Software Containerization Lesson 1

lara.ziosi@hcl.com
l.ziosi@vu.nl

lara.ziosi@hcl.com, l.ziosi@vu.nl

Lecturer’s biography

Lara Ziosi’s education:

- Laurea (M.Sc.) in Physics from the University of Bologna, Italy (1995)
- Dottorato (PhD) in Physics from the University of Bologna, Italy (1999)
- M.Sc. in Artificial Intelligence from the Cork Institute of Technology (now Munster Technological University), Ireland, 2020

Relevant professional certifications:

- Google Cloud Certified Cloud Architect – 2020 till 2022
- Kubernetes Certified Application Developer - 2021

Teaching experience:

- Lecturer – Developing Services for the Cloud from 2013 to 2019 – Vrije Universiteit Amsterdam
- Lecturer – Software Containerization from 2021 – Vrije Universiteit Amsterdam

Professional experience:

- Support engineer, Modeling tools, Rational Software (1999 - 2003)
- Support engineer, Modeling and development tools, IBM (2003 - 2013)
- Team Lead, UrbanCode support team EMEA, IBM (2013 - 2018)
- Team Lead, UrbanCode support team EMA, HCL Software Support (2018 - 2019)
- Senior Technical Architect, HCL Software Customer Support (2019 – 2022)
- Associate Director Software Engineering, HCL Software Customer Support (2022 – to date)

Projects based on containerization:

- DataMart using Apache Airflow with KubernetesPodOperators, Jenkins to build Docker containers, Helm to deploy charts of Apache Superset, Postgresql, Apache Airflow and PGO, the CrunchyData Postgres Operator
- Chatbot using Kubernetes and containerized versions of Rasa, Haystack, OpenSearch

Teaching Assistant’s support sessions (see Syllabus for updates)

Abdellah Lahnaoui - Groups 01 - 13
Monday 11:00 - 13:00 (except the first week, when it is on Wednesday at the same time)
https://vu-live.zoom.us/j/98741669184?pwd=Y294MjhxcWNpSjFwbXB4Mk5JVjd3dz09
Meeting ID: 987 4166 9184
Passcode: 983205

Andrea Marino - Groups 14 - 26
Monday 14:00 - 16:00 (except the first week, when it is on Wednesday at the same time)
https://vu-live.zoom.us/j/97268430506?pwd=M2lmaXNVZDVPR2pUMGRwUHRqTHdDZz09
Meeting ID: 972 6843 0506
Passcode: 025823

Andrei Bogdan - Groups 27 - 39
Friday 15:00 - 17:00
https://vu-live.zoom.us/j/7912342072?pwd=NFdkOUtwZzB4R053MVR3aEN6QitsUT09
Passcode: 827423

Ruben Horn - Groups 40 - 53
Friday 10:00 - 12:00
https://vu-live.zoom.us/j/91351839361?pwd=dS9ISThvcGZMcm5lUXNwM1M1dUFtUT09

https://canvas.vu.nl/courses/63149/assignments/syllabus

Introduction to the course: agenda

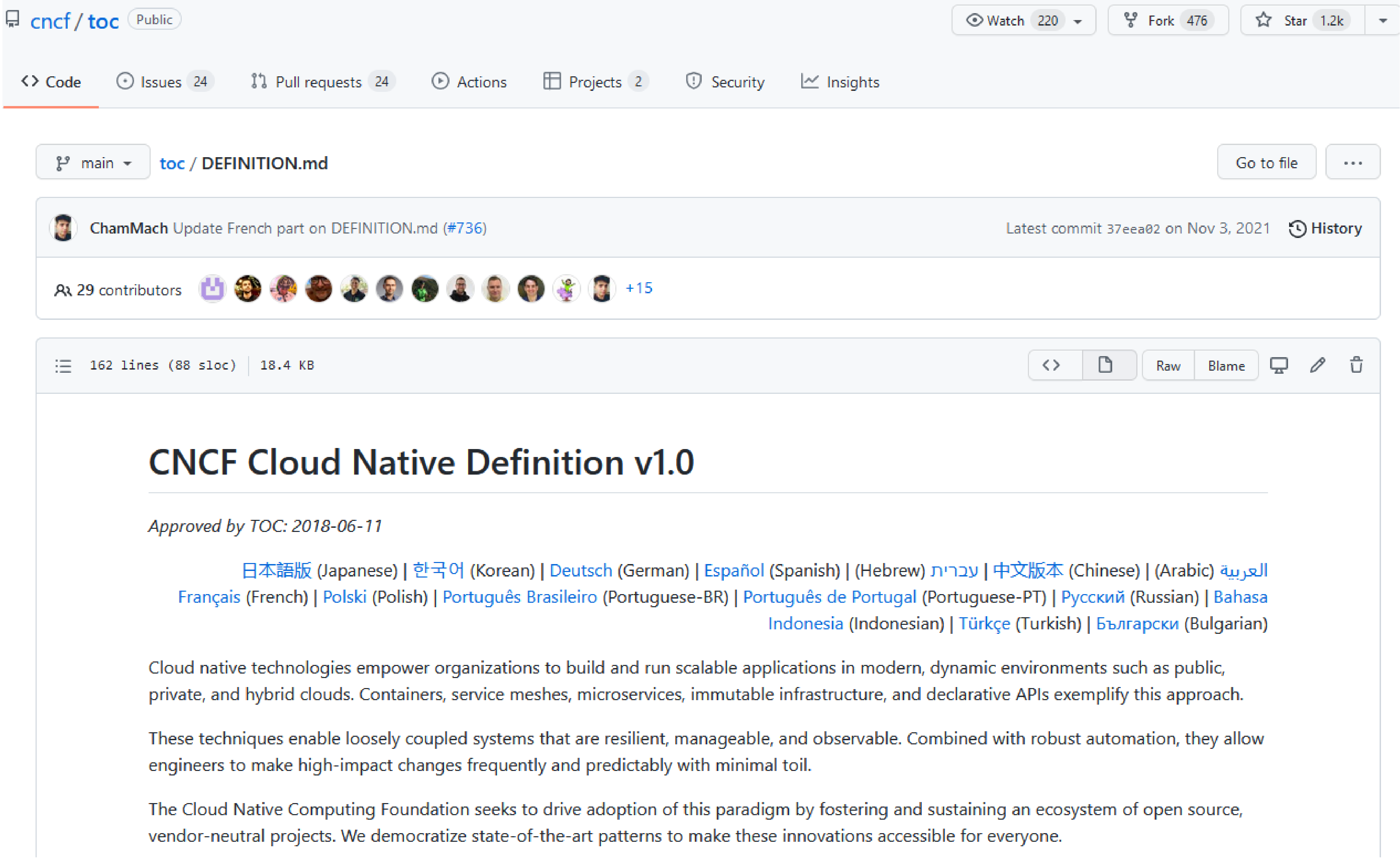
- Lesson 1:
  - Introduction to Software Containerization
  - Linux features that enable containerization
  - Introduction to Docker
- Lesson 2:
  - Kubernetes architecture
  - Nodes, Namespaces
  - Pods and containers
  - Deployments, Jobs, CronJobs
  - YAML syntax
- Lesson 3:
  - Networking
  - Services
  - CoreDNS
  - Ingress
- Lesson 4:
  - Storage
  - StatefulSet
  - ConfigMaps, Secrets

- Lesson 5:
  - Rolling updates
  - Canary deployments
  - Application deployment with Helm Charts
- Lesson 6:
  - Network Policies
  - Role Based Access Control (RBAC)
- Lesson 7:
  - Service Mesh: Istio
  - Metrics with Prometheus
  - Visualization with Grafana
  - Distributed tracing with Jaeger
- Lesson 8:
  - Written Exam

HCLSoftware

Introduction to Software Containerization

Cloud Native Definition v1.0



CNCF Cloud Native Definition v1.0

Approved by TOC: 2018-06-11

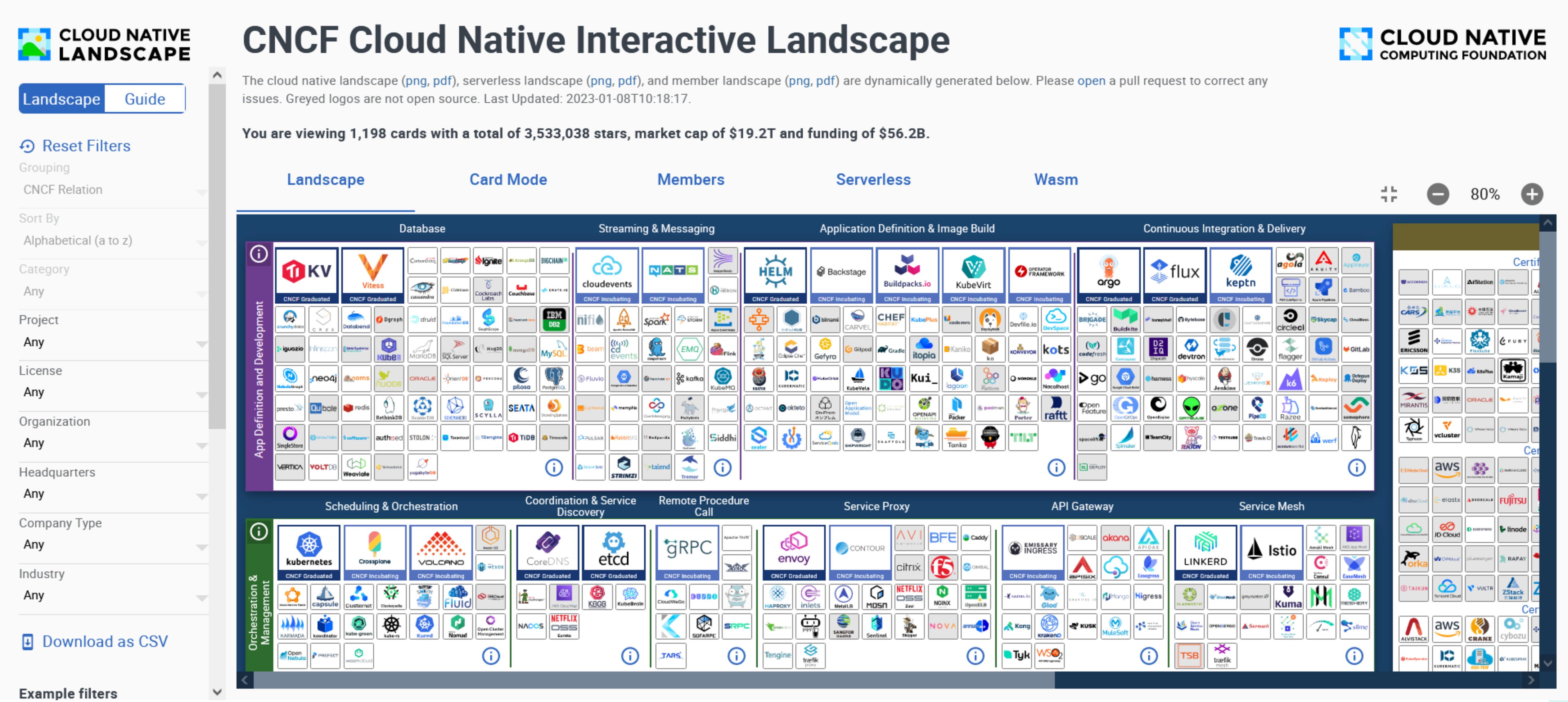
日本語版 (Japanese) | 한국어 (Korean) | Deutsch (German) | Español (Spanish) | (Hebrew) עברית | 中文版本 (Chinese) | (Arabic) العربية Français (French) | Polski (Polish) | Português Brasileiro (Portuguese-BR) | Português de Portugal (Portuguese-PT) | Русский (Russian) | Bahasa Indonesia (Indonesian) | Türkçe (Turkish) | Български (Bulgarian)

Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

The Cloud Native Computing Foundation seeks to drive adoption of this paradigm by fostering and sustaining an ecosystem of open source, vendor-neutral projects. We democratize state-of-the-art patterns to make these innovations accessible for everyone.

CNCF Cloud Native Interactive Landscape



CLOUD NATIVE LANDSCAPE
Landscape
Guide
Reset Filters
CLOUD NATIVE COMPUTING FOUNDATION
Landscape
Card Mode
Members
Serverless
Wasm
80%
Database
Streaming & Messaging
Application Definition & Image Build
Continuous Integration & Delivery
App Definition and Development
Scheduling & Orchestration
Coordination & Service Discovery
Remote Procedure Call
Service Proxy
API Gateway
Service Mesh
Orchestration & Management
Download as CSV
Example filters

CNCF Cloud Native Interactive Landscape

The cloud native landscape (png, pdf), serverless landscape (png, pdf), and member landscape (png, pdf) are dynamically generated below. Please open a pull request to correct any issues. Greyed logos are not open source. Last Updated: 2023-01-08T10:18:17.

You are viewing 1,198 cards with a total of 3,533,038 stars, market cap of $19.2T and funding of $56.2B.

Grouping
CNCF Relation

Sort By
Alphabetical (a to z)

Category
Any

Project
Any

License
Any

Organization
Any

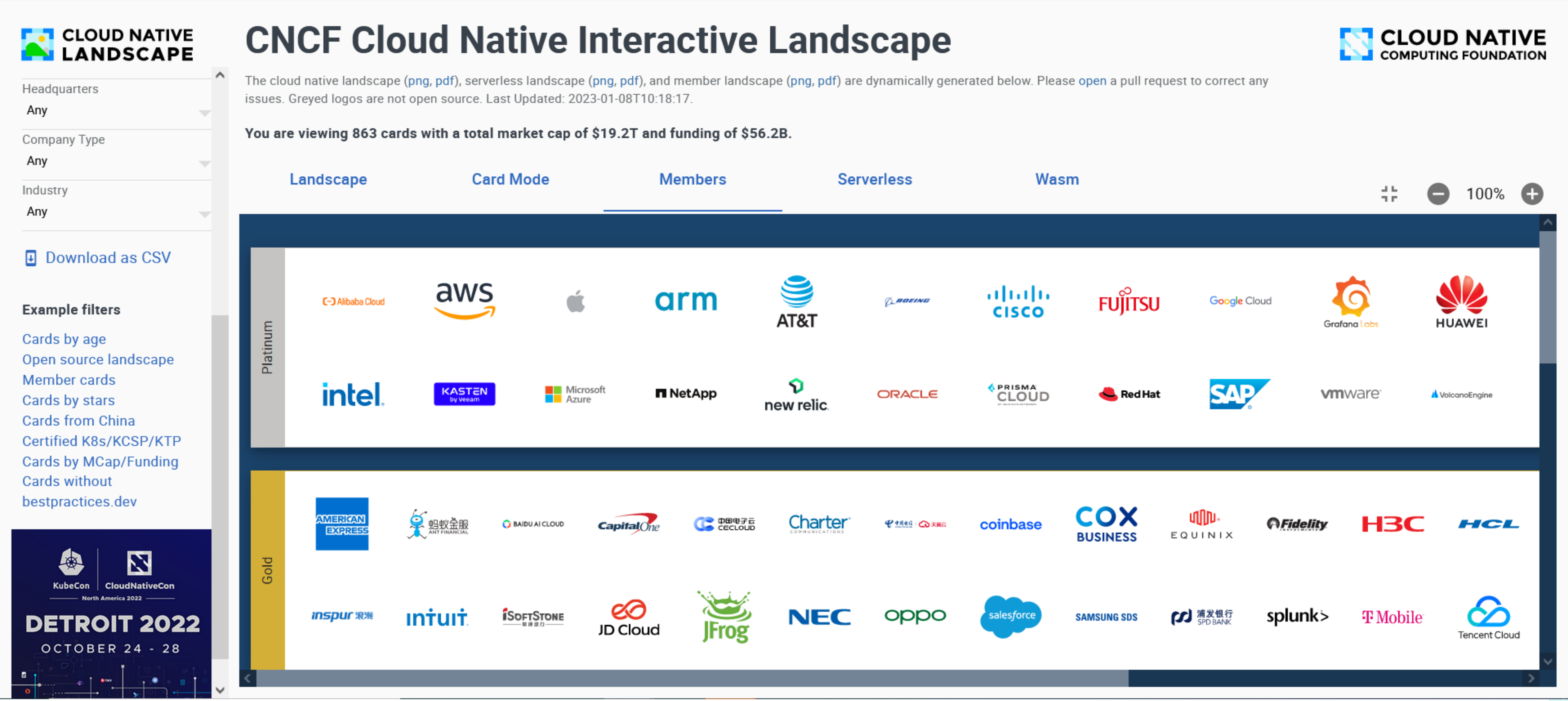
Headquarters
Any

Company Type
Any

Industry
Any

https://landscape.cncf.io

CNCF Cloud Native Interactive Landscape - Members



CLOUD NATIVE LANDSCAPE

Headquarters
Any

Company Type
Any

Industry
Any

Download as CSV

Example filters

- Cards by age
- Open source landscape
- Member cards
- Cards by stars
- Cards from China
- Certified K8s/KCSP/KTP
- Cards by MCap/Funding
- Cards without bestpractices.dev

CNCF Cloud Native Interactive Landscape

CLOUD NATIVE COMPUTING FOUNDATION

The cloud native landscape (png, pdf), serverless landscape (png, pdf), and member landscape (png, pdf) are dynamically generated below. Please open a pull request to correct any issues. Greyed logos are not open source. Last Updated: 2023-01-08T10:18:17.

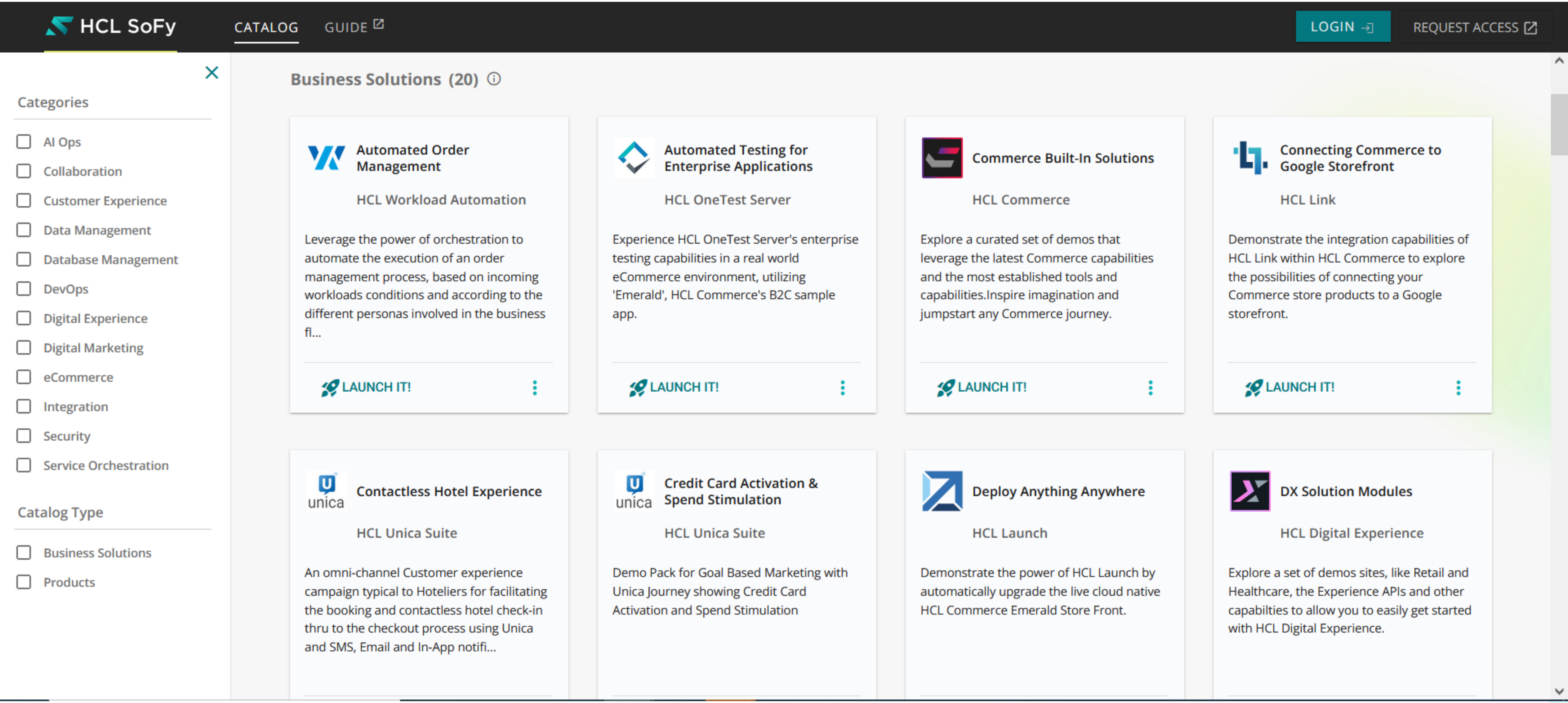
You are viewing 863 cards with a total market cap of $19.2T and funding of $56.2B.

Landscape | Card Mode | Members | Serverless | Wasm

100%

https://landscape.cncf.io/members?category=cncf-members&grouping=category

HCLSoFy – Business Solutions and Product Catalog



Business Solutions (20)

Categories

- □ AI Ops
- □ Collaboration
- □ Customer Experience
- □ Data Management
- □ Database Management
- □ DevOps
- □ Digital Experience
- □ Digital Marketing
- □ eCommerce
- □ Integration
- □ Security
- □ Service Orchestration

Catalog Type

- □ Business Solutions
- □ Products

Automated Order Management
HCL Workload Automation
Leverage the power of orchestration to automate the execution of an order management process, based on incoming workloads conditions and according to the different personas involved in the business fl...
LAUNCH IT!

Automated Testing for Enterprise Applications
HCL OneTest Server
Experience HCL OneTest Server's enterprise testing capabilities in a real world eCommerce environment, utilizing 'Emerald', HCL Commerce's B2C sample app.
LAUNCH IT!

Commerce Built-In Solutions
HCL Commerce
Explore a curated set of demos that leverage the latest Commerce capabilities and the most established tools and capabilities.Inspire imagination and jumpstart any Commerce journey.
LAUNCH IT!

