cloudflight

kubernetes, docker, cri/-o, containerd, OCI, runC, container

what they are and how they fit together

17. Cloud Native Computing Linz Meetup - October 18, 2022

Martin Strigl

cloudflight

Ehlo!

Martin Strigl

Master of disaster (internal IT + SRE)
Not afraid of touching php
Using linux container technics since 2010

Why this talk ?

- I was and am triggered by someone saying docker when they should say container
- At some point realized that it’s easier when you grow up with stuff and not have to start with a given full blown ecosystem

Picture Source: https://media.ed.edmunds-media.com/mercedes-benz/ns/mercedes-benz_r34_ns_80122_717.jpg

Picture Source: https://web-cdn.rimac-automobili.com/wp-content/uploads/2021/05/31101409/intro_slider_08_optimised-1600x900.jpg

cloudflight

Topics covered

01

A short lesson in History

02

(linux) containers at a low level

03

OCI and runtimes (runc, crun)

04

Container runtime Interface (containerd, cri-o)

05

Container engines (docker, podman)

06

Container Orchestration (Kubernetes)

A short lesson in History

- **1979 – chroot**
  - During development of Unix V7 the syscall chroot is added, thus enabling changing the root dir of a process and its children => beginning of process isolation
- **1982 - chroot for BSD**
  - Chroot is added to BSD
- **2000 - FreeBSD Jails**
  - Jails allows partitioning of a FreeBSD system into several independent, smaller systems – called “jails” – with the ability to assign an IP address for each system and configuration.
- **2001 - Linux vServer (Jacques Gélinas)**
  - jail mechanism that can partition resources (file systems, network addresses, memory)
  - Patched kernel needed
- **2004 - Solaris Container**
  - First public beta which combines system resource controls and boundary separation provided by zones
- **2005 – OpenVZ/Virtuozzo (SWSoft/Parallels)**
  - OS level virtualization by using isolation, resource management and checkpointing
  - Patches for kernel
  - PID, IPC, and network namespaces contribution 08/09
- **2006 - Process Containers (Paul Menage / Google)**
  - limiting, accounting and isolating resource usage (CPU, memory, disk I/O, network) of a collection of processes
  - renamed to cgroups and upstream 2.6.24
- **2008 – LXC (engineers from IBM)**
  - using cgroups and namespaces
  - works on a single Linux kernel without requiring any patches
- **2013 – Docker**
  - Initial LXC - replaced later by libcontainer
  - Largest benefits:
    - Ecosystem
    - Container and Image Management

A short lesson in History

- **2015 - The year of OCI, Kube v1.0 & CNCF**
  - June: Open Container Initiative is established
  - July 21: Kubernetes v1.0 gets released and google gets part of CNCF
  - November 9-11: KubeCon 2015 is the first inaugural community Kubernetes conf
- **2016 - Security**
  - Container Security gets more focus – dirty COW
- **2017 – Adoption of containerd and rkt by CNCF**
  - Docker donates containerd to CNCF
  - Rkt gets adopted by CNCF
- **2018 – Hybrid techs show up**
  - Hybrid techs that combine vm-like isolation with container stuff strat to evolve
  - Kata, gvisor, nabla

- **Links**
  - https://www.redhat.com/en/blog/history-containers
  - https://wiki.openvz.org/History
  - https://www.kernel.org/doc/ols/2007/ols2007v2-pages-45-58.pdf
  - https://www.zdnet.com/article/open-container-project-how-cloud-giants-are-joining-forces-against-lock-in-and-fragmentation/
  - https://opencontainers.org/
  - https://github.com/opencontainers/runtime-spec

cloudflight

Topics covered

01

A short lesson in History

02

(linux) containers at a low level

03

OCI and runtimes (runc, crun)

04

Container runtime Interface (containerd, cri-o)

05

Container engines (docker, podman)

06

Container Orchestration (Kubernetes)

(linux) containers at a low level

- **Namespaces**
  - “Namespaces are a feature of the Linux kernel that partitions kernel resources such that one set of processes sees one set of resources while another set of processes sees a different set of resources.” (Wikipedia)
  - Can be instrumented on cli by using unshare
  - Listing (since kernel 5.6)
    - Process ID (pid)
    - Mount (mnt)
    - Network (net)
    - Interprocess Communication (ipc)
    - Unix Time Sharing (UTS)
    - User ID (user)
    - Control Group (cgroup)
    - Time (time)

