阿里云 × CLOUD NATIVE COMPUTING FOUNDATION

云原生技术公开课

第 15 讲

深入解析Linux容器

华敏　阿里巴巴技术专家

1 资源隔离和限制

2 容器镜像

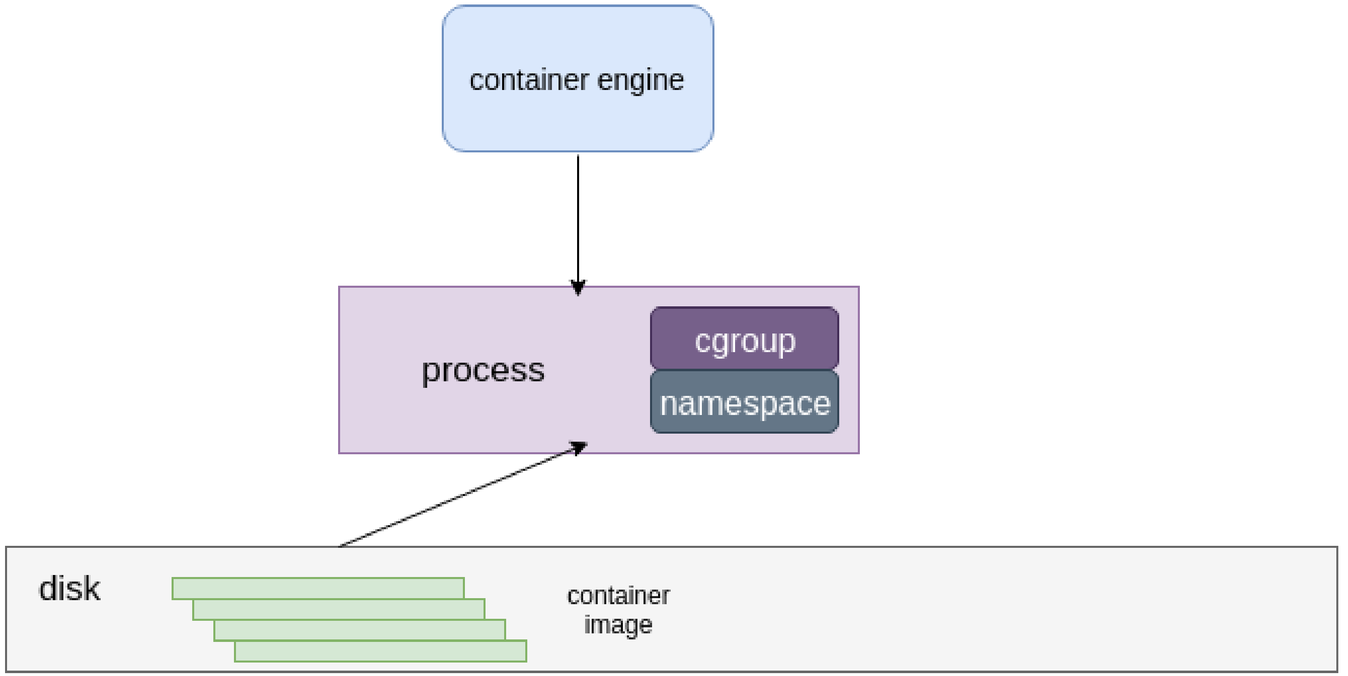
3 容器引擎

4 总结

容器

以docker为例

cgroup + namespace + docker image



namespace

1. mount
2. uts
3. pid
4. network
5. user
6. ipc
7. cgroup

disable cgroup namespace

```
$ docker exec 85d9619a0b6c cat /proc/self/cgroup
12:devices:/docker/85d9619a0b6c48ca251e62ec175911b23572b3b4c14f0c25837904c2431ed186
11:pids:/docker/85d9619a0b6c48ca251e62ec175911b23572b3b4c14f0c25837904c2431ed186
10:freezer:/docker/85d9619a0b6c48ca251e62ec175911b23572b3b4c14f0c25837904c2431ed186
9:perf_event:/docker/85d9619a0b6c48ca251e62ec175911b23572b3b4c14f0c25837904c2431ed186
8:blkio:/docker/85d9619a0b6c48ca251e62ec175911b23572b3b4c14f0c25837904c2431ed186
7:cpuset:/docker/85d9619a0b6c48ca251e62ec175911b23572b3b4c14f0c25837904c2431ed186
6:memory:/docker/85d9619a0b6c48ca251e62ec175911b23572b3b4c14f0c25837904c2431ed186
5:net_cls,net_prio:/docker/85d9619a0b6c48ca251e62ec175911b23572b3b4c14f0c25837904c2431ed186
4:rdma:/
3:cpu,cpuacct:/docker/85d9619a0b6c48ca251e62ec175911b23572b3b4c14f0c25837904c2431ed186
2:hugetlb:/docker/85d9619a0b6c48ca251e62ec175911b23572b3b4c14f0c25837904c2431ed186
