Container engines, runtimes and orchestrators

- an overview -

ESCAPE Project tech talks

Stefano Alberto Russo - INAF

https://sarusso.github.io/blog_container_engines_runtimes_orchestrators.html

Containers

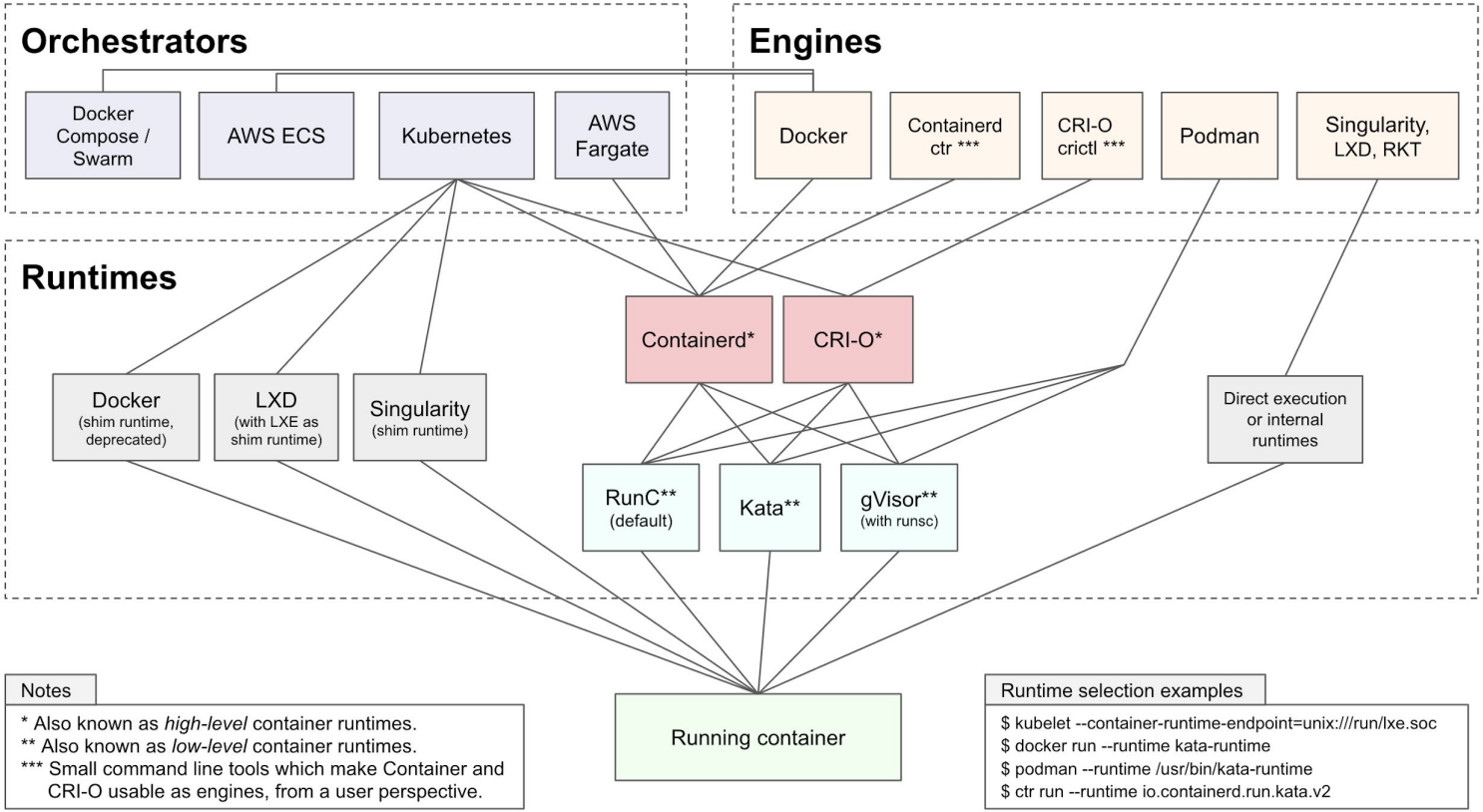
Technological definition

- Software containers are lightweight, standalone, executable packages of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. *(Docker.com)*

