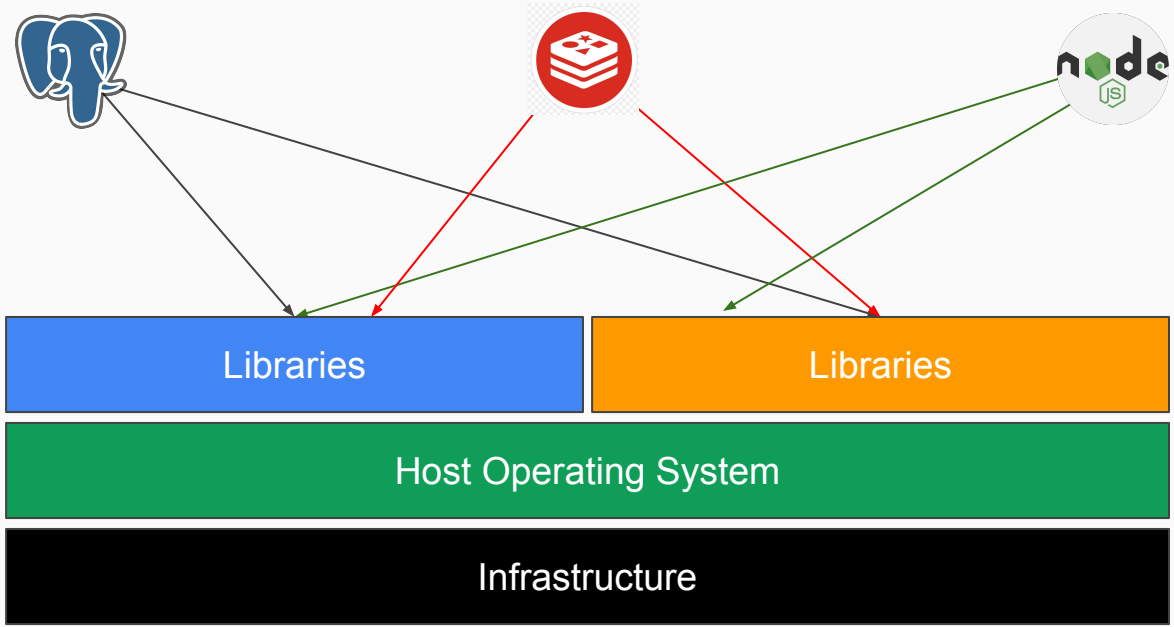
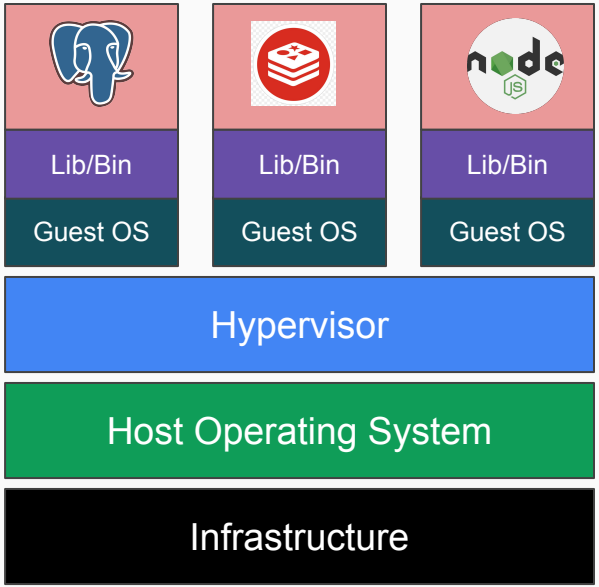
Linux Container The Hardway

Traditional Software Deployment - Dependency Hell



Traditional Software Deployment - Portability

Virtual Machine



- Advantages
  - Portable
  - Strictly Isolated
  - Multi-OS

- Disadvantages
  - Time consuming
  - Resource usage
  - Startup time
  - Scalability

What is container?

What is container?

An environment for executing processes with configurable isolation and resource limitations. ....

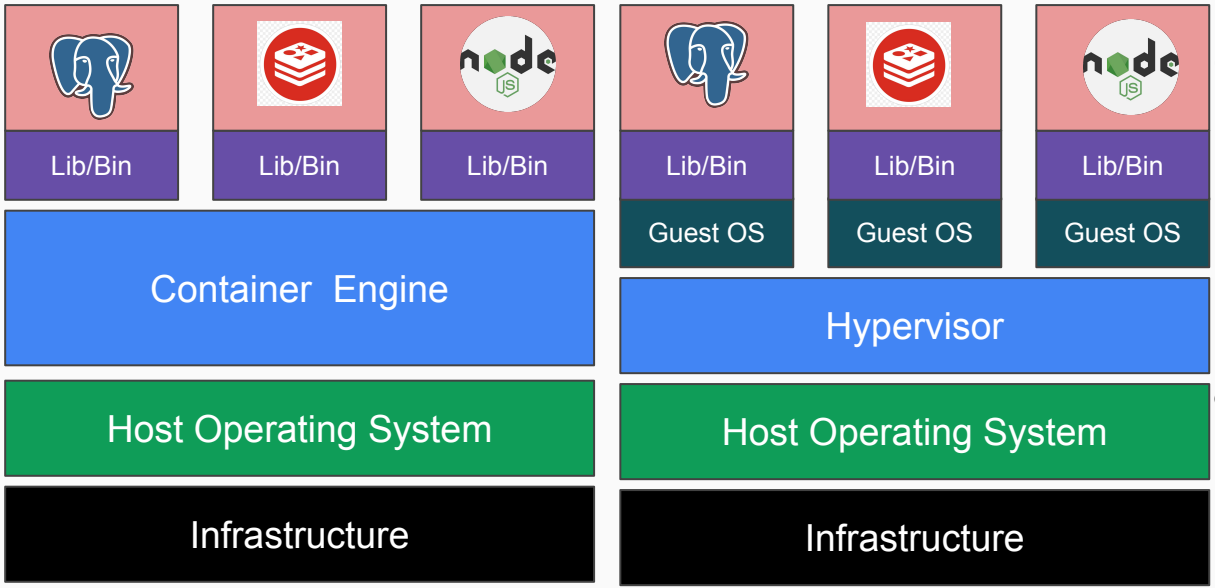
https://github.com/opencontainers/runtime-spec/blob/main/glossary.md#container

Why do we need container(s) ?

....The goal of a Standard Container is to encapsulate a software component and all its dependencies in a format ..... that any compliant runtime can run it without extra dependencies, regardless of the underlying machine and the contents of the container.

https://github.com/opencontainers/runtime-spec/blob/main/principles.md#the-5-principles-of-standard-containers

Linux Container & Virtual Machine



- Pros
  - Efficience resource usage
  - Faster startup
  - Scalability
- Cons
  - Less secure than VM?
  - Same as Host OS

Linux Container Technologies

- Unix chroot system - 1979
- Free BSD Jails - 2000
- Linux VServer - 2001
- OpenVZ - 2005