- **cgroups**
  - “cgroups (abbreviated from control groups) is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes” (Wikipedia)
  - Can be instrumented on cli by (lib)cgroup-tools
  - Features
    - Resource limiting
      - groups can be set to not exceed a configured memory limit, which also includes the file system cache, I/O bandwidth limit, CPU quota limit, or CPU set limit.
    - Prioritization
      - some groups may get a larger share of CPU utilization or disk I/O throughput
    - Accounting
      - measures a group's resource usage, which may be used, for example, for billing purposes
    - Control
      - freezing groups of processes, their checkpointing and restarting

(linux) containers at a low level

- **Bash runtime**
  - Minimal example => https://github.com/0n1shi/runb/blob/master/runb.sh
  - Excerpt of relevant steps
    - Network
      - ip netns add $CONTAINER_NET_NS && ip link add name $VETH type veth peer name $ETH
      - brctl addif $BRIDGE_NAME $VETH && ip link set $VETH up && ip link set $ETH netns $CONTAINER_NET_NS
      - ip netns exec $CONTAINER_NET_NS ip address add 10.0.0.2/24 dev $ETH
      - ip netns exec $CONTAINER_NET_NS ip link set $ETH up
      - ip netns exec $CONTAINER_NET_NS ip route add default via 10.0.0.1
    - Ressources
      - cgcreate -g "$CGROUP_CONTROLLERS:$CONTAINER_NAME"
      - cgclassify -g "$CGROUP_CONTROLLERS:$CONTAINER_NAME" $$
    - Mount
      - mkdir -p $ROOT_FS_DIR/proc && mount -t proc -o noexec,nosuid,nodev proc $ROOT_FS_DIR/proc
      - mknod -m 666 $ROOT_FS_DIR/dev/null c 1 3
      - touch $ROOT_FS_DIR/dev/pts/ptmx && mount --bind /dev/pts/ptmx $ROOT_FS_DIR/dev/pts/ptmx
    - Namespaces and startup
      - unshare --pid --uts --ipc --mount --cgroup --fork bash $CORE $CONTAINER_DIR $CONTAINER_NAME
      - exec ip netns exec $CONTAINER_NET_NS capsh --inh="$SET_CAPS" --drop="$DROP_CAPS" --uid="$UID" --gid="$GUID" --chroot="$ROOT_FS_DIR" -- -c "/bin/bash"

cloudflight

Topics covered

01

A short lesson in History

02

(linux) containers at a low level

03

OCI and runtimes (runc, crun)

04

Container runtime Interface (containerd, cri-o)

05

Container engines (docker, podman)

06

Container Orchestration (Kubernetes)

OCI and runtimes (runc, crun)

- **OCI – Open Container Initiative**
  - "Is an open governance structure for the express purpose of creating open industry standards around container formats and runtimes" (opencontainers.org)
  - Currently contains three specifications
    - Runtime Specification (runtime-spec)
      - Defines how a filesystem bundle has to look like
      - Defines how this filesystem bundle has to be run
      - Defines the lifecycles
      - https://github.com/opencontainers/runtime-spec/blob/main/spec.md
    - Image Specification (image-spec)
      - The OCI Image Format Specification strictly defines the requirements for an OCI Image (container image), which consists of a manifest, an optional image index, a set of filesystem layers, and a configuration. The schema for OCI Image components is fully supported by the APIs defined in the OCI Distribution Specification.
      - https://github.com/opencontainers/image-spec/blob/main/spec.md
    - Distribution Specification (distribution-spec)
      - The OCI Distribution Specification is also designed generically enough to be leveraged as a distribution mechanism for any type of content. The format of uploaded manifests, for example, need not necessarily adhere to the OCI Image Format Specification so long as it references the blobs which comprise a given artifact.
      - https://github.com/opencontainers/distribution-spec/blob/main/spec.md

OCI and runtimes (runc, crun)

- **OCI Runtimes**
  - runc
    - runc is a CLI tool for spawning and running containers on Linux according to the OCI specification.
    - Long time the default OCI runtime for spawning containers
    - Evolved from docker's work on libcontainer and the OCI
    - Written in go
    - https://github.com/opencontainers/runc
  - crun
    - A fast and low-memory footprint OCI Container Runtime fully written in C
    - One of the first runtimes that supported cgroupsv2
    - Part of the broader containers project driven mainly by redhat
    - is faster than runc and has a much lower memory footprint
    - https://github.com/containers/crun
  - rkt (deprecated)
    - was second runtime after docker which was utilized by k8s machine daemon kubelet – rktlet
  - railcar (deprecated)
    - OCI runtime implemented by oracle
    - Written in rust