High-level purpose

- Containers are a solution to the problem of how to get software to run reliably when moved from one computing environment to another. *(cio.com)*

The overview

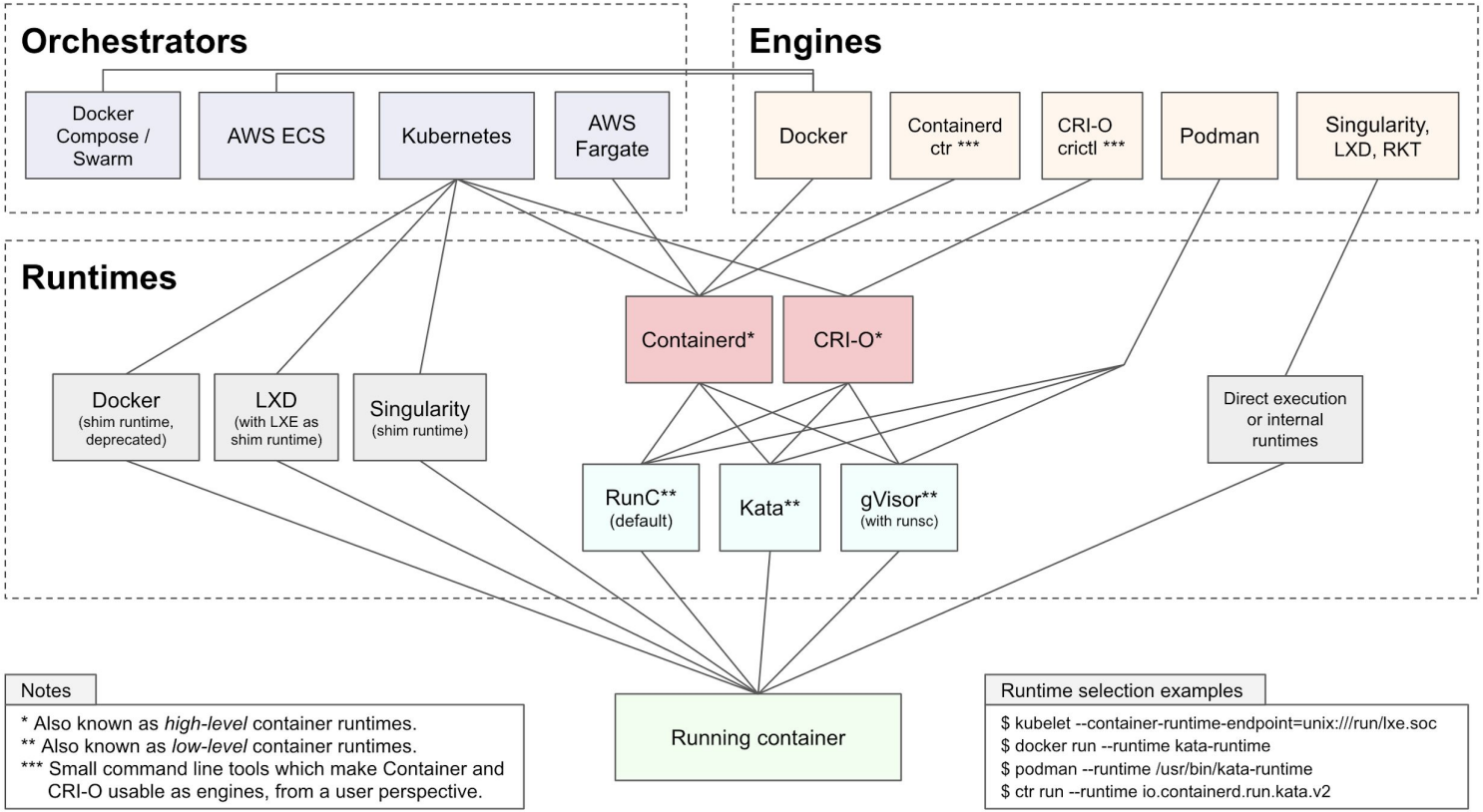


Definitions

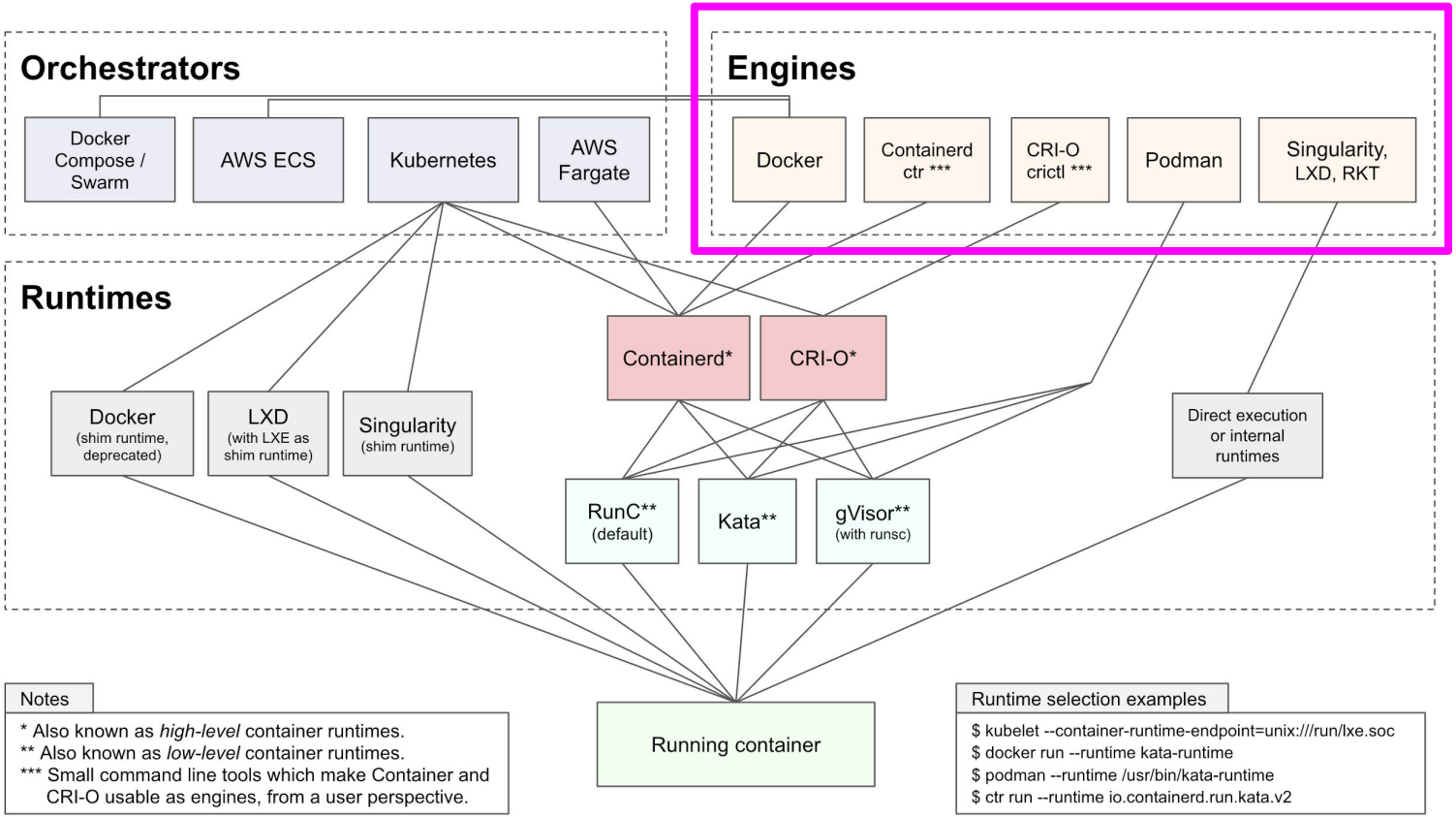
- A *container engine* is a piece of software that accepts user requests, including command line options, pulls images, and from the end user's perspective runs the container
- A *container runtime* is a software component which is in charge of managing the container lifecycle: configuring its environment, running it, stopping it, and so on. You can think about them as what's inside the engine (i.e valves and pistons). Runtimes can be further sub-divided in two types:
    - high level container runtimes, or container runtime interfaces (CRI) for Kubernetes. Following the engine analogy, you can think about them as the valves which in turn feeds the pistons.
    - low level container runtimes, or CRI runtimes for Kubernetes. You can think about them as the pistons which do the heavy lifting.

