MESOS & CONTAINERS

Overview of Mesos containerization and upcoming filesystem isolation support (a.k.a the docker like thing)

Yan Xu xujyan

WHAT IS A CONTAINER

- Loosely defined: a lightweight "VM" / OS-level virtualization / "chroot on steroids".
- To Mesos: a per-task/executor *isolated* execution environment.

DIMENSIONS OF CONTAINERIZATION

- **Performance** isolation: resource quota limiting. e.g. mem isolation.

- **Isolated visibility** from inside the container: stack separation, jailing. e.g., filesystem isolation.

- **Visibility** from the host: inspection, metrics.

Credit: http://cdn.diginomica.com/wp-content/uploads/2014/07/Fotolia-Oleksiy-Mark-50048132_Sub_M.jpg

CONTAINERIZATION: A CORE PREMISE OF MESOS RESOURCE MANAGEMENT

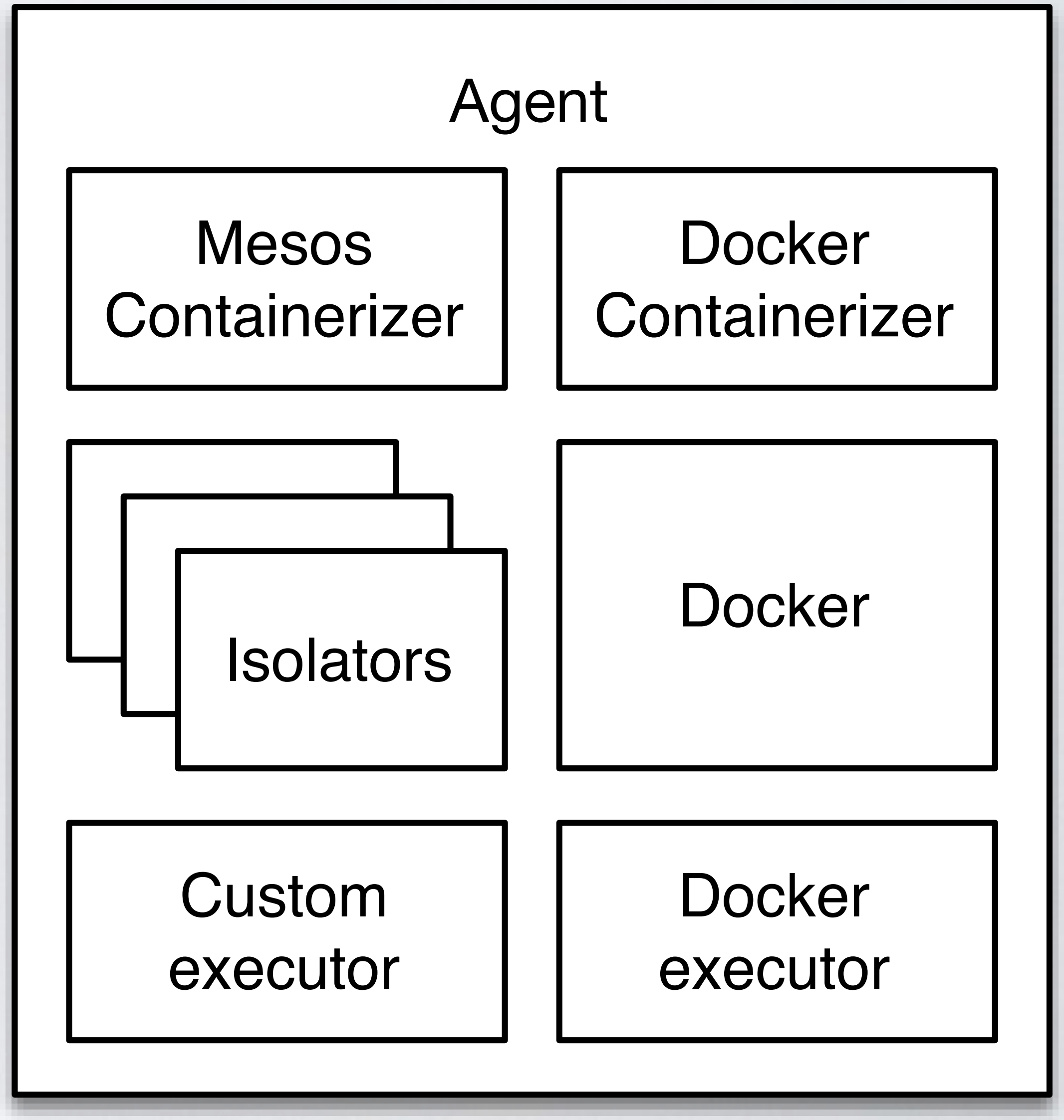
Can’t allocate resources without enforcement!

