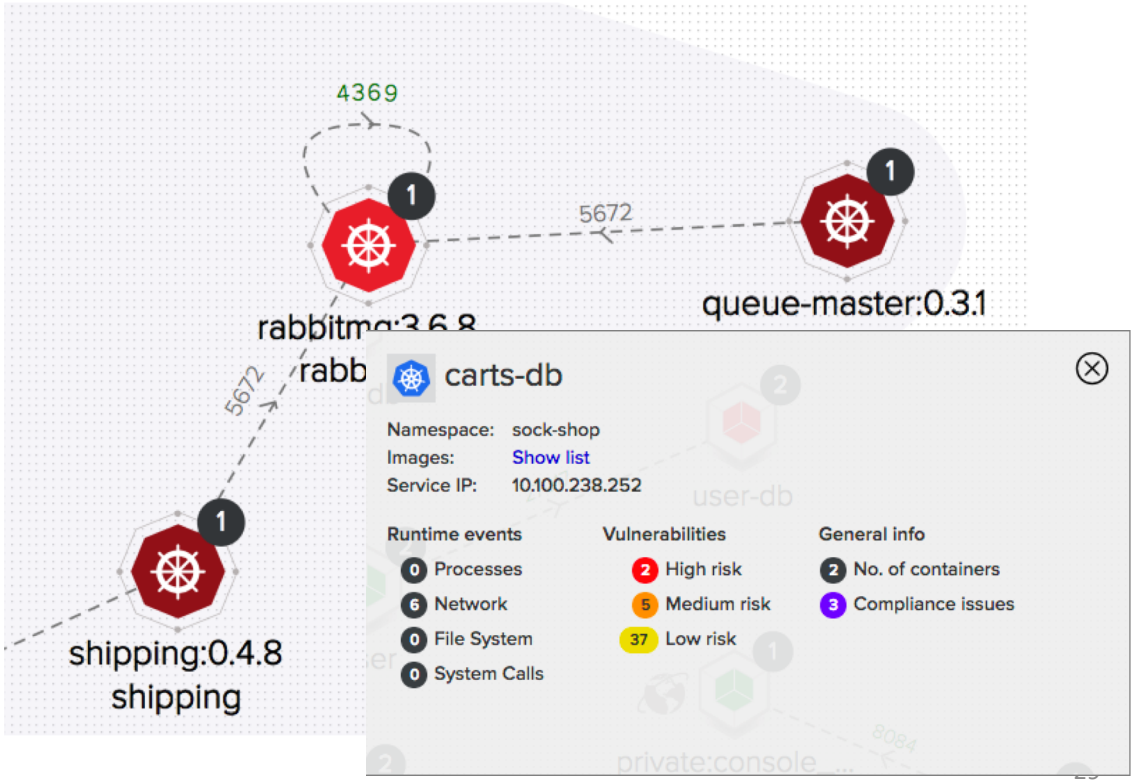
- Twistlock



Other Security Tools

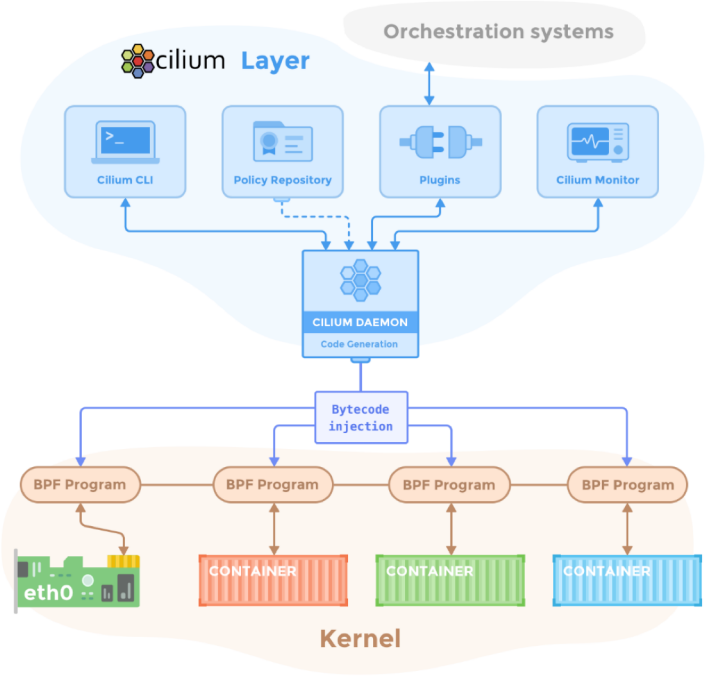
- Twistlock – Runtime radar
  - Container-centric environment visualization (w.r.t pods and services)
  - Can specify network rules
  - Can specify system call rules



