



# 目录

## ① 总览

核心技术  
容器架构

## ② 容器运行时

## ③ 容器引擎

Docker vs. Podman  
其他引擎

## ④ 其余产品



# 第 1 节 总览



# 第 1 节 总览

## 第 1 小节 核心技术



# 安全隔离 - namespace<sup>1</sup>

- 命名空间将全局系统资源包装在一个抽象中
- 命名空间内的进程看起来他们拥有自己独立的全局资源实例
- 每个进程只能访问命名空间内的局部资源

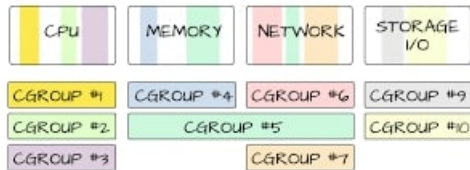
Namespace	Flag	Page	Isolates
Cgroup	CLONE_NEWCGROUP	cgroup_namespaces(7)	Cgroup root directory
IPC	CLONE_NEWIPC	ipc_namespaces(7)	System V IPC, POSIX message queues
Network	CLONE_NEWNET	network_namespaces(7)	Network devices, stacks, ports, etc.
Mount	CLONE_NEWNS	mount_namespaces(7)	Mount points
PID	CLONE_NEWPID	pid_namespaces(7)	Process IDs
Time	CLONE_NEWTIME	time_namespaces(7)	Boot and monotonic clocks
User	CLONE_NEWUSER	user_namespaces(7)	User and group IDs
UTS	CLONE_NEWUTS	uts_namespaces(7)	Hostname and NIS domain name

<sup>1</sup><https://man7.org/linux/man-pages/man7/namespaces.7.html>



# 性能隔离 - cgroup<sup>2</sup>

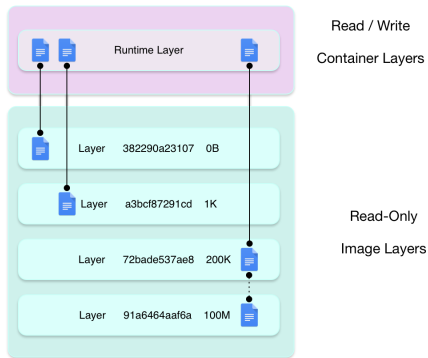
- cgroup 可以将进程组织成在限制和监控下使用各种类型资源的分层组
- 一个 cgroup 包含多个 task (线程), 可以组成的树状结构层级控制
- cgroup 可以对资源做限制 (最大值、分配比例)、优先级分配以及审计



<sup>2</sup><https://medium.com/@BeNitinAgarwal/understanding-the-docker-internals-7ceb052ee9fe>

# 镜像存储 - UnionFS<sup>3</sup>

- Unionfs 是轻量级高性能分层文件系统，可以将修改一次提交，层层叠加
- 修改镜像只需要添加新层，分发镜像只需要分发增量层
- 当容器基于镜像创建时，会在镜像的最上层添加一个可写层，所有对于运行时容器的修改其实都是对这个容器读写层的修改



<sup>3</sup><https://enqueuezero.com/container-and-unionfs.html>

# 第 1 节 总览

## 第 2 小节 容器架构





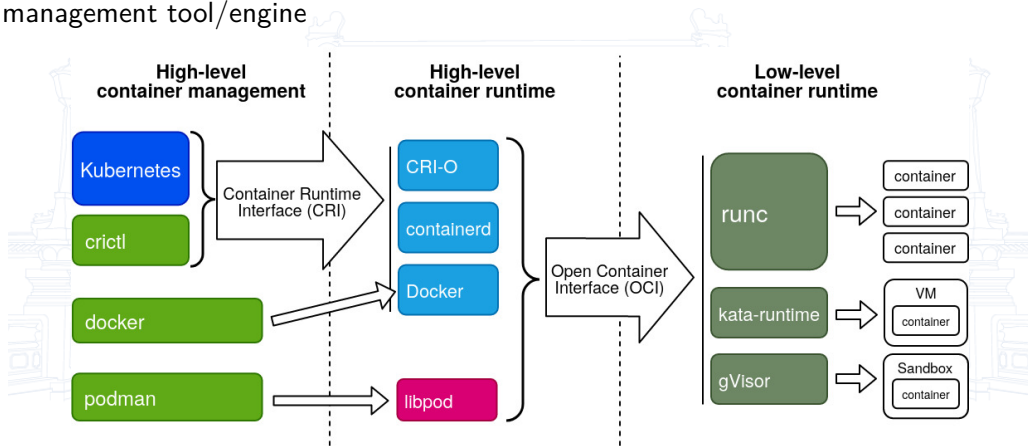
# 容器标准

- Open Container Initiative 开放容器标准
  - Runtime Specification: 如何运行一个解压的镜像
  - Image Format: 定义一个 OCI 镜像的组成
  - Distribution Specification: 标准化内容分发的 API 协议
- Container Runtime Interface 容器运行时接口
  - 用来控制创建和管理容器的不同运行时的 API