1:name=systemd:/docker/85d9619a0b6c48ca251e62ec175911b23572b3b4c14f0c25837904c2431ed186
```

enable cgroup namespace

```
11:devices:/
10:memory:/
9:hugetlb:/
8:pids:/
7:net_cls:/
6:freezer:/
5:perf_event:/
4:cpuset,cpu,cpuacct:/
3:net_cgroup:/
2:blkio:/
1:name=systemd:/
```

cgroup

2种 cgroup 驱动：

- systemd cgroup driver
- cgroupfs cgroup driver

容器中常用的 cgroup：

- cpu cpuset cpuacct
- memory
- device
- freezer
- blkio
- pid

不常用的 cgroup：

- net_cls
- net_prio
- hugetlb
- perf_event
- rdma

namespace 示例

man unshare

```
EXAMPLES
       # unshare --fork --pid --mount-proc readlink /proc/self
       1
              Establish a PID namespace, ensure we're PID 1 in it against newly mounted procfs instance.

       $ unshare --map-root-user --user sh -c whoami
       root
              Establish a user namespace as an unprivileged user with a root user within it.

       # touch /root/uts-ns
       # unshare --uts=/root/uts-ns hostanme FOO
       # nsenter --uts=/root/uts-ns hostname
       FOO
```

unshare使用

```
$ sudo unshare --mount-proc --pid --fork /bin/bash
root@www:~  time-> 二 8月 13 15:37:40
$ ps -ef
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 15:37 pts/28   00:00:00 /bin/bash
root          14     1  0 15:37 pts/28   00:00:00 ps -ef
```