OCI and runtimes (runc, crun)

- **Live Demo**
  - Running
    - mkdir "${MYNAME}"
    - cd "${MYNAME}"
    - mkdir rootfs && podman export $(podman create busybox) | tar -C rootfs -xf -
    - ${MYCRT} spec --rootless
    - less config.json
    - ${MYCRT} run ${MYNAME}_container
  - Showing Namespaces and Cgroups
    - MYCONTPID=$(${MYCRT} list -f json | jq -r '.[] | select(.id| startswith("'${MYNAME}'_container")) | .pid')
    - lsns -p $MYCONTPID
    - MYCONTCG=$(cat /proc/${MYCONTPID}/cgroup | sed 's/^[0-9]*::/g')
    - cat /sys/fs/cgroup${MYCONTCG}/cgroup.controllers
    - cat /sys/fs/cgroup${MYCONTCG}/memory.max

cloudflight

Topics covered

01

A short lesson in History

02

(linux) containers at a low level

03

OCI and runtimes (runc, crun)

04

Container runtime Interface (containerd, cri-o)

05

Container engines (docker, podman)

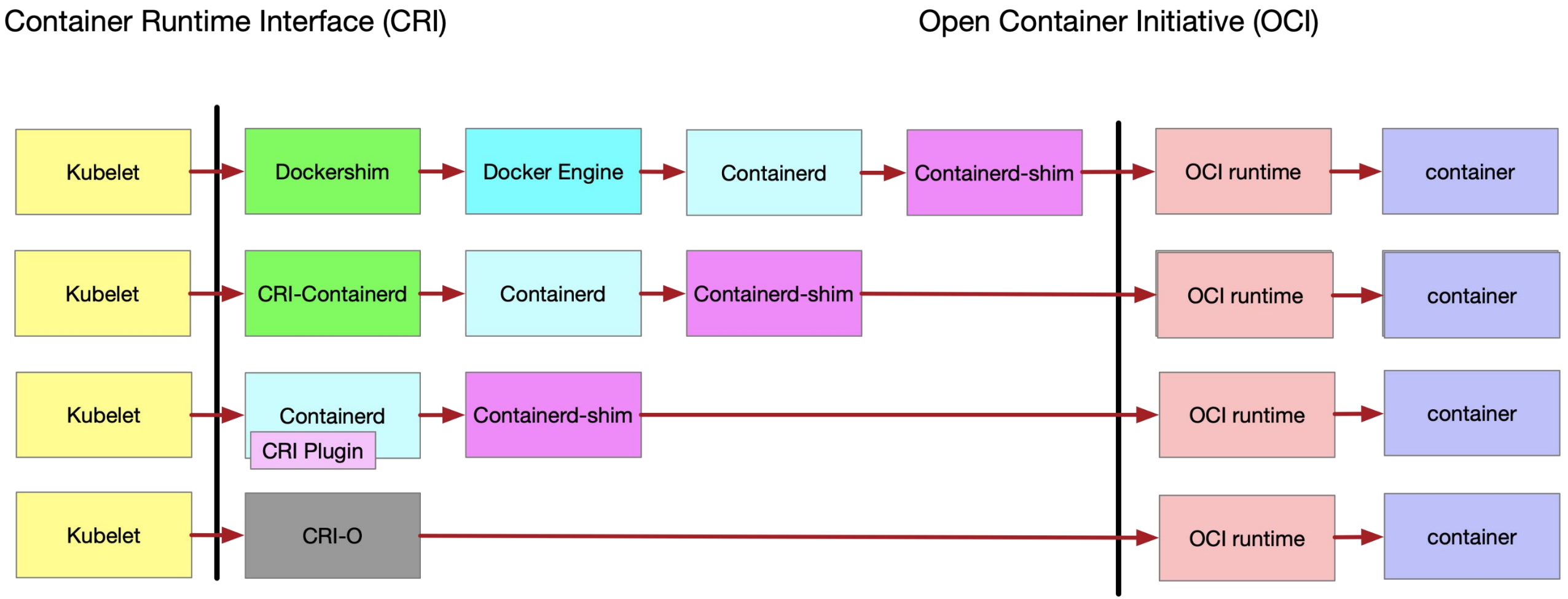
06

Container Orchestration (Kubernetes)