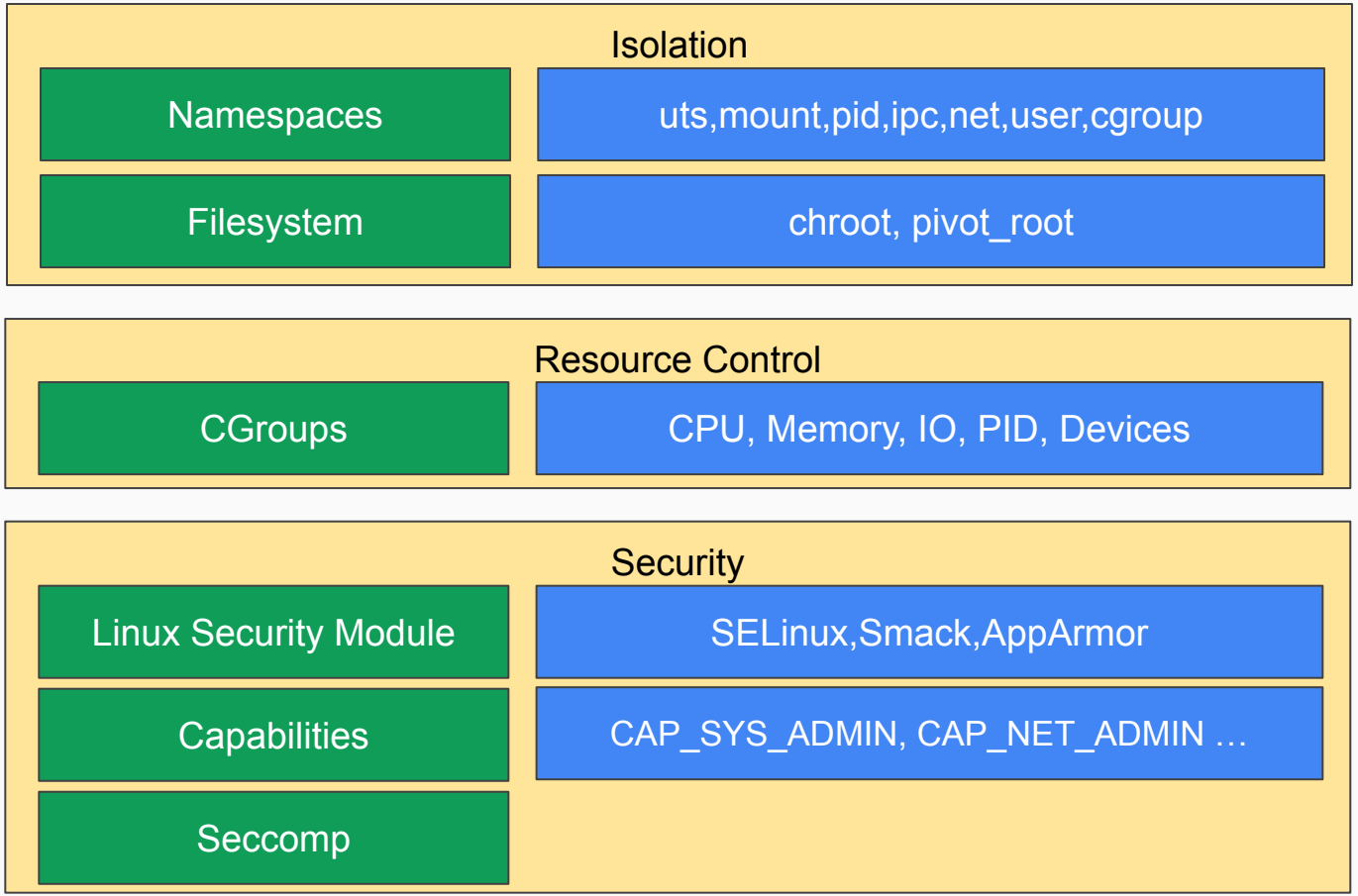
  … Introduction of control groups (2006)
- Linux Container (LXC) - The most stable container system (2008)
- Cloud Foundry Warden - 2011

  … The Golden Age
- Docker - 2013
- Windows support running (Windows & Linux) containers on Windows OS families

Why is Docker so popular ?

- Easy to use
- Good Documentations
- Tooling & CLI
  - Image Management (docker build/push/pull)
  - Public Image Registry
  - Standardizate Image Build with Dockerfile
- Ecosystem and Cloud Support
  - Docker Swarm - An orchestration tool to manage clusters of Docker containers
  - Docker Trusted Registry - A private registry for trusted Docker images
  - Docker Compose - A tool for launching applications with numerous containers that need to exchange data.
  - Docker Desktop - A tool for creating Docker-enabled virtual machines.

Linux Container - Main Components



Namespaces

Namespaces

- Namespacing is a mechanism to provide processes with different view on different system resources
- It’s a kind of process isolation
- Namespaces kind
  - Mount
  - Process ID
  - UTS
  - User
  - Network
  - CGroup
  - IPC (Inter-Process Communication)
  - Time
- Working with namespace
  - Create new namespace: **unshare**
  - Enter a namespace: **nsenter**

Demo Namespaces in Linux

Filesystem Isolation

Filesystem Isolation

*A **chroot** on Unix and Unix-like operating systems is an operation that changes the apparent root directory for the current running process and its children. A program that is run in such a modified environment cannot name (and therefore normally cannot access) files outside the designated directory tree*

https://en.wikipedia.org/wiki/Chroot

Control Group (CGroup)

Control Group

- `cpuset` - assigns individual processor(s) and memory nodes to task(s) in a group;
- `cpu` - uses the scheduler to provide cgroup tasks access to the processor resources;
- `cpuacct` - generates reports about processor usage by a group;
- `pid` - sets limit to number of processes in a group.
- `io` - sets limit to read/write from/to block devices;
- `memory` - sets limit on memory usage by a task(s) from a group;
- `devices` - allows access to devices by a task(s) from a group;
- `freezer` - allows to suspend/resume for a task(s) from a group;
- `net_cls` - allows to mark network packets from task(s) from a group;
- `net_prio` - provides a way to dynamically set the priority of network traffic per network interface for a group;
- `perf_event` - provides access to perf events to a group;
- `hugetlb` - activates support for huge pages for a group;

Interact with Control Group

Working with filesystem

```
root@basic:/home/vagrant# findmnt | grep cgroup
├─/sys/fs/cgroup                    tmpfs      tmpfs   ro,nosuid,nodev,noexec,mode=755
│ ├─/sys/fs/cgroup/unified          cgroup2    cgroup2 rw,nosuid,nodev,noexec,relatime,nsdelegate
│ ├─/sys/fs/cgroup/systemd          cgroup     cgroup  rw,nosuid,nodev,noexec,relatime,xattr,name=systemd
│ ├─/sys/fs/cgroup/perf_event       cgroup     cgroup  rw,nosuid,nodev,noexec,relatime,perf_event
│ ├─/sys/fs/cgroup/net_cls,net_prio cgroup     cgroup  rw,nosuid,nodev,noexec,relatime,net_cls,net_prio
│ ├─/sys/fs/cgroup/cpu,cpuacct      cgroup     cgroup  rw,nosuid,nodev,noexec,relatime,cpu,cpuacct
│ ├─/sys/fs/cgroup/rdma             cgroup     cgroup  rw,nosuid,nodev,noexec,relatime,rdma
│ ├─/sys/fs/cgroup/devices          cgroup     cgroup  rw,nosuid,nodev,noexec,relatime,devices
│ ├─/sys/fs/cgroup/memory           cgroup     cgroup  rw,nosuid,nodev,noexec,relatime,memory
│ ├─/sys/fs/cgroup/hugetlb          cgroup     cgroup  rw,nosuid,nodev,noexec,relatime,hugetlb
│ ├─/sys/fs/cgroup/freezer          cgroup     cgroup  rw,nosuid,nodev,noexec,relatime,freezer
│ ├─/sys/fs/cgroup/pids             cgroup     cgroup  rw,nosuid,nodev,noexec,relatime,pids
│ ├─/sys/fs/cgroup/blkio            cgroup     cgroup  rw,nosuid,nodev,noexec,relatime,blkio
│ └─/sys/fs/cgroup/cpuset           cgroup     cgroup  rw,nosuid,nodev,noexec,relatime,cpuset
```

Interact with Control Group

Helpers from *cgroup-tools*

```
root@basic:/home/vagrant# cgexec --help
Usage: cgexec [-h] [-g <controllers>:<path>] [--sticky] command [arguments] ...
Run the task in given control group(s)
  -g <controllers>:<path>       Control group which should be added
  -h, --help                    Display this help
  --sticky                      cgred daemon does not change pidlist and children tasks
root@basic:/home/vagrant# cgcreate --help
Usage: cgcreate [-h] [-f mode] [-d mode] [-s mode] [-t <tuid>:<tgid>] [-a <agid>:<auid>] -g <controllers>:<path> [-g ...]
Create control group(s)
  -a <tuid>:<tgid>              Owner of the group and all its files
  -d, --dperm=mode              Group directory permissions
  -f, --fperm=mode              Group file permissions
  -g <controllers>:<path>       Control group which should be added
  -h, --help                    Display this help
  -s, --tperm=mode              Tasks file permissions
  -t <tuid>:<tgid>              Owner of the tasks file
```