Connecting Commerce to Google Storefront
HCL Link
Demonstrate the integration capabilities of HCL Link within HCL Commerce to explore the possibilities of connecting your Commerce store products to a Google storefront.
LAUNCH IT!

Contactless Hotel Experience
HCL Unica Suite
An omni-channel Customer experience campaign typical to Hoteliers for facilitating the booking and contactless hotel check-in thru to the checkout process using Unica and SMS, Email and In-App notifi...

Credit Card Activation & Spend Stimulation
HCL Unica Suite
Demo Pack for Goal Based Marketing with Unica Journey showing Credit Card Activation and Spend Stimulation

Deploy Anything Anywhere
HCL Launch
Demonstrate the power of HCL Launch by automatically upgrade the live cloud native HCL Commerce Emerald Store Front.

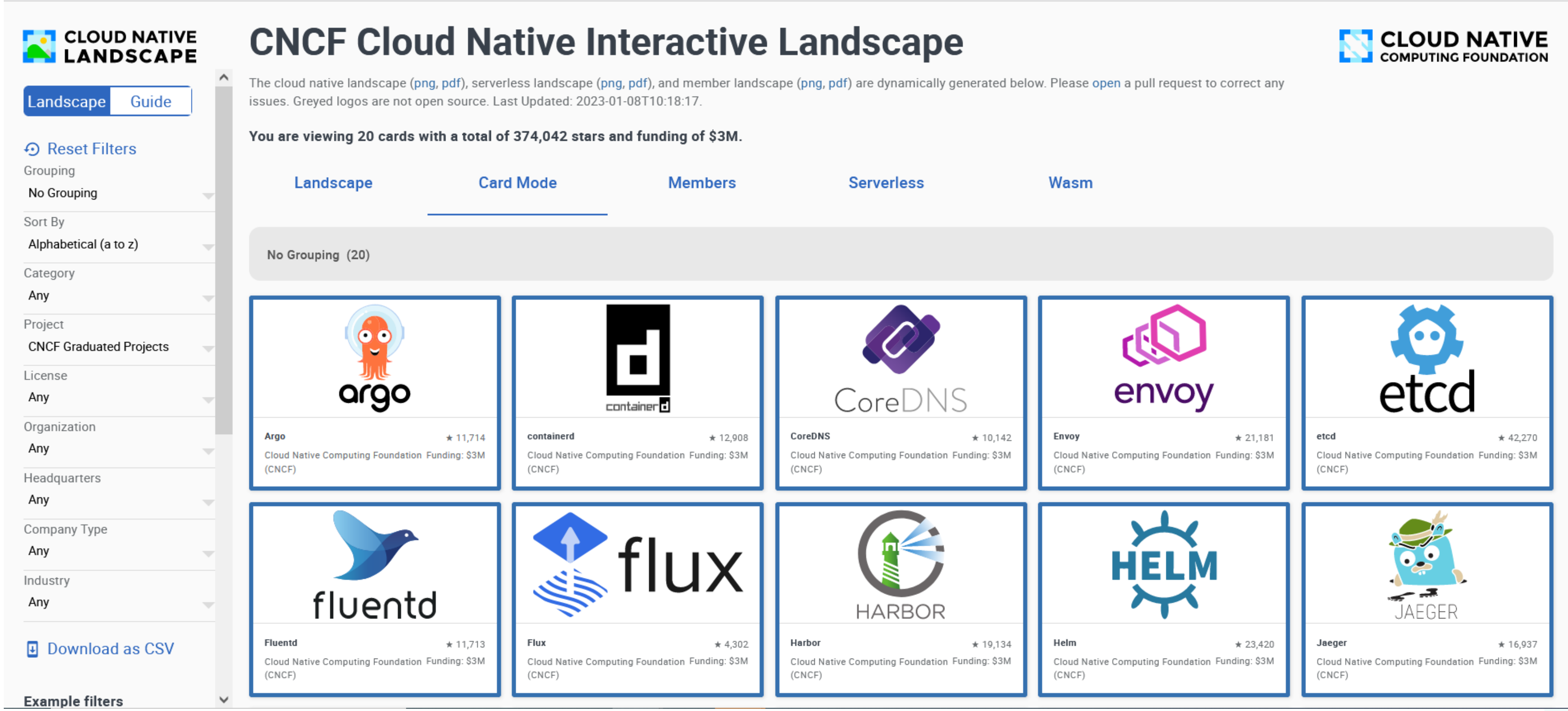
DX Solution Modules
HCL Digital Experience
Explore a set of demos sites, like Retail and Healthcare, the Experience APIs and other capabilties to allow you to easily get started with HCL Digital Experience.

https://hclsofy.com/catalog/content

CNCF graduated projects

In this class you will use at least:

- Kubernetes
- containerd
- etcd
- CoreDNS
- Helm
- Prometheus



CLOUD NATIVE LANDSCAPE

Landscape | Guide

Reset Filters

Grouping: No Grouping

Sort By: Alphabetical (a to z)

Category: Any

Project: CNCF Graduated Projects

License: Any

Organization: Any

Headquarters: Any

Company Type: Any

Industry: Any

Download as CSV

Example filters

CNCF Cloud Native Interactive Landscape

CLOUD NATIVE COMPUTING FOUNDATION

The cloud native landscape (png, pdf), serverless landscape (png, pdf), and member landscape (png, pdf) are dynamically generated below. Please open a pull request to correct any issues. Greyed logos are not open source. Last Updated: 2023-01-08T10:18:17.

You are viewing 20 cards with a total of 374,042 stars and funding of $3M.

Landscape | Card Mode | Members | Serverless | Wasm

No Grouping (20)

Argo ★ 11,714
Cloud Native Computing Foundation (CNCF) Funding: $3M

containerd ★ 12,908
Cloud Native Computing Foundation (CNCF) Funding: $3M

CoreDNS ★ 10,142
Cloud Native Computing Foundation (CNCF) Funding: $3M

Envoy ★ 21,181
Cloud Native Computing Foundation (CNCF) Funding: $3M

etcd ★ 42,270
Cloud Native Computing Foundation (CNCF) Funding: $3M

Fluentd ★ 11,713
Cloud Native Computing Foundation (CNCF) Funding: $3M

Flux ★ 4,302
Cloud Native Computing Foundation (CNCF) Funding: $3M

Harbor ★ 19,134
Cloud Native Computing Foundation (CNCF) Funding: $3M

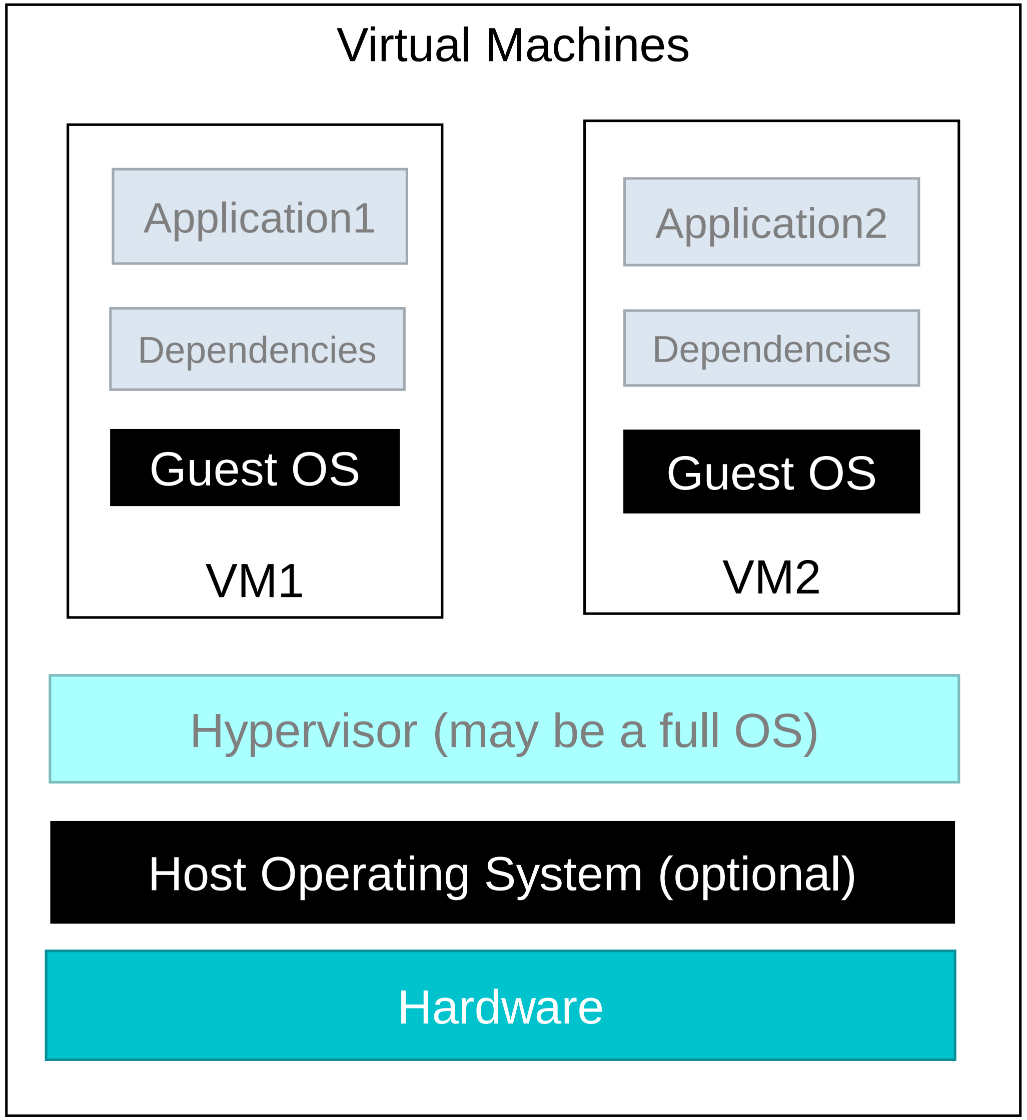
Helm ★ 23,420
Cloud Native Computing Foundation (CNCF) Funding: $3M

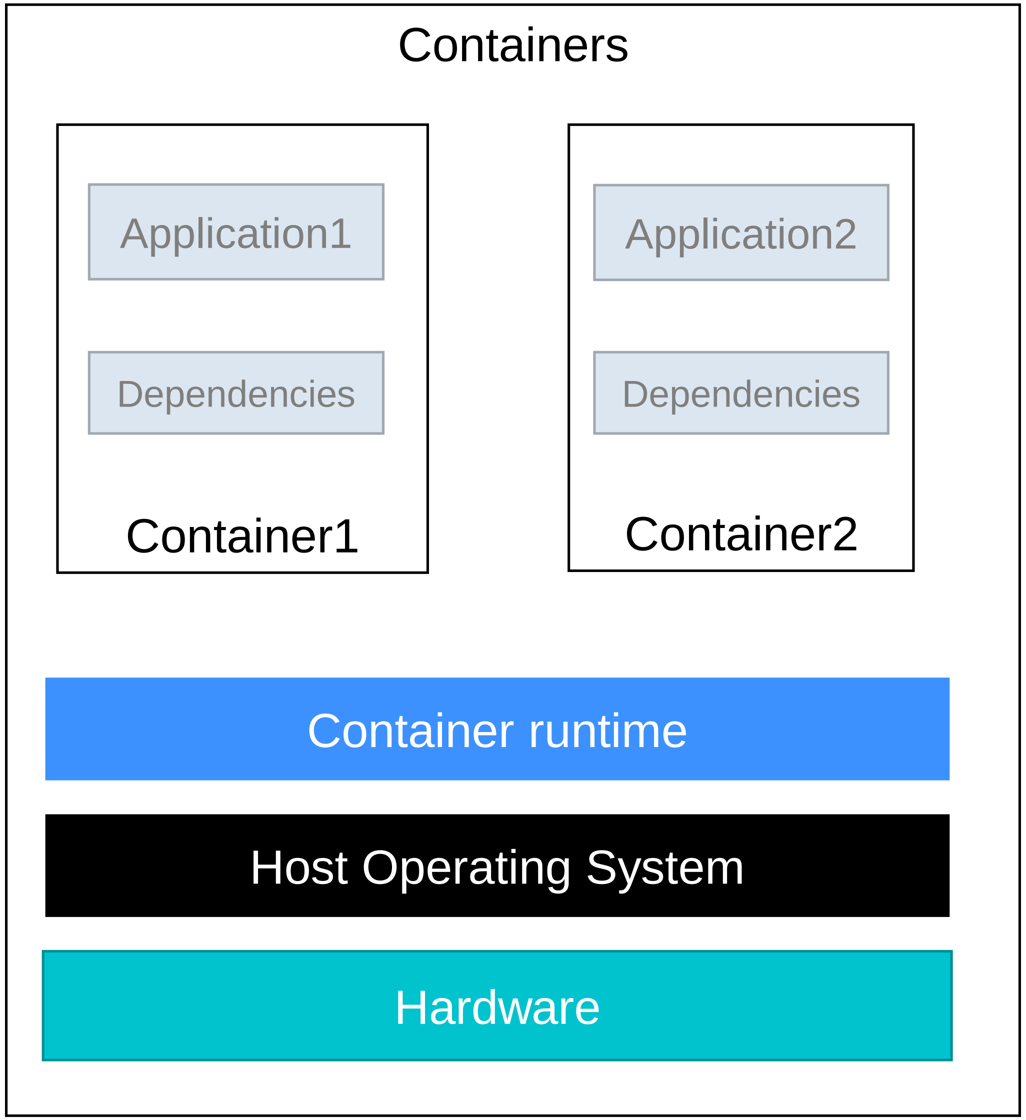
Jaeger ★ 16,937
Cloud Native Computing Foundation (CNCF) Funding: $3M

https://landscape.cncf.io/card-mode?project=graduated&grouping=no

https://www.cncf.io/projects/

Containers versus Virtual Machines





Full virtualization/paravirtualization vs OS-level virtualization (containers)

In full virtualization (emulation) and paravirtualization, isolation is achieved by means of loading one Guest operating system per Virtual Machine on top of the underlying Host operating system/Hypervisor.

This has considerable overhead in terms of resource consumption, startup time, and large amount of unnecessary software bundled with each VM.

Examples of Hypervisors:

- VMWare ESXi (no Host OS needed)
- Oracle VirtualBox
- KVM (Kernel-based Virtual Machine)
- IBM zVM
- Xen Hypervisor (paravirtualization)
- QEMU
- Microsoft Windows Hyper-V

Virtualization library:

- libvirt

In OS-level virtualization or containerization, isolation is achieved by separating resources within the same host operating system.

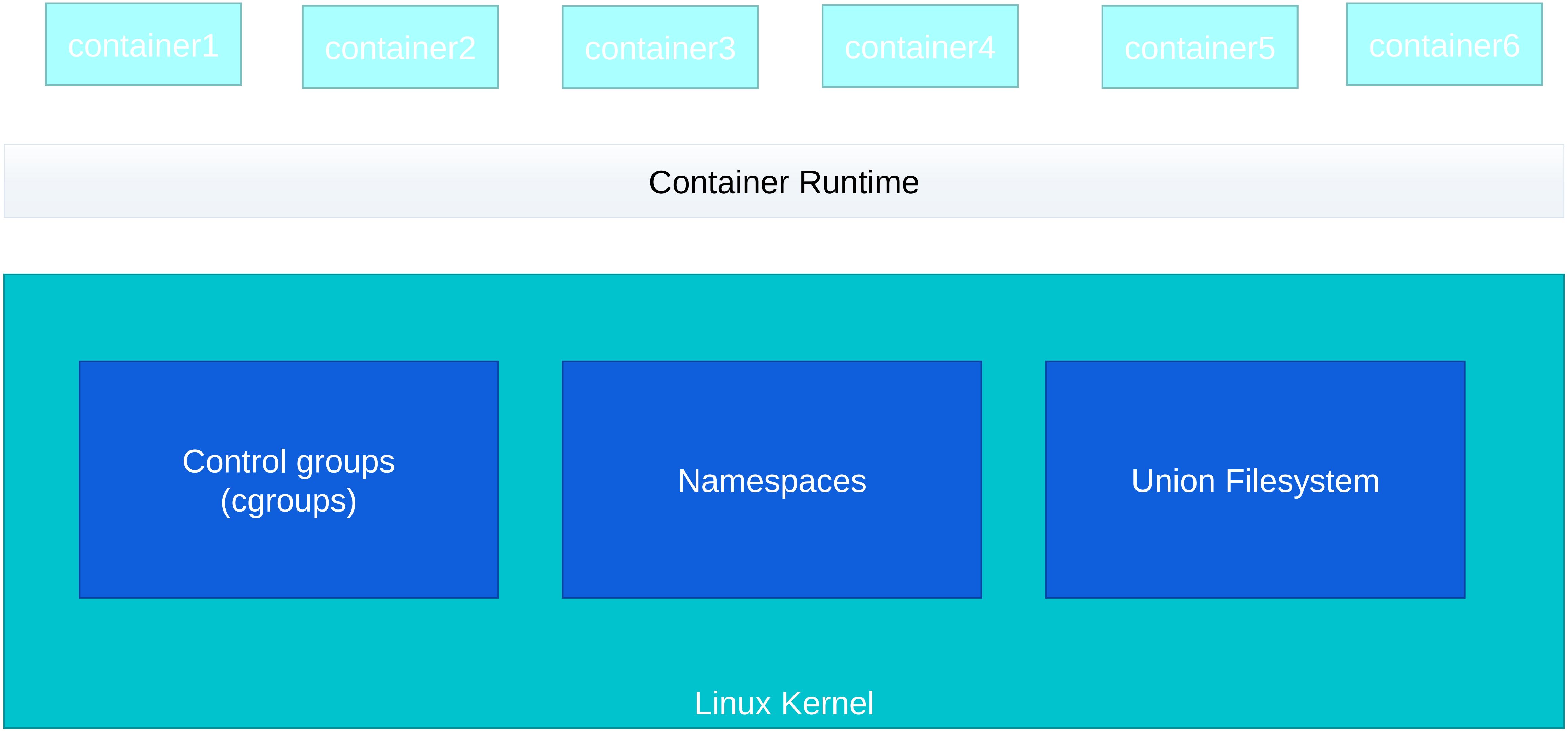
This reduces the amount of resources required for each application to the bare minimum and improves startup time and allows for orchestration of different containers.

Examples of Linux based platforms/tools that support containers:

- Docker
- LXC, LXD and LXCFS
- CoreOs Linux Containers -> Fedora CoreOS
- Kubernetes
- Red Hat OpenShift
- Rancher

In 2016 Microsoft added support for Windows Containers (requiring Docker and Hyper-V)

What Linux kernel features enable containers?



container1

container2

container3

container4

container5

container6

Container Runtime

Control groups (cgroups)

Namespaces

Union Filesystem

Linux Kernel

Linux namespaces

A namespace is a mechanism that allows the separation/isolation of resources that are otherwise in a global scope.
Currently there are 8 kinds of namespaces in the Linux Kernel, in historical order of appearance:

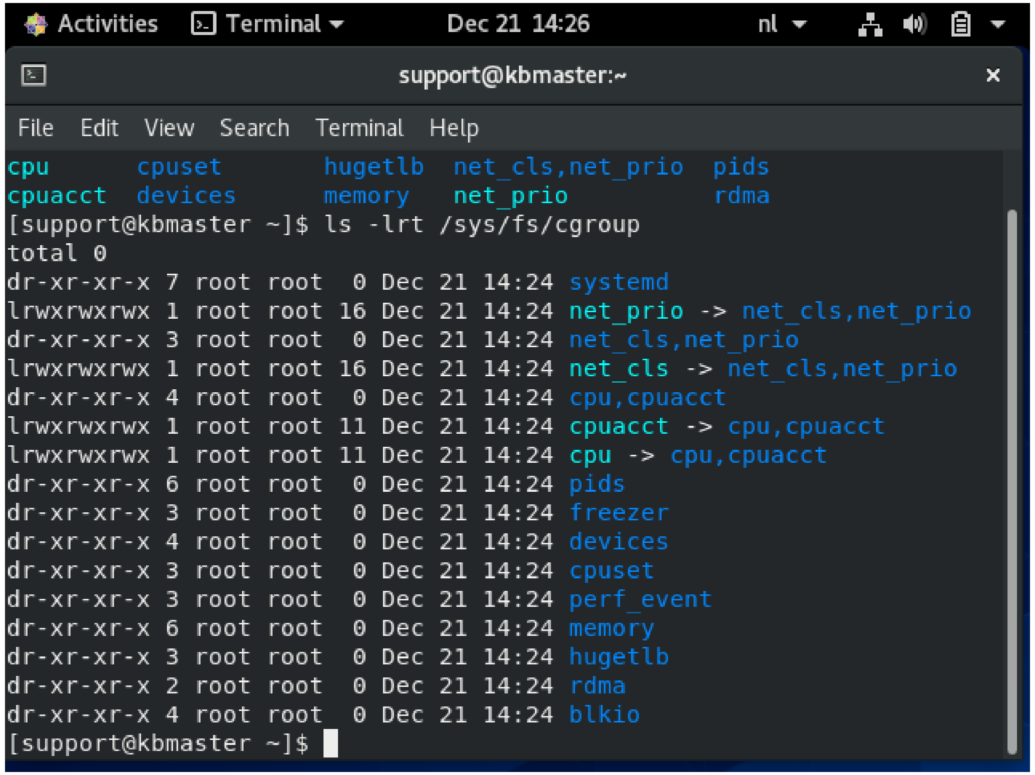
1. **Mount namespace (mnt)** – Linux 2.4.19: isolate the set of mount points that a process sees
2. **Interprocess Communication Namespace (ipc)** - Linux 2.6.19 : isolate message queues, semaphores, shared memory
3. **Unix Time Sharing namespace (UTS)** - Linux 2.6.19: isolates the hostname and NIS domain name of a process
4. **Process ID namespace (pid)** - Linux 2.6.24: isolate the process ID number space
5. **Network namespace (net)** – Linux 2.6.24 to 2.6.29: isolation of network devices, IP addresses, IP routing tables, /proc/net directory, port numbers, etc.
6. **User ID namespace (user)** - Linux 2.6.23 to 3.8: isolate the user and group ID number spaces. A process can have a normal user ID outside a user namespace while having a user ID of 0 (root) inside the namespace.
7. **Control group namespace (cgroup)** – Linux 4.6 March 2016: virtualize the view of a process's cgroups as seen via /proc/[pid]/cgroup and/proc/[pid]/mountinfo.
8. **Time namespace** – Linux 5.6 March 2020: virtualize the values of two system clocks: CLOCK_BOOTTIME and CLOCK_MONOTONIC