Container runtime Interface (containerd, cri-o)

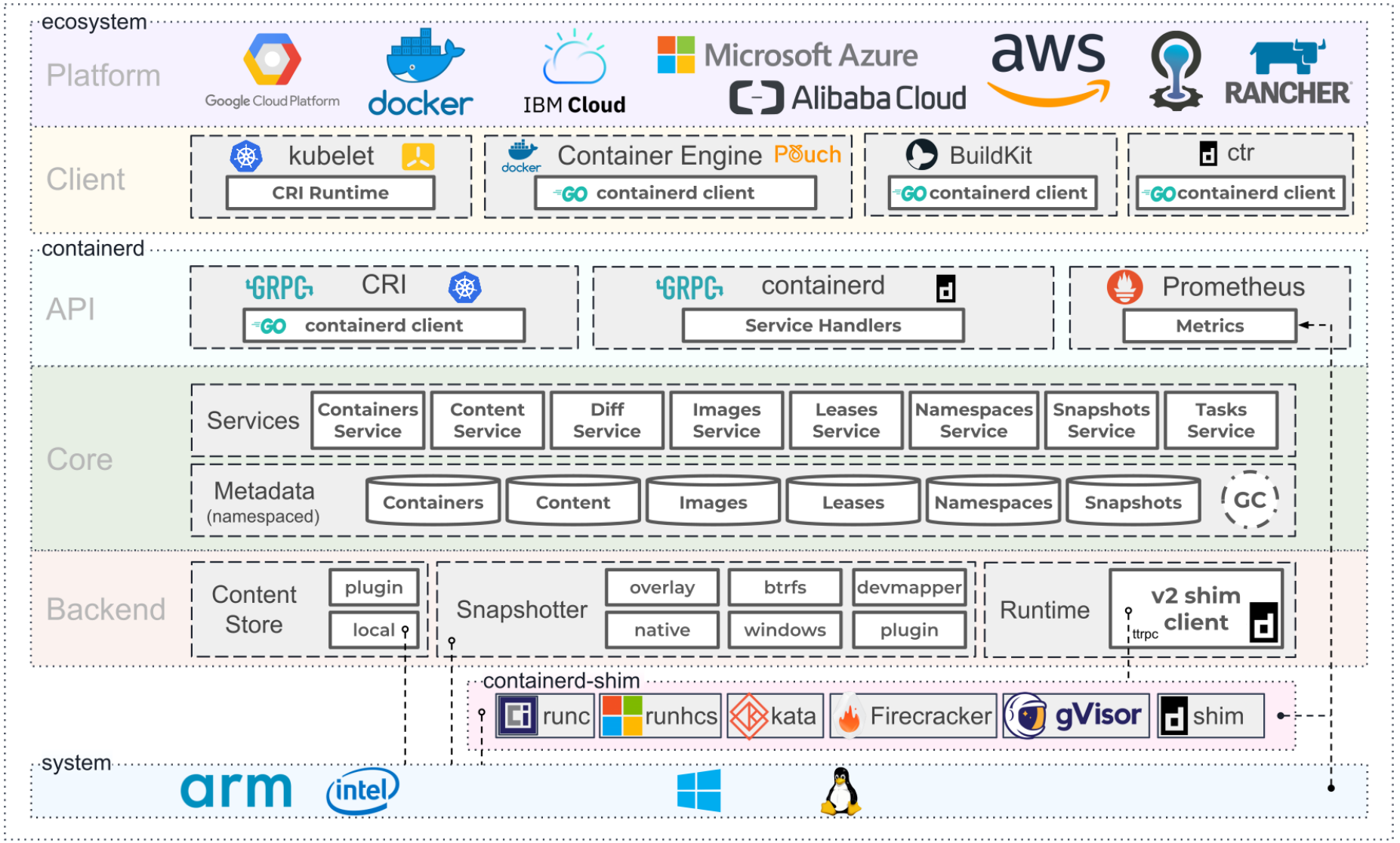
- **CRI – Kubernetes API**
  - "The CRI is a plugin interface which enables the kubelet to use a wide variety of container runtimes, without having a need to recompile the cluster components." (https://kubernetes.io/docs/concepts/architecture/cri/)
  - You need a working container runtime on each Node in your cluster, so that the kubelet can launch Pods and their containers
  - The Container Runtime Interface (CRI) is the main protocol for the communication between the kubelet and Container Runtime
  - The Kubernetes Container Runtime Interface (CRI) defines the main gRPC protocol for the communication between the cluster components kubelet and container runtime
  - https://github.com/kubernetes/cri-api/blob/c75ef5b/pkg/apis/runtime/v1/api.proto
  - Resolved the constraint that the container runtime (docker/rkt) had to be embedded in kubelet
  - Two common incarnations
    - containerd
      - Multi purpose, one tool fits (almost) all usecases
    - cri-o
      - Specialized for kubernetes

