Kata Containers总结

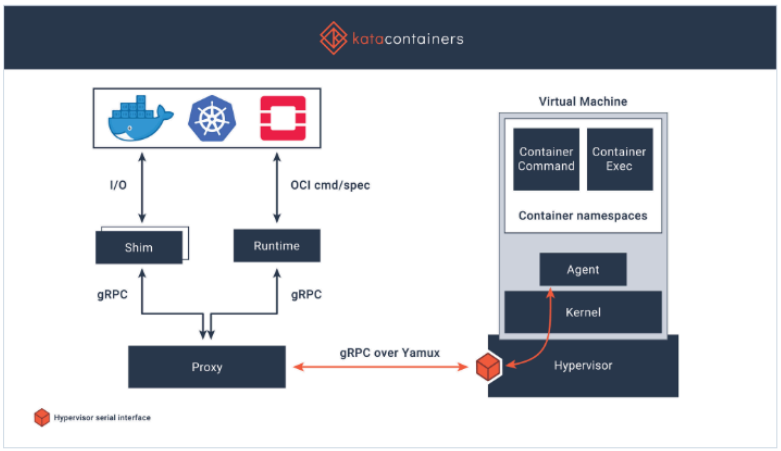
简介

`Kata Containers 是一个开源项目和社区，致力于构建轻量级虚拟机 (VM) 的标准实现，其感觉和性能类似于容器，但提供了 VM 的工作负载隔离和安全优势。`

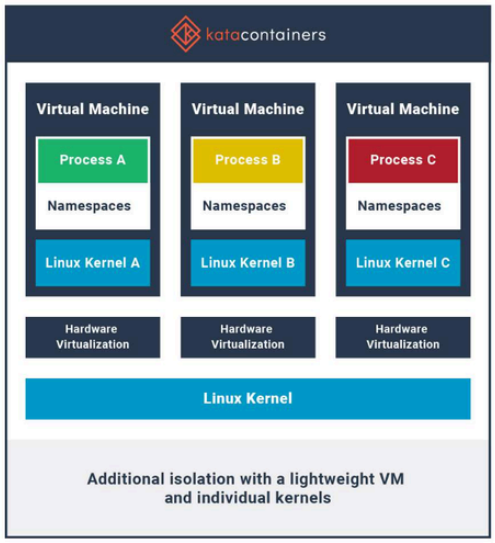
`Kata Containers 是一个开源容器运行时，构建可无缝插入容器生态系统的轻量级虚拟机。`

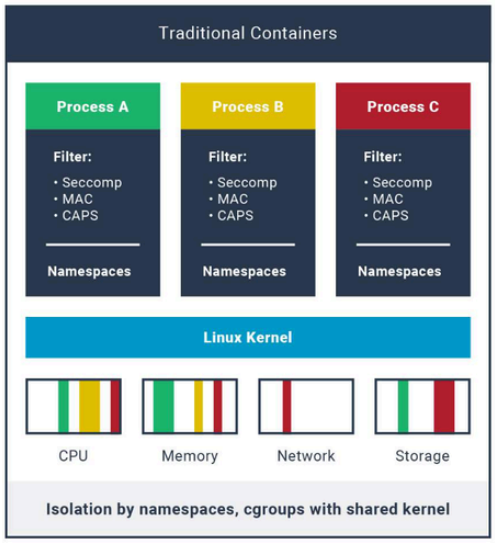
架构

kata-containers/docs/design/architecture at main · kata-containers/kata-containers