  Also note that some engines can behave as runtimes and can be thus used from within other engines, or orchestrators.
- A *container orchestrator* is a software in charge of managing set of containers across different computing resources, handling network and storage configurations which are under the hood delegated to the container runtimes (or in some nearly legacy cases, engines).

The overview



The overview



Engines: Docker

- A monolithic project in the beginning, it has since then refactored for flexibility
- The Docker GitHub repository was renamed in Moby and the internal, built-in runtime was extracted as Containerd.
- Docker identified indeed for a long time many things: a container engine, a runtime, a registry, an image format, a project and.. a company.
- As of today, the Docker Engine is to be intended as an open source software for Linux, while Docker Desktop is to be intended as the freemium product of the Docker, Inc. company for Mac and Windows platforms.

Personal take home message: the de-facto standard becoming less and less standard

Engines: Podman

- A daemonless container engine for developing, managing, and running OCI Containers on your Linux System.
- Containers can either be run as root or in rootless mode.
- A near drop-in replacement for the Docker which can run container in userspace.
- Users can become root inside the container even if outside they are standard users
- Podman has a few issues with user IDs (UID) management when running when running in rootless mode and UIDs close to 65536.
    - to allow the advanced package tool (APT) to work on Debian/Ubuntu-based containers, its UID must be reassigned not to clash with some forbidden ones [4], e.g.: `groupadd -g 600 _apt` and `usermod -g 600 _apt`.
    - by default the user outside the container is mapped to root inside the container, and non-root user mapped to to other UIDs

Personal take home message: a terrific piece of software, but beware UIDs.

Engines: Singularity / Apptainer

- Singularity (now forked as Apptainer) should be thought more of a virtual environment on steroids rather than as a container engine.
- It does not enforce (or even permit) robust isolation between the containers and the host, leaving large portions exposed → issues in security* and reproducibility
    - directories as the `/home` folder, `/tmp`, `/proc`, `/sys`, and `/dev` are all shared with the host, environment variables are exported as they are set on host, the PID namespace is not created from scratch, and the network and sockets are as well shared with the host.
- Singularity maps the user outside the container as the same user inside it, meaning that every time a container is run the user UID (and name) can change inside it, making it very hard to handle permissions.
- Two issues opened on the former Singularity project are quite self-explanatory: Same container, different results and Python3 script fails in singularity container on one machine, but works in same container on another.

Personal take home message: stay away from it, unless you really have to.

*what happens inside the container does not necessarily stay in the container. It is not “contained”.