1 资源隔离和限制

2 容器镜像

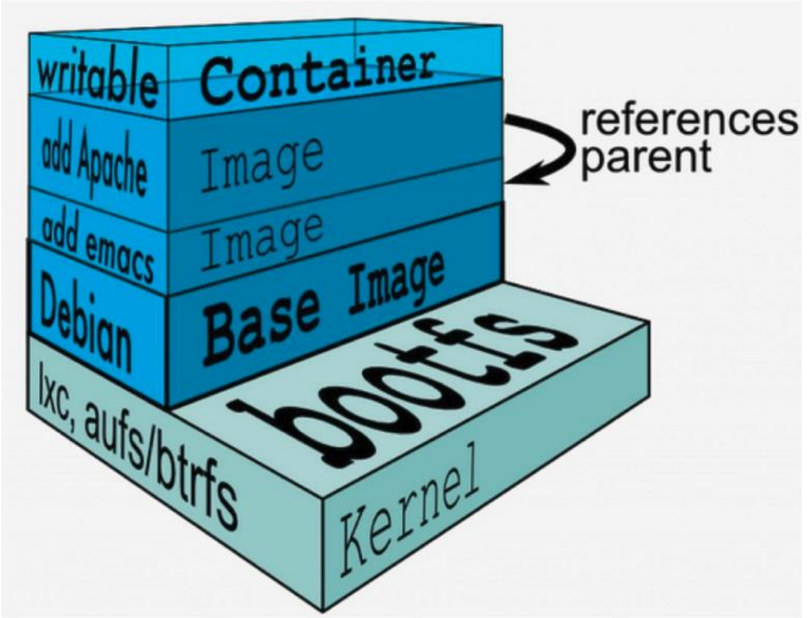
3 容器引擎

4 总结

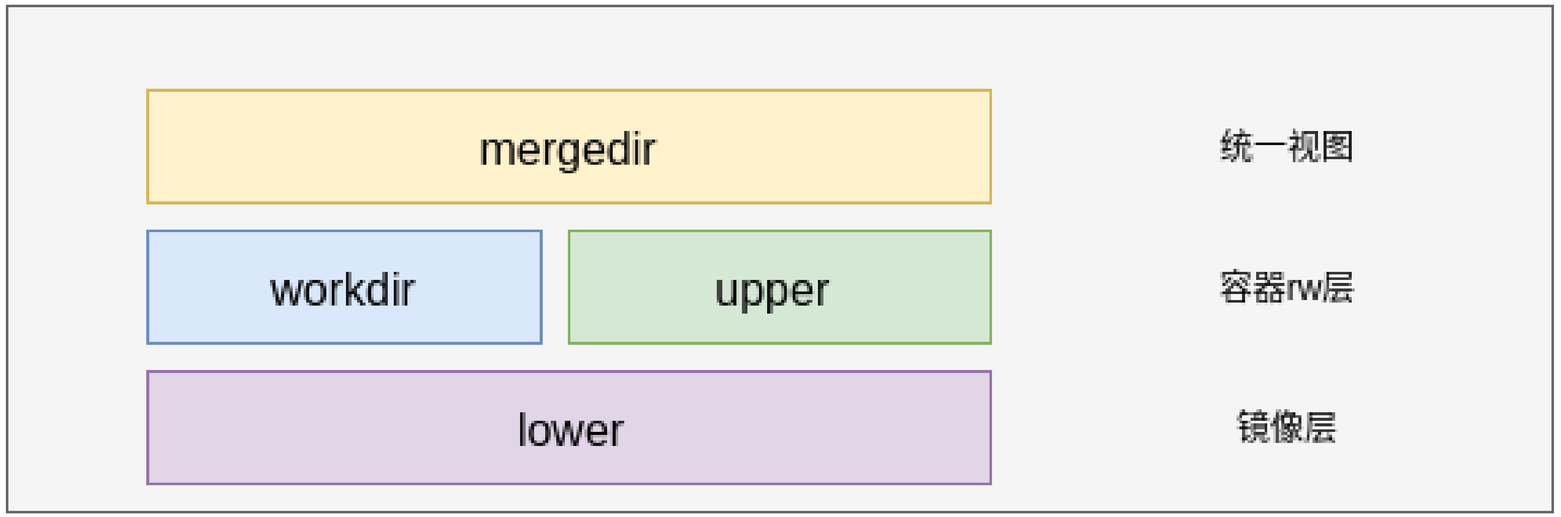
docker images

docker images:

- 基于联合文件系统
- 不同的层可以被其他镜像复用
- 容器的可写层可以做成镜像新的一层



以overlay为例



- merged：整合了lower 层和upper 读写层显示出来的视图
- upper：容器读写层
- workdir：类似中间层，对upper层的写入，先写入workdir，再移入upper层
- lower：镜像层

文件操作

- 读： 如果upper层没有副本，数据都从lower读上来
- 写： 容器创建出来时，upper层是空的，只有对文件进行写操作时，才会从lower层拷贝文件上来，对副本进行操作
- 删： 删操操作不影响lower层，删除操作通过对文件进行标记，使文件无法显示。有2种方式，whiteout 和设置目录的xattr “trusted.overlay.opaque”=y

操作步骤

1. 启动一个容器

```
#docker run -d busybox top
4efceba272982940c402f16d84a0f4341a310348079f0636ad3d88a4a5324d43
```

2. 查看容器rootfs的挂载点

```
#mount -l | grep 62c1e0209a4a41f0a4becc8d6ac67147cd3d4abd6f3e39bb08884ae7428b199e
overlay on /home/t4/docker/overlay2/62c1e0209a4a41f0a4becc8d6ac67147cd3d4abd6f3e39bb08884ae7428b199e/merged type overlay (rw,relatime,lowerdir=l/5SXNKIWOA5E2EKH2JHN5FNCR3Y:l/P5RLCC6KDEKFTYBI
IXVJ4R6RH6,upperdir=/home/t4/docker/overlay2/62c1e0209a4a41f0a4becc8d6ac67147cd3d4abd6f3e39bb08884ae7428b199e/diff,workdir=/home/t4/docker/overlay2/62c1e0209a4a41f0a4becc8d6ac67147cd3d4abd6f
3e39bb08884ae7428b199e/work)
```