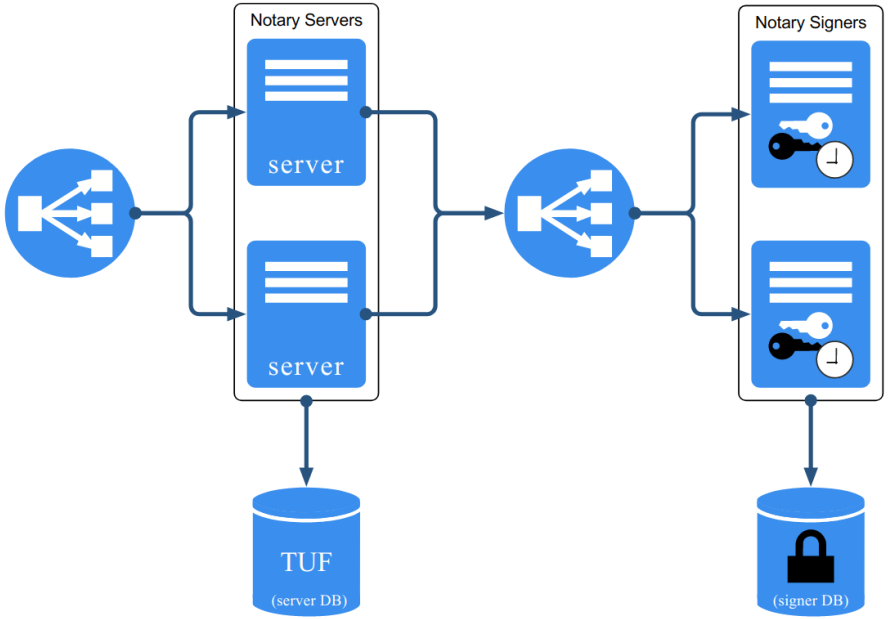


Other Security Tools

- Cillium
  - iptables doesn't work for containers
  - allows apps to talk to certain apps