Container runtime Interface (containerd, cri-o)



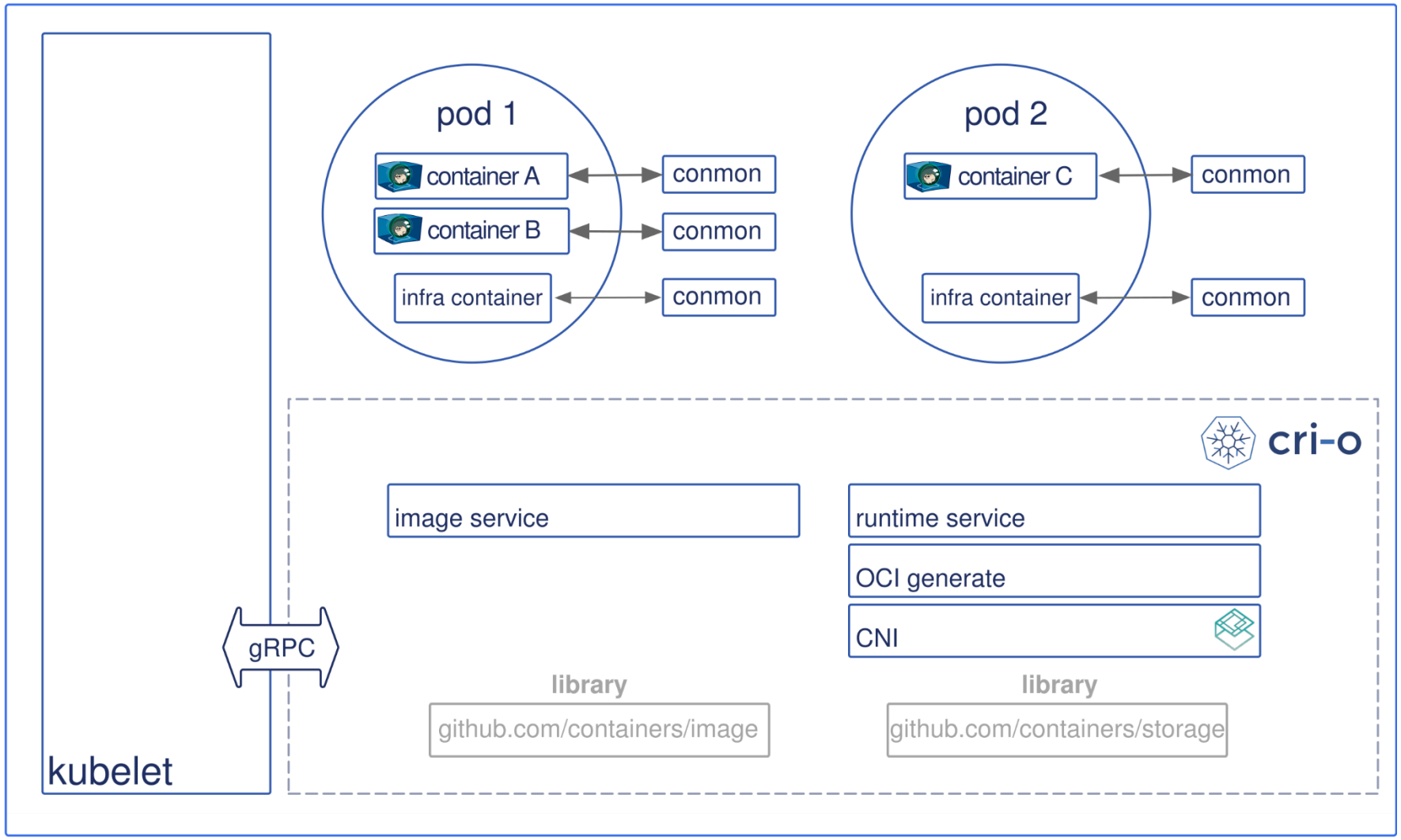
Picture Source: https://i.imgur.com/nUAYZiq.png