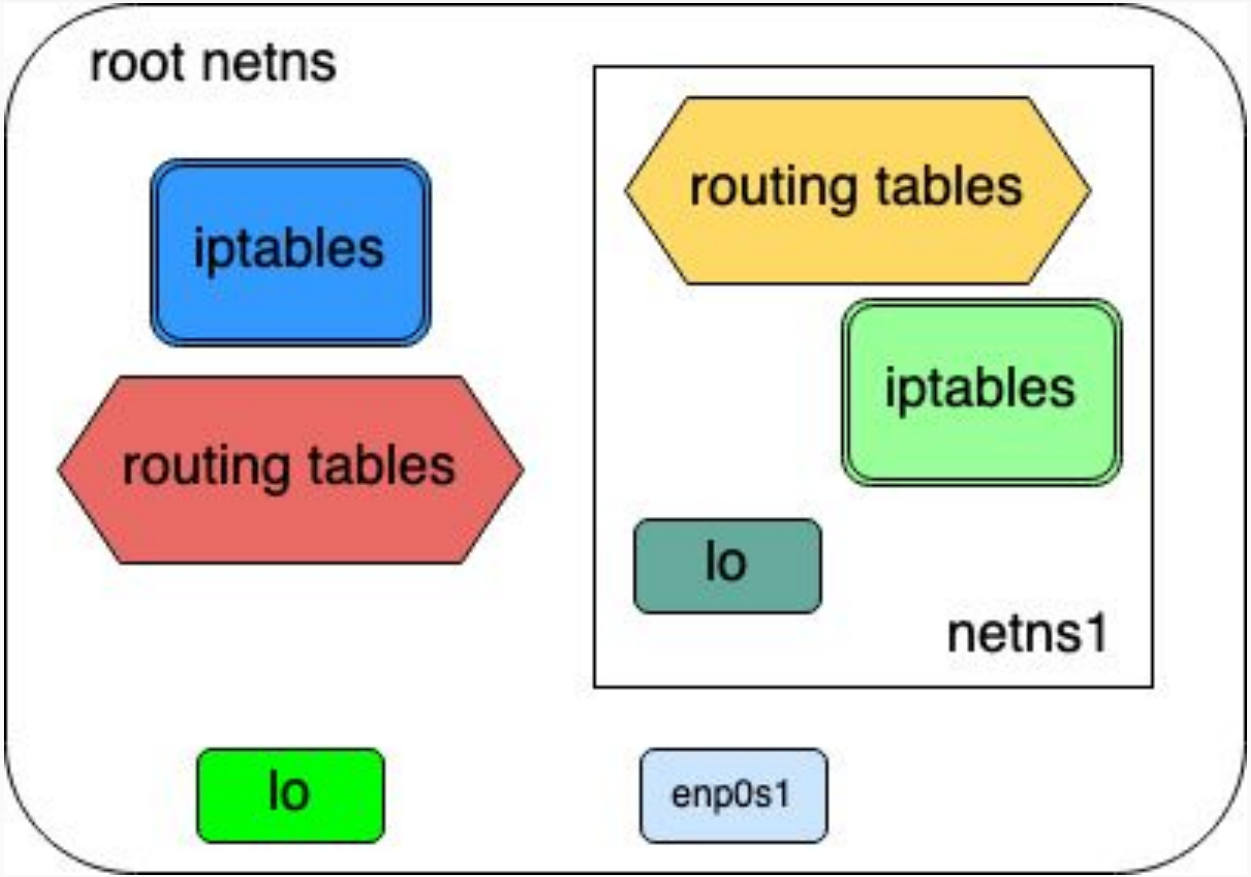
Container Networking

Which problems do we need to solve?

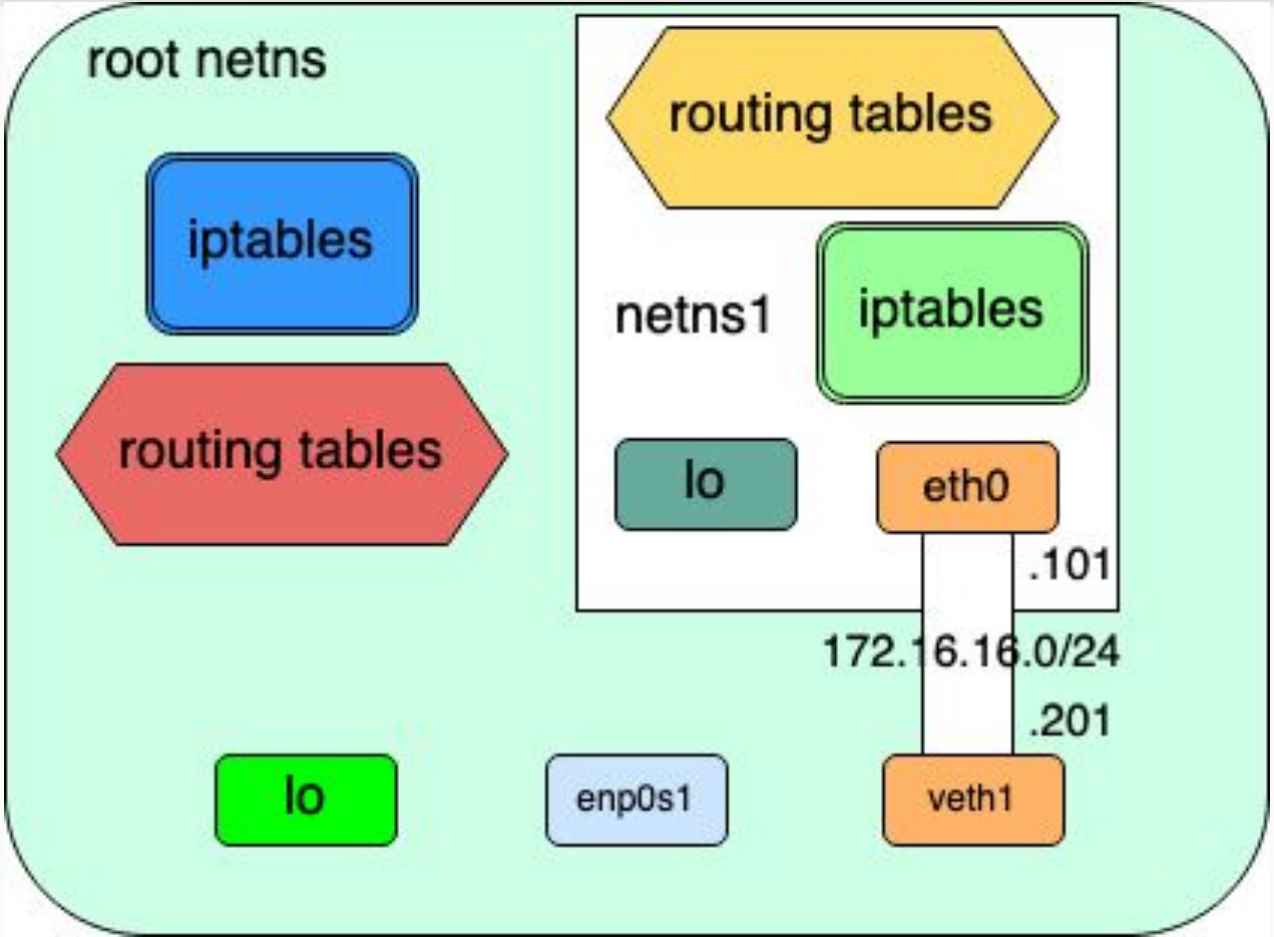
- How to make containers think each of them has a dedicated network stack?
- How container communicate with each others & with host
- How to reach the outside world (e.g. the Internet) from inside the container?
- How to reach containers from the outside world (*aka* port publishing)?