Linux namespaces - References

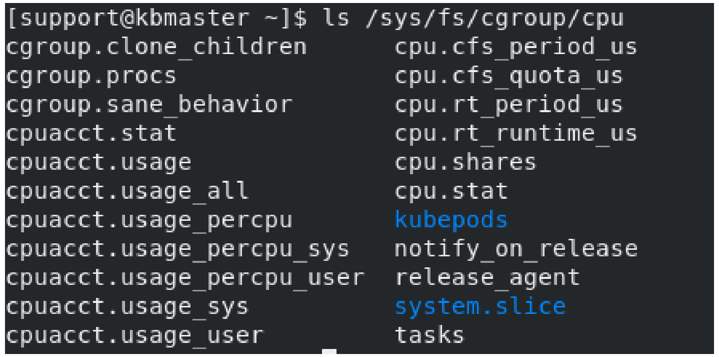
- https://man7.org/linux/man-pages/man7/namespaces.7.html
- Namespaces in operation: https://lwn.net/Articles/531114/
- Control groups namespaces:
  - V1: https://lwn.net/Articles/616099/
  - V2: https://lwn.net/Articles/621006/

cgroups and OOM killer

- cgroups are a feature of the Linux Kernel, used to limit the usage of resources by processes. They are used by systemd, Docker, Kubernetes and others.
- OOM killer is another feature of the Linuk kernel, responsible for killing processes when memory gets exhausted
- cgroups can be manipulated by interacting with the filesystem rooted at:
  /sys/fs/cgroup
- Every type of resource has a different cgroup controller such as:
  /sys/fs/cgroup/pids
  /sys/fs/cgroup/cpu
  /sys/fs/cgroup/memory
- cgroup v1 has a hierarchy per resource (as shown)
- Every PID is represented once in each hierarchy (shown in the tasks file of each cgroup)
- cgroup v2 has cgroups at the top level and resources per cgroup



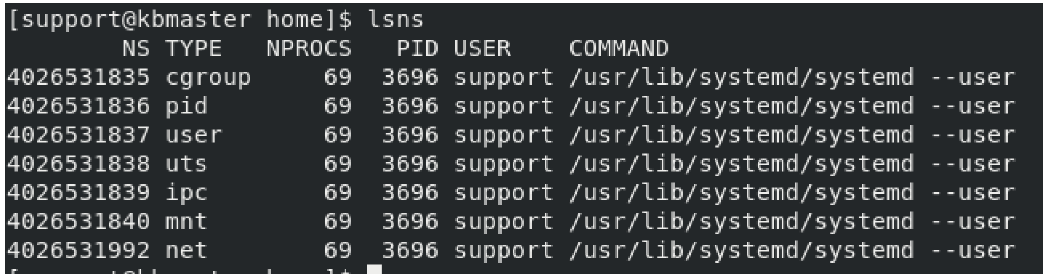
```
cpu      cpuset       hugetlb  net_cls,net_prio  pids
cpuacct  devices      memory   net_prio          rdma
[support@kbmaster ~]$ ls -lrt /sys/fs/cgroup
total 0
dr-xr-xr-x 7 root root  0 Dec 21 14:24 systemd
lrwxrwxrwx 1 root root 16 Dec 21 14:24 net_prio -> net_cls,net_prio
dr-xr-xr-x 3 root root  0 Dec 21 14:24 net_cls,net_prio
lrwxrwxrwx 1 root root 16 Dec 21 14:24 net_cls -> net_cls,net_prio
dr-xr-xr-x 4 root root  0 Dec 21 14:24 cpu,cpuacct
lrwxrwxrwx 1 root root 11 Dec 21 14:24 cpuacct -> cpu,cpuacct
lrwxrwxrwx 1 root root 11 Dec 21 14:24 cpu -> cpu,cpuacct
dr-xr-xr-x 6 root root  0 Dec 21 14:24 pids
dr-xr-xr-x 3 root root  0 Dec 21 14:24 freezer
dr-xr-xr-x 4 root root  0 Dec 21 14:24 devices
dr-xr-xr-x 3 root root  0 Dec 21 14:24 cpuset
dr-xr-xr-x 3 root root  0 Dec 21 14:24 perf_event
dr-xr-xr-x 6 root root  0 Dec 21 14:24 memory
dr-xr-xr-x 3 root root  0 Dec 21 14:24 hugetlb
dr-xr-xr-x 2 root root  0 Dec 21 14:24 rdma
dr-xr-xr-x 4 root root  0 Dec 21 14:24 blkio
[support@kbmaster ~]$
```



```
[support@kbmaster ~]$ ls /sys/fs/cgroup/cpu
cgroup.clone_children      cpu.cfs_period_us
cgroup.procs               cpu.cfs_quota_us
cgroup.sane_behavior       cpu.rt_period_us
cpuacct.stat               cpu.rt_runtime_us
cpuacct.usage              cpu.shares
cpuacct.usage_all          cpu.stat
cpuacct.usage_percpu       kubepods
cpuacct.usage_percpu_sys   notify_on_release
cpuacct.usage_percpu_user  release_agent
cpuacct.usage_sys          system.slice
cpuacct.usage_user         tasks
```

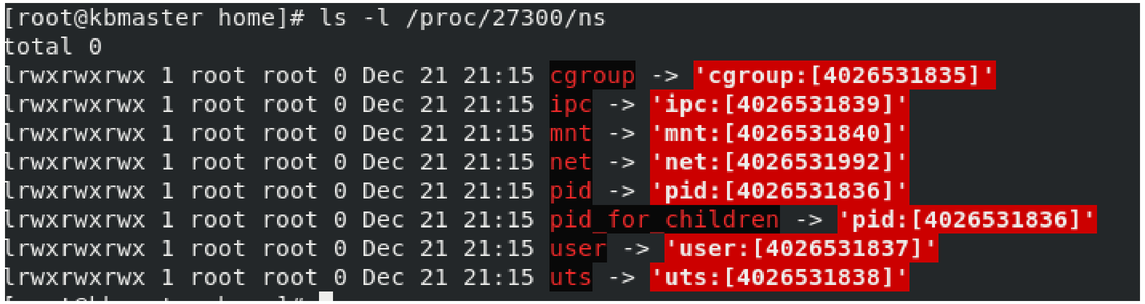
More on Linux namespaces

- If you have installed the util-linux package you can see Linux namespaces using the lsns command



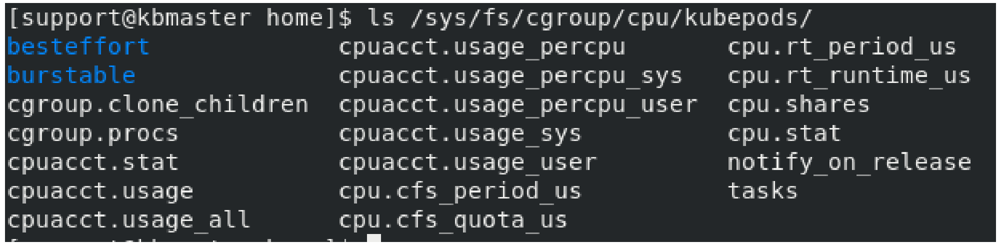
```
[support@kbmaster home]$ lsns
        NS TYPE   NPROCS   PID USER    COMMAND
4026531835 cgroup     69  3696 support /usr/lib/systemd/systemd --user
4026531836 pid        69  3696 support /usr/lib/systemd/systemd --user
4026531837 user       69  3696 support /usr/lib/systemd/systemd --user
4026531838 uts        69  3696 support /usr/lib/systemd/systemd --user
4026531839 ipc        69  3696 support /usr/lib/systemd/systemd --user
4026531840 mnt        69  3696 support /usr/lib/systemd/systemd --user
4026531992 net        69  3696 support /usr/lib/systemd/systemd --user
```

- For a given process, you can just list the contents of /proc/PID/ns
  while being logged as root



```
[root@kbmaster home]# ls -l /proc/27300/ns
total 0
lrwxrwxrwx 1 root root 0 Dec 21 21:15 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Dec 21 21:15 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 Dec 21 21:15 mnt -> 'mnt:[4026531840]'
lrwxrwxrwx 1 root root 0 Dec 21 21:15 net -> 'net:[4026531992]'
lrwxrwxrwx 1 root root 0 Dec 21 21:15 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Dec 21 21:15 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Dec 21 21:15 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Dec 21 21:15 uts -> 'uts:[4026531838]'
```

- On a system with Kubernetes installed, you can see: /sys/fs/cgroup/cpu/kubepods
  This is a cgroup of type cpu and it contains two cgroups:
  besteffort
  burstable
  These are Quality Of Service (QoS) for Kubernetes pods (there is also guaranteed)
  https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/



```
[support@kbmaster home]$ ls /sys/fs/cgroup/cpu/kubepods/
besteffort             cpuacct.usage_percpu       cpu.rt_period_us
burstable              cpuacct.usage_percpu_sys   cpu.rt_runtime_us
cgroup.clone_children  cpuacct.usage_percpu_user  cpu.shares
cgroup.procs           cpuacct.usage_sys          cpu.stat
cpuacct.stat           cpuacct.usage_user         notify_on_release
cpuacct.usage          cpu.cfs_period_us          tasks
cpuacct.usage_all      cpu.cfs_quota_us
```

Union File System (UnionFS)

UnionFS is a File System that creates a union of separate directories known as branches.
There a few variants: AUFS, OverlayFS.
OverlayFS uses an **upper** and **lower** filesystem (directory tree).
When the same name exists in both, the object in the ‘upper’ tree is visible while the object in the ‘lower’ filesystem is either **hidden** or, in the case of directories, **merged** with the ‘upper’ object.
The lower filesystem can be any filesystem supported by Linux and does not need to be writable. The upper filesystem will normally be writable.

Usage in Docker images:
Docker Engine can use multiple UnionFS variants, including AUFS, and OverlayFS.
Docker provides two storage drivers for OverlayFS: **overlay**, and **overlay2**.
Overlay2 is more efficient than overlay in terms of inode utilization.
To use overlay2, you need version 4.0 or higher of the Linux kernel, or RHEL or CentOS using version 3.10.0-514 and above.

“The **overlay2 driver natively supports up to 128 lower OverlayFS layers**. This capability provides better performance for layer-related Docker commands such as `docker build` and `docker commit`, and consumes fewer inodes on the backing filesystem.”

A little Docker history

- **DotCloud Inc**. was launched in 2011 as a multi-language Platform as a Service (PaaS), supporting MongoDB, Redis, MySQL.
- It released Docker as open source project (Apache 2 license) in 2013 and it changed the company name to **Docker Inc**.
- Docker initially ran on **LXC** but in v. 0.9 it added **libcontainer**, a library written in Go https://www.docker.com/blog/docker-0-9-introducing-execution-drivers-and-libcontainer/
  “Thanks to libcontainer, Docker out of the box can now manipulate namespaces, control groups, capabilities, apparmor profiles, network interfaces and firewalling rules – all in a consistent and predictable way, and without depending on LXC”.
- In 2015, Docker donated the **container image specification** and runtime code now known as **runc**, to the **Open Container Initiative (OCI).**
- In 2017 Docker donated the **containerd** project to CNCF. containerd is a container runtime that leverages runc and is the core container runtime of the **Docker Engine**.
- **Docker Desktop** currently includes **Kubernetes.**
- In 2021 Docker ended free **Docker Desktop** use for business customers on Windows and MacOs.Linux usage remains free. Docker replaced its Free plan with a Personal plan for individuals (including Education). The Personal Plan is free of charge: https://www.docker.com/pricing/

A little Kubernetes history

- Kubernetes is an open source project for **orchestrating containers** that evolved from the **Google Borg project**.
- Kubernetes is written in Go https://golang.org/
- Kubernetes is a distributed system made of a (group of) master node(s) and one or more worker nodes, that run containers grouped in pods.

- **Kubernetes** in Greek means **helmsman**: person who steers a ship.
  - Acronym: k8s (start with first letter, end with last letter and number of letters in between)

- Kubernetes was open sourced by Google in 2014.
- Kubernetes v1.0 was released on July 21, 2015 and at the same time Google partnered with the Linux Foundation to create the **Cloud Native Computing Foundation** (https://cncf.io) and donated Kubernetes to it.
- Kubernetes is a **Graduated** project of the **CNCF.**
- Kubernetes version 1.26 was released in December 2022 (https://kubernetes.io/releases/).

- **Helm**, https://helm.sh/ is the Kubernetes Package Manager, released in 2016. Version 3 shipped in 2019 with significant changes.

OCI: Open Container Initiative

https://opencontainers.org/
The OCI was launched on June 22nd 2015 by Docker, CoreOS and other leaders in the container industry.
It provides two specifications:

Container Runtime specification
https://github.com/opencontainers/runtime-spec/blob/master/spec.m

- defines how:
  - to download an OCI Image
  - to unpack that image into an OCI Runtime “filesystem bundle”
  - the OCI Runtime runs the “filesystem bundle”

Image Format Specification
https://github.com/opencontainers/image-spec/blob/master/spec.md

- defines how to create an OCI Image, typically using a build system, resulting in:
  - an image manifest (metadata about the contents and dependencies of the image)
  - a filesystem (layer) serialization
  - an image configuration (application arguments, environments, etc)

Runc: CLI tool written in Go for spawning and running containers according to the OCI specification
https://github.com/opencontainers/runc

Container Runtimes

Kubernetes 1.5 (2016) introduces the CRI (Container Runtime Interface), an API that enables kubelet to use different container runtimes without having to recompile. It is implemented using gRPC as remote procedure call API and protocol buffers as serialization mechanism.

Container runtimes supported by Kubernetes as of Dec 2020:

- containerd
- CRI-O
- docker

Note on docker runtime:

The docker runtime will be deprecated in Kubernetes 1.20 and removed in Kubernetes 1.22.
The docker runtime is based on containerd, but it also adds extra features around it, which make it not compliant with the CRI. For this reason, Kubernetes needs to add **a docker-shim** to be able to use the docker runtime.
The docker-shim will be removed, so that the docker runtime will not be usable anymore.
Existing docker images will remain usable in Kubernetes and the docker build command will continue to be supported to create such images.

https://kubernetes.io/docs/setup/production-environment/container-runtimes/

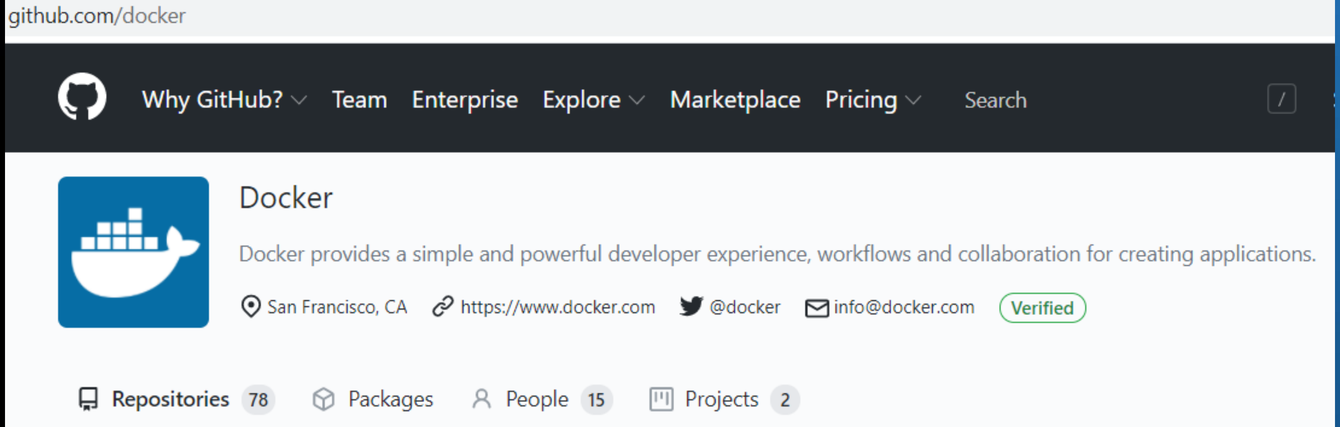
References

- https://en.wikipedia.org/wiki/Comparison_of_platform_virtualization_software
- A sysadmin's guide to containers https://opensource.com/article/18/8/sysadmins-guide-containers
- Linux Container Primitives: cgroups, namespaces, and more! (AWS) https://www.youtube.com/watch?v=x1npPrzyKfs
- Kubernetes On Cgroup v2 - Giuseppe Scrivano, Red Hat https://www.youtube.com/watch?v=u8h0e84HxcE
- cgroupv2: Linux's new unified control group hierarchy (QCON London 2017) https://www.youtube.com/watch?v=ikZ8_mRotT4
- Kernel Korner - Unionfs: Bringing Filesystems Together (2004) https://www.linuxjournal.com/article/7714
- Docker: Use the OverlayFS storage driver https://docs.docker.com/storage/storagedriver/overlayfs-driver/
- Overlay Filesystem https://www.kernel.org/doc/html/latest/filesystems/overlayfs.html?highlight=overlayfs

- Kubernetes:
  - Burns, Brendan, et al. "Borg, omega, and kubernetes." *Queue* 14.1 (2016): 70-93. https://dl.acm.org/doi/pdf/10.1145/2898442.2898444
  - Matt Asay, The secret to Kubernetes' success (2020) https://www.infoworld.com/article/3530379/the-secret-to-kubernetes-success.html
  - Burns, Brendan, Joe Beda, and Kelsey Hightower. *Kubernetes*. Dpunkt, 2018. http://www.academia.edu/download/61644710/Livro_Kubernetes20191231-77912-r0gj39.pdf

HCLSoftware

Introduction to Docker



github.com/docker

Docker

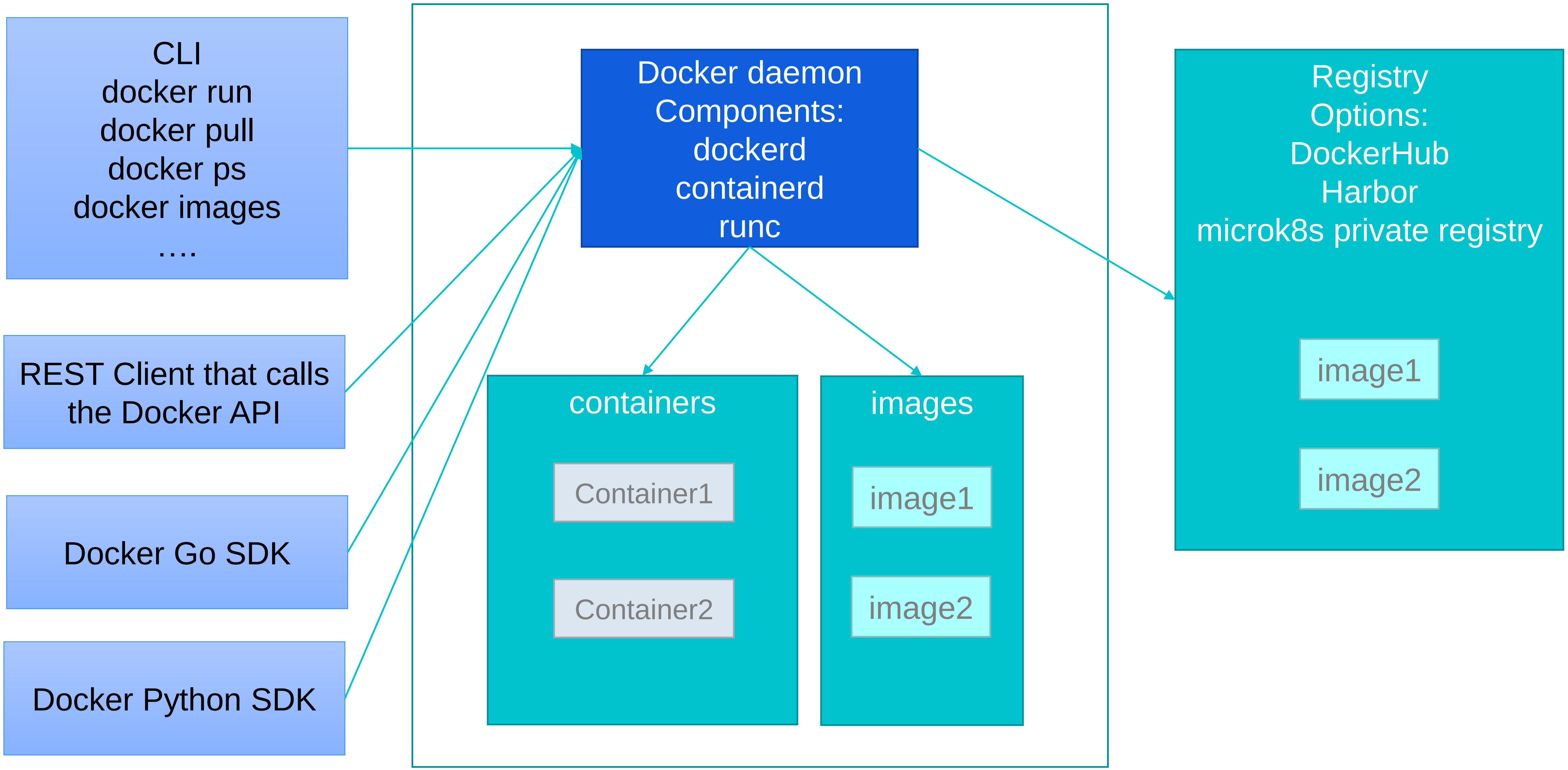
Docker provides a simple and powerful developer experience, workflows and collaboration for creating applications.