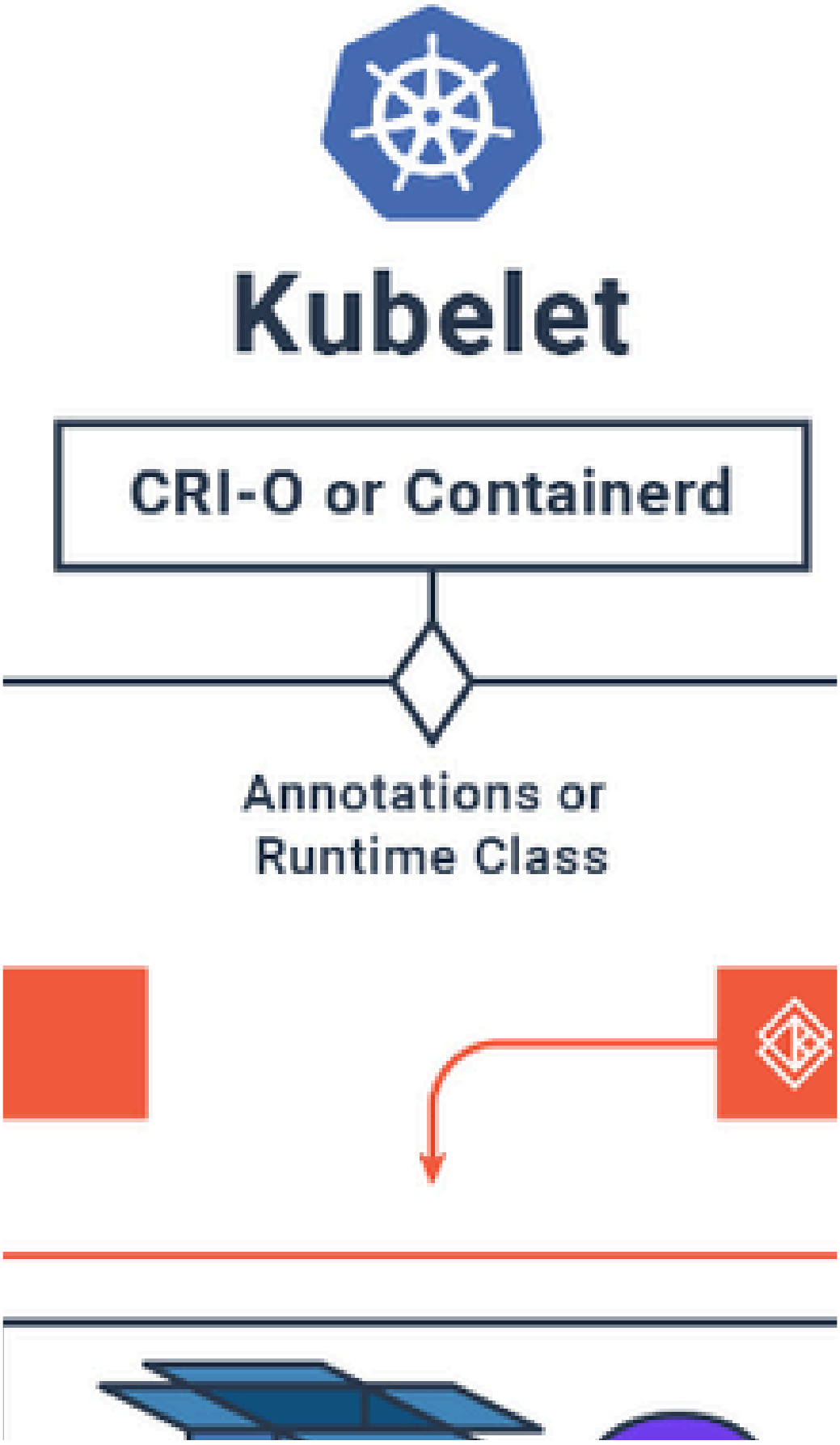


与传统容器对比





与kubernetes集成



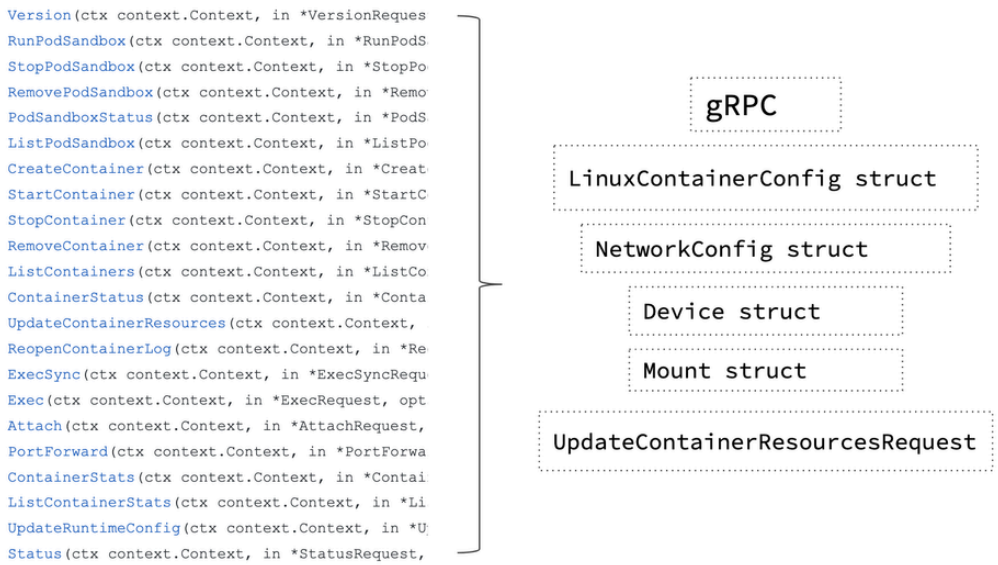
虚拟化

kata-containers/docs/design/virtualization.md at main · kata-containers/kata-containers

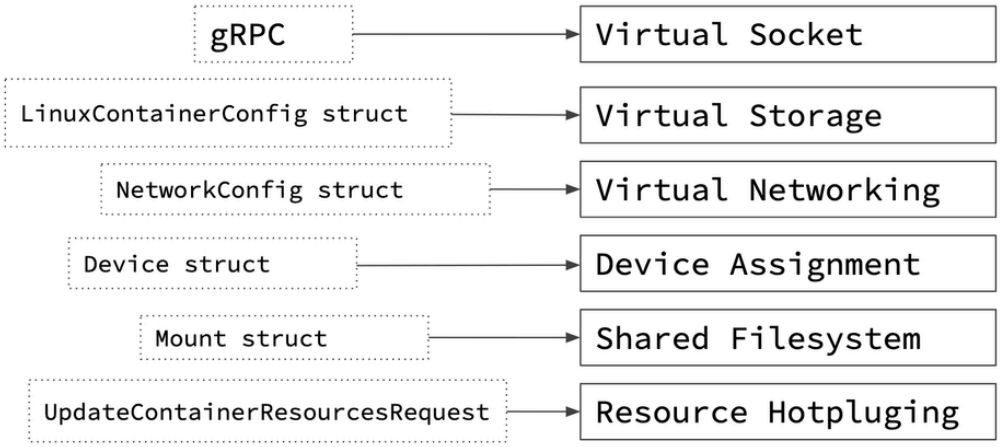
Kata 容器是在传统命名空间容器提供的隔离之上创建的第二层隔离。硬件虚拟化接口是这个附加层的基础。 Kata 将启动一个轻量级虚拟机，并使用guest 的 Linux 内核来创建容器工作负载，或者在多容器 Pod 的情况下创建工作负载。在 Kubernetes 和 Kata 实现中，沙箱是在 pod 级别进行的。在 Kata 中，这个沙箱是使用虚拟机创建的。

Kata 容器的典型部署将通过容器运行时接口 (CRI) 实现在 Kubernetes 中进行。在每个节点上，Kubelet 将与 CRI 实现者（例如 containerd 或 CRI-O）进行交互，而 CRI 实现者又将与 Kata Containers（基于 OCI 的运行时）进行交互。