Hand-On

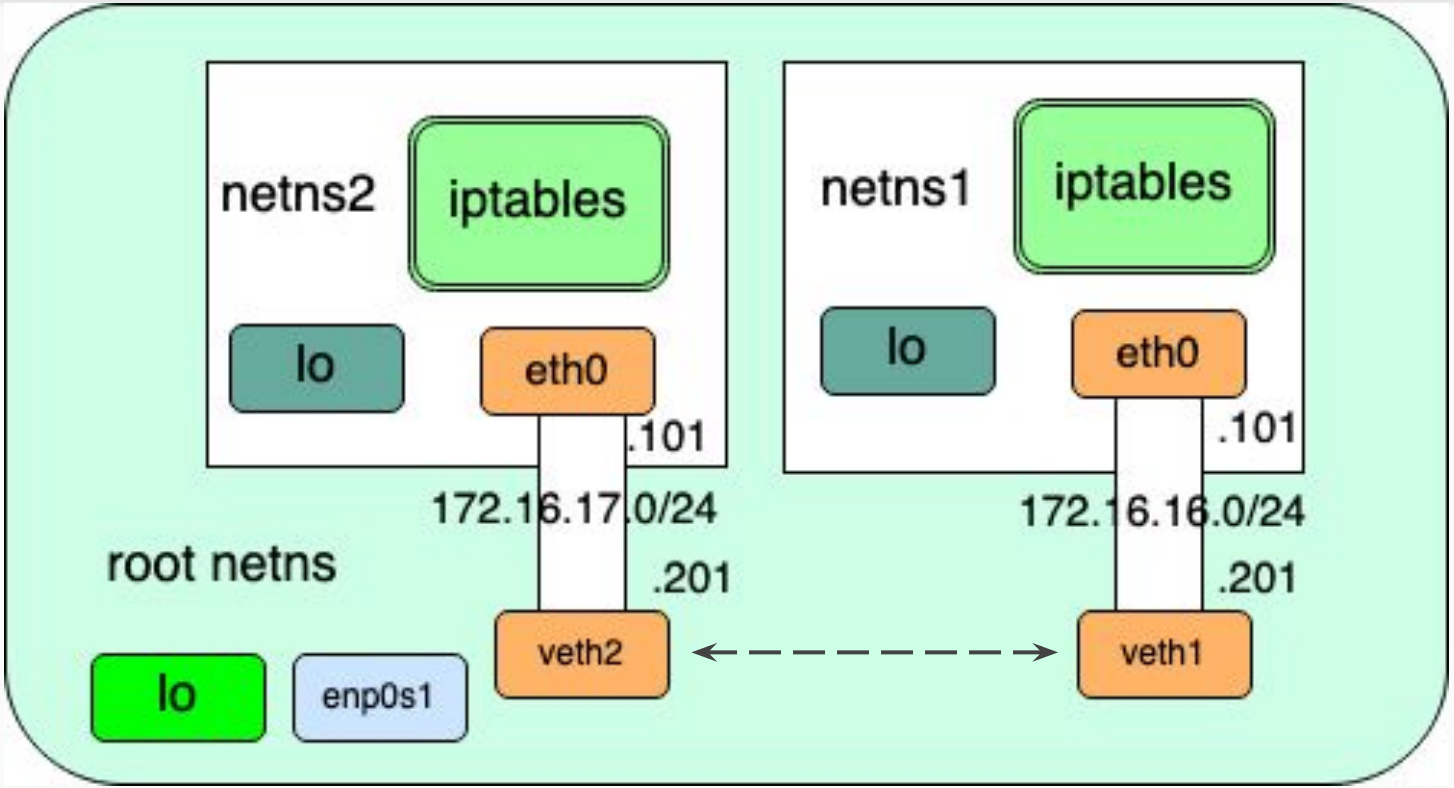
Container Network - Dedicated Networking Stack



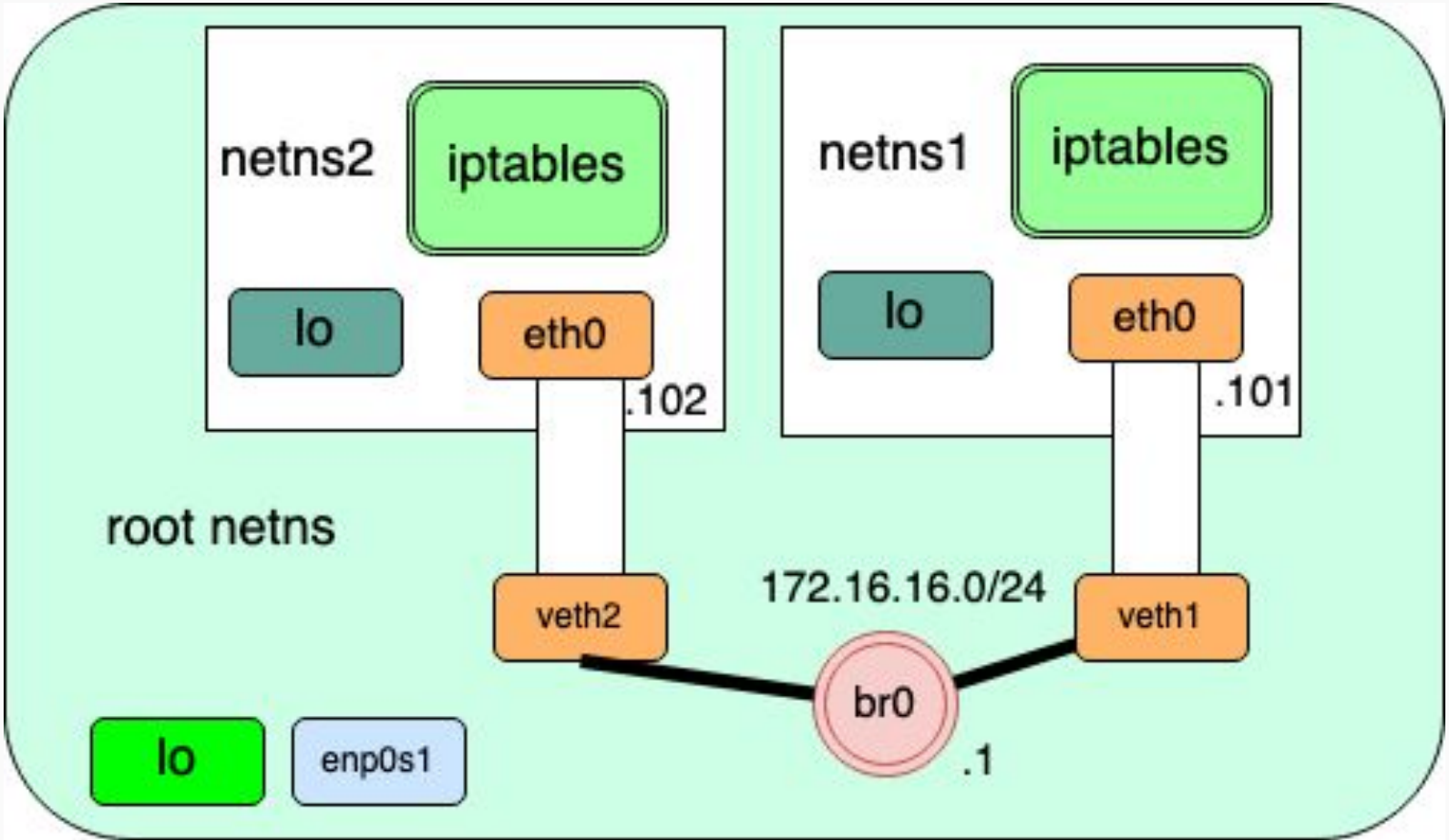
Container Network - Communication With Host



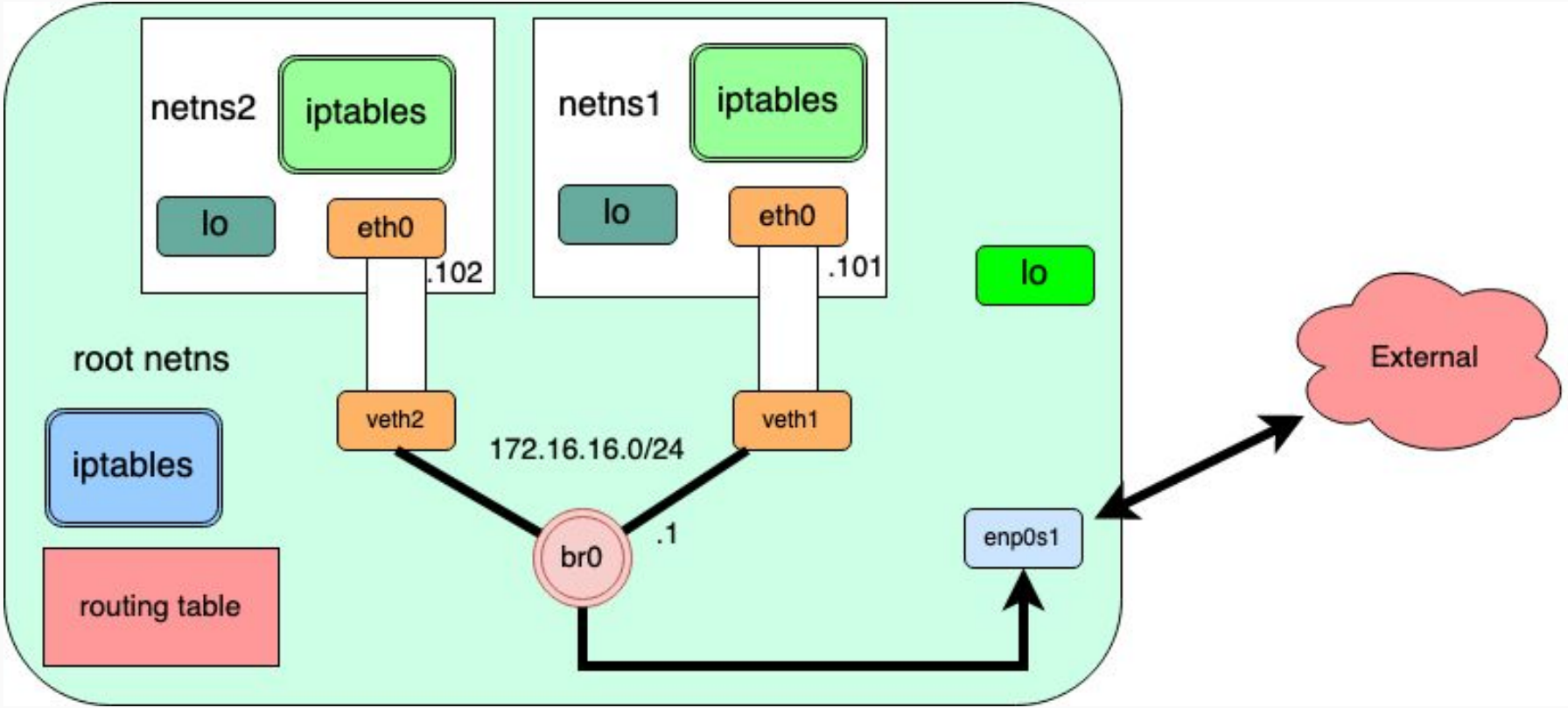
Container Network - Communication With Other Containers