San Francisco, CA https://www.docker.com @docker info@docker.com Verified

Docker ecosystem

- **Docker Engine**:
  - client-server application with:
    - server with a long-running daemon process **dockerd**. Note that dockerd requires **containerd**.
    - APIs that programs can use to talk to the Docker daemon.
    - command line interface (CLI) client **docker**.
  - swarm mode for natively managing a cluster of Docker Engines called a **swarm**
- **Docker Compose**:
  - tool for defining and running multi-container Docker applications using yaml files
- **Docker Desktop**:
  - application for Mac or Windows environment that enables users to build and share containerized applications.
  - includes Docker Engine, Docker CLI client, Docker Compose, Notary, Kubernetes, and Credential Helper.
- **Docker Hub**:
  - service for finding and sharing container images
  - repository of container images with contributions from community developers, open source projects and independent software vendors (ISV)
  - free public repositories for storing and sharing images and subscription plans for private repositories.

Docker Engine Architecture



CLI
docker run
docker pull
docker ps
docker images
….

REST Client that calls the Docker API

Docker Go SDK

Docker Python SDK

Docker daemon
Components:
dockerd
containerd
runc

containers

Container1

Container2

images

image1

image2

Registry
Options:
DockerHub
Harbor
microk8s private registry

image1

image2

Docker command line and REST API

As a user, you need to know the various commands exposed as parameters of the docker executable.
https://docs.docker.com/engine/reference/commandline/docker/
The main ones are:

- docker container: perform various operations con a container
- docker pull: pull an image from a registry
- docker push: push an image to a registry
- docker build: build an image from a Dockerfile
- docker run: run a command in a new container (creates a container)
- docker images: list the images available on the Docker host
- docker ps: list the existing containers (running or terminated)
- docker exec: run a new command in a docker container
- docker tag: create a target image that refers to source image
- docker login/logout: login/lout to/from a docker registry
- docker start/stop/rename: start/stop/rename a container
- docker volume: manage persistent storage for containers
- docker logs: get the logs of a container

The Docker Engine REST API is documented here: https://docs.docker.com/engine/api/v1.41/
The Docker Python and Go SDKs are documented here: https://docs.docker.com/engine/api/sdk/

Running your first docker container: docker run

```
lara@kube-master:~$ sudo docker run hello-world
[sudo] password for lara:
Unable to find image 'hello-world:latest'
locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest:
sha256:1a523af650137b8accdaed439c17d684df61ee4d
74feac151b5b337bd29e7eec
Status: Downloaded newer image for hello-
world:latest

Hello from Docker!
This message shows that your installation
appears to be working correctly.
To generate this message, Docker took the
following steps:
 1. The Docker client contacted the Docker
daemon.
 2. The Docker daemon pulled the "hello-world"
image from the Docker Hub. (amd64)
 3. The Docker daemon created a new container
from that image which runs the executable that
produces the output you are currently reading.
 4. The Docker daemon streamed that output to
the Docker client, which sent it to your
terminal.

To try something more ambitious, you can run an
Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with
a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Listing images with docker images

```
lara@kube-master:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
lara@kube-master:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
hello-world         latest              bf756fb1ae65        11 months ago
13.3kB
lara@kube-master:~$ sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.
…..
```

After you have run the container, the image remains available in the local registry and it can reused by other containers.
If you run the container again, the message about downloading the image no longer appears.

Removing images with docker rm

```
lara@kube-master:~$ sudo docker image rm hello-world
Error response from daemon: conflict: unable to remove repository reference "hello-
world" (must force) - container 4501e4f626c8 is using its referenced image
bf756fb1ae65
```

docker rm <image-name> can be used to remove an image from the registry.

If you try to remove the image from the previous example, you cannot because it is still used by some container.

But when you run docker ps, you don’t see any containers because they have all stopped executing after printing the message.

Listing containers with docker ps

```
lara@kube-master:~$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                     PORTS               NAMES
202e0f676664        hello-world         "/hello"            2 minutes ago       Exited (0) 2 minutes ago                       busy_tereshkova
4501e4f626c8        hello-world         "/hello"            7 minutes ago       Exited (0) 7 minutes ago                       blissful_lalande
```

If you run ‘docker ps –a’, you can see all containers.

The containers that stopped execution still exist and hold a reference to the image.

Deleting containers with docker container rm and images with docker image rm

```
lara@kube-master:~$ sudo docker container rm 202e0f676664
202e0f676664
lara@kube-master:~$ sudo docker container rm 4501e4f626c8
4501e4f626c8
lara@kube-master:~$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
lara@kube-master:~$ sudo docker image rm hello-world
Untagged: hello-world:latest
Untagged: hello-
world@sha256:1a523af650137b8accdaed439c17d684df61ee4d74feac151b5b337bd29e7eec
Deleted: sha256:bf756fb1ae65adf866bd8c456593cd24beb6a0a061dedf42b26a993176745f6b
Deleted: sha256:9c27e219663c25e0f28493790cc0b88bc973ba3b1686355f221c38a36978ac63
lara@kube-master:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
```

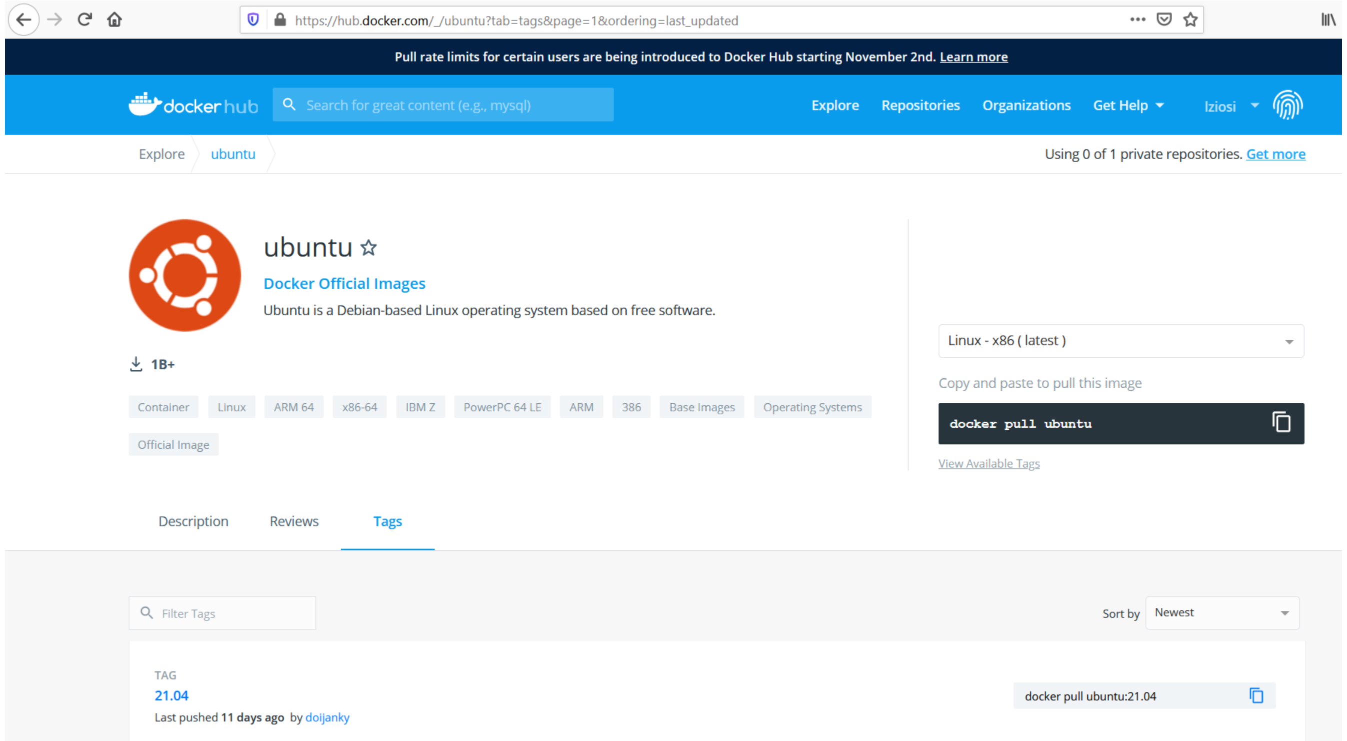
docker pull from Docker Hub

- The docker pull command allows you to download an image from a repository.

- By default images are pulled from: https://hub.docker.com/

- Example:
  docker pull ubuntu:21.04

- Note that Docker has introduced limits to the pull frequency:
  - 100 container image requests per six hours for anonymous usage
  - 200 container image requests per six hours for free Docker accounts.



https://hub.docker.com/_/ubuntu?tab=tags&page=1&ordering=last_updated

Pull rate limits for certain users are being introduced to Docker Hub starting November 2nd. Learn more

Using 0 of 1 private repositories. Get more

ubuntu ☆

Docker Official Images

Ubuntu is a Debian-based Linux operating system based on free software.

1B+

Linux - x86 (latest)

Copy and paste to pull this image

```
docker pull ubuntu
```

TAG
21.04
Last pushed 11 days ago by doijanky

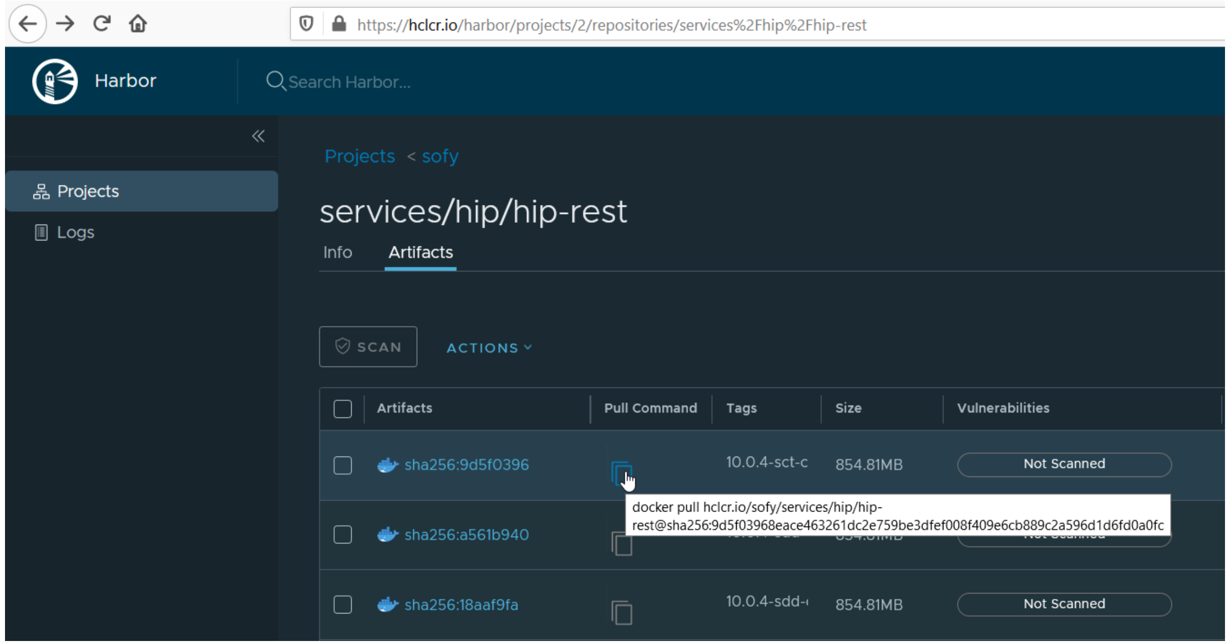
```
docker pull ubuntu:21.04
```

docker pull from a private or other registry

- You can specify a private or other commercial registry when using the pull command.
- For example HCL has established a registry at https://hclcr.io using Harbor:

```
docker pull hclcr.io/sofy/services/hip/hip-rest@sha256:9d5f03968eace463261dc2e759be3dfef008f409e6cb889c2a596d1d6fd0a0fc
```

- Harbor is an open source registry
- Harbor is a CNCF graduated project
- https://goharbor.io/
- Harbor can scan images for security vulnerabilities
- Harbor can now host both images and Helm Charts



https://hclcr.io/harbor/projects/2/repositories/services%2Fhip%2Fhip-rest

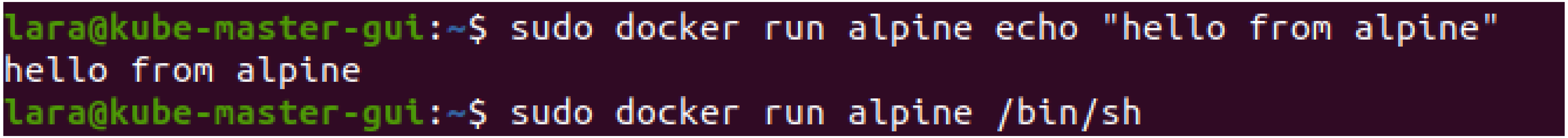
services/hip/hip-rest

□	Artifacts	Pull Command	Tags	Size	Vulnerabilities
□	sha256:9d5f0396		10.0.4-sct-c	854.81MB	Not Scanned
□	sha256:a561b940		[illegible]	[illegible]	[illegible]
□	sha256:18aaf9fa		10.0.4-sdd-(	854.81MB	Not Scanned

docker pull hclcr.io/sofy/services/hip/hip-rest@sha256:9d5f03968eace463261dc2e759be3dfef008f409e6cb889c2a596d1d6fd0a0fc

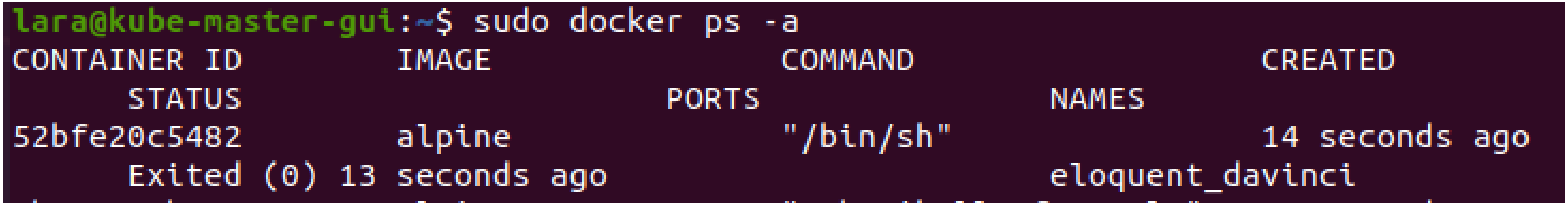
Executing a shell inside a container 1/2

The docker run command accepts a command after the image name.
It executes that command inside the container.
If the command is a shell, then it returns immediately and the container exits, which is not useful.



```
lara@kube-master-gui:~$ sudo docker run alpine echo "hello from alpine"
hello from alpine
lara@kube-master-gui:~$ sudo docker run alpine /bin/sh
```

Executing docker ps –a you can see all containers, including those that have exited but have not been removed.

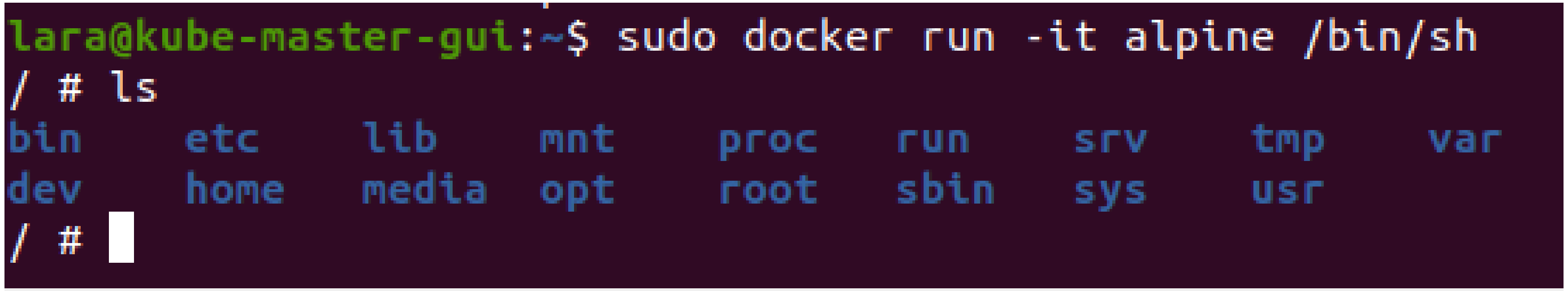


```
lara@kube-master-gui:~$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND                CREATED
     STATUS                     PORTS               NAMES
52bfe20c5482        alpine              "/bin/sh"              14 seconds ago
     Exited (0) 13 seconds ago                      eloquent_davinci
```

Executing a shell inside a container 2/2

To execute the shell inside the container so that you can then execute commands inside that shell, you need to pass the parameters “–it” to the docker run command.
Then you will generally see different command prompt, which means that you are now executing commands inside the container.
When you type “exit”, you will return to the command prompt of the host operating system.
In the example below, the command “ls” is executed inside the container.
Remember that any changes you make inside the container will be wiped away when the container exits, unless you take extra steps to either:

- Save data to persistent storage
- Create a new container image that includes the changes