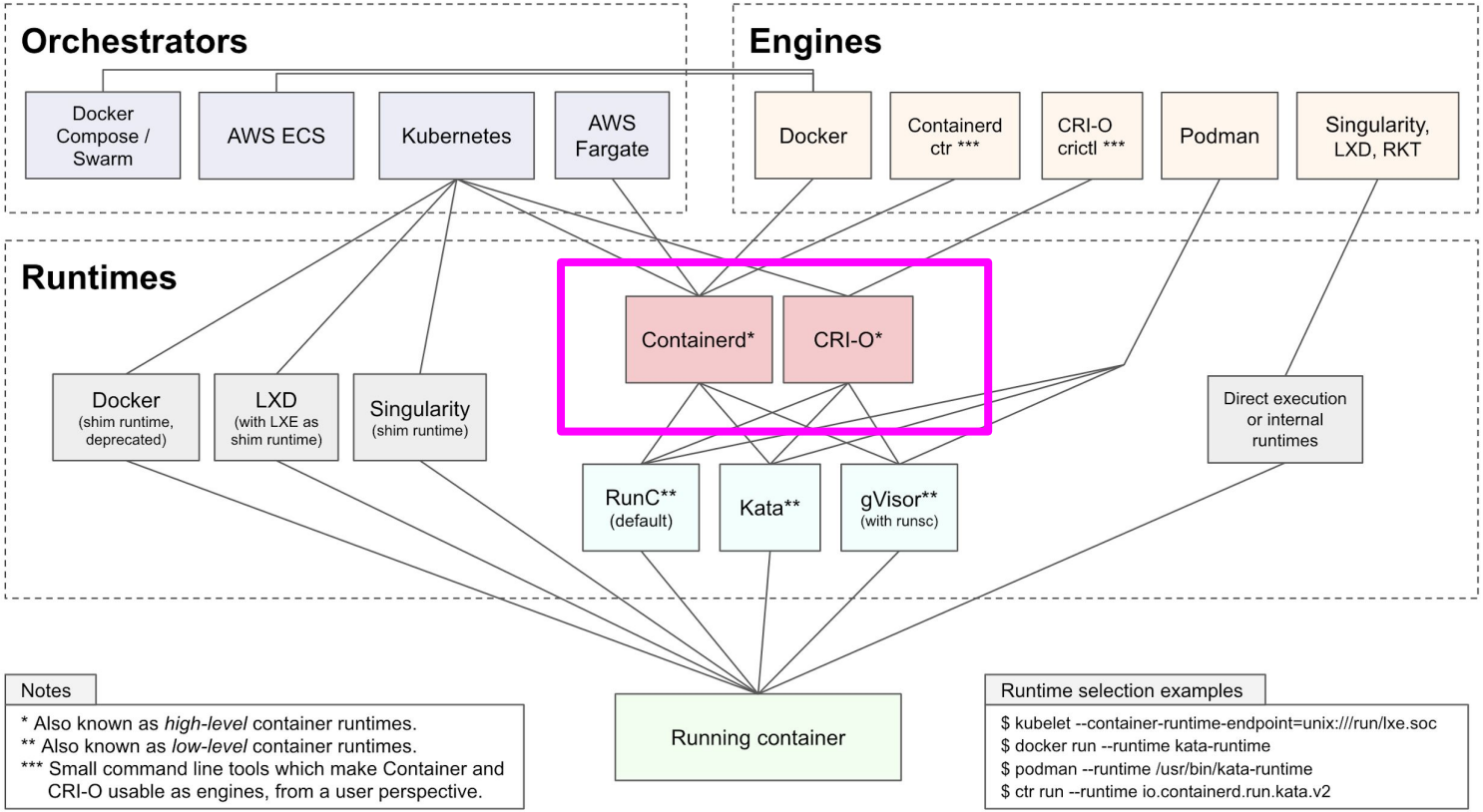
Engines: others

Containerd crt
Containerd, which will be introduced in the runtimes section, it is not intended to be directly used as an engine (being a runtime), but with the Containerd CLI (`ctr`) utility it can behave as such.

CRI-O crictl
CRI-O, which will be introduced in the runtimes section as well, is not intended to be directly used as an engine too. However, with the `crictl` command line utility it can behave as such, mainly for debugging purpose. To underline that CRI-O is not intended to be directly used from a command line (being a runtime), the official CRI-O code repository states that "any CLIs built as part of this project are only meant for testing this project and there will be no guarantees on the backward compatibility with it".

LXD and RKD
LXD is something tangential to a container engine, as it allow to manage both containers and virtual machines, offering "a unified user experience around full Linux systems running inside containers or virtual machines". LXD uses LXC as internal runtime. Rocket (RKD) was instead a command line utility for running containers on Linux directly using kernel-level calls, similarly as for LXC, and is as of today an ended project.

The overview



High level runtimes:

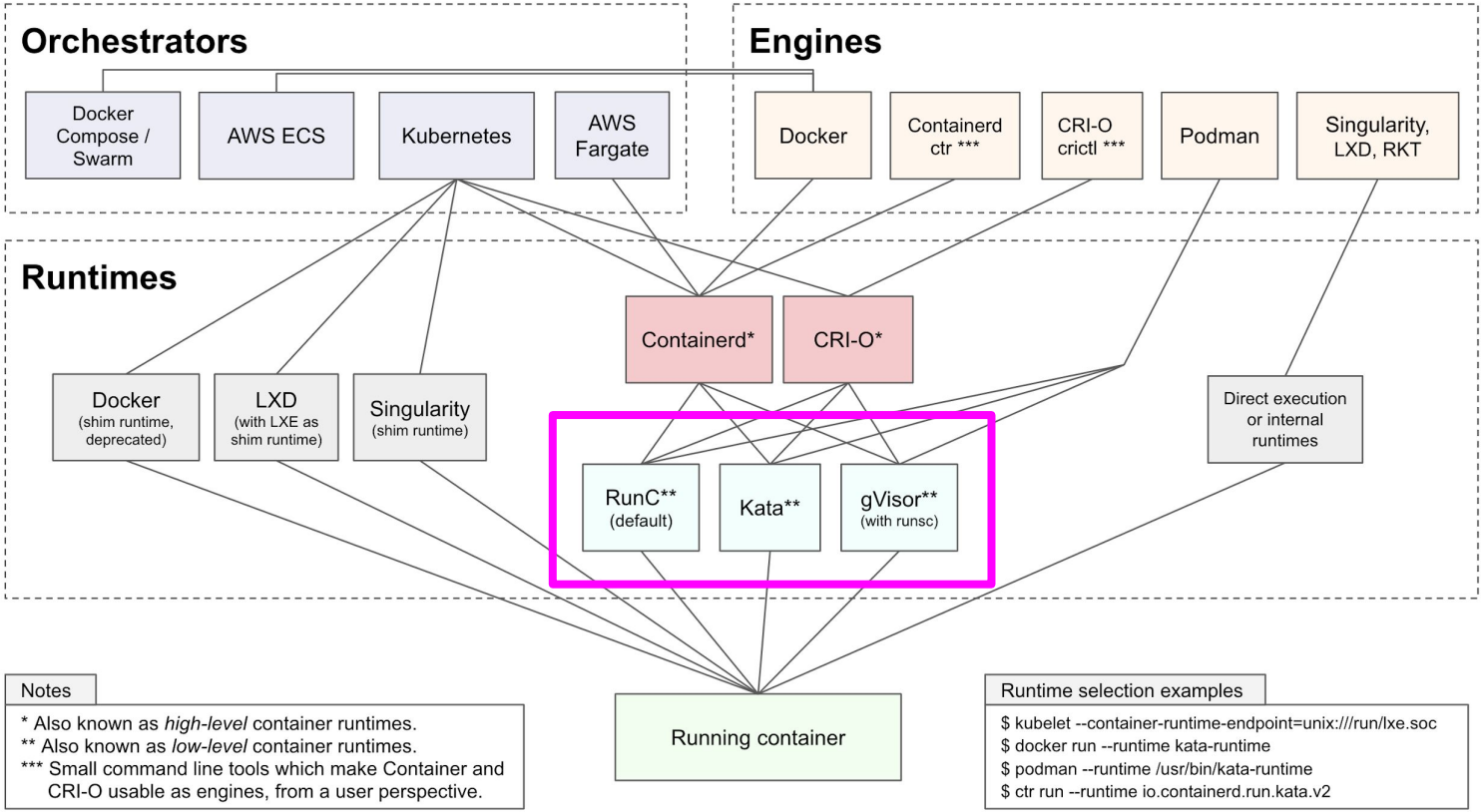
Containerd

- an high-level container runtime originated from Docker, and extracted out from Docker itself for flexibility over the years.
- A default Docker engine installation will install Containerd as well.
- Containerd is also the default Kubernetes CRI runtime. Containerd uses runC as its default low-level runtime.

CRI-O

- an "implementation of the Kubernetes CRI (Container Runtime Interface) to enable using OCI (Open Container Initiative) compatible runtimes".
- It basically tried to fill some gaps along the Kubernetes development and is now a direct competitor (if it makes sense to call it as such) to Containerd. CRI-O uses runC as its default low-level runtime as well.

The overview



Low level runtimes:

runC

- an OCI-compatible container runtime. It implements the OCI specification and runs the container processes.
- RunC is called the reference implementation of OCI.