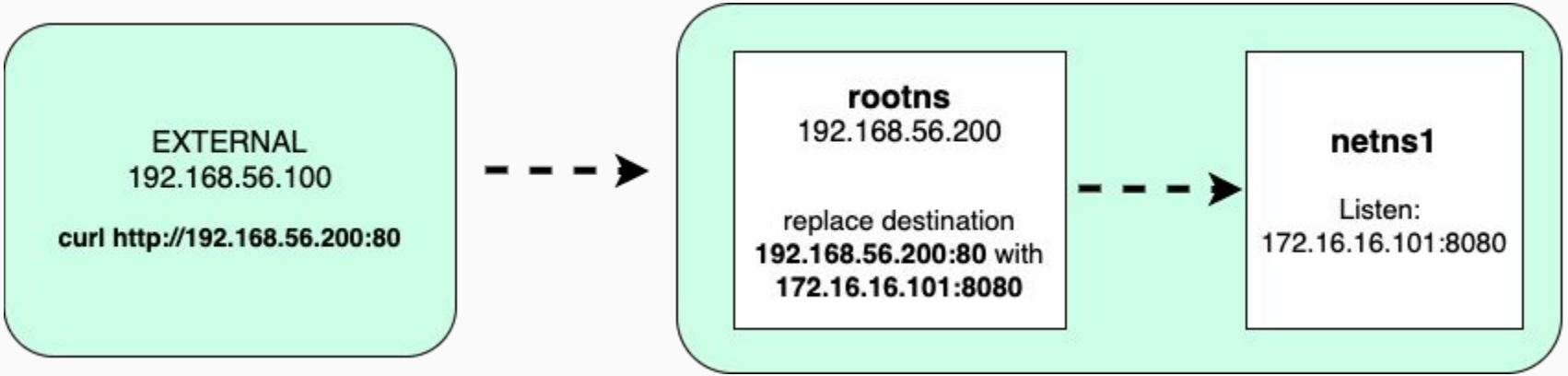
Container Network - Communication With Other Containers



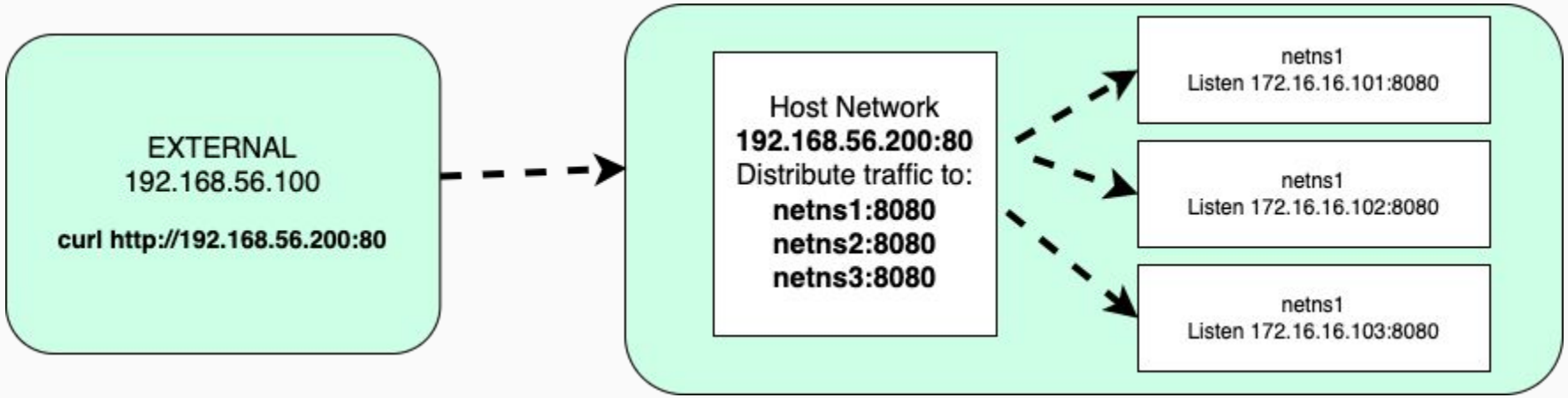
Container Network - External communication