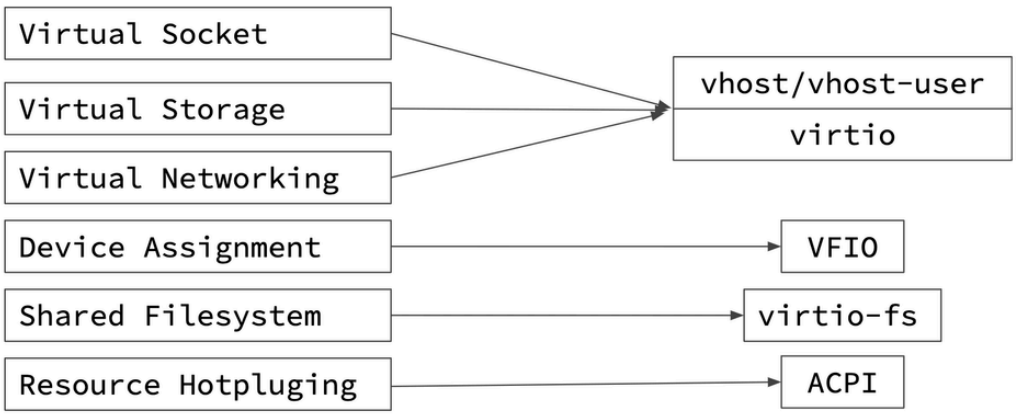
kata需要提供以下结构：



这些构造可以进一步映射到与虚拟机交互所需的设备：



这些概念映射到特定的半虚拟化设备或虚拟化技术：



基于 QEMU 的 Kata Containers 与 Kubernetes 完全兼容。

根据宿主机架构的不同，Kata Containers 支持各种机器类型，例如 x86 系统上的 q35、ARM 系统上的 virt 以及 IBM Power 系统上的 pseries。默认的 Kata Containers 机器类型是 q35。可以通过编辑运行时配置文件来更改机器类型及其机器加速器。

使用的设备和特性有：

- virtio VSOCK or virtio serial
- virtio block or virtio SCSI
- virtio net
- virtio fs or virtio 9p (recommend: virtio fs)
- VFIO
- hotplug
- machine accelerators

Kata 容器中使用machine accelerators和hotplug来管理资源限制、缩短启动时间并减少内存占用。

machine accelerators：

```
机器加速器是特定于架构的，可用于提高性能并启用机器类型的特定功能。 Kata Containers 使用以下机器加速器：

NVDIMM：此机器加速器特定于 x86，仅受 q35 机器类型支持。 nvdimm 用于向虚拟机提供根文件系统作为持久内存设备。
```

hotplug devices:

Kata Containers VM 以最少量的资源启动，从而缩短启动时间并减少内存占用。随着容器启动的进行，设备会热插拔到虚拟机。例如，当指定的 CPU 约束包含额外的 CPU 时，可以热添加它们。 Kata Containers 支持热添加以下设备：

- Virtio block
- Virtio SCSI
- VFIO
- CPU

kata支持的hypervisor：

Solution	release introduced	brief summary
Cloud Hypervisor	1.10	upstream Cloud Hypervisor with rich feature support, e.g. hotplug, VFIO and FS sharing
Firecracker	1.5	upstream Firecracker, rust-VMM based, no VFIO, no FS sharing, no memory/CPU hotplug
QEMU	1.0	upstream QEMU, with support for hotplug and filesystem sharing
StratoVirt	3.3	upstream StratoVirt with FS sharing and virtio block hotplug, no VFIO, no CPU/memory resize

兼容性

```
Kata Containers 运行时与 OCI 运行时规范兼容，因此可以通过 CRI-O 和 containerd 实现与 Kubernetes 容器运行时接口 (CRI) 无缝协作。

Kata Containers 提供了“shimv2”兼容的运行时。
```

Shim v2架构

containerd 运行时 shimv2 架构或 shim API 架构通过定义兼容运行时实现必须提供的一组 shimv2 API 解决了旧架构的问题。 shimv2 架构不是为每个新容器多次调用运行时二进制文件，而是运行运行时二进制文件的单个实例（对于任意数量的容器）。这提高了性能并解决了状态处理问题。

shimv2 API 与 OCI 运行时 API 类似，将容器生命周期拆分为不同的动作。容器管理器不是多次调用运行时，而是创建一个套接字并将其传递给 shimv2 运行时。套接字是一个双向通信通道，它使用基于 gRPC 的协议，允许容器管理器向运行时发送 API 调用，运行时使用同一通道将结果返回给容器管理器。

shimv2 架构允许每个虚拟机运行多个容器，以支持需要在 pod 内运行多个容器的容器引擎。

通过新的架构，Kubernetes 可以启动 Pod 和 OCI 兼容的容器，每个 Pod 使用一个运行时 shim，而不是 2N+1 shim。即使 VSOCK 不可用，也不需要独立的 kata-proxy 进程。

工作负载

工作负载是用户请求在容器中运行的命令，并在 OCI 捆绑包的配置文件中指定。

工作负载根文件系统

工作负载通过其周围的容器环境与guest虚拟机环境隔离。容器运行的guest虚拟机环境也与容器管理器运行的外部宿主机环境隔离。

系统概览

环境

Type	Name	Virtualized	Containerized	rootfs	Rootfs device type	Mount type	Description
Host	Host	no `[1]`	no	Host specific	Host specific	Host specific	The environment provided by a standard, physical non virtualized system.
VM root	Guest VM	yes	no	rootfs inside the guest image	Hypervisor specific `[2]`	`ext4`	The first (or top) level VM environment created on a host system.
VM container root	Container	yes	yes	rootfs type requested by user (`ubuntu` in the example)	`kataShared`	virtio FS	The first (or top) level container environment created inside the VM. Based on the OCI bundle.

```
将工作负载容器化在虚拟机内的原因是：

将工作负载与虚拟机环境完全隔离。
在 Pod 中的容器之间提供更好的隔离。
允许通过其 cgroup 限制来管理和监控工作负载。
```

容器创建

1、`用户通过运行命令请求创建容器`。

2、`容器管理器守护进程运行 Kata 运行时的单个实例。`

3、`Kata 运行时加载其配置文件。`

4、`容器管理器在运行时调用一组 shimv2 API 函数。`

5、`Kata 运行时启动配置的虚拟机管理程序。`

6、

```
虚拟机管理程序使用guest资源创建并启动（引导）VM：

虚拟机管理程序 DAX 将guest镜像共享到 VM 中，成为 VM rootfs（安装在 /dev/pmem* 设备上），这称为 VM 根环境。

虚拟机管理程序使用 virtio FS 将 OCI 包安装到 VM rootfs 内的容器特定目录中。

这个容器特定的目录将成为容器rootfs，称为容器环境。
```

7、`agent作为 VM 启动的一部分启动。`

8、

```
运行时调用agent的 CreateSandbox API 来请求agent创建容器：

agent在包含容器 rootfs 的容器特定目录中创建容器环境。

容器环境将工作负载托管在容器 rootfs 目录中。

agent在容器环境内生成工作负载。
```