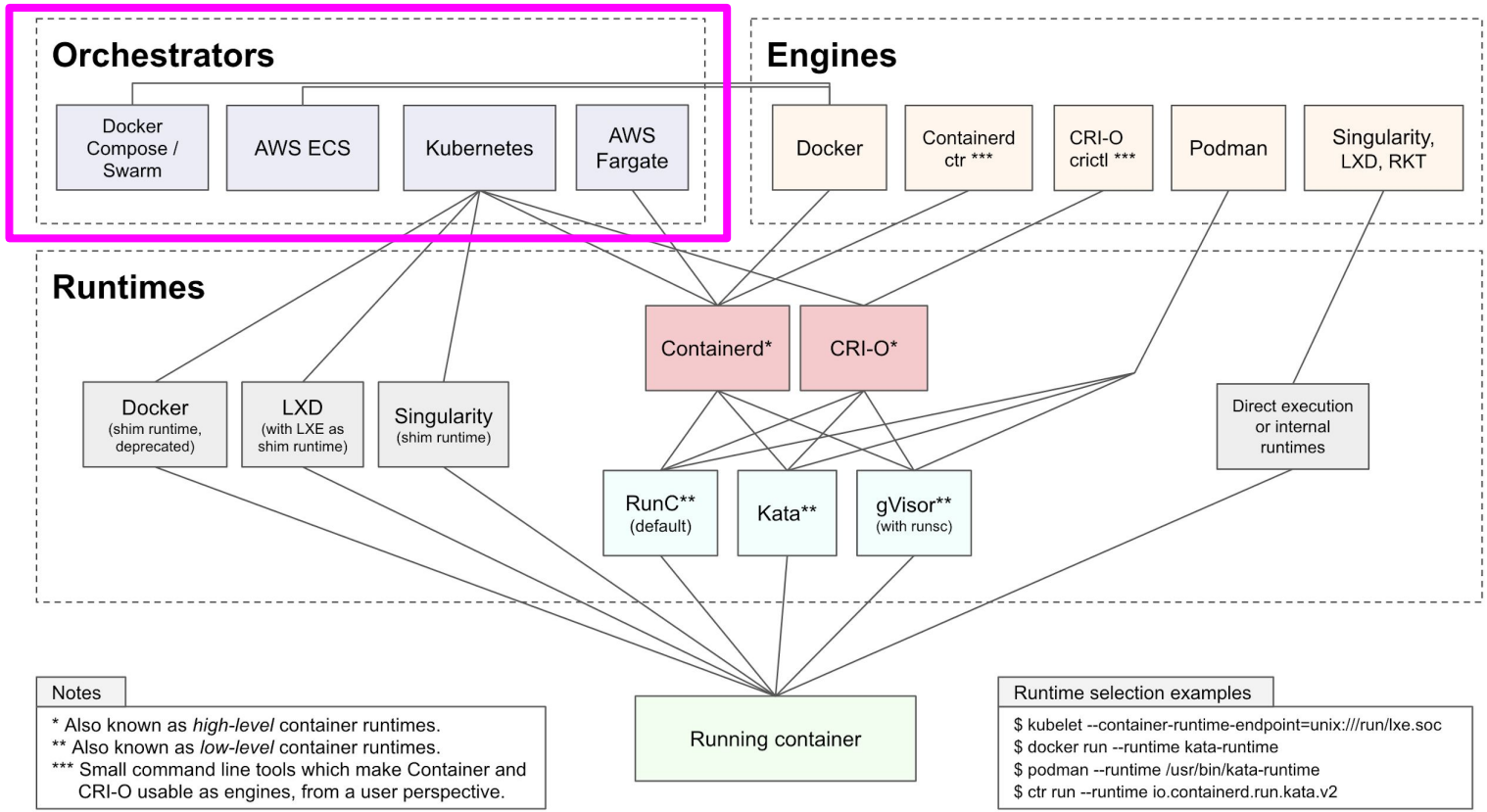
gVisor

- a runtime developed by Google which implements kernel virtualisation.
- each container has its own kernel, unlike other container runtimes where the kernel is usually shared between the host and the containers.
- allows for more security than other runtimes while allowing to share host resources without pre-allocation.

Kata

- a runtime implementing hardware virtualisation (aka: a virtual machine).
- The idea is to have a runtime which behave as running software containers but that under the hood spawn a new virtual machine and run the container inside it.
- requires pre-allocation of resources, and in particular of the memory which is set by default to 2GB per container.