```
lara@kube-master-gui:~$ sudo docker run -it alpine /bin/sh
/ # ls
bin    etc    lib    mnt    proc   run    srv    tmp    var
dev    home   media  opt    root   sbin   sys    usr
/ #
```

Executing a container in detached mode and publishing ports

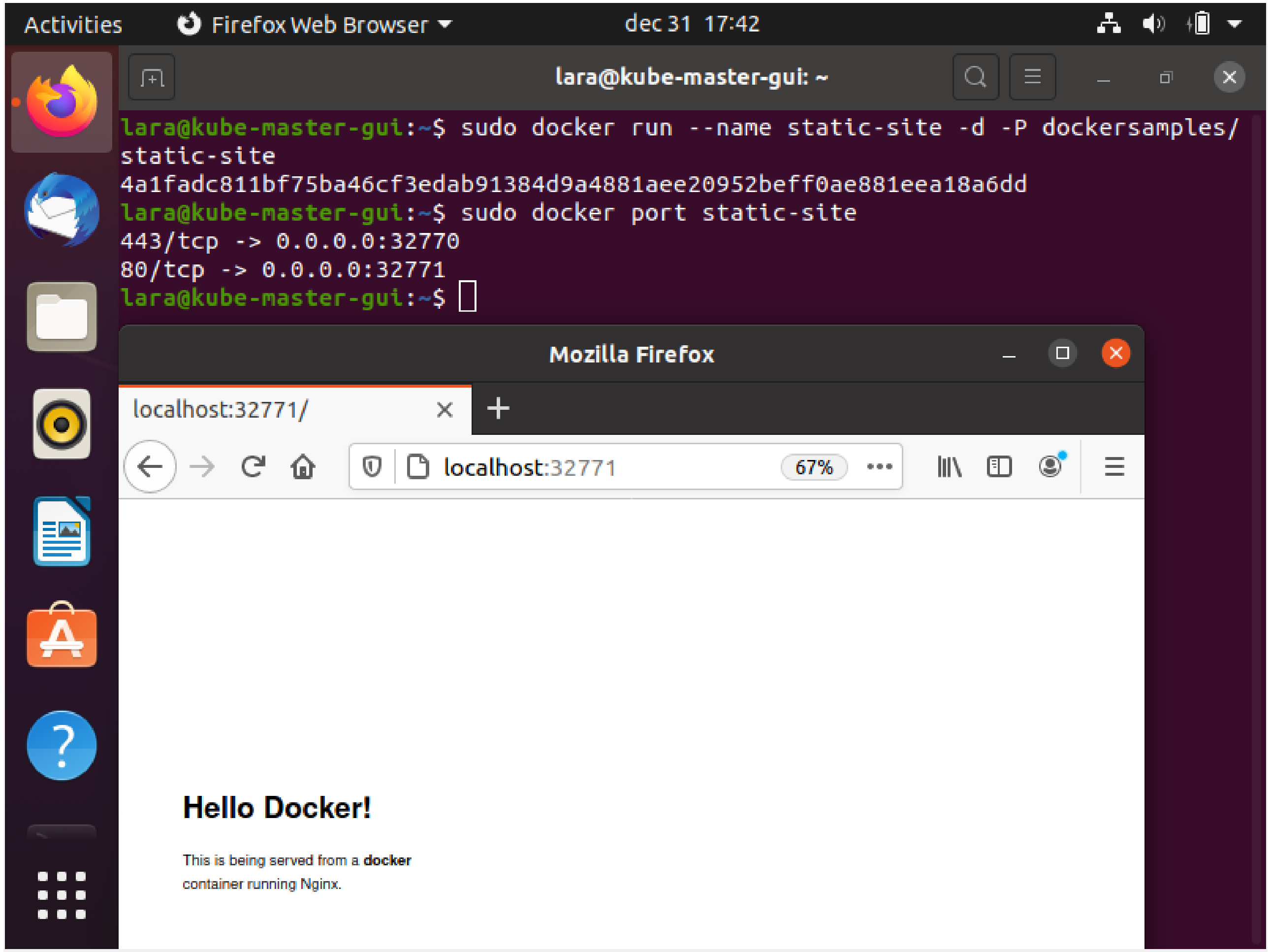
- The “-d” option of docker run causes the container to be detached from the shell of the host operating system. This means that the standard input, output, error of the container don’t flow to the shell of the host operating system. You can use **docker attach** to attach back.
- The “-P” option of docker run causes random ports to be assigned to the exposed container ports.
- The **docker port** command prints the port mappings. The output of docker port is of the form:

  port-in-container/protocol -> ip-address-of-host:port-of-host

  If the exposed port is for HTTP or HTTPS, you can then load the page on the host using the URL:

  http(s)://ip-address-of-host:port-of-host

  0.0.0.0 means any network interface



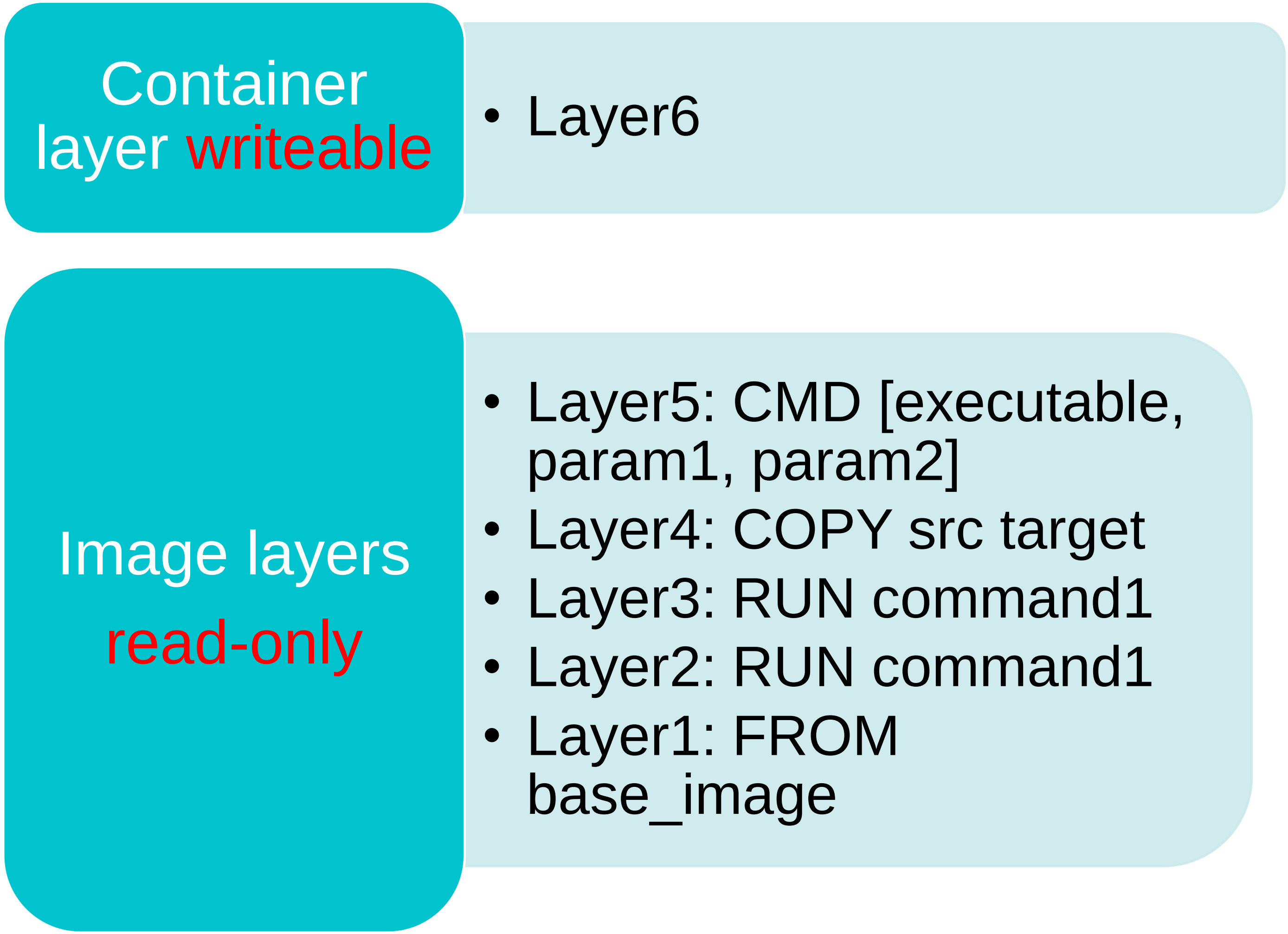
```
lara@kube-master-gui:~$ sudo docker run --name static-site -d -P dockersamples/
static-site
4a1fadc811bf75ba46cf3edab91384d9a4881aee20952beff0ae881eea18a6dd
lara@kube-master-gui:~$ sudo docker port static-site
443/tcp -> 0.0.0.0:32770
80/tcp -> 0.0.0.0:32771
lara@kube-master-gui:~$ 
```

Hello Docker!

This is being served from a **docker** container running Nginx.

Docker images: Layered architecture

- Docker images are built out of layers
- Images can be created from declarative files called dockerfiles
- Most commands in a dockerfile result in a new layer being added to the image
- Examples of commands are:
  - FROM, RUN, COPY, CMD
- The image is created by running the docker build command
- After docker builds the image, all the layers are read-only
- When docker runs a container, it adds a new writeable layer to the read-only image
- Any files created or modified in the container layer are lost when the container is stopped



Container layer writeable

- Layer6

Image layers read-only

- Layer5: CMD [executable, param1, param2]
- Layer4: COPY src target
- Layer3: RUN command1
- Layer2: RUN command1
- Layer1: FROM base_image

Example of Dockerfile

https://github.com/docker/labs/blob/master/beginner/flask-app/Dockerfile

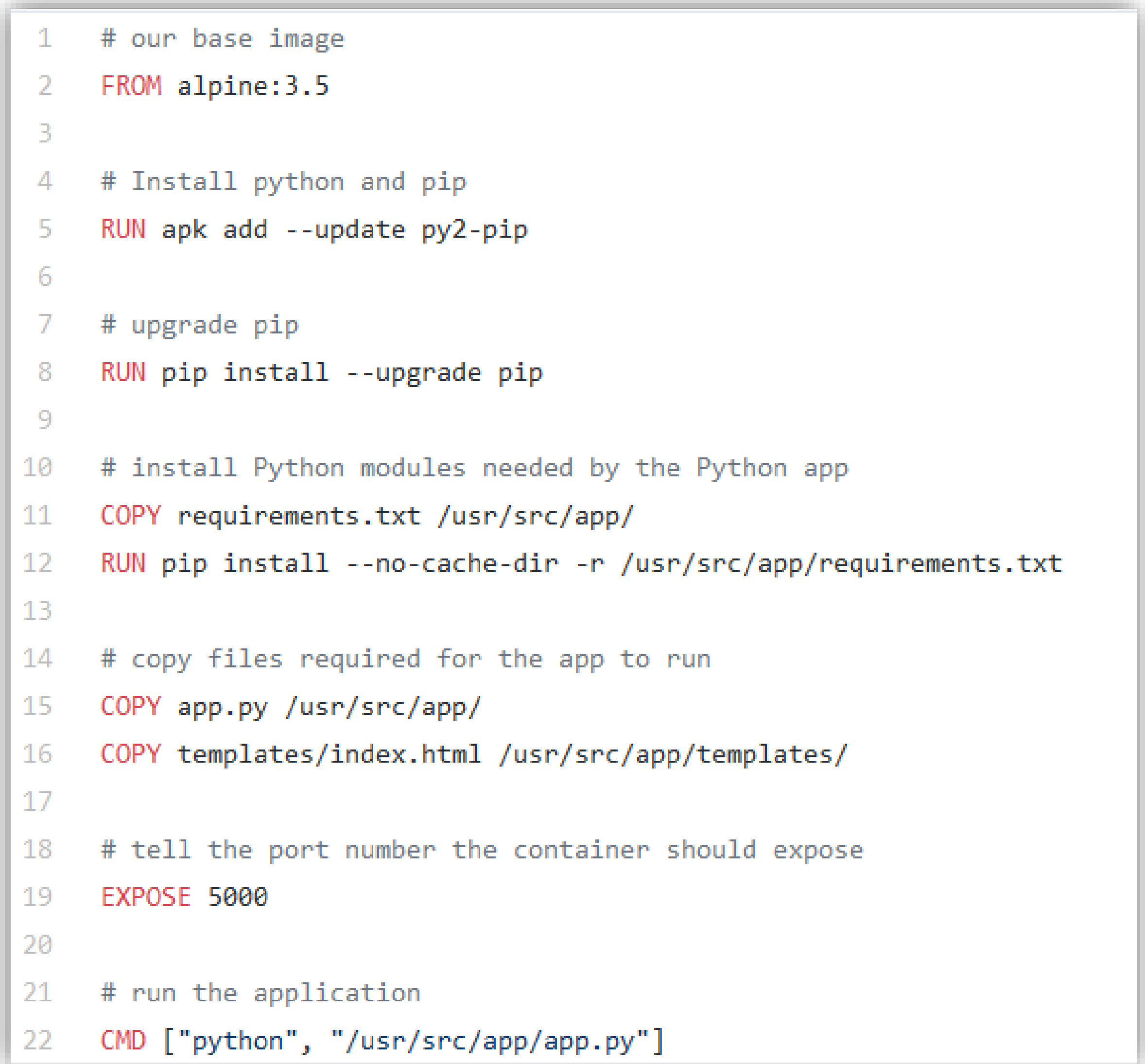
A Dockerfile must begin with the FROM statement that can pull a base image from a registry.

Every RUN command executes a series of commands that creates a new layer.

The COPY command copies external files into the container. It also creates a layer.

The EXPOSE command indicates which port the container listens on, but it does not actually publish the port.

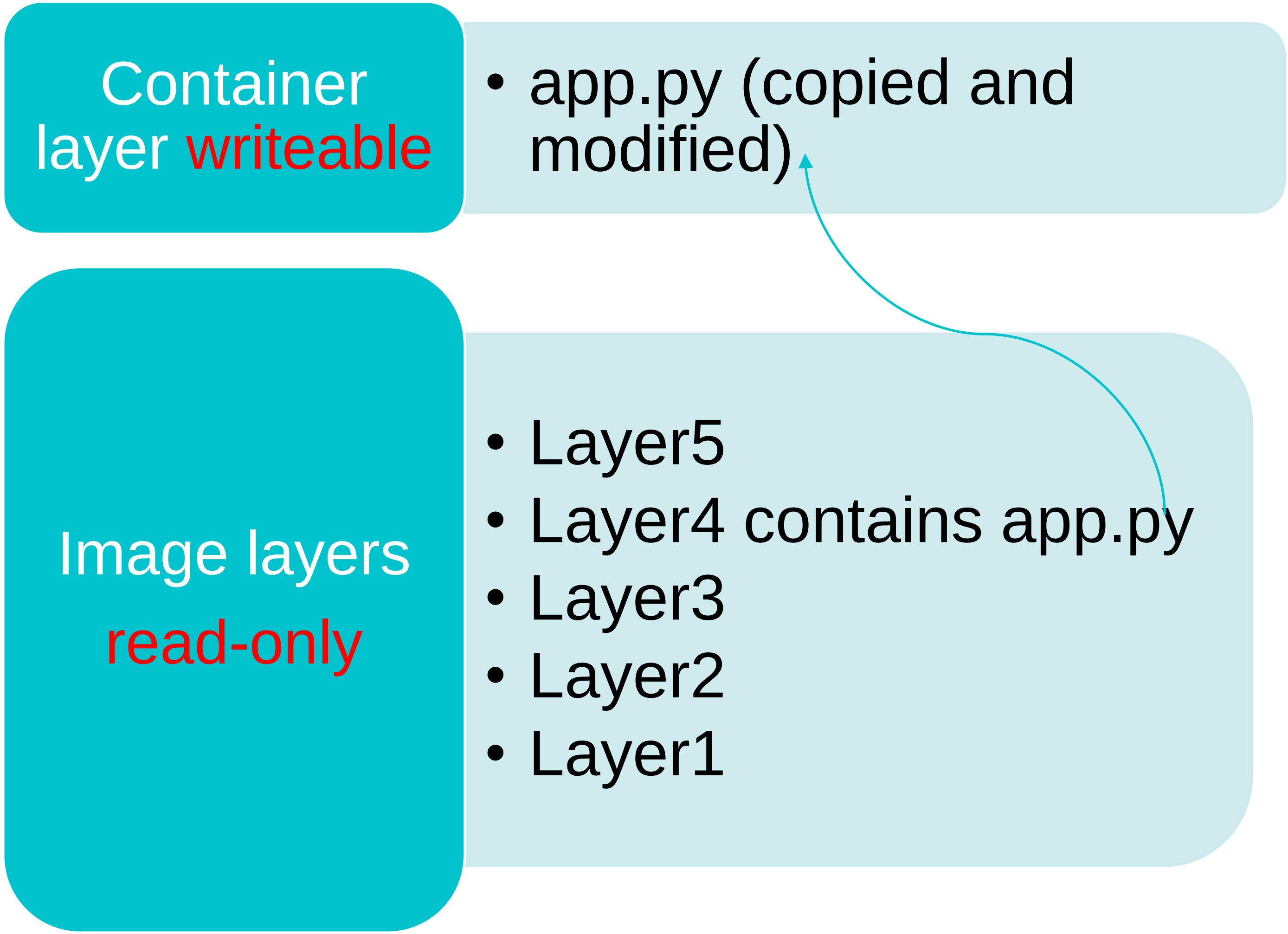
The CMD command provides defaults for the container at runtime. It may include the executable and parameters.
If the executable is omitted, then you must add ENTRYPOINT.
https://docs.docker.com/engine/reference/builder/



```
# our base image
FROM alpine:3.5

# Install python and pip
RUN apk add --update py2-pip

# upgrade pip
RUN pip install --upgrade pip

# install Python modules needed by the Python app
COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt

# copy files required for the app to run
COPY app.py /usr/src/app/
COPY templates/index.html /usr/src/app/templates/

# tell the port number the container should expose
EXPOSE 5000

# run the application
CMD ["python", "/usr/src/app/app.py"]
```

COPY-ON-WRITE (CoW) strategy

- **Read access**: if any layer needs read access to a file in a lower layer, it just uses it directly.
- **Write access**: if any layer needs to modify a file in a lower layer the file is copied into the layer that needs to use it, and then it is modified. This can happen during the image build process or while the container is running.
- **Layer reuse**: when you pull an image from a repository, all the layers are pulled separately. If a layer already exists in the local Docker storage, it is used from the cache without pulling it again.



Container layer writeable

- app.py (copied and modified)

Image layers

read-only

- Layer5
- Layer4 contains app.py
- Layer3
- Layer2
- Layer1

Building images with docker build 1/2

https://docs.docker.com/engine/reference/commandline/build/

```
lara@kube-master:~/flask-app$ sudo docker build -t lara/myfirstapp .
Sending build context to Docker daemon   7.68kB
Step 1/8 : FROM alpine:3.5
3.5: Pulling from library/alpine
8cae0e1ac61c: Pull complete
Digest: sha256:66952b313e51c3bd1987d7c4ddf5dba9bc0fb6e524eed2448fa660246b3e76ec
Status: Downloaded newer image for alpine:3.5
 ---> f80194ae2e0c
Step 2/8 : RUN apk add --update py2-pip
 ---> Running in 345d7e2f357d
fetch http://dl-cdn.alpinelinux.org/alpine/v3.5/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.5/community/x86_64/APKINDEX.tar.gz
(1/12) Installing libbz2 (1.0.6-r5)
....
(12/12) Installing py2-pip (9.0.0-r1)
Executing busybox-1.25.1-r2.trigger
OK: 62 MiB in 23 packages
Removing intermediate container 345d7e2f357d
 ---> 6232533d3c9d
Step 3/8 : COPY requirements.txt /usr/src/app/
 ---> c15a262549c5
```

Building images with docker build 2/2

```
Step 4/8 : RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt
 ---> Running in 34bcb09ecb41
Collecting Flask==0.10.1 (from -r /usr/src/app/requirements.txt (line 1))
  Downloading
https://files.pythonhosted.org/packages/db/9c/149ba60c47d107f85fe52564133348458f093dd5e6b57a5b60ab9ac517bb/Flask-
0.10.1.tar.gz (544kB)
.....
Installing collected packages: Werkzeug, MarkupSafe, Jinja2, itsdangerous, Flask
  Running setup.py install for MarkupSafe: started
….'
Successfully installed Flask-0.10.1 Jinja2-2.11.2 MarkupSafe-1.1.1 Werkzeug-1.0.1 itsdangerous-1.1.0
Removing intermediate container 34bcb09ecb41
 ---> de7f9bd822c7
Step 5/8 : COPY app.py /usr/src/app/
 ---> 113a904893e9
Step 6/8 : COPY templates/index.html /usr/src/app/templates/
 ---> e6e6e91c95a2
Step 7/8 : EXPOSE 5000
 ---> Running in da9135d0ea22
Removing intermediate container da9135d0ea22
 ---> f1354c38cd87
Step 8/8 : CMD ["python", "/usr/src/app/app.py"]
 ---> Running in 7ee92078073b
Removing intermediate container 7ee92078073b
 ---> 25e621b1555d
Successfully built 25e621b1555d
Successfully tagged lara/myfirstapp:latest
```

Docker swarm

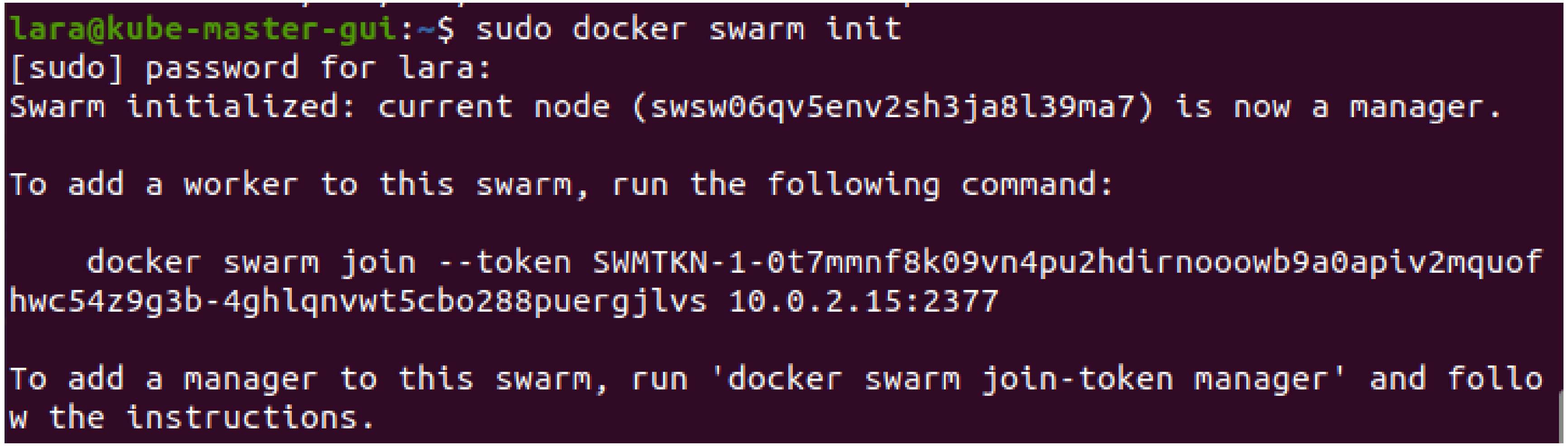
Docker swarm is a specific way of running the docker engine that distributes containers across multiple:

- Manager nodes (can also act as Worked nodes)
- Worker nodes

Docker swarm uses the Raft consensus algorithm to get the managers to agree on the status of data.

The main commands are:

- docker swarm init: initialize the current node ad manager of a new swarm
- docker swarm join: join a swarm as worker or manager depending on the token type provided
- docker swarm leave: used to remove a worker or a manager from a swarm



```
lara@kube-master-gui:~$ sudo docker swarm init
[sudo] password for lara:
Swarm initialized: current node (swsw06qv5env2sh3ja8l39ma7) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-0t7mmnf8k09vn4pu2hdirnooowb9a0apiv2mquof
hwc54z9g3b-4ghlqnvwt5cbo288puergjlvs 10.0.2.15:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follo
w the instructions.
```

Docker compose

Docker compose allows for orchestration of multiple containers, including the setup of the networking between them.

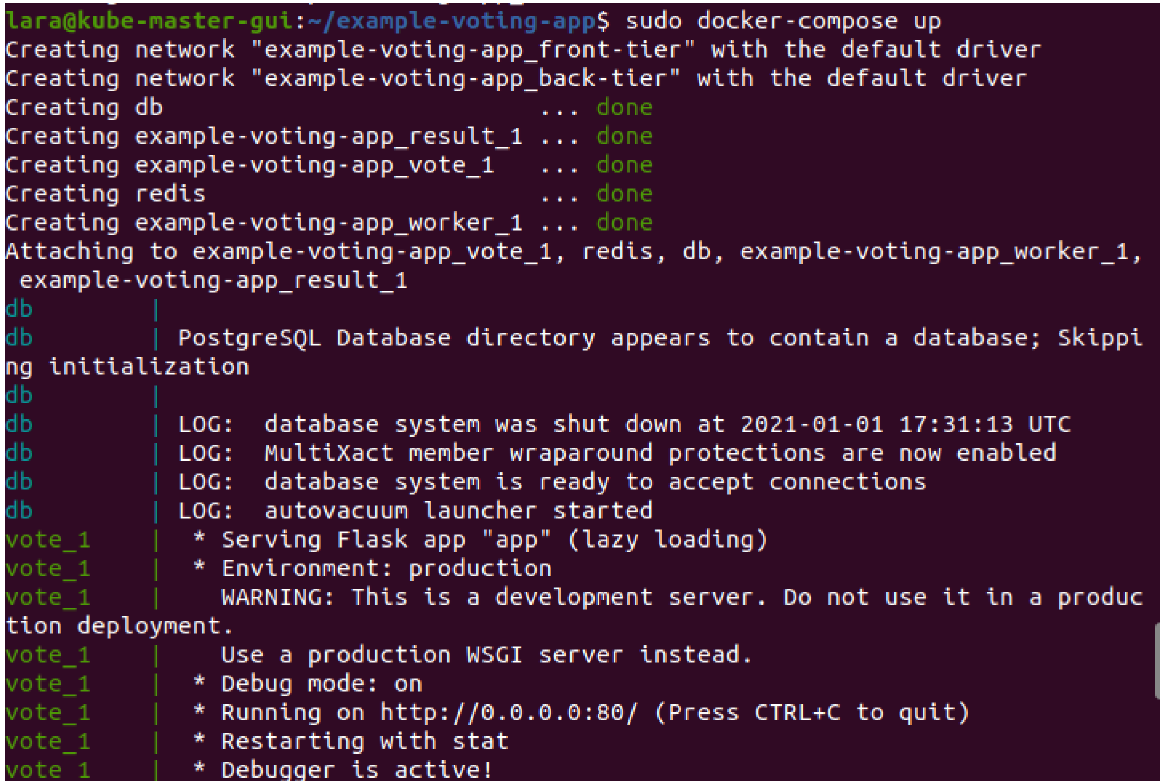
The list of containers is described in a so called docker-compose file that uses the YAML syntax.

Docker compose introduces its own Command line interface with the following main commands:

- **docker-compose up**: creates and runs the containers described by a docker-compose yaml file. Use with –d if you want to run the containers detached, else when you stop the command the containers also stop. Note that all logs of all containers are merged together.
- **docker-compose down**: stops and destroys the containers previously created by up
- **docker-compose start**: starts the containers described by the docker-compose yaml file and previous created by up
- **docker-compose stop**: stop running containers without removing them. They can be started again with start.
- **docker-compose logs**: shows the logs of all the containers
- **docker-compose ps**: lists the containers

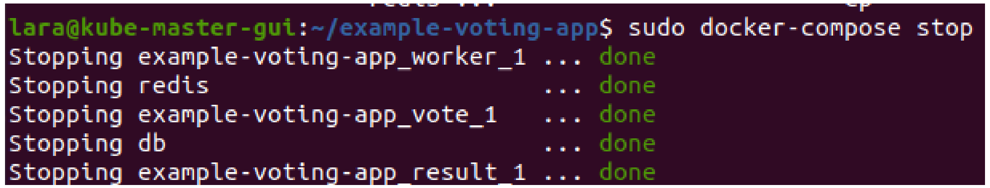
For more details, see: https://docs.docker.com/compose/reference/overview/

Docker compose command examples

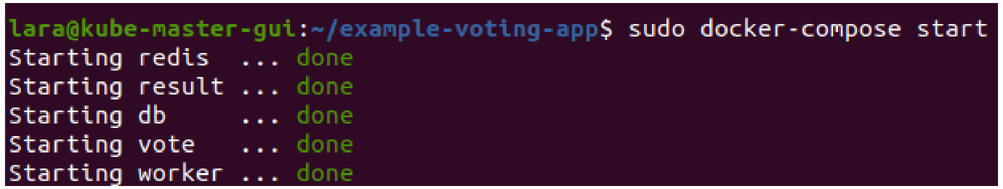