注意：

`Agent创建的容器环境相当于runc OCI运行时创建的容器环境； Linux cgroup 和命名空间由guest内核在 VM 内创建，以将工作负载与创建容器的 VM 环境隔离开来。`

9、`容器管理器将容器的控制权返回给运行 ctr 命令的用户。`

注意：

```
此时，容器已经运行并且：

工作负载进程在容器环境中运行。
用户现在可以与工作负载交互（使用ctr 命令）。
在虚拟机内运行的agent监视工作负载进程。
运行时正在等待agent的 WaitProcess API 调用完成。
```

容器关闭

```
终止容器环境有两种可能的方式：

当工作负载退出时。

这是标准的或优雅的关闭方法。

当容器管理器强制删除容器时。
```

工作负载退出

```
agent将检测工作负载进程何时退出，捕获其退出状态（请参阅 wait(2)），并通过将该值指定为对运行时发出的 WaitProcess 代理 API 调用的响应，将该值返

然后，运行时通过 Wait shimv2 API 调用将该值传递回容器管理器。

一旦工作负载完全退出，就不再需要虚拟机，并且运行时会清理环境（包括终止虚拟机管理程序进程）。
```

注意：

`启用agent跟踪后，关闭行为会有所不同。`

容器管理器请求关闭

`如果容器管理器请求删除容器，运行时将通过向代理发送 DestroySandbox ttRPC API 请求来向代理发出信号。`

Guest资产

`guest资产包括由虚拟化管理器程序使用的guest镜像和guest内核`。

虚拟化管理器

`配置文件中指定的虚拟机管理程序创建一个虚拟机来托管容器环境内的agent和工作负载。`

注意：

虚拟机管理程序进程在与主机环境略有不同的环境中运行： 它在与主机不同的 cgroup 环境中运行。 它被赋予与主机不同的网络命名空间。 如果 OCI 配置指定了 SELinux 标签，则虚拟机管理程序进程将使用该标签运行（而不是在虚拟机管理程序的 VM 内运行的工作负载）。

Agent

`Kata Containers agent (kata-agent) 采用 Rust 编程语言编写，是一个在虚拟机内运行的长时间运行的进程。它充当管理容器以及在这些容器中运行的工作负载的主管。每个创建的虚拟机仅运行一个agent进程。`

agent通信协议

`agent使用基于 ttRPC 的协议与其他 Kata 组件（主要是运行时）进行通信。`

注意：

`如果您希望了解有关此协议的更多信息，一种实用的方法是在测试系统上使用agent控制工具进行实验。该工具仅用于测试和开发目的，可以向agent发送任意 ttRPC 代理 API 命令。`

运行时

`Kata Containers 运行时（containerd-shim-kata-v2 二进制文件）是 shimv2 兼容的运行时。`

注意：

`Kata Containers 运行时有时称为 Kata shim。这两个术语都是正确的，因为containerd-shim-kata-v2是一个容器运行时，并且该运行时实现了 containerd shim v2 API。`

```
运行时大量使用 virtcontainers 包，它提供了一个通用的、与运行时规范无关的硬件虚拟化容器库。

运行时负责启动虚拟机管理程序及其 VM，并通过 VSOCK 套接字使用基于 ttRPC 的协议与agent进行通信，VSOCK 套接字在 VM 和主机之间提供通信链接。

该协议允许运行时向agent发送容器管理命令。该协议还用于在容器和容器管理器（例如 CRI-O 或 containerd）之间传输标准 I/O 流（stdout、stderr、stdin）。
```

实用程序

`kata-runtime 二进制文件是一个实用程序，它提供管理命令来操作和查询 Kata Containers 安装。`

注意：

`在 Kata 1.x 中，该程序还充当主运行时，但由于改进的 shimv2 架构，不再需要它。`

exec命令

`exec 命令允许管理员或开发人员进入容器工作负载无法访问的 VM 根环境。`

policy命令

`策略设置命令允许管理员或开发人员为VM根环境设置策略。这样，我们就可以通过策略启用/禁用 kata-agent API。`

配置

`配置文件还用于启用运行时调试输出。`

进程概览

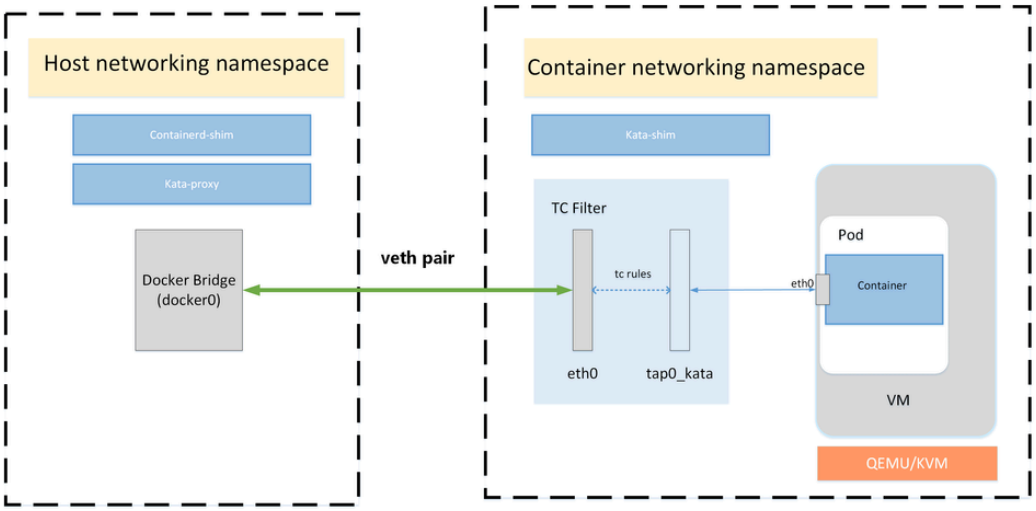
Description	Host	VM root environment	VM container environment
Container manager	`containerd`		

Kata Containers	runtime, `virtiofsd`, hypervisor	agent	
User workload			`ubuntu sh`

网络

kata-containers/docs/design/architecture/networking.md at main · kata-containers/kata-containers