The overview



Orchestrators

Docker compose

- allows to define simple multi-service applications where all the containers run on the same node. It creates a dedicated network for the containers on the host from which they can all talk each others, and a `docker-compose.yml` file describes how to assemble them. It the simplest orchestrator, and very useful for local and simple deployments.
- Docker compose has support only for the Docker APIs, and Podman can work with it by emulating Docker.

Docker Swarm

- Docker Swarm is similar to Docker compose but it can manage multi-node deployments, or on other words a cluster of Docker engines called a "swarm".
- As for Docker compose, Docker Swarm supports only the Docker APIs.

Kubernetes

- the full-featured solution for container orchestration, supporting a variety of settings, network topologies and container runtimes. In 2021 it dropped support for Docker, which generated some panic over the internet. What it actually happened is that it dropped support for Dockershim in favour of directly using Containerd, and nothing changed for the users.

Orchestrators (hosted)

AWS ECS

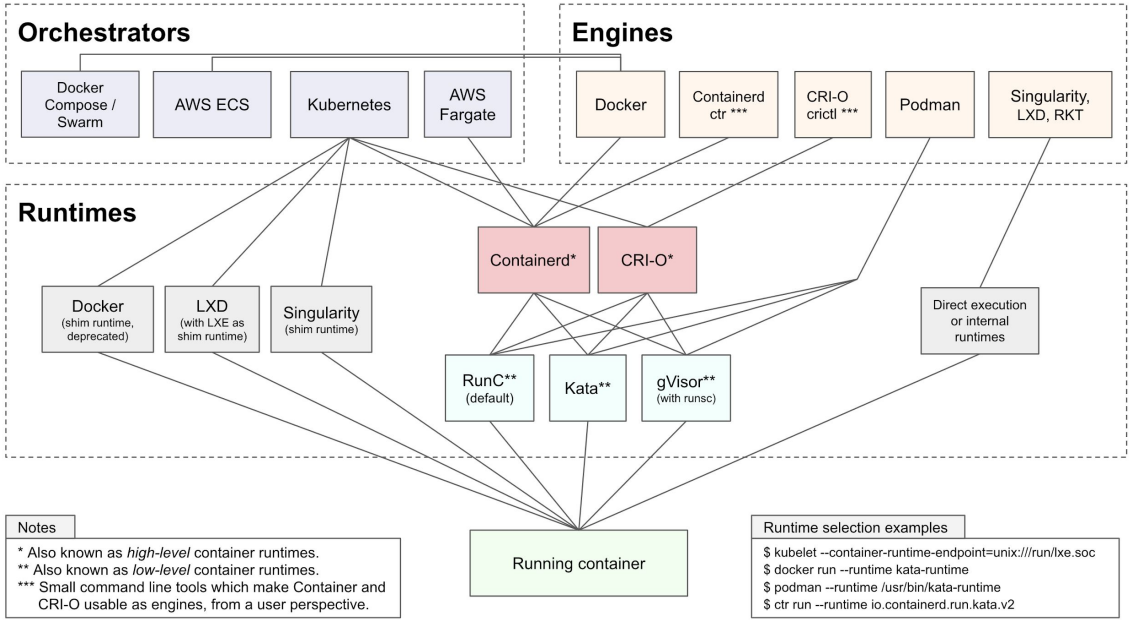
- Amazon Web Services's Elastic Compute Service is Amazon's internal implementation of a Kubernetes-like solution.
- Amazon virtual machines require the Docker Engine to be installed in order to be managed using ECS.
- Alternatively, customers can directly use a virtual machine image pre-build by Amazon which is already configured for using it with ECS. The main point is that AWS ECS use the Docker engine, and not a container runtime.

AWS Fargate

- AWS Fargate is likely one of these "definitive" solutions that will become the new normal for a large number of use cases (as it happened with RDS).
- Fargate allow executing containers in a serverless fashion, on AWS computing infrastructure, and to entirely forget about the underlying OS (and hardware, of course).
- Interestingly enough, by probably being a project younger than ECS, it could make the strategic move of stopping to rely on container engines in favour of adopting container runtimes. In particular, with Fargate platform version 1.4 in April 2020, they replaced the Docker Engine with Containerd as Fargate’s container execution engine.

Thanks!

Questions?



Stefano Alberto Russo - INAF

https://sarusso.github.io/blog_container_engines_runtimes_orchestrators.html