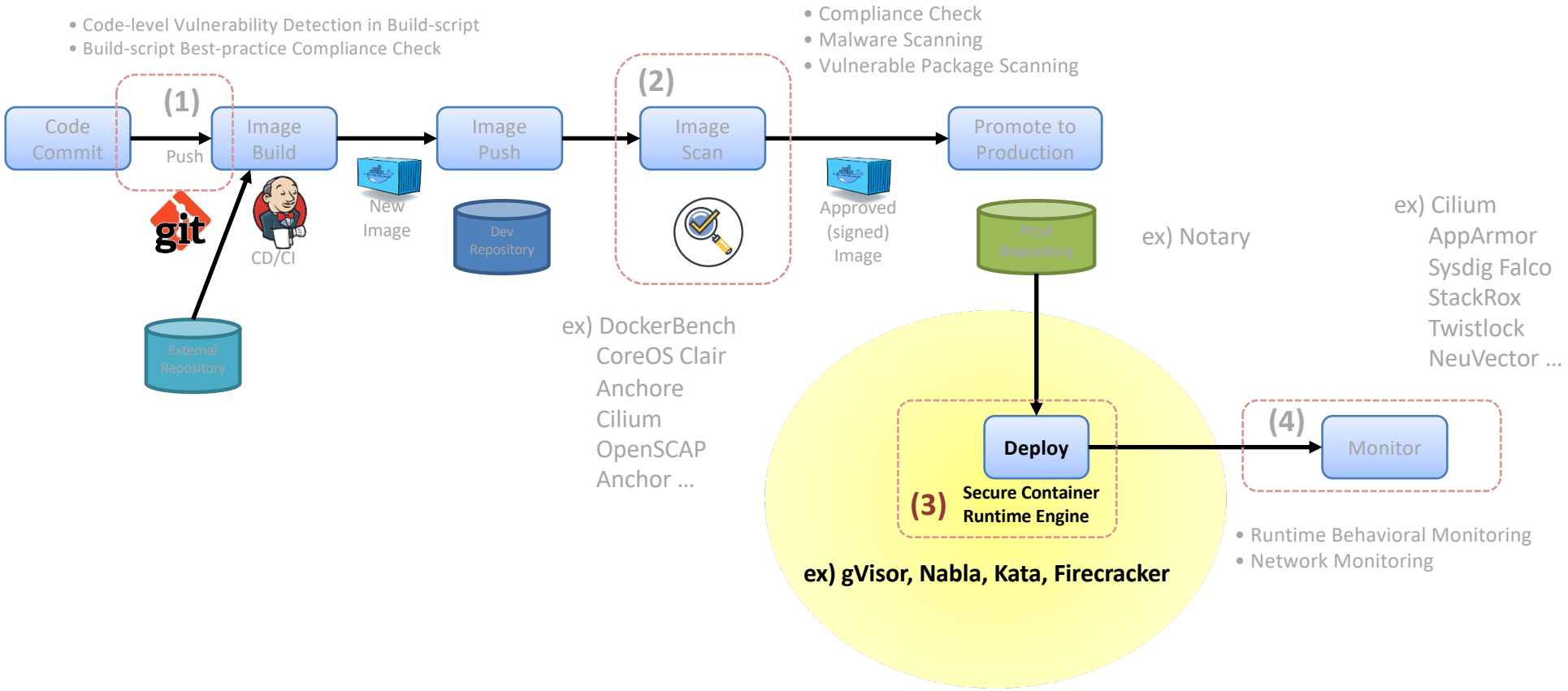


- Notary
  - De facto Docker image signing framework
  - Digitally sign image collections
  - Consumers verify the origin and content integrity