Container runtime Interface (containerd, cri-o)



Picture Source: https://containerd.io/img/architecture.png

Container runtime Interface (containerd, cri-o)



- Kubernetes contacts the kubelet to launch a pod.
- The kubelet forwards the request to the CRI-O daemon VIA kubernetes CRI (Container runtime interface) to launch the new POD.
- CRI-O uses the containers/image library to pull the image from a container registry.
- The downloaded image is unpacked into the container's root filesystems, stored in COW file systems, using containers/storage library.
- After the rootfs has been created for the container, CRI-O generates an OCI runtime specification json file describing how to run the container using the OCI Generate tools.
- CRI-O then launches an OCI Compatible Runtime using the specification to run the container proceses. The default OCI Runtime is runc.
- Each container is monitored by a separate conmon process. The conmon process holds the pty of the PID1 of the container process. It handles logging for the container and records the exit code for the container process.
- Networking for the pod is setup through use of CNI, so any CNI plugin can be used with CRI-O.saf

Picture Source: https://cri-o.io/assets/images/architecture.png

cloudflight

Topics covered

01

A short lesson in History

02

(linux) containers at a low level

03

OCI and runtimes (runc, crun)

04

Container runtime Interface (containerd, cri-o)

05

Container engines (docker, podman)

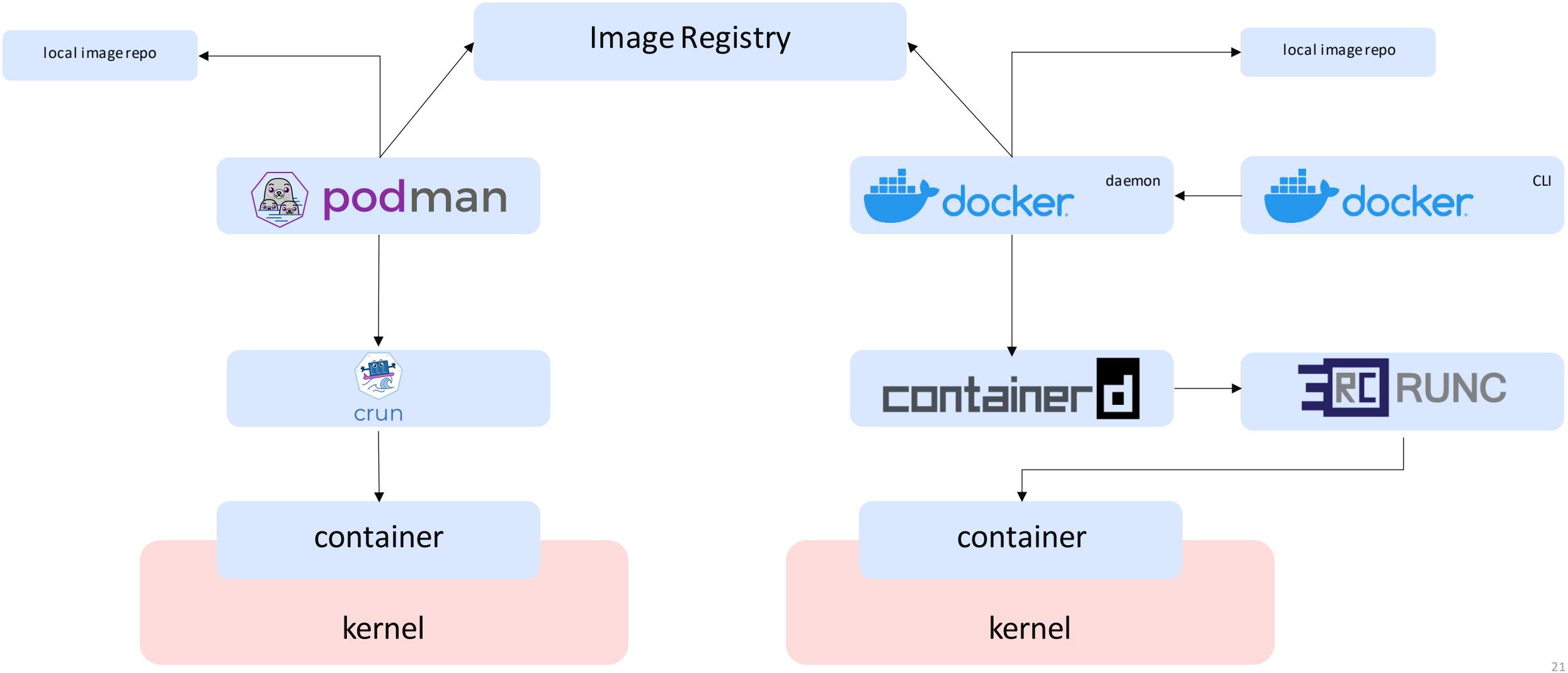
06

Container Orchestration (Kubernetes)

Container engines (docker, podman)

- **What is it**
  - You might have noticed by now that some areas were untouched up until now, like the following
    - Image Management
      - building
      - signing
      - Distribution
    - Storage Management
    - Network Management
  - A tool (or collection of tools) that covers these aspects too is nowadays called Container Engine
  - Docker provides these things as a full featured single executable
  - Other approaches instead follow more the unix philosophy of having more small tools which do one thing well, like e.g.:
    - podman – image running (whereby today it already covers a lot of other aspects too)
    - buildah – image building
    - skopeo – image distribution

Container engines (docker, podman)



cloudflight

Topics covered

01

A short lesson in History

02

(linux) containers at a low level

03

OCI and runtimes (runc, crun)

04

Container runtime Interface (containerd, cri-o)

05

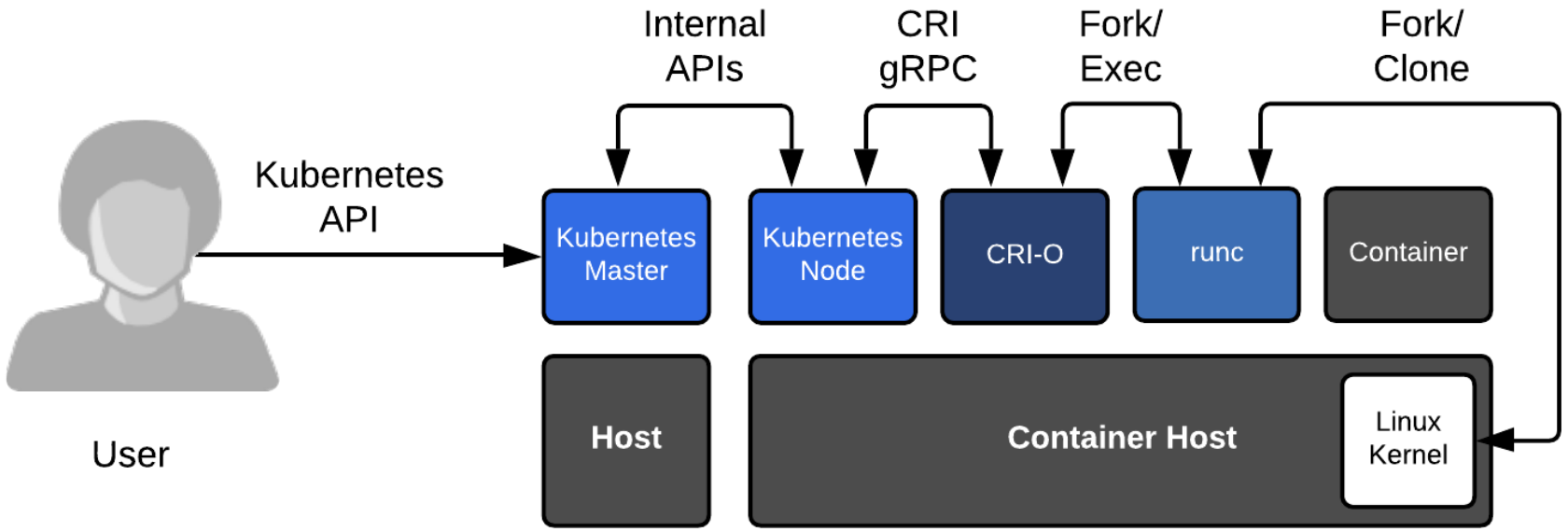
Container engines (docker, podman)