```
lara@kube-master-gui:~/example-voting-app$ sudo docker-compose up
Creating network "example-voting-app_front-tier" with the default driver
Creating network "example-voting-app_back-tier" with the default driver
Creating db                           ... done
Creating example-voting-app_result_1 ... done
Creating example-voting-app_vote_1   ... done
Creating redis                        ... done
Creating example-voting-app_worker_1 ... done
Attaching to example-voting-app_vote_1, redis, db, example-voting-app_worker_1,
 example-voting-app_result_1
db          |
db          | PostgreSQL Database directory appears to contain a database; Skippi
ng initialization
db          |
db          | LOG:  database system was shut down at 2021-01-01 17:31:13 UTC
db          | LOG:  MultiXact member wraparound protections are now enabled
db          | LOG:  database system is ready to accept connections
db          | LOG:  autovacuum launcher started
vote_1      |  * Serving Flask app "app" (lazy loading)
vote_1      |  * Environment: production
vote_1      |    WARNING: This is a development server. Do not use it in a produc
tion deployment.
vote_1      |    Use a production WSGI server instead.
vote_1      |  * Debug mode: on
vote_1      |  * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
vote_1      |  * Restarting with stat
vote_1      |  * Debugger is active!
```

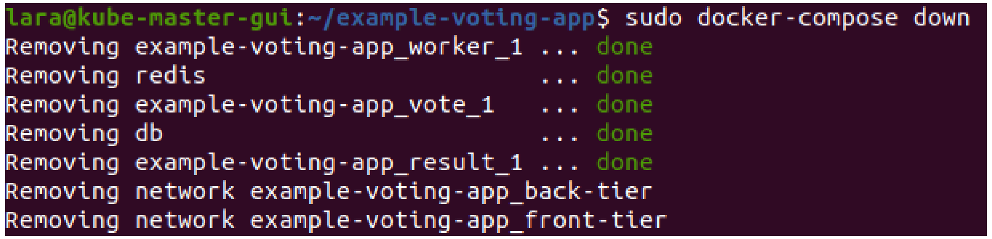
If you use CTRL+C while “up” is running, all the containers will exit and you will regain control of the command prompt.



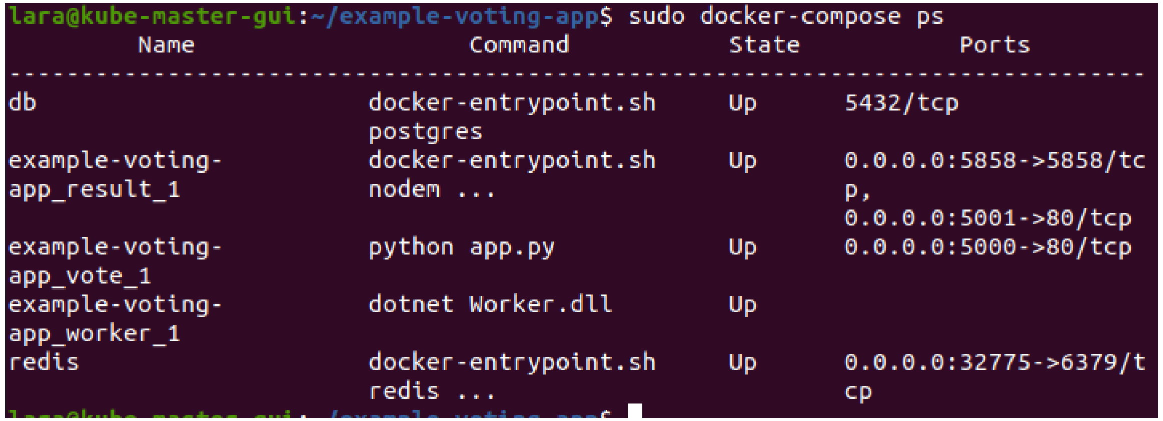
```
lara@kube-master-gui:~/example-voting-app$ sudo docker-compose stop
Stopping example-voting-app_worker_1 ... done
Stopping redis                        ... done
Stopping example-voting-app_vote_1   ... done
Stopping db                           ... done
Stopping example-voting-app_result_1 ... done
```



```
lara@kube-master-gui:~/example-voting-app$ sudo docker-compose start
Starting redis  ... done
Starting result ... done
Starting db     ... done
Starting vote   ... done
Starting worker ... done
```



```
lara@kube-master-gui:~/example-voting-app$ sudo docker-compose down
Removing example-voting-app_worker_1 ... done
Removing redis                        ... done
Removing example-voting-app_vote_1   ... done
Removing db                           ... done
Removing example-voting-app_result_1 ... done
Removing network example-voting-app_back-tier
Removing network example-voting-app_front-tier
```



```
lara@kube-master-gui:~/example-voting-app$ sudo docker-compose ps
         Name                      Command             State          Ports
-------------------------------------------------------------------------------
db                          docker-entrypoint.sh    Up      5432/tcp
                            postgres
example-voting-             docker-entrypoint.sh    Up      0.0.0.0:5858->5858/tc
app_result_1                nodem ...                       p,
                                                            0.0.0.0:5001->80/tcp
example-voting-             python app.py           Up      0.0.0.0:5000->80/tcp
app_vote_1
example-voting-             dotnet Worker.dll       Up
app_worker_1
redis                       docker-entrypoint.sh    Up      0.0.0.0:32775->6379/t
                            redis ...                       cp
lara@kube-master-gui:~/example-voting-app$
```

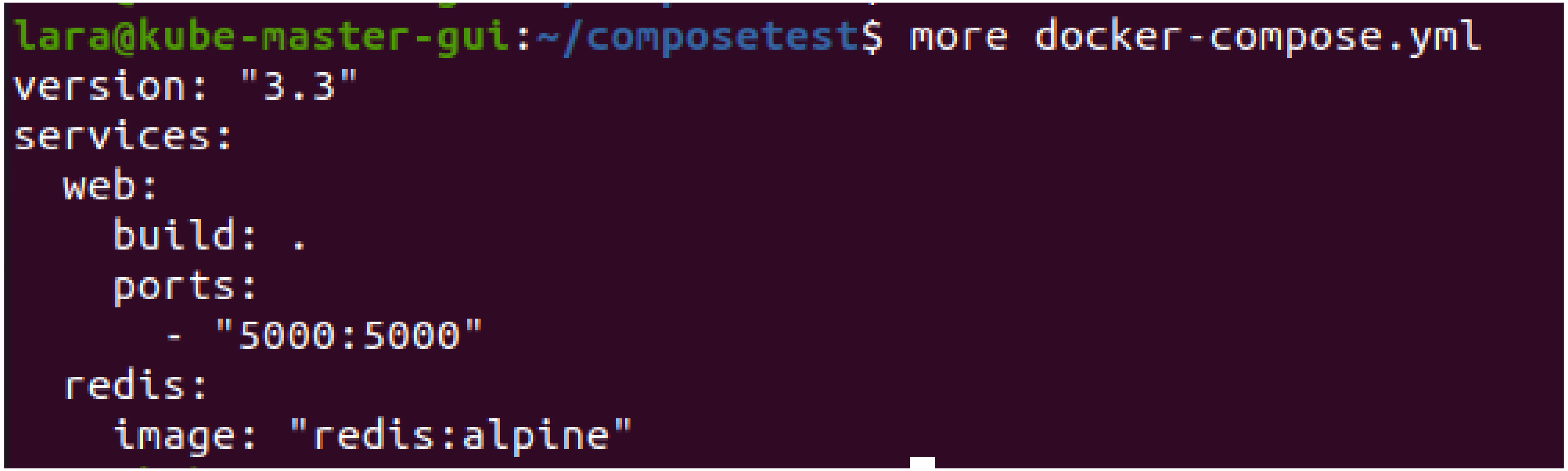
docker-compose YAML file

The docker-compose.yaml file contains the definition of **services**, **networks**, **volumes**, **configs**, **secrets** according to the following specification: https://github.com/compose-spec/compose-spec/blob/master/spec.md

Services are compute resources based on containers, with additional configuration. They can be declared based on:

- A existing image that will be pulled from a registry if it’s not already present locally.
- A Dockerfile and local source files that will be built on the fly by the “docker-compose up” command.

Services can bind a port on the host to a port exposed by the container, so that they can be accessed from clients external to the container.



```
lara@kube-master-gui:~/composetest$ more docker-compose.yml
version: "3.3"
services:
  web:
    build: .
    ports:
      - "5000:5000"
  redis:
    image: "redis:alpine"
```

Docker persistent storage

Docker containers can write to the writeable layer using a storage driver (currently overalay2 is most used).
This data is lost when the container stops executing and its performance is inferior to writing to the host disk.
To persist data so that it survives after the container exits, you need to use persistent storage:

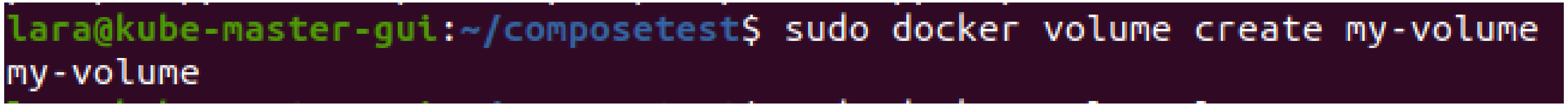
- **Volumes**:
    - stored in a part of the host filesystem which is managed by Docker:
        - /var/lib/docker/volumes/ on standard Linux installations
        - /var/snap/docker/common/var-lib-docker/volumes in microk8s
    - Non-Docker processes should not modify this part of the filesystem.
- **Bind mounts**:
    - Can be stored anywhere on the host filesystem
    - Can be modified by other processes than docker

- There is also an additional type of storage that is not persistent:
    - **tmpfs mounts**:
        - Only stored in memory, never stored on the host filesystem.
    - This is useful to temporarily store sensitive files that you don’t want to persist in either the host or the container writable layer. This information cannot e shared among containers
    - It performs better than writing to the container’s writeable layer.

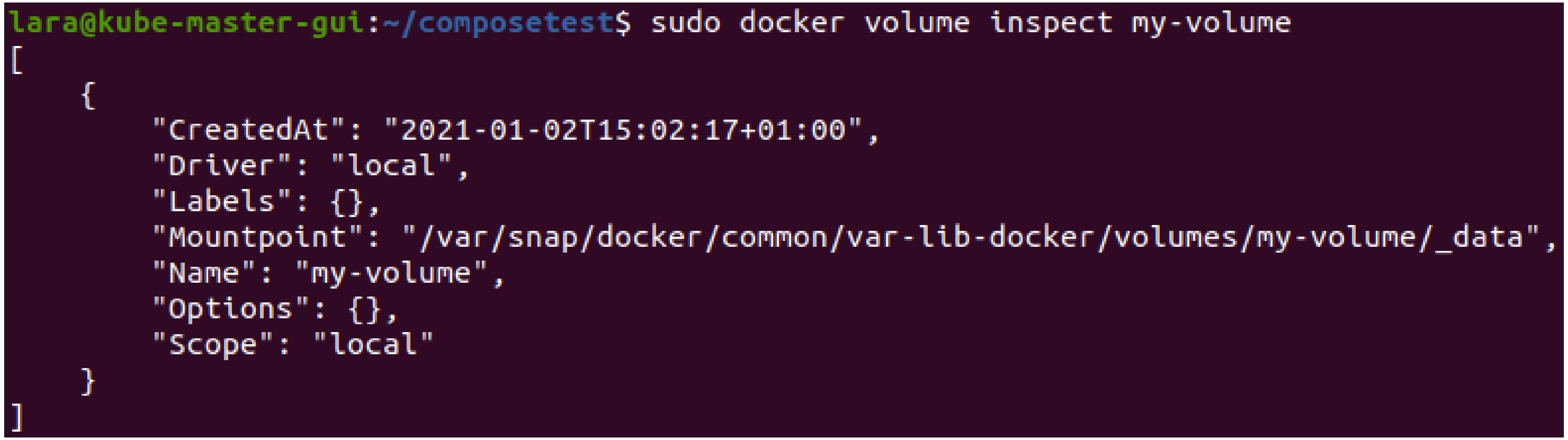
Docker volumes

Docker volumes can be created in three different ways:

- Using the **docker volume create** command
- During container creation (**docker run**) or within the volumes section of a **docker-compose.yml file**
- During service creation (with docker swarm). Note: Don’t confuse the **docker service create** command which applies to docker swarm with services created in a docker-compose.yml file.



```
lara@kube-master-gui:~/composetest$ sudo docker volume create my-volume
my-volume
```



```
lara@kube-master-gui:~/composetest$ sudo docker volume inspect my-volume
[
    {
        "CreatedAt": "2021-01-02T15:02:17+01:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/snap/docker/common/var-lib-docker/volumes/my-volume/_data",
        "Name": "my-volume",
        "Options": {},
        "Scope": "local"
    }
]
```

HCLSoftware

Docker volume mounts

If you create a volume manually, you can use it by mounting it onto a container.

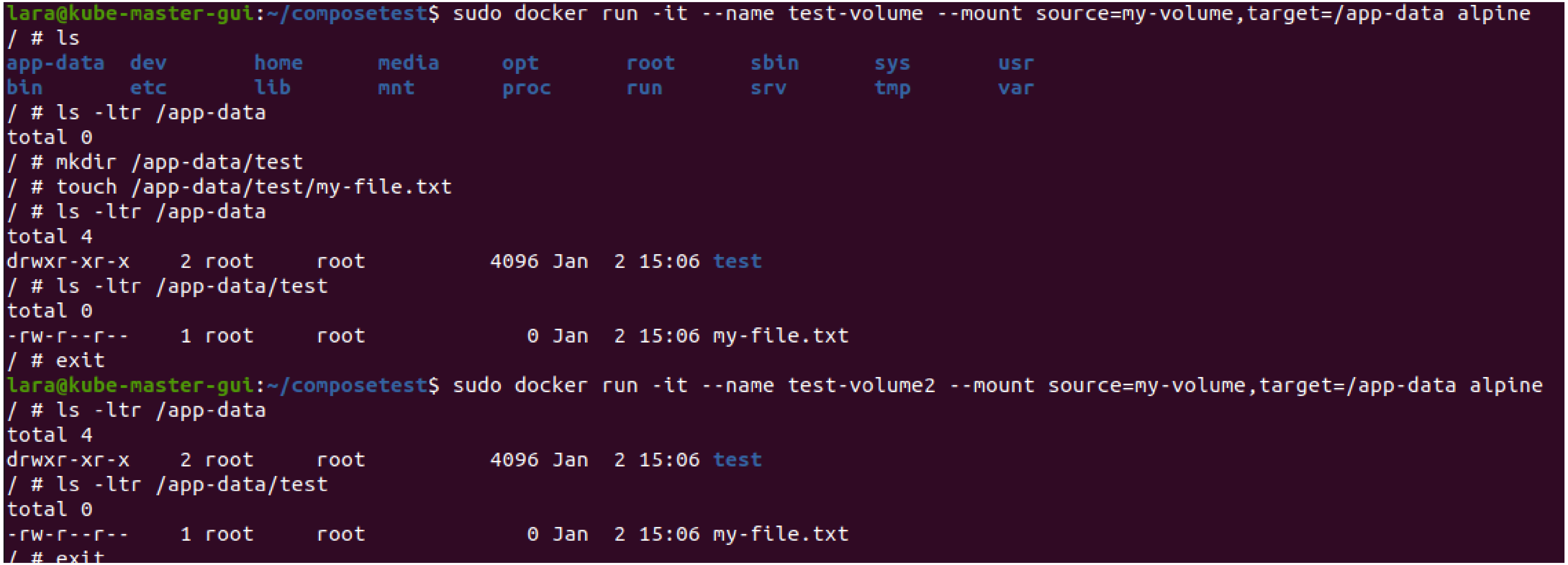
Use the --mount argument of the docker run command, which can be specified as follows:

--mount source=<volume-name>,target=<mount-point-in-container>

(for more complex forms of the mount argument, see documentation).

If the volume does not exist yet, it will be created automatically. The same volume can be used by other containers.

The container can work on this directory, referring to the target of the mount argument (/app-data in the example):



```
lara@kube-master-gui:~/composetest$ sudo docker run -it --name test-volume --mount source=my-volume,target=/app-data alpine
/ # ls
app-data  dev       home      media     opt       root      sbin      sys       usr
bin       etc       lib       mnt       proc      run       srv       tmp       var
/ # ls -ltr /app-data
total 0
/ # mkdir /app-data/test
/ # touch /app-data/test/my-file.txt
/ # ls -ltr /app-data
total 4
drwxr-xr-x    2 root     root          4096 Jan  2 15:06 test
/ # ls -ltr /app-data/test
total 0
-rw-r--r--    1 root     root             0 Jan  2 15:06 my-file.txt
/ # exit
lara@kube-master-gui:~/composetest$ sudo docker run -it --name test-volume2 --mount source=my-volume,target=/app-data alpine
/ # ls -ltr /app-data
total 4
drwxr-xr-x    2 root     root          4096 Jan  2 15:06 test
/ # ls -ltr /app-data/test
total 0
-rw-r--r--    1 root     root             0 Jan  2 15:06 my-file.txt
/ # exit
```

Docker volume removal

Stopping the container leaves the volume intact. If desired, you can manually remove the volume.

The required sequence of commands is:

```
docker container stop <container-id>

docker container rm <container-id>

docker volume rm <volume-name>
```

If you want to remove all volumes no longer referenced by any container do:

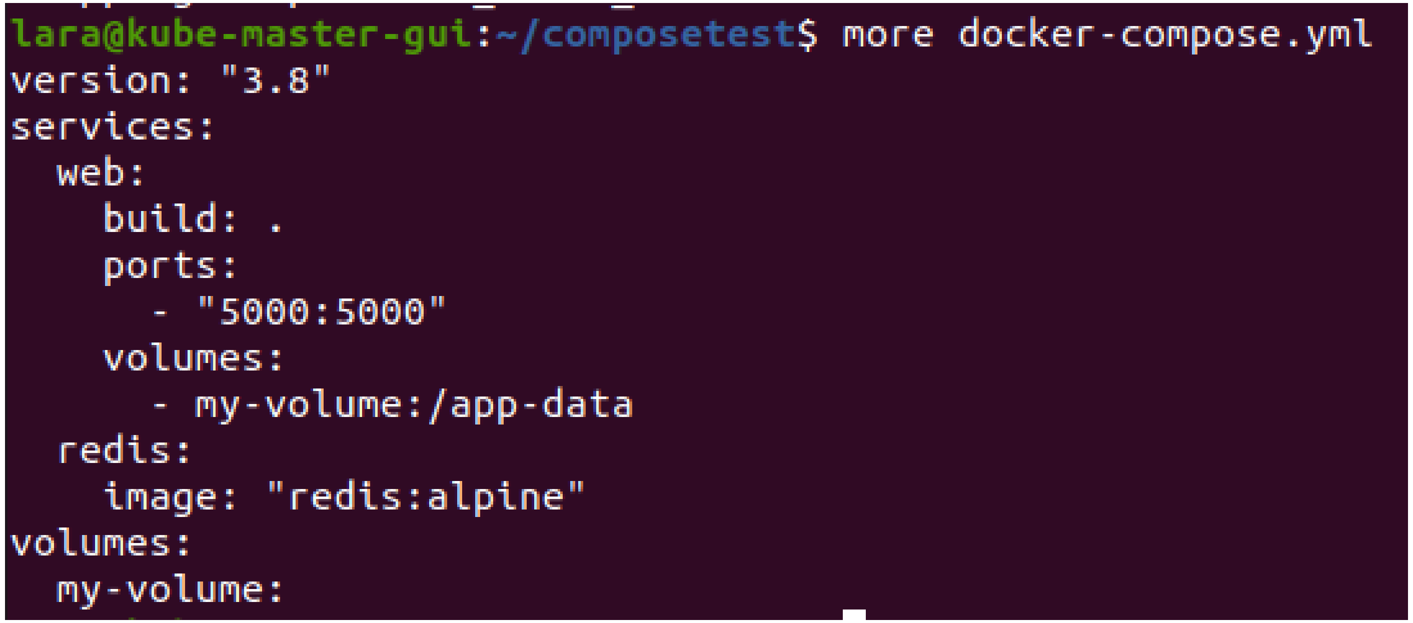
```
docker volume prune
```

Docker volume creation and mount with docker compose 1/2

To create a new volume you need to add a top-level “volumes:” section to the docker-compose.yaml file.

To mount the volume, you need to reference the volume name in the “volumes:” section of a service declaration. The syntax of the reference is:

- <volume-name>:<absolute-path-inside-the-container>