## 调用链<sup>4</sup>

- high-level/low-level runtime
- management tool/engine



<sup>4</sup><https://merlijn.sebrechts.be/blog/2020-01-docker-podman-kata-cri-o/>

## 第 2 节 容器运行时



## 主流产品<sup>5</sup>

## Runtime - Container Runtime (11)



<b>containerd</b>	★ 9,428
Cloud Native Computing Foundation (CNCF)	Funding: \$3M



cri-o

**CRI-O** ★ 3,641  
Cloud Native  
Computing  
Foundation  
(CNCF) Funding: \$3M



## Firecracker

<b>Firecracker</b>	★16,200
Amazon Web Services	MCap: \$1.7T



<b>gVisor</b>	★ 11,745
Google	MCap: \$1.9T

 INCLAVARE

**Inclavare  
Containers** ★397  
Funding: \$3M  
Cloud Native  
Computing  
Foundation  
(CNCF)



kata

**Kata Containers** ★2,101  
Open Infrastructure  
Foundation



**lxd**

lxd ★3,018  
Canonical Funding: \$12.8M



OPEN CONTAINER INITIATIVE  
RUDC

**runc** ★ 8,464  
Open Container Initiative  
(OCI)



**Singularity**

**Singularity** ★2,218

Sylabs



# SmartOS

SmartOS	★1,437
Jovent	Funding: \$131M



WasmEdgeRuntime

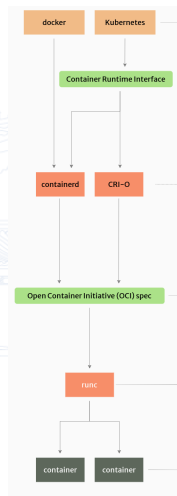
**WasmEdge Runtime** ★1,767  
Funding: \$3M

Cloud Native  
Computing  
Foundation  
(CNCF)

<sup>5</sup><https://landscape.cncf.io/>

# containerd/CRI-O/runc<sup>6</sup>


- 最主流的工业级标准的容器运行时
- CRI-O 主要面向 Kubernetes，完整实现 CRI 接口功能，并且严格兼容 OCI 标准
- containerd 在性能上更好，社区支持更广泛
- 链接：🔗 🔗 🔗

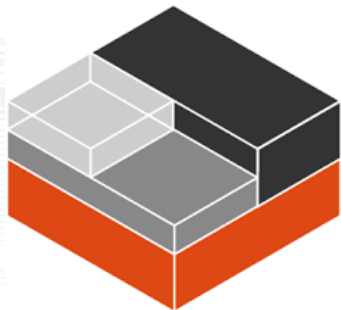


<sup>6</sup><https://www.tutorialworks.com/difference-docker-containerd-runc-crio-oci/>




# lxc/lxd

- lxc 是最早的容器隔离技术，但部署在单机上，不能无法有效支持跨主机之间的容器迁移，管理复杂，没有版本控制
- lxd 是 lxc 的下一代，改善了用户体验
  - 优势：对系统的封装更加完善；加强了多线程上支持，可并发执行多程序；数据持久化储存方案更好
  - 劣势：专注于部署 linux 虚拟机，而不是部署应用
- 链接：




# Singularity

- 致力于高性能计算 HPC 的容器运行时
- 优势：针对以计算为中心的企业和 HPC 工作负载进行了优化
- 劣势：没有提供非常完善的隔离和虚拟化，没有网络虚拟化，社区支持较少
- 链接：




# kata-runtime

- 致力于用轻量级虚拟机构建安全的容器运行时，使用硬件虚拟化技提供更强的隔离，适用于注重安全的场景
- 在专用内核中运行，提供网络、I/O 和内存的隔离
- 提供与标准 Linux 容器相同的性能；增加了隔离，而无需标准虚拟机的性能负担
- 链接：





# WasmEdge Runtime

- 轻量级、高性能和可扩展的 WebAssembly 运行时，适用于云原生、边缘和去中心化应用程序
- 注重于面向 Web 与计算密集型应用，如游戏引擎，可能不适用于其他场景
- 链接：