06

Container Orchestration (Kubernetes)

Container Orchestration (Kubernetes)

- **Putting it all together in a Kubernetes context**



How containers run in a Kubernetes cluster

Picture Source: https://www.redhat.com/cms/managed-files/image2_10_0.png

cloudflight

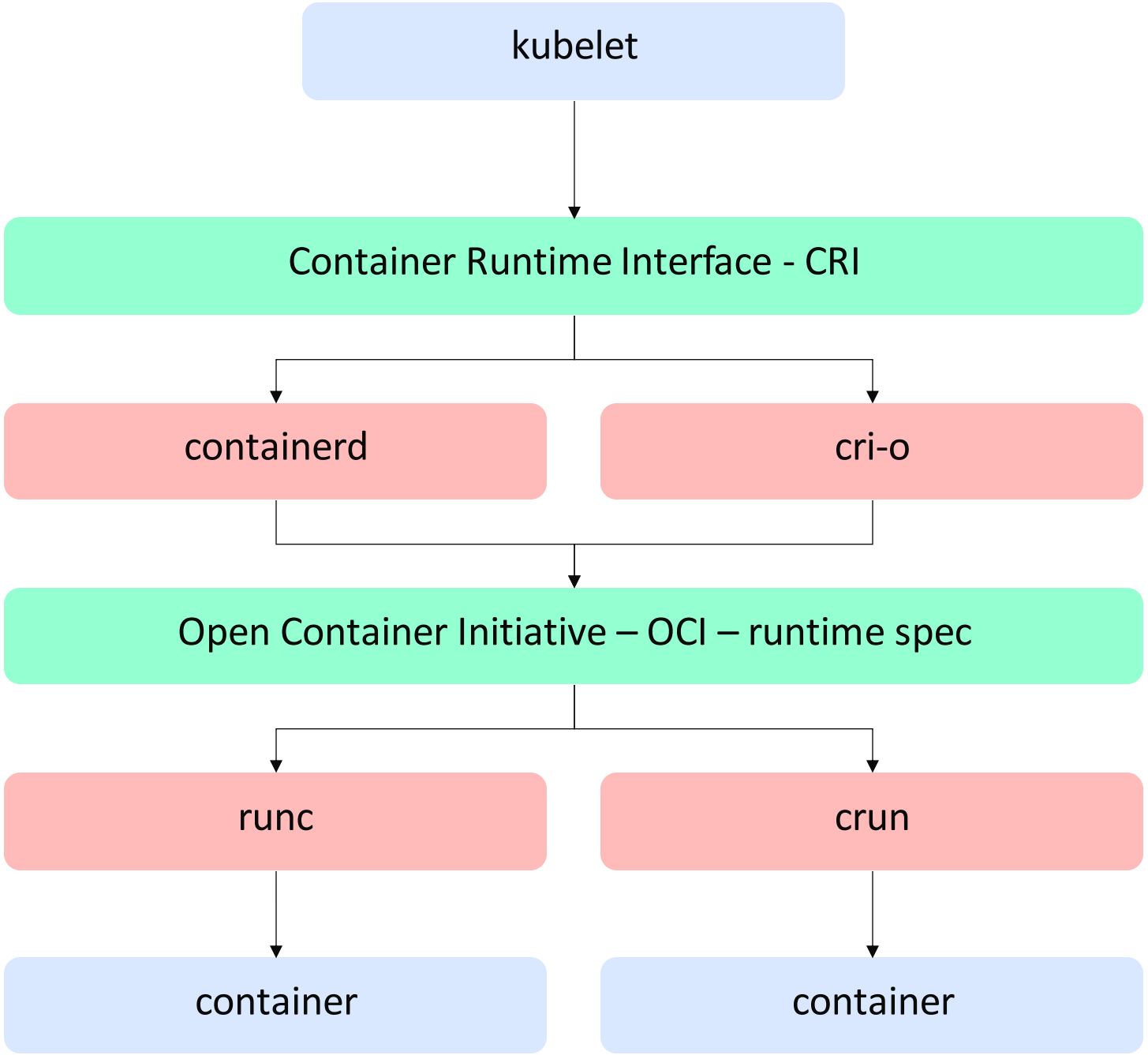
Conclusio

Conclusio

container do not exclusively mean docker

Conclusio

- **TL / DR**



cloudflight

Thank you for your attention