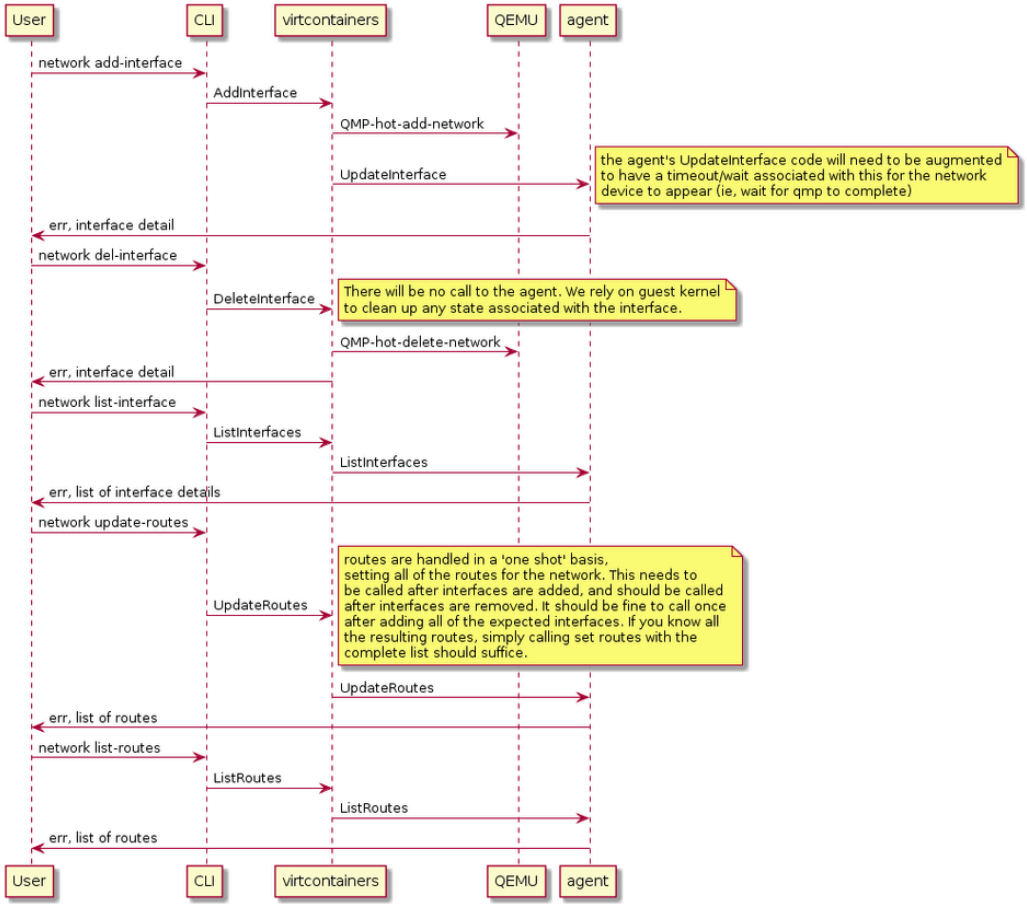
```
容器通常位于它们自己的（可能是共享的）网络命名空间中。在容器生命周期的某个时刻，容器引擎将设置该命名空间以将容器添加到与主机网络隔离的网

为了为容器设置网络，容器引擎调用网络插件。网络插件通常会创建一个虚拟以太网 (veth) 对，将 veth 对的一端添加到容器网络命名空间中，而 veth 对的另

这是一种非常以命名空间为中心的方法，因为许多虚拟机管理程序或 VM 管理器 (VMM)（例如 virt-manager）无法处理 veth 接口。通常，TAP 接口是为虚拟

为了克服典型容器引擎期望与虚拟机之间的不兼容性，Kata Containers 网络使用流量控制透明地将 veth 接口与 TAP 接口连接起来：
```




```
部署 TC 过滤规则后，容器网络和虚拟机之间就会创建重定向。例如，网络插件可以将设备 eth0 放置在容器的网络命名空间中，该命名空间是 VETH 设备的

Kata Containers 维护对 MACVTAP 的支持，这是 Kata 中使用的早期实现。通过这种方法，Kata 创建了一个 MACVTAP 设备来直接连接到 eth0 设备。 TC-filt

由于相对于 TC-filter 和 MACVTAP 缺乏性能，Kata Containers 已弃用对 Bridge 的支持。

Kata Containers 支持 CNM 和 CNI 进行网络管理。
```

网络hotplug：

```
Kata Containers 开发了一组网络子命令和 API，用于添加、列出和删除访客网络端点以及操作访客路由表。

下图说明了 Kata Containers 网络热插拔工作流程。
```



存储

kata-containers/docs/design/architecture/storage.md at main · kata-containers/kata-containers

```
Kata Containers 与现有标准和运行时兼容。从存储的角度来看，这意味着容器工作负载可以使用的存储量没有限制。

由于 cgroup 无法设置存储分配限制，因此如果您希望限制容器使用的存储量，请考虑使用现有设施，例如配额（1）限制或设备映射器限制。
```

如果配置了基于块的图形驱动程序，则使用 `virtio-scsi` 将工作负载映像（例如 `busybox:latest`）共享到 `VM` 内的容器环境中。

```
如果未配置基于块的图形驱动程序，则会使用 virtio-fs (VIRTIO) 覆盖文件系统挂载点来共享工作负载映像。agent使用此安装点作为容器进程的根文件系统。

对于 virtio-fs，运行时为创建的每个虚拟机启动一个 virtiofsd 守护进程（在主机上下文中运行）。
```

```
devicemapper 快照程序是一个特例。快照程序使用专用块设备而不是格式化文件系统，并在块级别而不是文件级别进行操作。这些知识用于直接使用底层块

热插拔
Kata Containers 能够热插拔添加和热插拔删除块设备。这使得在虚拟机启动后启动的容器可以使用块设备。

用户可以通过在容器内调用 mount(8) 来检查容器是否使用 devicemapper 块设备作为其 rootfs。如果使用 devicemapper 块设备，根文件系统 (/) 将从 /dev/vda 挂
```

Kubernetes支持

kata-containers/docs/design/architecture/kubernetes.md at main · kata-containers/kata-containers

OCI注解

```
为了让 Kata Containers 运行时（或任何基于 VM 的 OCI 兼容运行时）能够了解是否需要创建完整的 VM，或者是否必须在现有 pod 的 VM 内创建新容器，C

在调用其运行时之前，CRI-O 始终会将 io.kubernetes.cri-o.ContainerType 注释添加到它从 Kubelet CRI 请求生成的 config.json 配置文件中。 io.kubernetes.cri-o.
```