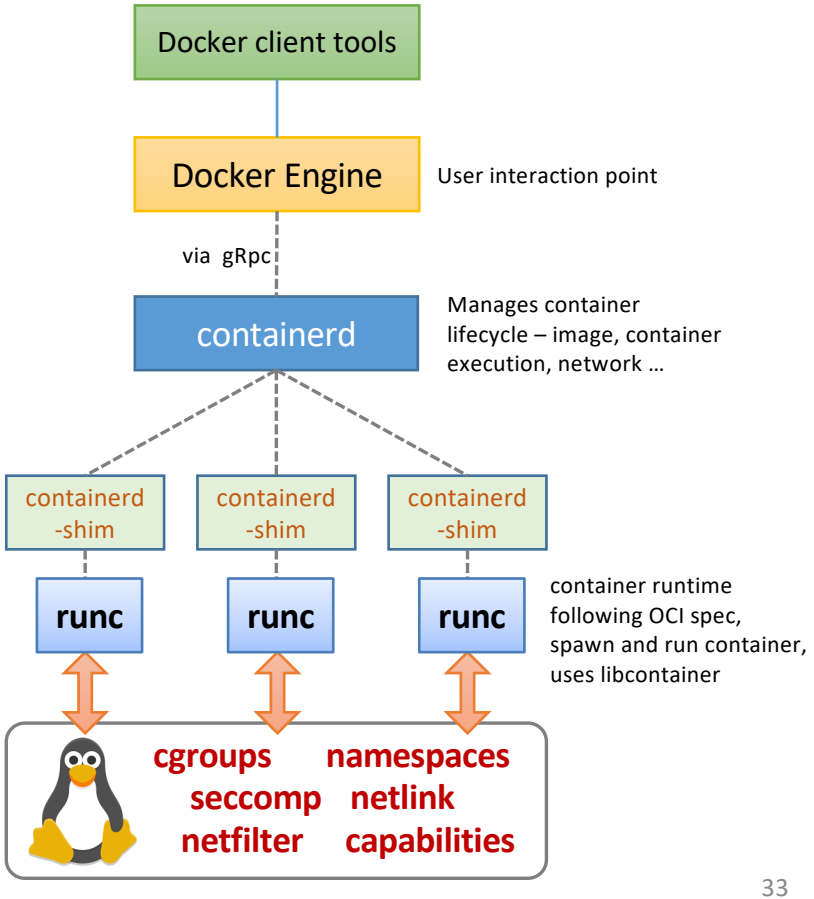
Container Runtime Security

Container Security Domains



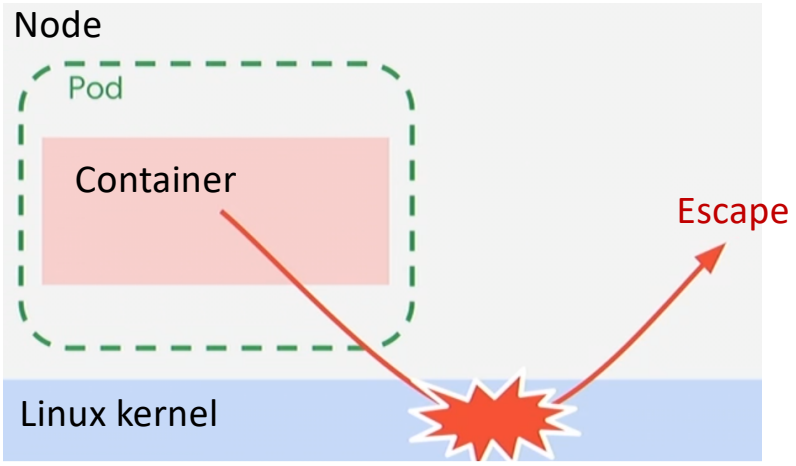
Container Runtime

- Container Runtime
  - Module that set up namespaces and cgroups (using libcontainer)
  - Transient – Once container is up, it disappears
  - OCI-compliance
- Runtimes
  - runc: default docker container runtime
  - runsc: gVisor runtime
  - runnc: Nabla container runtime
  - kata-runtime: Kata container runtime
  - rkt



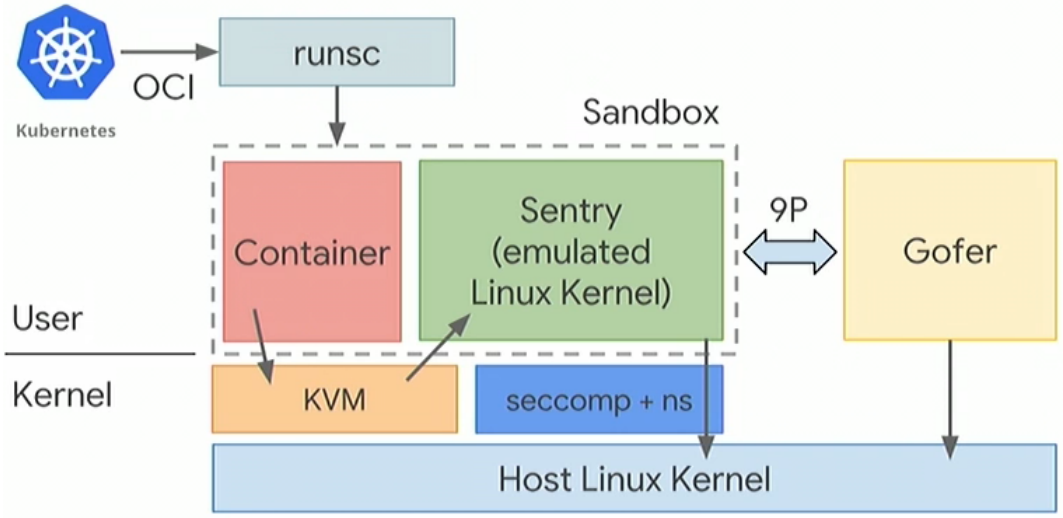
Attack Model

- Attack model for Secure Container Runtime
  - Scenario
    - Service model that needs to run unsafe code uploaded from outside
  - Container Escape
    - Exploit bugs in Linux Kernel via syscalls
    - Obtain elevated privilege