## WasmEdgeRuntime



# crun

- 完全用 C 编写的高性能且低内存占用的 OCI 容器运行时
- 可以用作一个库包含在程序中，而无需容器引擎来管理，支持 cgroups v2
- 相比 runc 有 49.4% 的性能提升
- 链接：🔗

crun vs. runc	
runtime	100 /bin/true
crun	0:01.69
runc	0:3.34



## 第 3 节 容器引擎




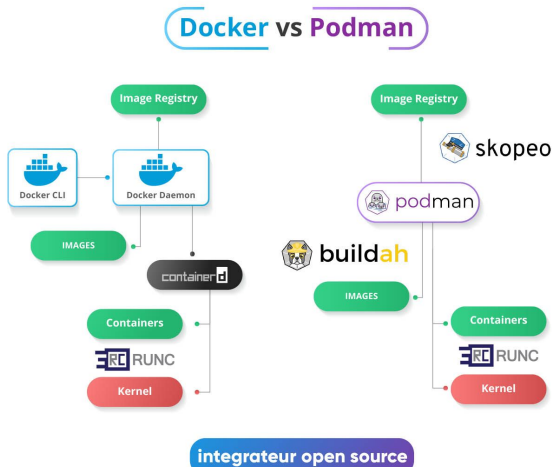
## 第 3 节 容器引擎

### 第 1 小节 Docker vs. Podman



# Docker<sup>7</sup>

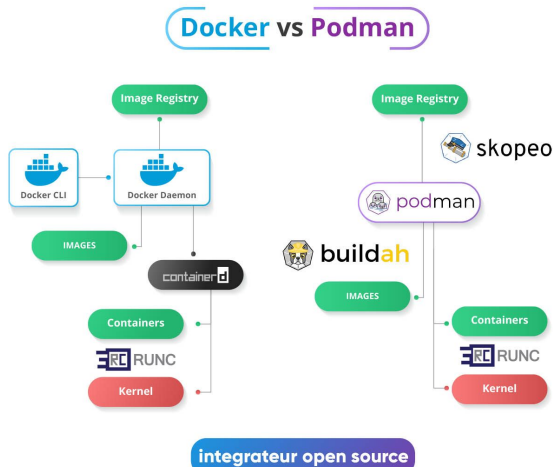
- Docker CLI 个用来与 Docker Daemon 进行交互
- Docker Daemon 是一个常驻的后台进程，帮助管理和创建 Docker 镜像、容器、网络 and 存储卷
- Docker 是单体应用，它无需额外工具即可处理整个容器生命周期
- 链接：



<sup>7</sup><https://ios.dz/from-docker-to-podman/>

# Podman<sup>8</sup>

- Podman 直接与镜像注册表、容器和镜像存储进行交互, 直接使用 runC 容器运行时
- buildah 项目实现 dockerfile 的脚本化执行
- skopeo 项目负责处理镜像相关的工作
- 链接: 🔗



<sup>8</sup><https://ios.dz/from-docker-to-podman/>

# 比较

Docker	Podman
逐渐支持多平台	只支持 Linux
有守护进程，存在单点故障	没有守护进程，没有单点故障
单体应用	需要 buildah 和 skopeo 的帮助
可以构建容器镜像	不能构建容器镜像
以 root 方式运行，有诸多安全漏洞	可以以非 root 模式运行，更安全



## 第 3 节 容器引擎


### 第 2 小节 其他引擎





# Pouch

## 企业级富容器引擎


- 隔离性强
  - 用户态增强容器的隔离维度，比如网络带宽、磁盘使用量等
  - 给内核提交 patch，修复容器的资源可见性问题，cgroup 方面的 bug
  - 实现基于 Hypervisor 的容器，通过创建新内核来实现容器隔离
- 富容器技术
  - 富容器指在 Linux 内核上创建一个与虚拟机体验完全一致的容器，实现容器镜像实现业务的快速交付、容器环境兼容企业原有运维体系
- P2P 镜像分发
  - 利用基于 P2P 的分发系统 Dragonfly 实现企业大规模的快速容器镜像分发
- 链接：

# PouchContainer



# balena-engine

为嵌入式和物联网设计的容器引擎

- 占用空间小：比 Docker CE 小 3.5 倍
- 多架构支持：适用于多种架构
- 写入时磁盘保护
- 镜像拉取容错：保证原子性和持久性
- 保守内存使用：防止镜像拉取期间频繁替换 cache，应用程序可以在低内存情况下不受干扰运行
- 链接：



## 第 4 节 其余产品



## 其余轻量级虚拟化

namespace 对容器的限制并不完美，运行在容器上的恶意程序会威胁主机安全

- firecracker(microVM)🔗
  - 一个基于 KVM 的新型 VMM，在宿主机上提供轻量级的虚拟化支持
  - 面向 serverless 场景，专为短周期任务设计，占用 5MB 内存，125ms 启动，能够支持快速启动进行快速计算
- gVisor(VM-like)🔗
  - 一个用于提供安全隔离的轻量级容器运行时沙箱
  - gVisor 用 sandbox 提供了和 VM 方案相当隔离等级，在用户空间模拟一些系统调用，也做到了轻量级和较小内存消耗



Firecracker



gVisor



## 基于硬件 Enclave 的隔离

- Enclave 基于权限控制和加密提供可信执行环境
- 面向机密计算的 Enclave 容器能为其中的敏感和机密数据提供基于密钥学算法的强安全隔离，阻止不可信的实体访问用户的数字资产
- Inclave Containers<sup>®</sup>是阿里云研发的面向机密计算场景的开源容器运行时



INCLAVARE  
CONTAINERS





谢谢