3. 查看新文件写入

```
#docker exec 4efceba27298 sh -c 'echo "new_file" > /new-file'

[root@r10e19288.sqa.zmf /root]
#cat /home/t4/docker/overlay2/62c1e0209a4a41f0a4becc8d6ac67147cd3d4abd6f3e39bb08884ae7428b199e/diff/new-file
new_file

[root@r10e19288.sqa.zmf /root]
#cat /home/t4/docker/overlay2/62c1e0209a4a41f0a4becc8d6ac67147cd3d4abd6f3e39bb08884ae7428b199e/merged/new-file
new_file
```

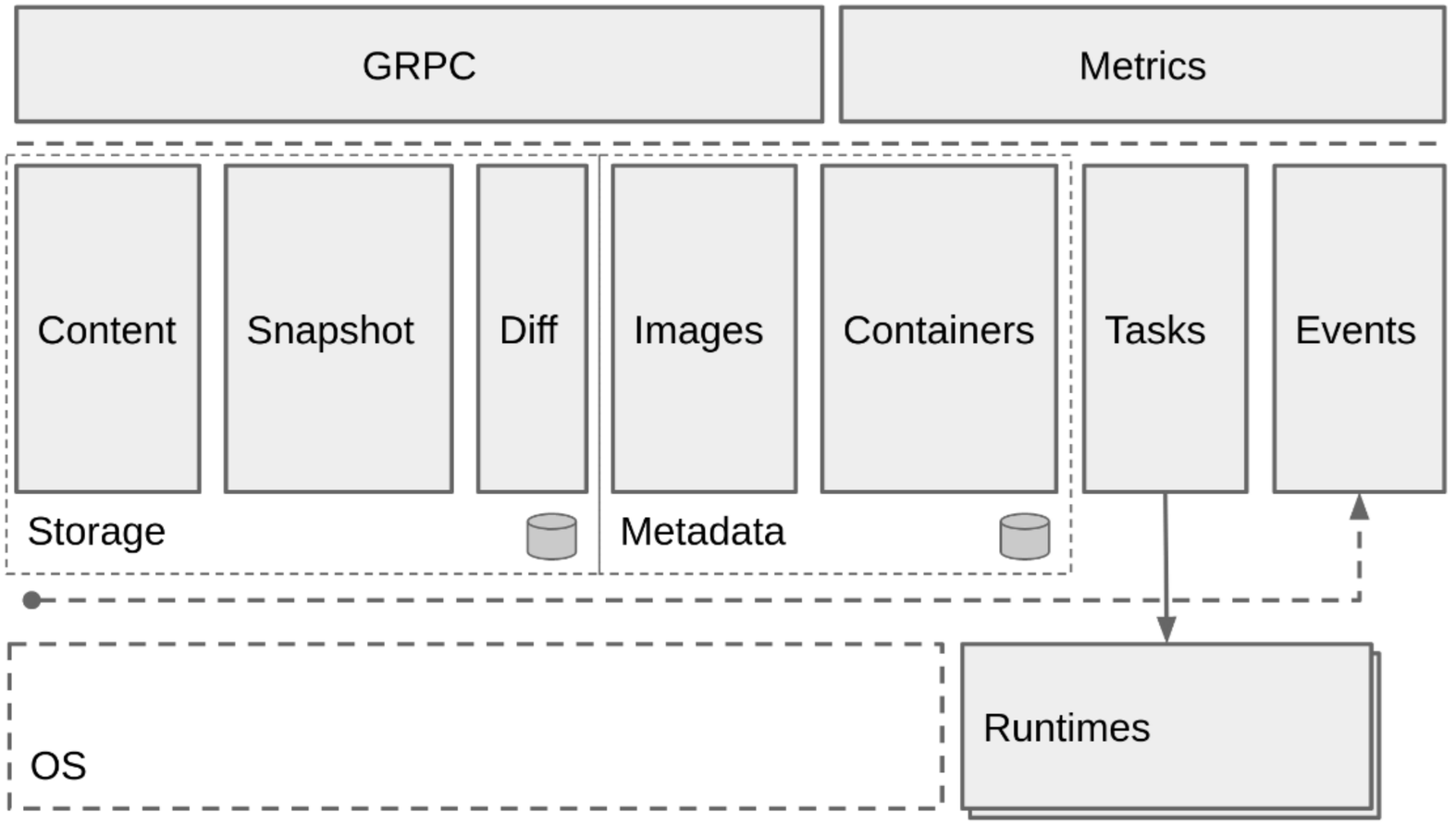
1 资源隔离和限制

2 容器镜像

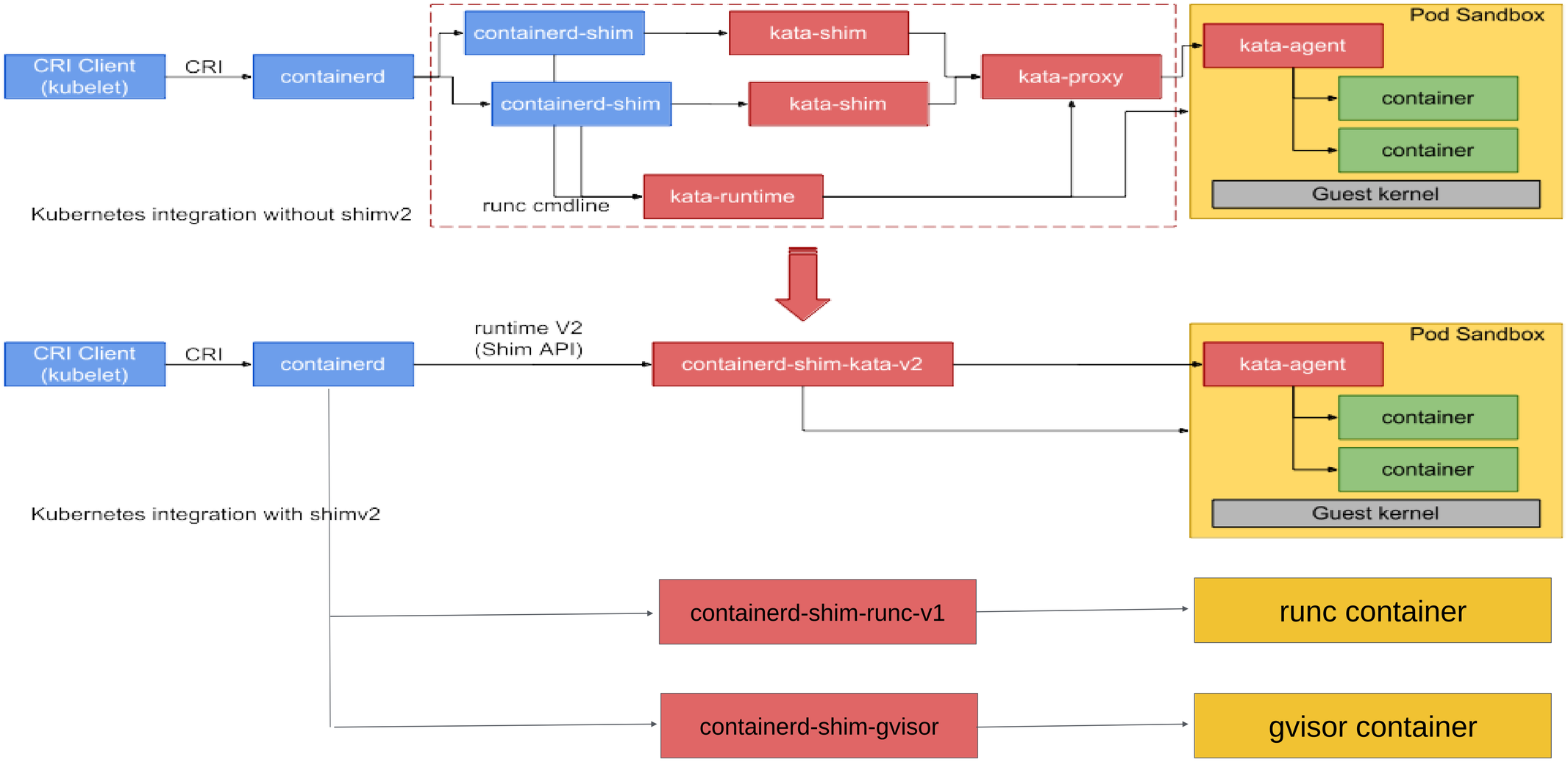
3 容器引擎

4 总结

containerd 容器架构详解