Container Network - Port Forwarding

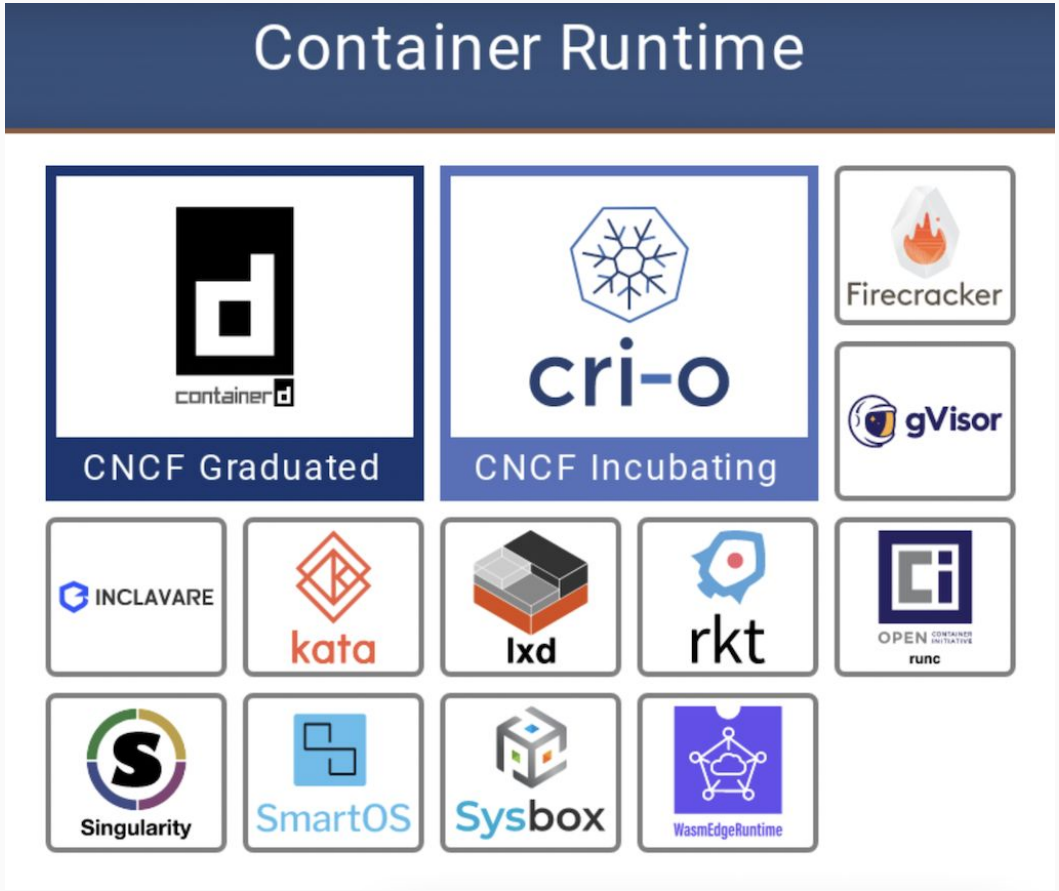


Container Network - Port Forwarding

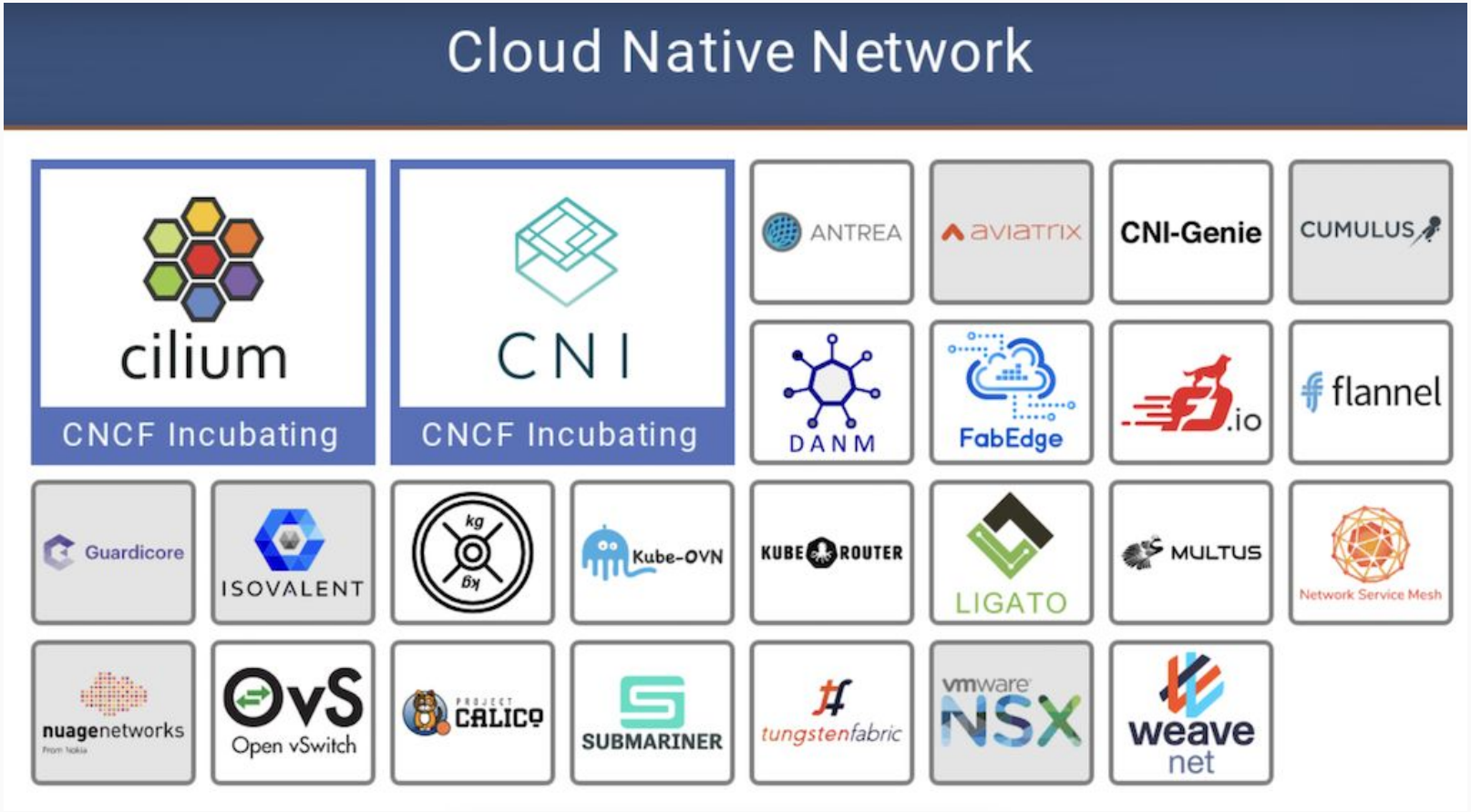


CNI - Container Network Interface

Why do we need CNI ?



Why do we need CNI ?

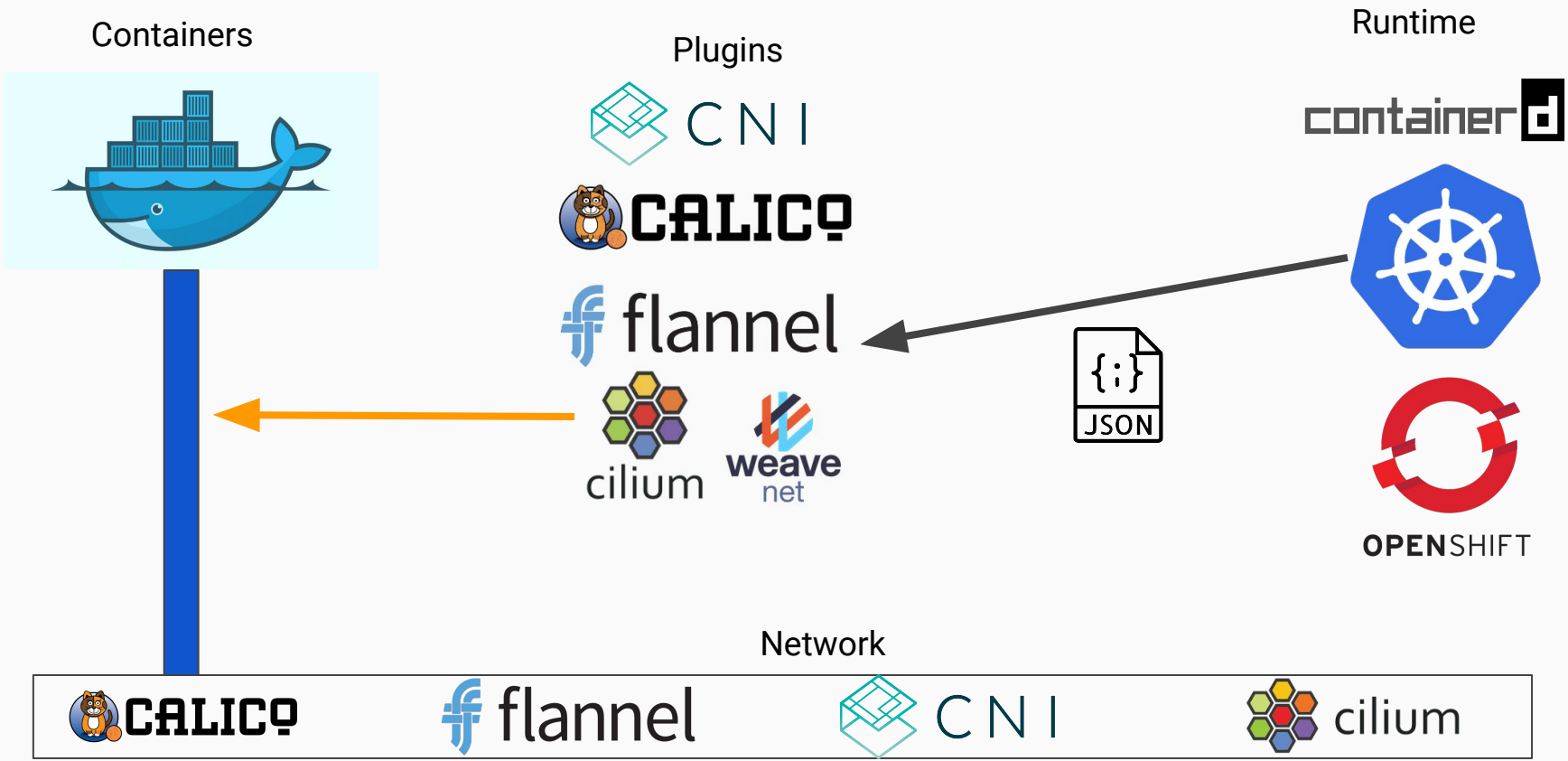


CNI - Container Network Interface

The CNI project has 2 major parts

- The CNI specification documents
  - Go Libraries - https://github.com/containernetworking/cni/tree/main/libcni
  - A specification - https://github.com/containernetworking/cni/blob/main/SPEC.md
- A set of based plugins - https://github.com/containernetworking/plugins
  - Interface management: create network interface/bridge/vlan
  - IP Address allocation: allocate IP address interfaces
  - Others chained plugin
    - tuning : tuning sysctl network parameters
    - portmap : iptables based port mapping
    - bandwidth : bandwidth-limiting for ingress/egress traffic
    - firewall: using iptables to allow/deny traffic to/from container
    - sbr: doing source-based routing