Annotation value	Kata VM created?	Kata container created?
`sandbox`	yes	yes (inside new VM)
`container`	no	yes (in existing VM)

混合基于VM和基于命名空间的运行时

注意：

从 Kubernetes 1.12 开始，支持 Kubernetes RuntimeClass，用户可以指定运行时，而无需非标准化注释。

通过 RuntimeClass，用户可以将 Kata Containers 定义为 RuntimeClass，然后显式指定必须将 pod 创建为 Kata Containers pod。详细信息请参考：https://github.com/kata-containers/kata-containers/blob/main/docs/how-to/containerd-kata.md

追踪

kata-containers/docs/tracing.md at main · kata-containers/kata-containers

附录

DAX

Kata Containers 利用 Linux 内核 DAX（直接访问文件系统）功能将宿主机环境中的guest镜像高效地映射到guest VM 环境中，成为 VM 的 rootfs。

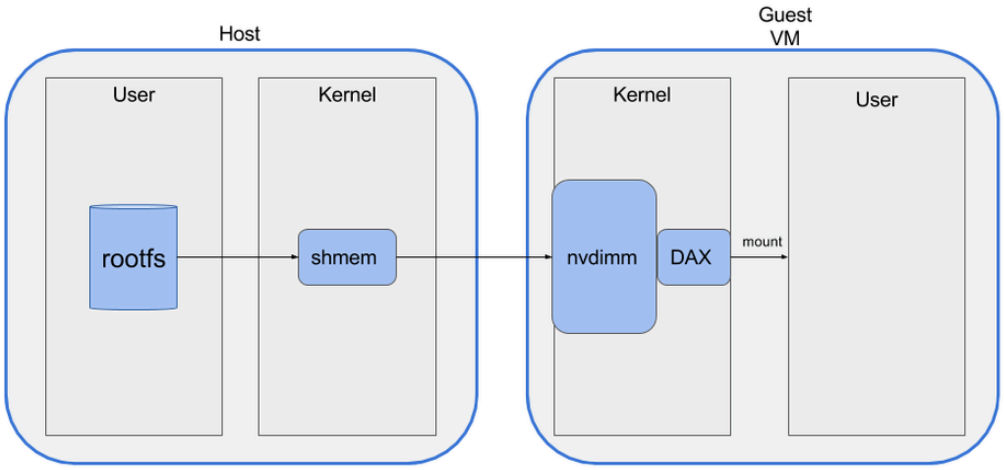
Hypervisor	Feature used	rootfs device type
Cloud Hypervisor (CH)	`dax` `FsConfig` configuration option	PMEM (emulated Persistent Memory device)
QEMU	NVDIMM memory device with a memory file backend	NVDIMM (emulated Non-Volatile Dual In-line Memory Module device)

上表中的功能是等效的，因为它们提供了内存映射虚拟设备，用于 DAX 将 VM 的 rootfs 映射到 VM guest内存地址空间。

然后引导 VM，指定 root= 内核参数以使guest内核使用适当的模拟设备作为其 rootfs。

DAX优势

```
与更传统的 VM 文件和设备映射机制相比，使用 DAX 映射文件具有许多优势：

映射为直接访问设备允许GUEST直接访问主机内存页面（例如通过就地执行（XIP）），绕过GUEST内核的页面缓存。这种零拷贝提供了时间和空间的优化

映射为 VM 内的直接访问设备允许使用页面错误按需加载来自主机的页面，而不必通过虚拟化设备发出请求（导致昂贵的 VM 退出/超级调用），从而提供

在主机上利用 mmap(2) 的 MAP_SHARED 共享内存选项允许主机有效地共享页面。
```



NVDIMM with QEMU详情请见：QEMU

Agent Control tool

`代理控制工具是一个测试和开发工具，可用于了解有关 Kata Containers 系统的更多信息。`

特性

安全：在专用内核中运行，提供网络、I/O 和内存的隔离，并可以利用虚拟化 VT 扩展的硬件强制隔离。

兼容性：`支持行业标准，包括 OCI 容器格式、Kubernetes CRI 接口以及传统虚拟化技术。`

性能：提供与标准 Linux 容器一致的性能；增强隔离性，且不会产生标准虚拟机的性能负担。

简单：`消除了在成熟的虚拟机中嵌套容器的要求；标准接口使其易于插入和启动。`

限制

kata-containers/docs/Limitations.md at main · kata-containers/kata-containers

`拥有两个标准使支持 Docker 环境的挑战变得复杂，因为运行时必须支持官方 OCI 规范和 runc 提供的非标准扩展。`

限制列表：https://github.com/pulls?utf8=%E2%9C%93&q=is%3Aopen+label%3Alimitation+org%3Akata-containers 连接您的 Github 帐户

关键限制：

1、`Kata 容器不支持 Podman`。

2、`运行时不提供检查点和恢复命令。有人讨论使用虚拟机保存和恢复来为我们提供类似 criu 的功能，这可能会提供一个解决方案。`

3、`运行时未完全实现 events 命令。不完全支持 OOM 通知和 Intel RDT 统计信息。`

4、`目前，仅不支持块 I/O 权重。所有其他配置均受支持并且工作正常。`

5、不支持Host network，无法从虚拟机内直接访问主机网络配置。

6、`Kata Containers 不支持网络命名空间共享。如果 Kata 容器设置为共享 runc 容器的网络命名空间，则运行时将有效地接管分配给该命名空间的所有网络接口并将它们绑定到 VM。因此，runc 容器失去其网络连接。`

7、`由于虚拟机在 CPU 和内存分配以及跨主机系统共享方面存在差异，因此为这些命令实现等效方法可能具有挑战性。`

CPU限制：kata-containers/docs/design/vcpu-handling-runtime-go.md at main · kata-containers/kata-containers

8、`Kata 容器目前不支持 Kubernetes VolumeMount.subPath。`

9、`Kata 中的特权支持与 runc 容器本质上不同。容器在guest vm中以增强的功能运行，并被授予对guest vm设备而不是宿主机设备的访问权限。在 Kubernetes 中使用 securityContextprivileged=true 也是如此。`

kata特权设置：kata-containers/docs/how-to/privileged.md at main · kata-containers/kata-containers