```
lara@kube-master-gui:~/composetest$ more docker-compose.yml
version: "3.8"
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - my-volume:/app-data
  redis:
    image: "redis:alpine"
volumes:
  my-volume:
```

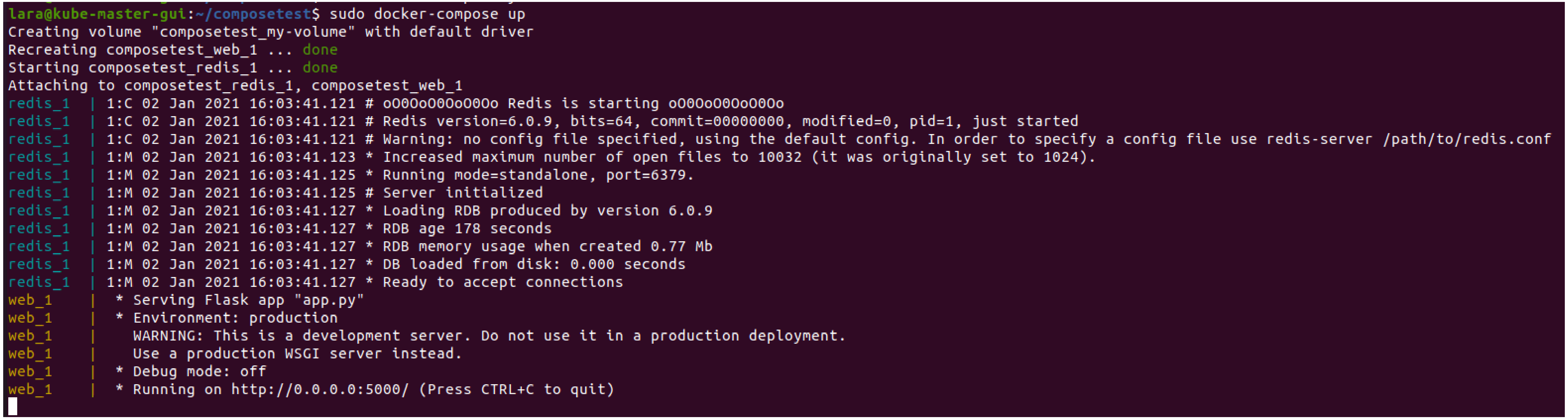
Docker volume creation and mount with docker compose 2/2

When you execute **docker-compose up**, it creates a new volume called like:

<project-name>_<volume-name>

where <project-name> by default is equal to name of the directory where the docker-compose.yml file resides.

It then binds the volume to the service in which the reference was declared.



```
lara@kube-master-gui:~/composetest$ sudo docker-compose up
Creating volume "composetest_my-volume" with default driver
Recreating composetest_web_1 ... done
Starting composetest_redis_1 ... done
Attaching to composetest_redis_1, composetest_web_1
redis_1  | 1:C 02 Jan 2021 16:03:41.121 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
redis_1  | 1:C 02 Jan 2021 16:03:41.121 # Redis version=6.0.9, bits=64, commit=00000000, modified=0, pid=1, just started
redis_1  | 1:C 02 Jan 2021 16:03:41.121 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
redis_1  | 1:M 02 Jan 2021 16:03:41.123 * Increased maximum number of open files to 10032 (it was originally set to 1024).
redis_1  | 1:M 02 Jan 2021 16:03:41.125 * Running mode=standalone, port=6379.
redis_1  | 1:M 02 Jan 2021 16:03:41.125 # Server initialized
redis_1  | 1:M 02 Jan 2021 16:03:41.127 * Loading RDB produced by version 6.0.9
redis_1  | 1:M 02 Jan 2021 16:03:41.127 * RDB age 178 seconds
redis_1  | 1:M 02 Jan 2021 16:03:41.127 * RDB memory usage when created 0.77 Mb
redis_1  | 1:M 02 Jan 2021 16:03:41.127 * DB loaded from disk: 0.000 seconds
redis_1  | 1:M 02 Jan 2021 16:03:41.127 * Ready to accept connections
web_1    |  * Serving Flask app "app.py"
web_1    |  * Environment: production
web_1    |    WARNING: This is a development server. Do not use it in a production deployment.
web_1    |    Use a production WSGI server instead.
web_1    |  * Debug mode: off
web_1    |  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

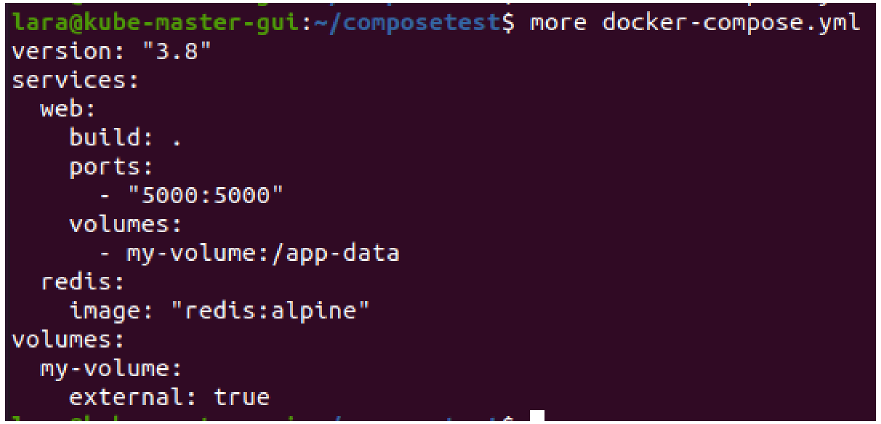
Docker volume mount with existing volumes in docker compose

To make compose use an existing volume, add the keyword:

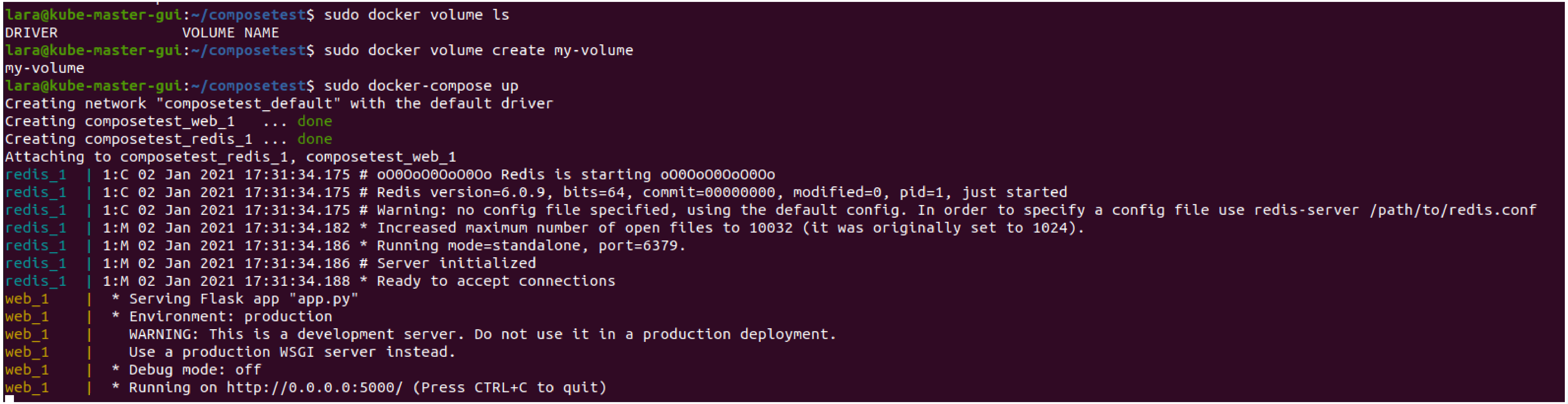
external: true

Under the volume name in the volumes: top-level section

In this case this volume will be used and not created.



```
lara@kube-master-gui:~/composetest$ more docker-compose.yml
version: "3.8"
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - my-volume:/app-data
  redis:
    image: "redis:alpine"
volumes:
  my-volume:
    external: true
```

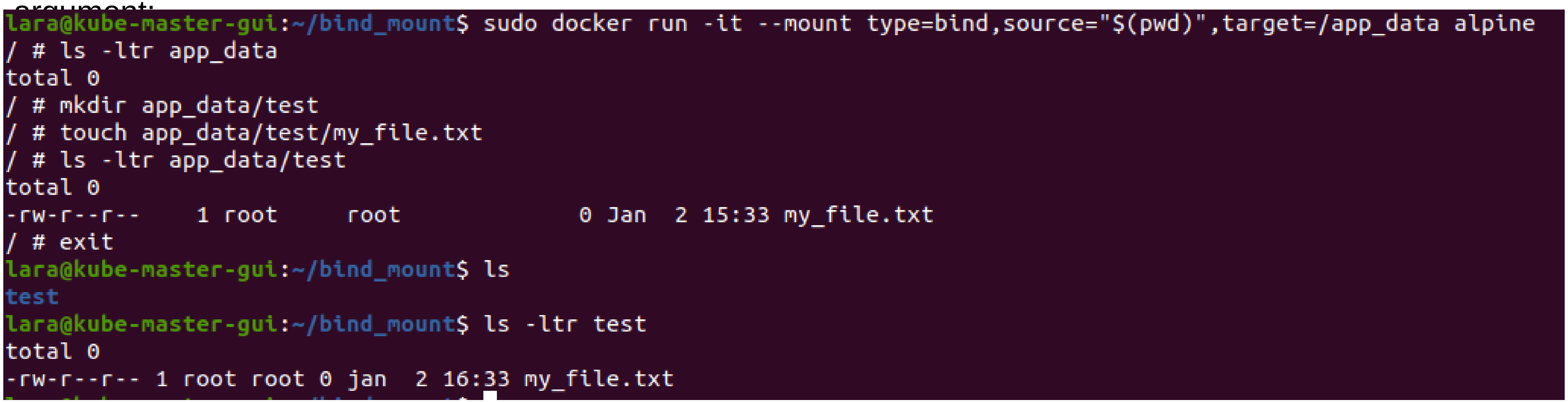


```
lara@kube-master-gui:~/composetest$ sudo docker volume ls
DRIVER              VOLUME NAME
lara@kube-master-gui:~/composetest$ sudo docker volume create my-volume
my-volume
lara@kube-master-gui:~/composetest$ sudo docker-compose up
Creating network "composetest_default" with the default driver
Creating composetest_web_1   ... done
Creating composetest_redis_1 ... done
Attaching to composetest_redis_1, composetest_web_1
redis_1  | 1:C 02 Jan 2021 17:31:34.175 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
redis_1  | 1:C 02 Jan 2021 17:31:34.175 # Redis version=6.0.9, bits=64, commit=00000000, modified=0, pid=1, just started
redis_1  | 1:C 02 Jan 2021 17:31:34.175 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
redis_1  | 1:M 02 Jan 2021 17:31:34.182 * Increased maximum number of open files to 10032 (it was originally set to 1024).
redis_1  | 1:M 02 Jan 2021 17:31:34.186 * Running mode=standalone, port=6379.
redis_1  | 1:M 02 Jan 2021 17:31:34.186 # Server initialized
redis_1  | 1:M 02 Jan 2021 17:31:34.188 * Ready to accept connections
web_1    |  * Serving Flask app "app.py"
web_1    |  * Environment: production
web_1    |    WARNING: This is a development server. Do not use it in a production deployment.
web_1    |    Use a production WSGI server instead.
web_1    |  * Debug mode: off
web_1    |  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Docker bind mount

Docker bind mounts are simpler but provide less functionality than volume mounts.
They simply map a folder on the host to a folder inside the container. They cannot be managed using the docker cli.
Docker cannot guarantee that other processes don’t alter the files.

To use a bind mount when creating a new container you need to add the `type=bind` to the `--mount` argument:

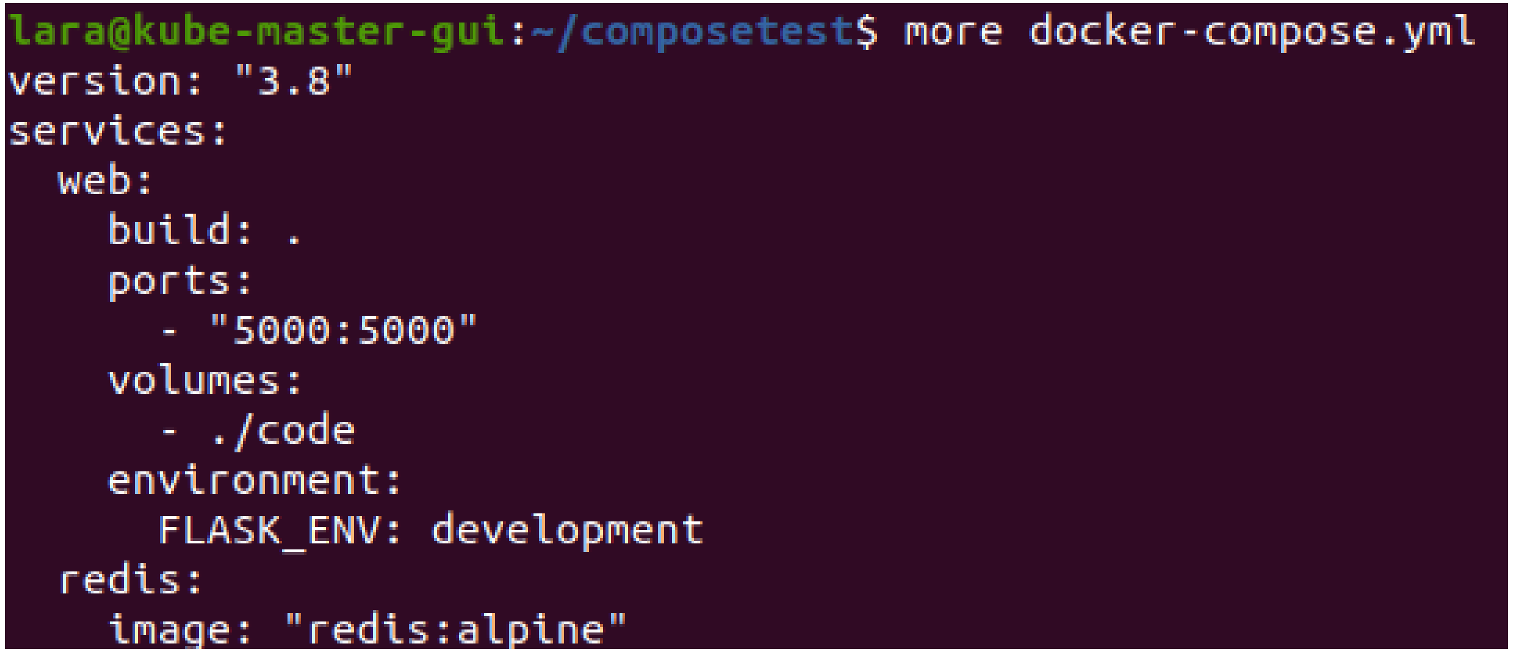


```
lara@kube-master-gui:~/bind_mount$ sudo docker run -it --mount type=bind,source="$(pwd)",target=/app_data alpine
/ # ls -ltr app_data
total 0
/ # mkdir app_data/test
/ # touch app_data/test/my_file.txt
/ # ls -ltr app_data/test
total 0
-rw-r--r--    1 root     root             0 Jan  2 15:33 my_file.txt
/ # exit
lara@kube-master-gui:~/bind_mount$ ls
test
lara@kube-master-gui:~/bind_mount$ ls -ltr test
total 0
-rw-r--r-- 1 root root 0 jan  2 16:33 my_file.txt
```

In this case you can inspect/alter the contents of the directory from the host operating system.

Docker bind mount with docker compose

A docker-compose.yaml file can use a bind mount with the following syntax:



```
lara@kube-master-gui:~/composetest$ more docker-compose.yml
version: "3.8"
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - ./code
    environment:
      FLASK_ENV: development
  redis:
    image: "redis:alpine"
```

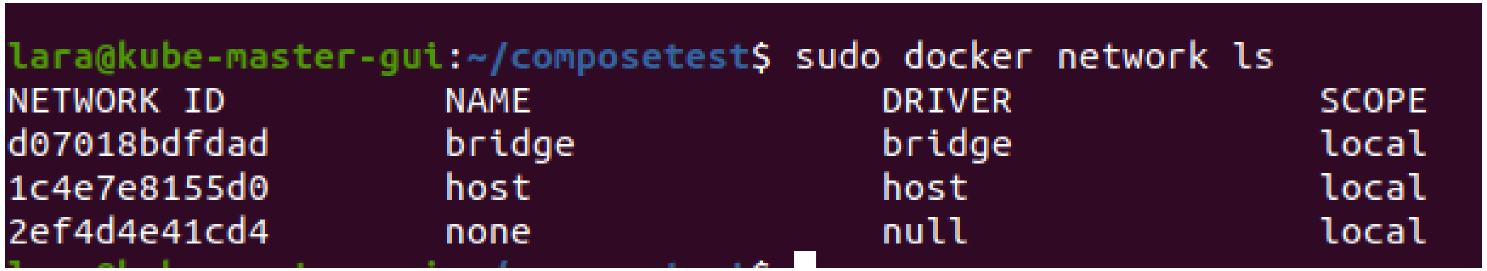
In this example, the current working directory "." on the host is mapped to the "/code" directory inside the container.

The source code file /code/app.py can be automatically reloaded by Flask due to the FLASK_ENV environment variable being set to development.

After you start the containers with **docker-compose up**, if you modify ~/composetest/app.py on the host and reload the page at http://localhost:5000, you will see the effect of the modified source code.

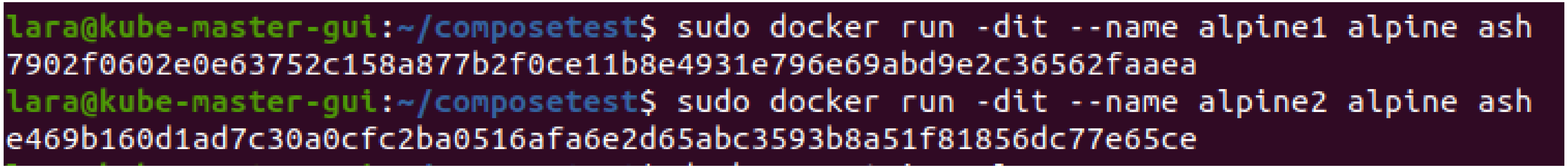
Docker networking: bridge default network 1/2

By default docker creates the following networks:



```
lara@kube-master-gui:~/composetest$ sudo docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
d07018bdfdad        bridge              bridge              local
1c4e7e8155d0        host                host                local
2ef4d4e41cd4        none                null                local
```

By default when you start a container it gets attached to the default bridge network:



```
lara@kube-master-gui:~/composetest$ sudo docker run -dit --name alpine1 alpine ash
7902f0602e0e63752c158a877b2f0ce11b8e4931e796e69abd9e2c36562faaea
lara@kube-master-gui:~/composetest$ sudo docker run -dit --name alpine2 alpine ash
e469b160d1ad7c30a0cfc2ba0516afa6e2d65abc3593b8a51f81856dc77e65ce
```

Note that –dit means run the container detached, but provide a tty and allow for input to be sent to the container. You can later attach and interact with the container.
Ash is the shell in the Alpine Linux image (instead of bash available in Ubuntu). Alpine Linux is a very small footprint Linux distribution suitable for running containers.

Docker networking: bridge default network 2/2

The command:

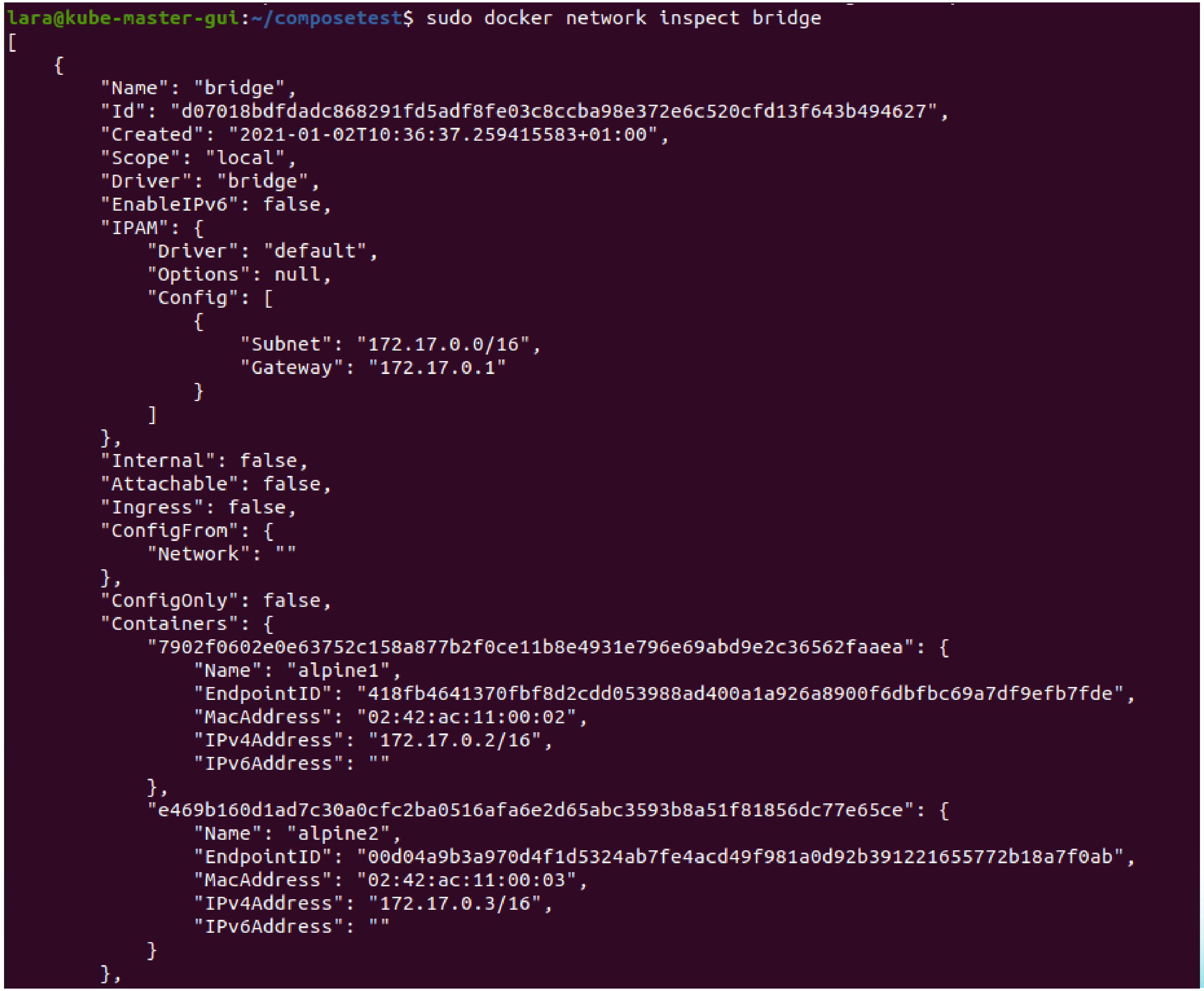
```
docker network inspect bridge
```

shows that the two containers we just created are attached to the default bridge network.

Each container has its own IP address in the Subnet defined by the default bridge network.

Using this network, the two containers can refer to each other via IP.

Using the default bridge network is not recommended for production.

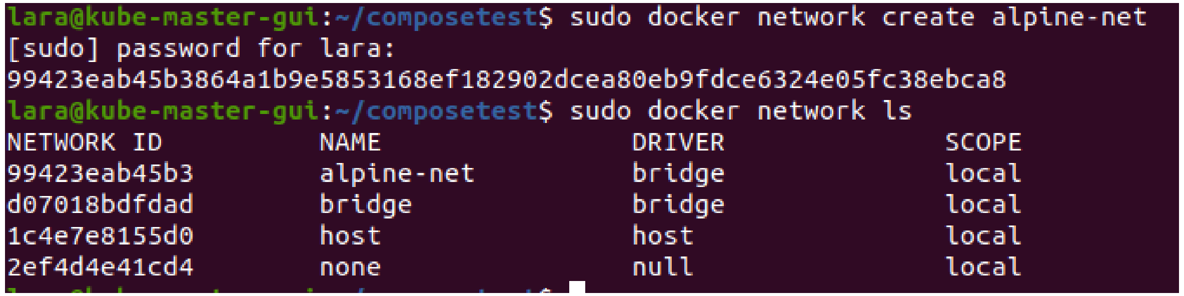


```
lara@kube-master-gui:~/composetest$ sudo docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "d07018bdfdadc868291fd5adf8fe03c8ccba98e372e6c520cfd13f643b494627",
        "Created": "2021-01-02T10:36:37.259415583+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "7902f0602e0e63752c158a877b2f0ce11b8e4931e796e69abd9e2c36562faaea": {
                "Name": "alpine1",
                "EndpointID": "418fb4641370fbf8d2cdd053988ad400a1a926a8900f6dbfbc69a7df9efb7fde",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            },
            "e469b160d1ad7c30a0cfc2ba0516afa6e2d65abc3593b8a51f81856dc77e65ce": {
                "Name": "alpine2",
                "EndpointID": "00d04a9b3a970d4f1d5324ab7fe4acd49f981a0d92b391221655772b18a7f0ab",
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "IPv6Address": ""
            }
        },
```

HCLSoftware

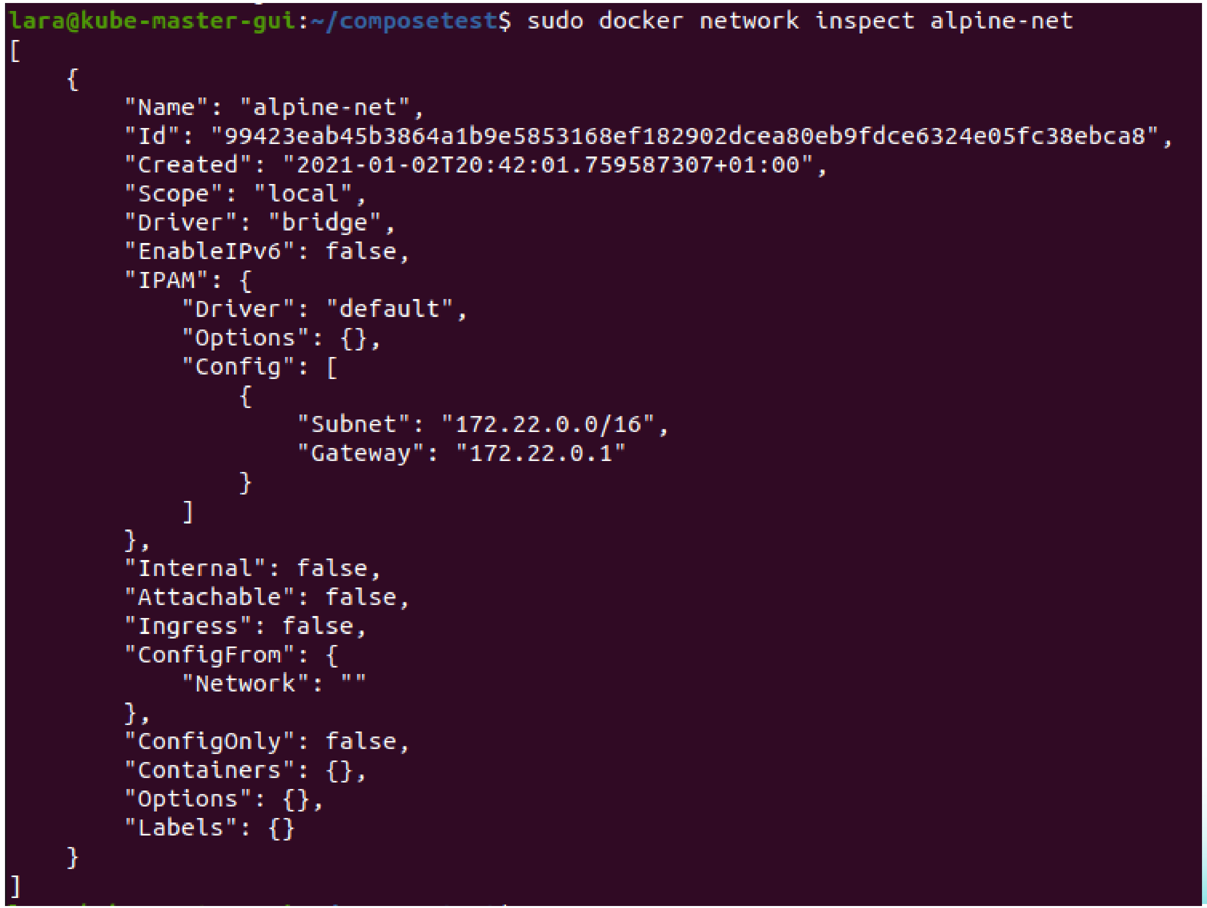
Docker networking: user bridge networks

You can create user defined bridge networks as follows:



```
lara@kube-master-gui:~/composetest$ sudo docker network create alpine-net
[sudo] password for lara:
99423eab45b3864a1b9e5853168ef182902dcea80eb9fdce6324e05fc38ebca8
lara@kube-master-gui:~/composetest$ sudo docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
99423eab45b3        alpine-net          bridge              local
d07018bdfdad        bridge              bridge              local
1c4e7e8155d0        host                host                local
2ef4d4e41cd4        none                null                local
```

Note that the user created bridge network uses a different subnet than the default bridge network. The CIDR notation 172.22.0.0./16 corresponds to 2^16 = 65,536 different IP addresses, of which the first one is used as the Gateway.



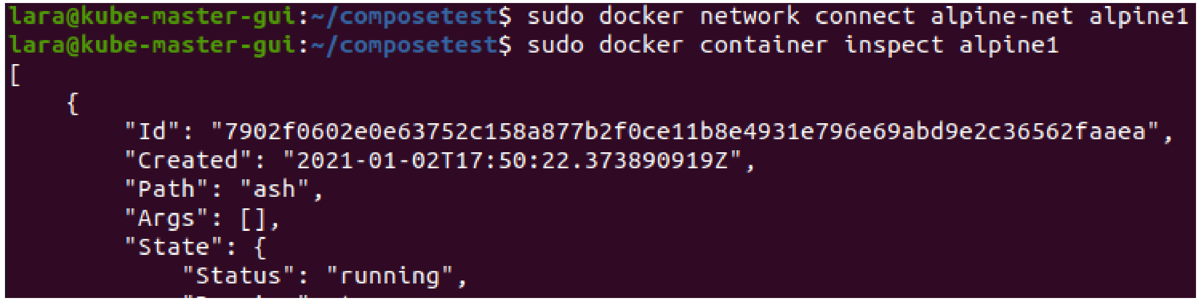
```
lara@kube-master-gui:~/composetest$ sudo docker network inspect alpine-net
[
    {
        "Name": "alpine-net",
        "Id": "99423eab45b3864a1b9e5853168ef182902dcea80eb9fdce6324e05fc38ebca8",
        "Created": "2021-01-02T20:42:01.759587307+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.22.0.0/16",
                    "Gateway": "172.22.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

Connecting a container to an existing user bridge network

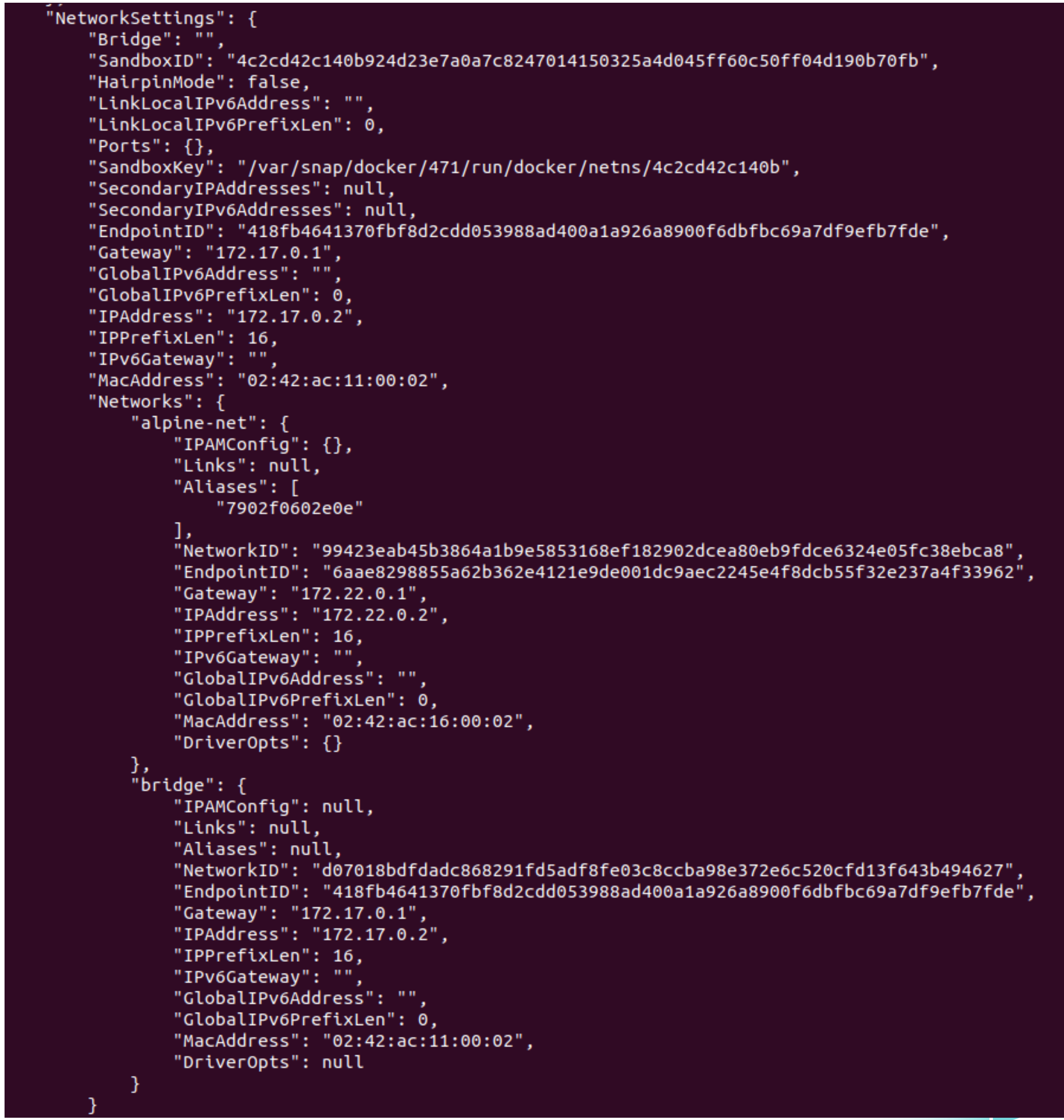
You can connect an existing container to the newly created network using the command

docker network connect <network-name> <container-name>

If you inspect the container you will see both networks listed:



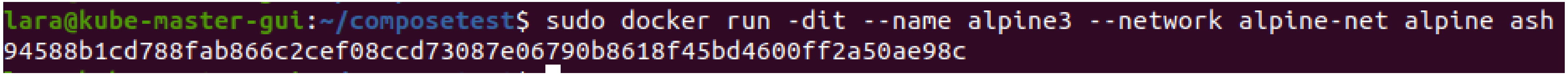
```
lara@kube-master-gui:~/composetest$ sudo docker network connect alpine-net alpine1
lara@kube-master-gui:~/composetest$ sudo docker container inspect alpine1
[
    {
        "Id": "7902f0602e0e63752c158a877b2f0ce11b8e4931e796e69abd9e2c36562faaea",
        "Created": "2021-01-02T17:50:22.373890919Z",
        "Path": "ash",
        "Args": [],
        "State": {
            "Status": "running",
```



```
    "NetworkSettings": {
        "Bridge": "",
        "SandboxID": "4c2cd42c140b924d23e7a0a7c8247014150325a4d045ff60c50ff04d190b70fb",
        "HairpinMode": false,
        "LinkLocalIPv6Address": "",
        "LinkLocalIPv6PrefixLen": 0,
        "Ports": {},
        "SandboxKey": "/var/snap/docker/471/run/docker/netns/4c2cd42c140b",
        "SecondaryIPAddresses": null,
        "SecondaryIPv6Addresses": null,
        "EndpointID": "418fb4641370fbf8d2cdd053988ad400a1a926a8900f6dbfbc69a7df9efb7fde",
        "Gateway": "172.17.0.1",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "IPAddress": "172.17.0.2",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "MacAddress": "02:42:ac:11:00:02",
        "Networks": {
            "alpine-net": {
                "IPAMConfig": {},
                "Links": null,
                "Aliases": [
                    "7902f0602e0e"
                ],
                "NetworkID": "99423eab45b3864a1b9e5853168ef182902dcea80eb9fdce6324e05fc38ebca8",
                "EndpointID": "6aae8298855a62b362e4121e9de001dc9aec2245e4f8dcb55f32e237a4f33962",
                "Gateway": "172.22.0.1",
                "IPAddress": "172.22.0.2",
                "IPPrefixLen": 16,
                "IPv6Gateway": "",
                "GlobalIPv6Address": "",
                "GlobalIPv6PrefixLen": 0,
                "MacAddress": "02:42:ac:16:00:02",
                "DriverOpts": {}
            },
            "bridge": {
                "IPAMConfig": null,
                "Links": null,
                "Aliases": null,
                "NetworkID": "d07018bdfdadc868291fd5adf8fe03c8ccba98e372e6c520cfd13f643b494627",
                "EndpointID": "418fb4641370fbf8d2cdd053988ad400a1a926a8900f6dbfbc69a7df9efb7fde",
                "Gateway": "172.17.0.1",
                "IPAddress": "172.17.0.2",
                "IPPrefixLen": 16,
                "IPv6Gateway": "",
                "GlobalIPv6Address": "",
                "GlobalIPv6PrefixLen": 0,
                "MacAddress": "02:42:ac:11:00:02",
                "DriverOpts": null
            }
        }
```

Creating a new container with a user defined bridge network

You can create a new container with a user defined bridge network by specifying the --network argument to the **docker run** command:



```
lara@kube-master-gui:~/composetest$ sudo docker run -dit --name alpine3 --network alpine-net alpine ash
94588b1cd788fab866c2cef08ccd73087e06790b8618f45bd4600ff2a50ae98c
```

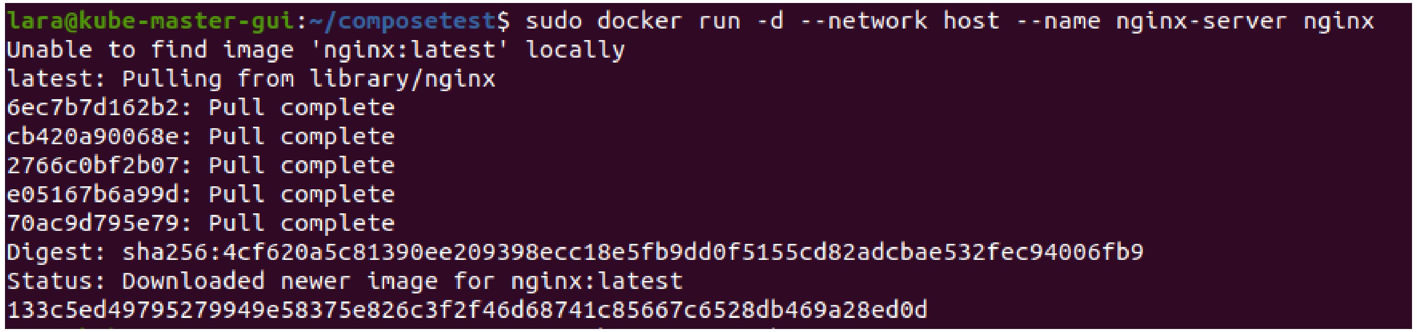
When containers are part of the same user defined bridge network, they can refer to each other both via IP and via container name.

This feature is known as **automatic service discovery**.

Docker host network

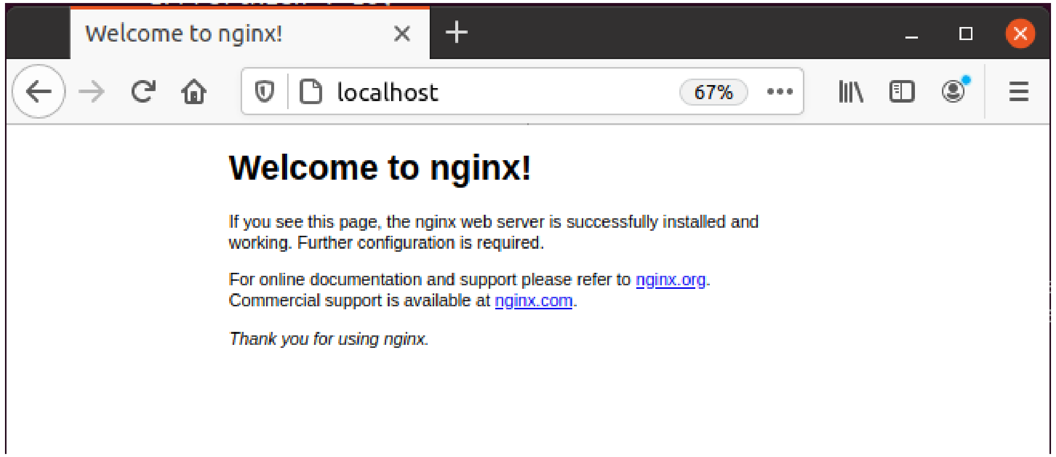
If you connect a container to the host network, the container network is NOT isolated from the host network.

This allows you to access any service running in the container as if it was running directly on the host, without publishing the port open in the container to a port of the host.



```
lara@kube-master-gui:~/composetest$ sudo docker run -d --network host --name nginx-server nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
6ec7b7d162b2: Pull complete
cb420a90068e: Pull complete
2766c0bf2b07: Pull complete
e05167b6a99d: Pull complete
70ac9d795e79: Pull complete
Digest: sha256:4cf620a5c81390ee209398ecc18e5fb9dd0f5155cd82adcbae532fec94006fb9
Status: Downloaded newer image for nginx:latest
133c5ed49795279949e58375e826c3f2f46d68741c85667c6528db469a28ed0d
```

Since by default the nginx image runs a web server on port 80, you can open the web browser and connect to:
http://localhost:80
from the host machine and you will connect directly to the nginx web server running in the container.



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Assignments before the next lesson

Complete the following assignments:

1. Become part of a groups of max. 3 students for the purpose of creating the course project.
2. Install a Virtual machine with Ubuntu Desktop. Install docker and microk8s using snap. Alternatively, install docker and microk8s natively on Linux, Windows or Mac.
3. Complete the docker labs: https://github.com/docker/labs/tree/master/beginner
4. Post your questions to the discussion forum of Lesson 1
5. Start thinking of an application that you will containerize for the project/presentation. The application should have at least one database, one middleware (such as a REST API) and one Web Frontend.

HCLSoftware

hcltechsw.com