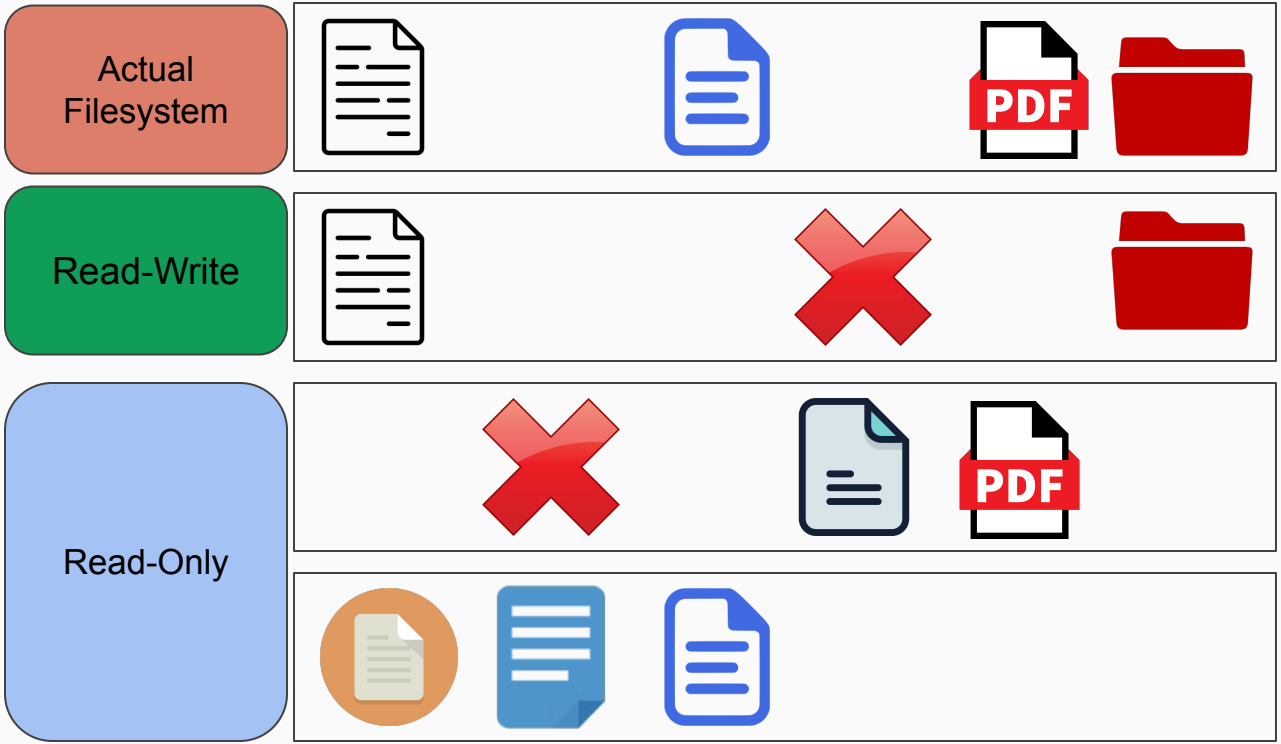
How CNI fit into the picture?



Demo - Using CNI with Docker

Union Mount

What the heck is it?



Yeah, I knew Union Mount … but Why?

- Contain images are quite big
  - *It's expensive to allocate that much space every time we'd like to create a container from these images*
- Thanks to union filesystem(s)
  - *we only needs to create thin layer on top of the image and rest of it can be shared between all the containers.*
- Reduce startup time
  - *as there's no need to copy the image files and data*
- Copy-on-write strategy if you want to modify those shared read-only files

How does it implement in reality

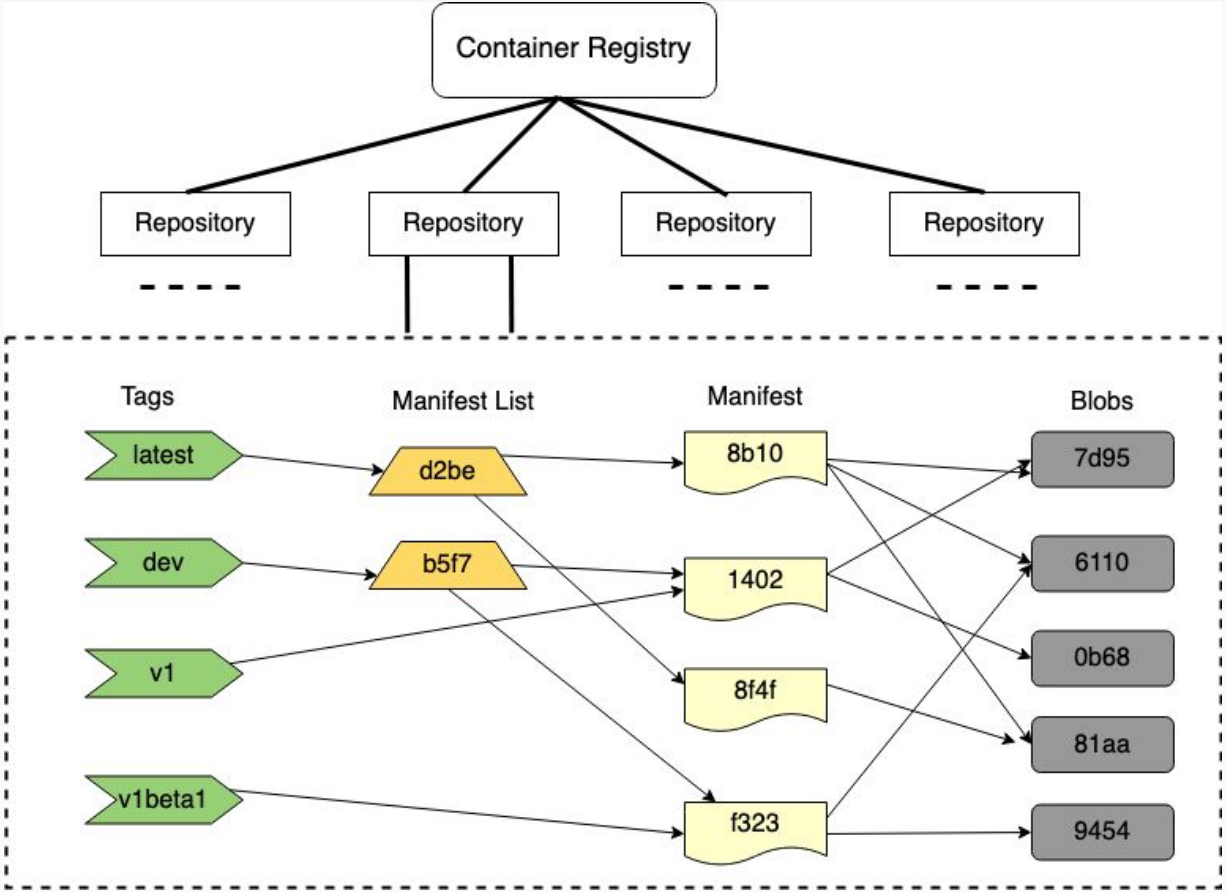
- UnionFS
  - The original union filesystem
  - UnionFS doesn't seem to be actively developed anymore
- AUFS
  - A re-implemenation of original UnionFS that added many new features….
  - But was rejected for merging into mainline Linux kernel. *(Its code was criticized for being "dense, unreadable, [and] uncommented")*
  - Was default driver for Docker on Ubuntu/Debian but was replaced by OverlayFS
- OverlayFS
  - This is the filesystem used by default overlay2 Docker driver
  - It generally has better performance then aufs and has some nice features such as page cache sharing.
- ZFS
  - Created by Sun Microsystems (now Oracle)
  - Can not be shipped as part of Linux kernel because of licensing.
  - You could however use the ZFS on Linux (ZoL) project, but not ready for production.
- Btrfs
  - Btrfs is a default filesystem of Fedora 33.
  - It also has some useful features such as block-level operations, defragmentation, writeable snapshots and a lot more.

How Docker container filesystem looks like?