10、

```
对于基于 VM 的系统，将 cgroup、CPU、内存和存储等资源限制应用于工作负载并不总是那么简单。 Kata 容器在虚拟机内的隔离环境中运行。这与 Kata 容

在某些情况下，可能需要将约束应用到多个级别。在其他情况下，硬件隔离VM提供与所请求的约束等效的功能。

以下示例概述了可以应用的一些不同领域的约束：

虚拟机内部

限制guest内核。这可以通过通过用于引导guest内核的内核命令行传递特定值来实现。或者，可以在早期启动时应用 sysctl 值。

容器内部

限制虚拟机内创建的容器。

虚拟机外部：

通过应用主机级约束来约束虚拟机管理程序进程。

限制虚拟机管理程序内运行的所有进程。

这可以通过指定特定的虚拟机管理程序配置选项来实现。

请注意，在某些情况下，可能需要对多个先前区域应用特定约束，以实现所需的隔离和资源控制级别。
```

安装

kata-containers/docs/install at main · kata-containers/kata-containers

前置条件

kata-containers/src/runtime/README.md at main · kata-containers/kata-containers

运行时

二进制名称

Binary name	Description
`containerd-shim-kata-v2`	The shimv2 runtime
`kata-runtime`	utility program
`kata-monitor`	metrics collector daemon

介绍

containerd-shim-kata-v2 二进制文件是 Kata Containers shimv2 运行时。它利用 virtcontainers 包提供符合标准的高性能运行时，用于创建在 Linux 主机上运行的硬件虚拟化 Linux 容器。 该运行时兼容 OCI、CRI-O 和 Containerd，使其能够分别与 Docker 和 Kubernetes 无缝协作。

配置

配置文件是 `configuration.toml` ，包含运行时自身，agent和hypervisor的配置项。

```
shimv2 运行时在以下位置查找其配置（按顺序）：

（1）用于创建 pod 沙箱的 OCI 配置文件（config.json 文件）中指定的 io.dataContainers.config.config_path 注释。

（2）传递给运行时的 containerd shimv2 选项。

（3）KATA_CONF_FILE 环境变量的值。

（4）默认配置路径。
```

```
kata-runtime 实用程序在以下位置（按顺序）查找其配置：

（1）由 --config 命令行选项指定的路径。

（2）KATA_CONF_FILE 环境变量的值。

（3）默认配置路径。
```

注意：

`对于这两个二进制文件，将使用第一个存在的路径。`

`为了在不更改配置文件本身的情况下更改配置，支持插入配置文件片段。解析配置文件后，如果配置文件所在目录中有一个名为 config.d 的子目录，则其内容将按字母顺序加载，并且每个项目将被解析为配置文件。从这些配置文件片段加载的设置会覆盖从主配置文件和早期片段加载的设置。鼓励用户使用熟悉的命名约定来订购片段（例如 config.d/10-this、config.d/20-that 等）。`

`由于运行时支持无状态系统，因此它会在多个位置检查此配置文件，其中两个位置内置于运行时中。标准系统的默认位置是 /usr/share/defaults/kata-containers/configuration.toml。但是，如果 /etc/kata-containers/configuration.toml 存在，则优先。`

日志

```
Kata containerd shimv2运行时通过containerd记录日志，其日志将被发送到containerd日志所指向的任何地方。但是，shimv2 运行时也始终使用 kata 标识符记录

注意：Kata 日志记录需要启用 containerd debug。

要查看 shimv2 运行时日志：

$ sudo Journalctl -t kata
```

调试

kata-containers/docs/Developer-Guide.md at main · kata-containers/kata-containers

包安装方法

Installation method	Description	Automatic updates	Use case
Using official distro packages	Kata packages provided by Linux distributions official repositories	yes	Recommended for most users.
Automatic	Run a single command to install a full system	**No!**	For those wanting the latest release quickly.
Using kata-deploy	The preferred way to deploy the Kata Containers distributed binaries on a Kubernetes cluster	**No!**	Best way to give it a try on kata-containers on an already up and running Kubernetes cluster.

Kata Deploy安装

kata-containers/tools/packaging/kata-deploy/README.md at main · kata-containers/kata-containers

```
Kata Deploy 提供了一个 Dockerfile，其中包含运行 Kata 容器所需的所有二进制文件和工件，以及参考 DaemonSet，可用于在正在运行的 Kubernetes 集群上安

使用 Kata Deploy 在 Kubernetes 集群上安装 Kata 容器。
```

注意：

`仅当节点使用 containerd 或 CRI-O CRI-shims 时，通过 DaemonSets 的安装才会成功在节点上安装 katacontainers.io/kata-runtime。`

Vanilla Kubernetes cluster安装

安装最新版：

```
$ kubectl apply -f https://raw.githubusercontent.com/kata-containers/kata-containers/main/tools/packaging/kata-deploy/kata-rbac/base/kata-rbac.yaml
$ kubectl apply -f https://raw.githubusercontent.com/kata-containers/kata-containers/main/tools/packaging/kata-deploy/kata-deploy/base/kata-deploy.yaml
```

安装稳定版：

```
$ kubectl apply -f https://raw.githubusercontent.com/kata-containers/kata-containers/main/tools/packaging/kata-deploy/kata-rbac/base/kata-rbac.yaml
$ kubectl apply -f https://raw.githubusercontent.com/kata-containers/kata-containers/main/tools/packaging/kata-deploy/kata-deploy/base/kata-deploy-stable.yaml
```

确认kata-deploy已就绪：

```
$ kubectl -n kube-system wait --timeout=10m --for=condition=Ready -l name=kata-deploy pod
```

运行工作负载

```
工作负载通过在 Pod 规范中设置适当的 runtimeClass 对象来指定它们想要使用的运行时。提供的 runtimeClass 示例定义了一个节点选择器来匹配节点标签 ka

runtimeClass 是 Kubernetes 中的内置类型。要应用每个 Kata Containers 运行时类：

$ kubectl apply -f https://raw.githubusercontent.com/kata-containers/kata-containers/main/tools/packaging/kata-deploy/runtimeclasses/kata-runtimeClasses.yaml
```