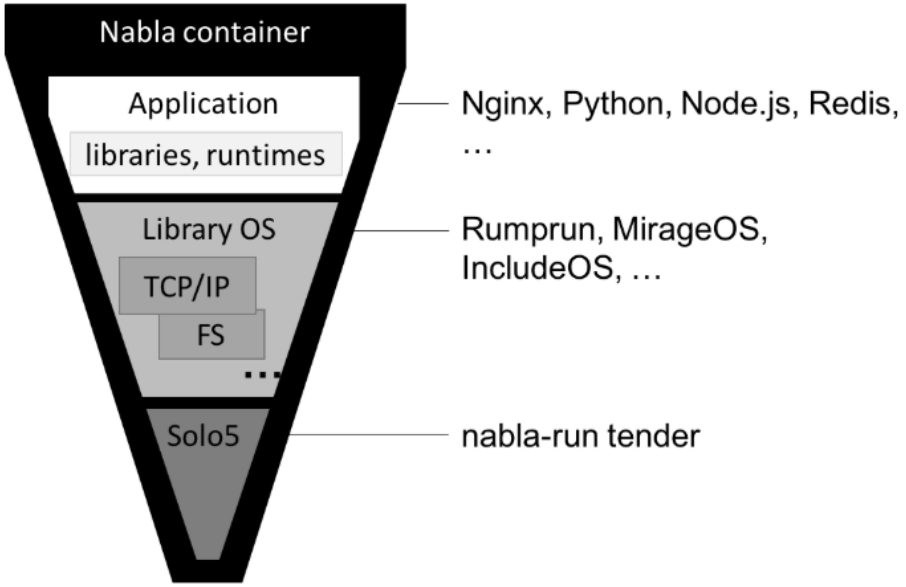
gVisor

- gVisor
  - Syscall interception via ptrace or KVM
  - Sentry: micro kernel
    - 200+ syscalls implemented
    - 10~30 actual syscalls to Host kernel
  - Gofer handles File I/O and network I/O



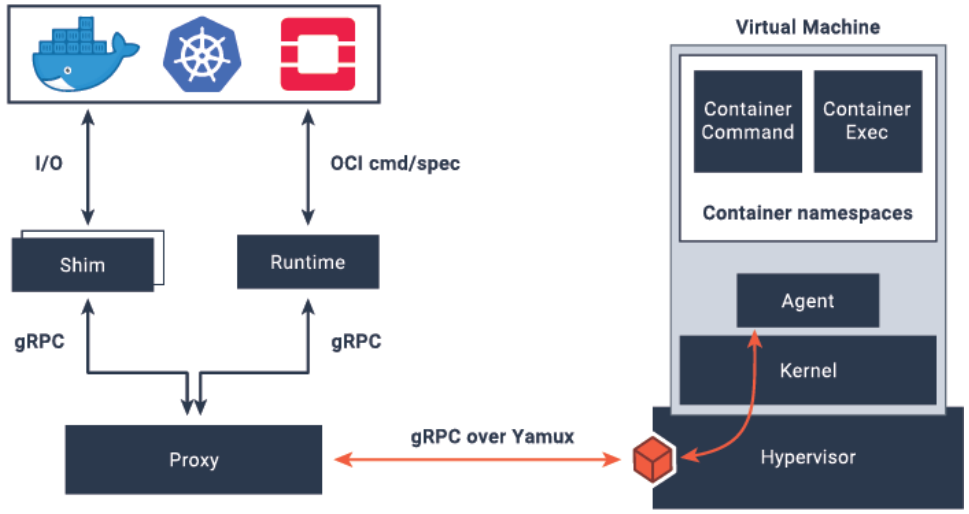
Nabla Container

- Nabla container
  - Nabla containers use library OS (aka unikernel) techniques, specifically those from the Solo5 Project
    - To avoid system calls and thereby reduce the attack surface
  - Nabla containers use 7 system calls
    - ‘read’, ’write’, ‘exit_group’, ‘clock_gettime’, ‘ppoll’, ‘pwrite64’, ‘pread64’
    - All others are blocked via a Linux seccomp policy
  - Library OS (unikernel)
    - Specialized, single-address-space machine images constructed by using library operating systems.
    - Minimal set of libraries required for their application to run.
  - Rumprun
    - Default unikernel in Nabla container



Kata Container

- Kata container
  - VM-based
  - Replace runc with Kata-runtime
  - Heavy memory optimization
    - No guest page cache
    - Shared Rootfs
    - memory deduplication via KSM
- Kata container Components
  - Agent: daemon inside VM that manages /create container processes inside the VM
  - Runtime: OCI-compliant, handles commands to launch container, create shims
  - Shim: representation of container processes inside VM, forwards stdin, stdout and signals



Summary and Conclusion

- Container technology is the dominant technology in the market today
- Mainstream container (orchestration) technologies
  - Docker, Kubernentes
- HPC community moving towards containers
- Security is the most critical concern
  - Weak isolation property
  - Various exiting system security tools are adapted into the container world
  - There are several efforts to building secure container runtimes