Toy Container Runtime

Container Images

Container Image

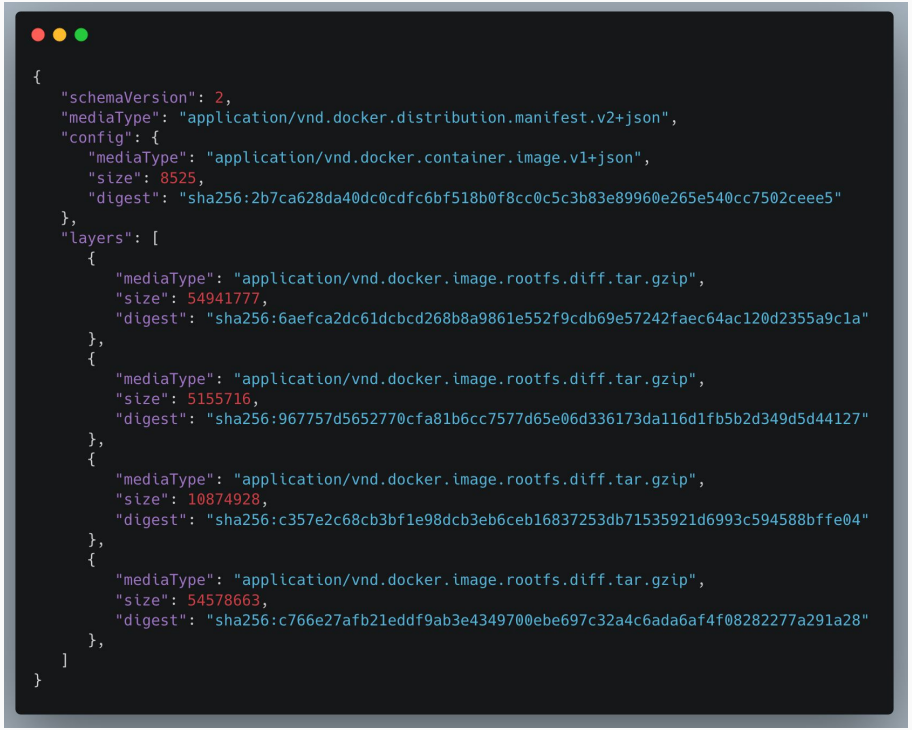


Container Image

- Layers
  - Container filesystem is split into multiple parts
  - Each layer is store in a single gzipped tarball
- Container Configuration: Contain metadata about images
  - OS/Architecture
  - Exposed ports/Entrypoint/Volumes/Env Variables/Labels/WorkingDir/User/Env Vars
  - A ordered list of digests (usually SHA-256) of **uncompressed** layer tarballs
  - A list of statement/command used to build the image - image history
- Blobs
  - Artifacts store in image registry (layers & manifest & configurations)
- Digests
  - Hash value of blobs
  - Used to uniquely identify blobs
  - Docker image use SHA-256 (OCI format support SHA-512 also)
  - Example: *sha256:09addbcf0db5a11803f29bddbdbfd31adce7e40d68750359f9a4eb4dcc54078f*
- Manifest: define the components that make up an image on a container registry
- Manifest list (fat manifest): which references image manifests for platform-specific versions of an image.

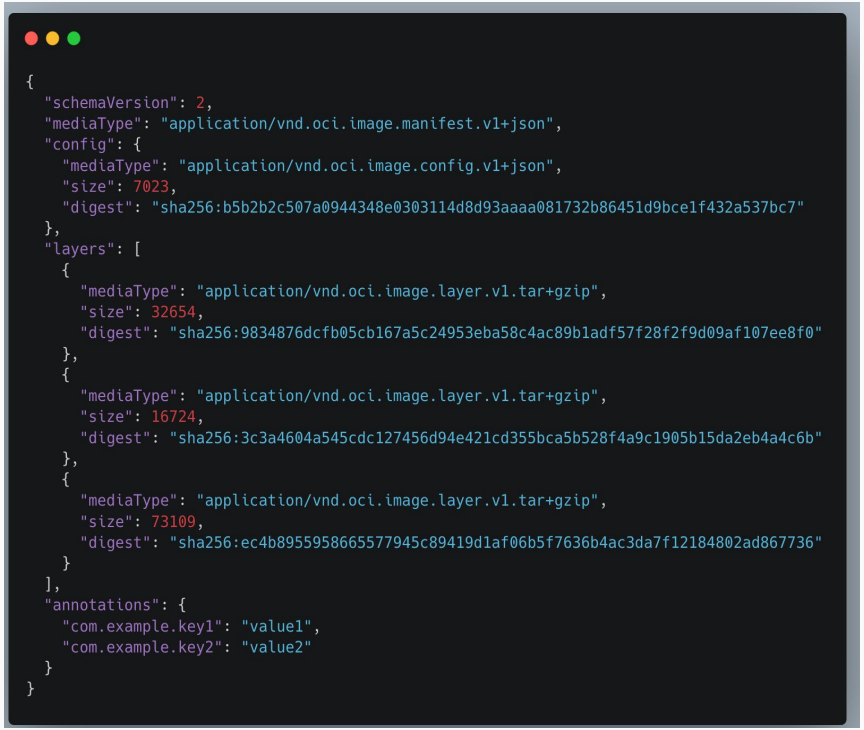
Image Manifest

Image Manifest V2, Schema 2



```
{
   "schemaVersion": 2,
   "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
   "config": {
      "mediaType": "application/vnd.docker.container.image.v1+json",
      "size": 8525,
      "digest": "sha256:2b7ca628da40dc0cdfc6bf518b0f8cc0c5c3b83e89960e265e540cc7502ceee5"
   },
   "layers": [
      {
         "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
         "size": 54941777,
         "digest": "sha256:6aefca2dc61dcbcd268b8a9861e552f9cdb69e57242faec64ac120d2355a9c1a"
      },
      {
         "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
         "size": 5155716,
         "digest": "sha256:967757d5652770cfa81b6cc7577d65e06d336173da116d1fb5b2d349d5d44127"
      },
      {
         "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
         "size": 10874928,
         "digest": "sha256:c357e2c68cb3bf1e98dcb3eb6ceb16837253db71535921d6993c594588bffe04"
      },
      {
         "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
         "size": 54578663,
         "digest": "sha256:c766e27afb21eddf9ab3e4349700ebe697c32a4c6ada6af4f08282277a291a28"
      },
   ]
}
```

OCI Image Manifest



```
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.oci.image.manifest.v1+json",
  "config": {
    "mediaType": "application/vnd.oci.image.config.v1+json",
    "size": 7023,
    "digest": "sha256:b5b2b2c507a0944348e0303114d8d93aaaa081732b86451d9bce1f432a537bc7"
  },
  "layers": [
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "size": 32654,
      "digest": "sha256:9834876dcfb05cb167a5c24953eba58c4ac89b1adf57f28f2f9d09af107ee8f0"
    },
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "size": 16724,
      "digest": "sha256:3c3a4604a545cdc127456d94e421cd355bca5b528f4a9c1905b15da2eb4a4c6b"
    },
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "size": 73109,
      "digest": "sha256:ec4b8955958665577945c89419d1af06b5f7636b4ac3da7f12184802ad867736"
    }
  ],
  "annotations": {
    "com.example.key1": "value1",
    "com.example.key2": "value2"
  }
}
```

Image Manifest List

Docker Image Manifest List v2	OCI Image Index


```
{
  "manifests": [
    {
      "digest": "sha256:76f9e442fdf5c12efb5949407b0bb7ad28a413b8a5387a4243b1d43a14654060",
      "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
      "platform": {
        "architecture": "amd64",
        "os": "linux"
      },
      "size": 2218
    },
    {
      "digest": "sha256:04c13ef2cc265f31aa90decf9ba111f715b82d8c80dc2f84b13b1bce31b22f77",
      "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
      "platform": {
        "architecture": "arm64",
        "os": "linux",
        "variant": "v8"
      },
      "size": 2218
    }
  ],
  "mediaType": "application/vnd.docker.distribution.manifest.list.v2+json",
  "schemaVersion": 2
}
```

```
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.oci.image.index.v1+json",
  "manifests": [
    {
      "mediaType": "application/vnd.oci.image.manifest.v1+json",
      "size": 7143,
      "digest": "sha256:e692418e4cbaf90ca69d05a66403747baa33ee08806650b51fab815ad7fc331f",
      "platform": {
        "architecture": "ppc64le",
        "os": "linux"
      }
    },
    {
      "mediaType": "application/vnd.oci.image.manifest.v1+json",
      "size": 7682,
      "digest": "sha256:5b0bcabd1ed22e9fb1310cf6c2dec7cdef19f0ad69efa1f392e94a4333501270",
      "platform": {
        "architecture": "amd64",
        "os": "linux"
      }
    }
  ],
  "annotations": {
    "com.example.key1": "value1",
    "com.example.key2": "value2"
  }
}
```

Meeting Container Registry API V2

Dockerfile Good Practices

Dockerfile Good Practices

- Using multi-stages Dockerfile
- Using ‘.dockerignore’
- Using Dockerfile linter
  - hadolint: https://github.com/hadolint/hadolint
- Using COPY/ADD options to change file metadata (permissions/owner)
- Write Cache Friendly Dockerfile
- Using BuildKit cache mount
- Using secrets/ssh mount