shim v1/v2 是什么

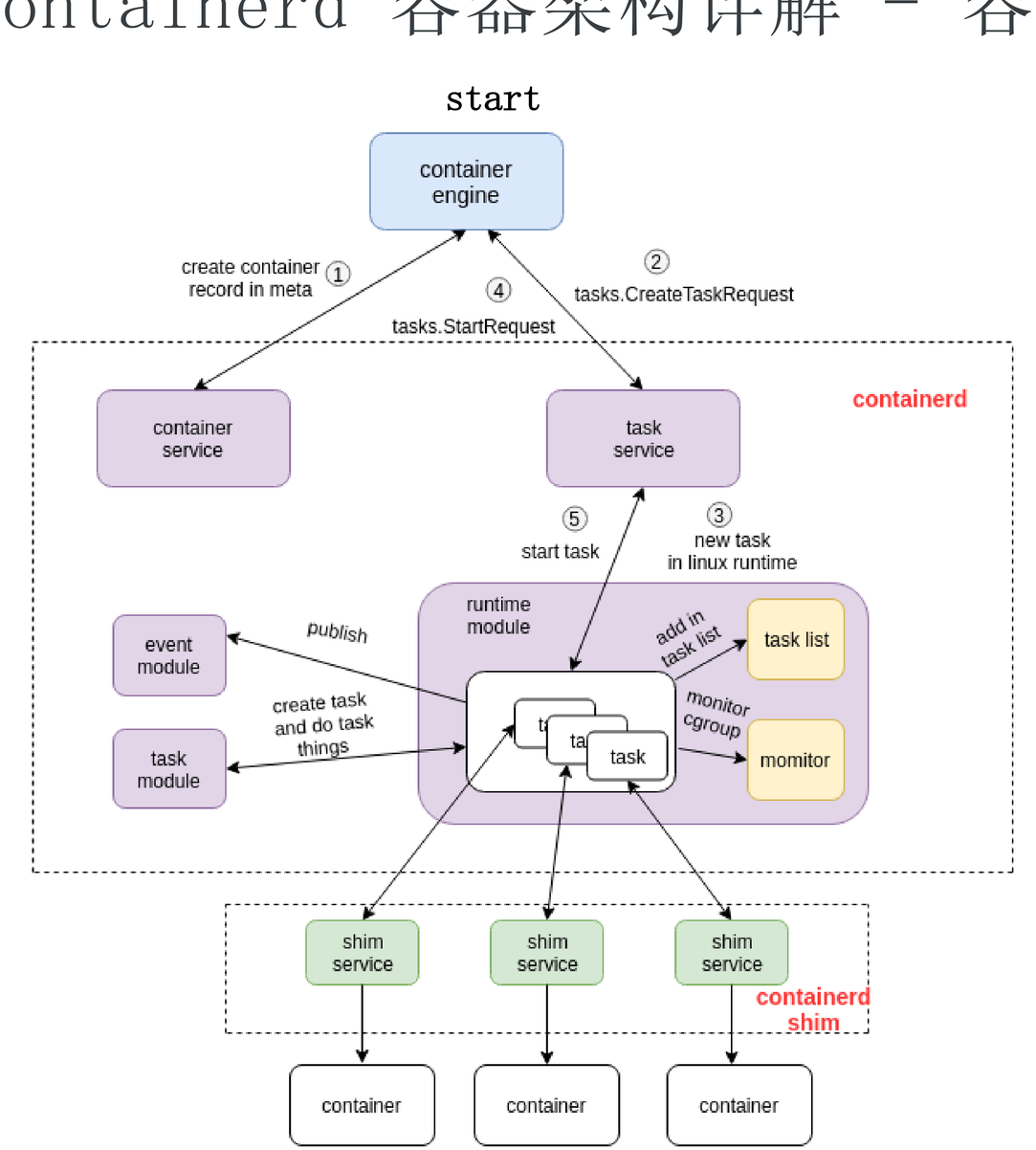


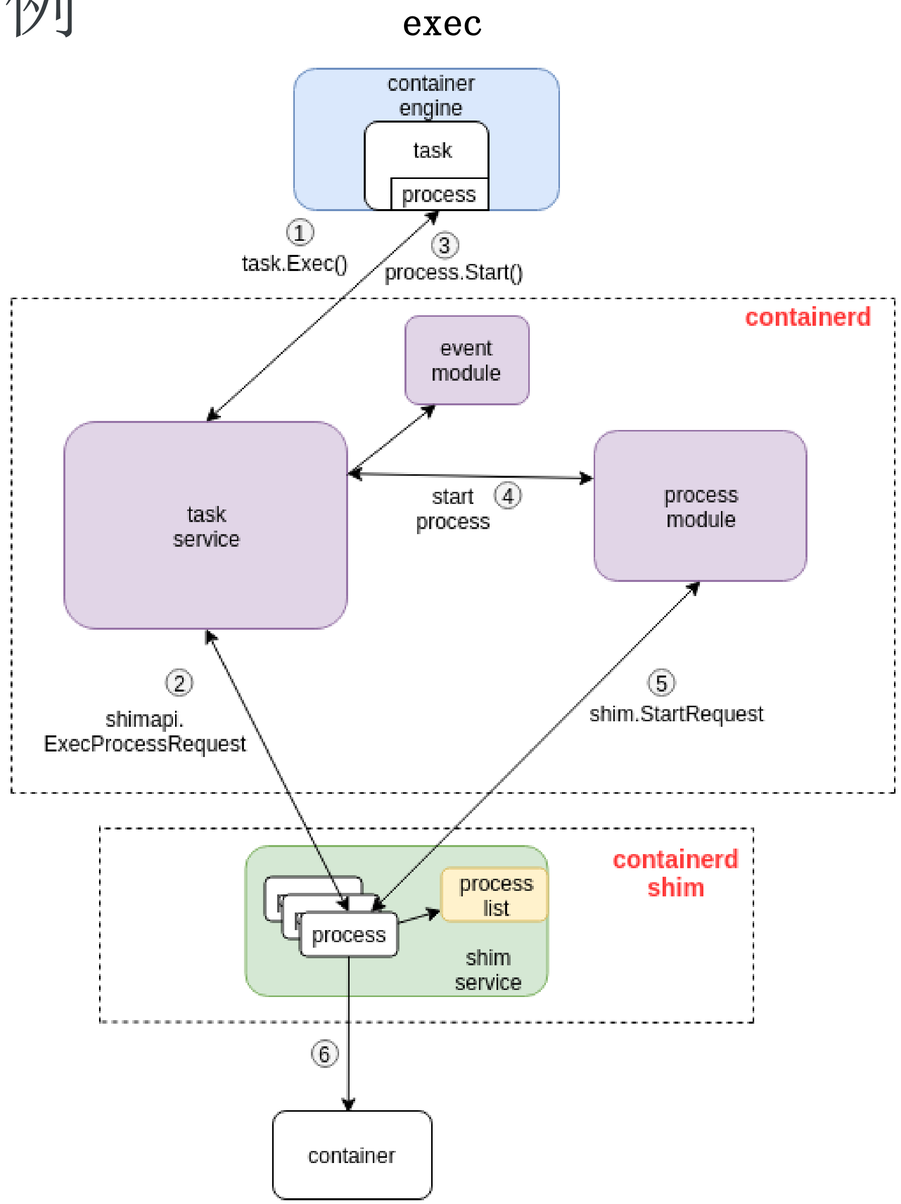
OCI 是什么

An open governance structure for the express purpose of creating open industry standards around container formats and runtime.

- oci runtime 标准：https://github.com/opencontainers/runtime-spec
- oci image 标准：https://github.com/opencontainers/image-spec
- distribution 标准：https://github.com/opencontainers/distribution-spec

containerd 容器架构详解 - 容器流程示例





1 资源隔离和限制

2 容器镜像

3 容器引擎

4 总结

关注“阿里巴巴云原生”公众号
获取第一手技术资料