`以下 YAML 片段显示了如何指定工作负载应将 Kata 与 QEMU 结合使用：`

```
spec:
  template:
    spec:
      runtimeClassName: kata-qemu
```

通过kata-qemu运行工作负载：

```
$ kubectl apply -f https://raw.githubusercontent.com/kata-containers/kata-containers/main/tools/packaging/kata-deploy/examples/test-deploy-kata-qemu.yaml
```

移除工作负载：

```
$ kubectl delete -f https://raw.githubusercontent.com/kata-containers/kata-containers/main/tools/packaging/kata-deploy/examples/test-deploy-kata-qemu.yaml
```

从kubernetes集群移除kata

以移除稳定版kata为例，最新版kata步骤类似：

1、删除kata-deploy

```
$ kubectl delete -f https://raw.githubusercontent.com/kata-containers/kata-containers/main/tools/packaging/kata-deploy/kata-deploy/base/kata-deploy-stable.yaml
$ kubectl -n kube-system wait --timeout=10m --for=delete -l name=kata-deploy pod
```

2、清理集群（清理节点标签等）

```
$ kubectl apply -f https://raw.githubusercontent.com/kata-containers/kata-containers/main/tools/packaging/kata-deploy/kata-cleanup/base/kata-cleanup-stable.yaml
```

3、删除cleanup daemonset，RBAC和runtime classes

```
$ kubectl delete -f https://raw.githubusercontent.com/kata-containers/kata-containers/main/tools/packaging/kata-deploy/kata-cleanup/base/kata-cleanup-stable.yaml
$ kubectl delete -f https://raw.githubusercontent.com/kata-containers/kata-containers/main/tools/packaging/kata-deploy/kata-rbac/base/kata-rbac.yaml
$ kubectl delete -f https://raw.githubusercontent.com/kata-containers/kata-containers/main/tools/packaging/kata-deploy/runtimeclasses/kata-runtimeClasses.yaml
```

kata-deploy细节

Dockerfile

`提供了用于创建部署在 DaemonSet 中的容器映像的 Dockerfile。该镜像包含运行 Kata Containers 所需的所有工件，所有这些工件均从 Kata Containers 发布页面（https://github.com/kata-containers/kata-containers/releases）中提取。`

宿主机工件：

- `cloud-hypervisor`，`firecracker`，`qemu`，`stratovirt` and supporting binaries
- `containerd-shim-kata-v2` (go runtime and rust runtime)
- `kata-collect-data.sh`
- `kata-runtime`

虚拟机工件：

- `kata-containers.img` and `kata-containers-initrd.img`: pulled from Kata GitHub releases page
- `vmlinuz.container` and `vmlinuz-virtiofs.container`: pulled from Kata GitHub releases page

DaemonSets and RBAC

`为 kata-deploy 引入了两个 DaemonSet，以及一个 RBAC 以方便将标签应用到节点。`

kata deploy:

`此 DaemonSet 在节点上安装必要的 Kata 二进制文件、配置文件和虚拟机工件。安装后，DaemonSet 会添加节点标签 katacontainers.io/kata-runtime=true 并重新配置 CRI-O 或 containerd 以注册三个运行时类：kata-clh（用于 Cloud Hypervisor 隔离）、kata-qemu（用于 QEMU 隔离）、 kata-fc（用于 Firecracker 隔离）和 kata-stratovirt（用于 StratoVirt 隔离）。最后一步，DaemonSet 重新启动 CRI-O 或 containerd。删除后，DaemonSet 会删除 Kata 二进制文件和 VM 工件，并将节点标签更新为 katacontainers.io/kata-runtime=cleanup。`

kata clean:

`节点运行的 DaemonSet 具有标签 katacontainers.io/kata-runtime=cleanup。这些 DaemonSet 会删除 katacontainers.io/kata-runtime 标签并重新启动 CRI-O 或 containerd systemctl 守护进程。您无法在 Kata 安装程序 DaemonSet 的 preStopHook 期间执行这些重置，这需要最终清理 DaemonSet。`

更多信息

升级文档：kata-containers/docs/Upgrading.md at main · kata-containers/kata-containers

Kata Containers 3.0 rust运行时安装：kata-containers/docs/install/kata-containers-3.0-rust-runtime-installation-guide.md at main · kata-containers/kata-containers

操作

kata-containers/docs/how-to at main · kata-containers/kata-containers

场景

kata-containers/docs/use-cases/GPU-passthrough-and-Kata.md at main · kata-containers/kata-containers

kata-containers/docs/use-cases/using-SRIOV-and-kata.md at main · kata-containers/kata-containers

kata-containers/docs/use-cases/using-Intel-QAT-and-kata.md at main · kata-containers/kata-containers

kata-containers/docs/use-cases/using-SPDK-vhostuser-and-kata.md at main · kata-containers/kata-containers

kata-containers/docs/use-cases/using-Intel-SGX-and-kata.md at main · kata-containers/kata-containers

开发

`Kata Containers` 的整个端到端流程：kata-containers/docs/design/end-to-end-flow.md at main · kata-containers/kata-containers

Kata Containers设计细节：kata-containers/docs/design/README.md at main · kata-containers/kata-containers

Kata Containers风险模型：kata-containers/docs/threat-model/threat-model.md at main · kata-containers/kata-containers

Kata Containers开发指南：kata-containers/docs/Developer-Guide.md at main · kata-containers/kata-containers

Kata Containers编码：kata-containers/docs/code-pr-advice.md at main · kata-containers/kata-containers kata-containers/docs/Unit-Test-Advice.md at main · kata-containers/kata-containers

社区

GitHub - kata-containers/community

贡献指南：community/CONTRIBUTING.md at main · kata-containers/community

邮件列表：http://lists.katacontainers.io

Slack：https://join.slack.com/t/katacontainers/shared_invite/zt-16w1u6usn-sK871qbMxVN8KsCP5Gr56A

IRC：Kata Containers - Open Source Container Runtime Software

会议：https://etherpad.opendev.org/p/Kata_Containers_Architecture_Committee_Mtgs

参考文献

Kata Containers

Kata Containers - Open Source Container Runtime Software

kata-containers/docs/install at main · kata-containers/kata-containers

kata-containers/docs at main · kata-containers/kata-containers