A BRIEF HISTORY OF MESOS CONTAINERIZATION

- LXC (2010)
- Cgroups (2012)
- Linux namespaces (2013)
- Docker* (2014)

THE TALE OF TWO CONTAINERIZERS

- MesosContainerizer (default)
- DockerContainerizer
- Dynamically chosen based on ContainerInfo if both are specified via `--containerizers`.



CURRENT MESOS CONTAINERIZER LINEUP

- **Performance** isolation
    - cpu, mem, disk quota, network egress bandwidth
- **Isolated visibility** from inside
    - pid, network (port mapping)
- **Visibility** from the host
    - perf_event, other cgroup stats and network stats, etc.

DOCKER IS GREAT, BUT...

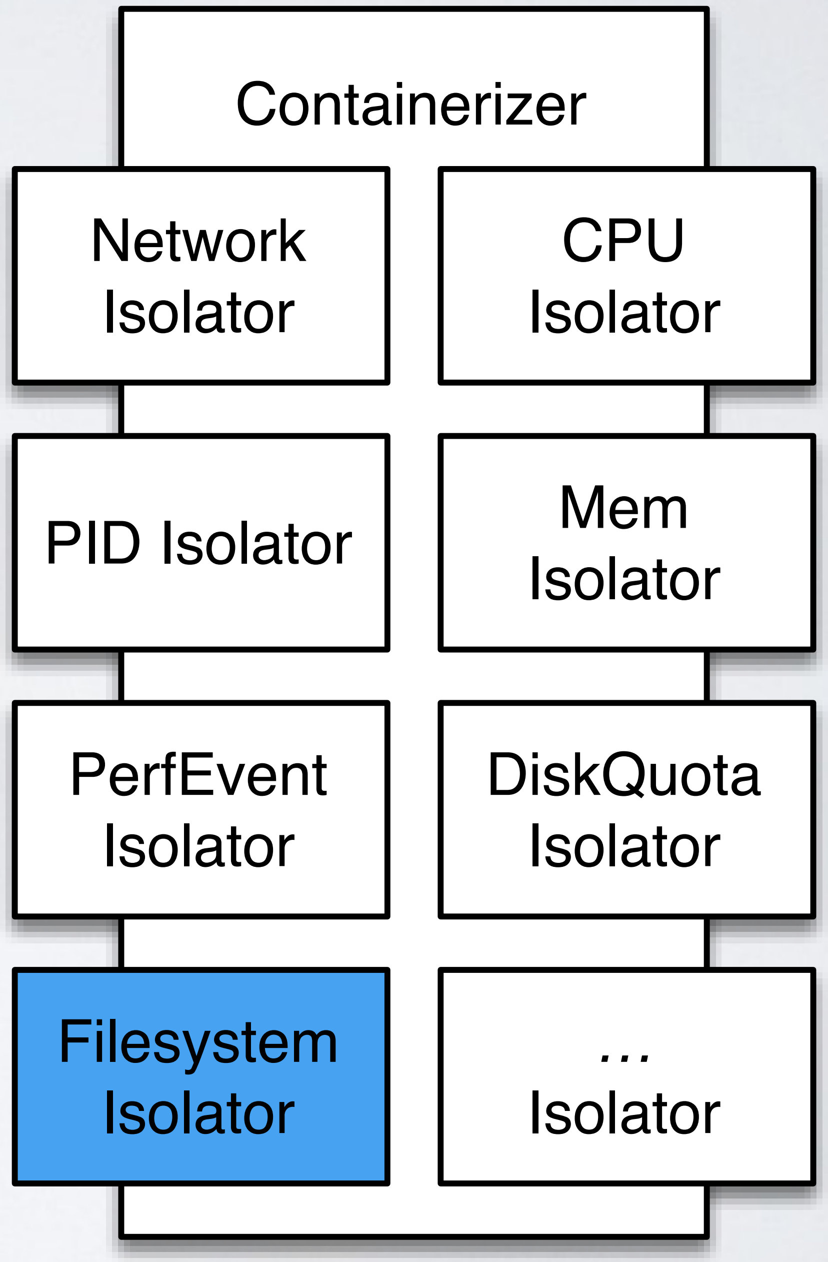
- Requires Docker installation and maintenance.
- Tasks die with Docker daemon (upgrade, etc.)
- Limited performance isolation done by Mesos.
- Cannot compose with Mesos isolators (disk quota, port mapping).
- Complexity in managing task lifecycle.
- Hard to take advantage of other Mesos features: disk quota enforcement with persistent volumes; IP per container, etc.

A UNIVERSAL MESOS CONTAINERIZER

- An all-encompassing containerizer for performance isolation, visibility isolation and metering.
- Compossible: each isolation is implemented as an Isolator and configured independently.
- Container resources are mutable during container lifecycle.
- Tightly integrated with Mesos task/executor.

MESOS CONTAINERIZER

- "The Docker thing": filesystem isolation.
- Extensible: new isolators such as are added and configured independently.
- Filesystem isolator also handles cases without a new rootfs.



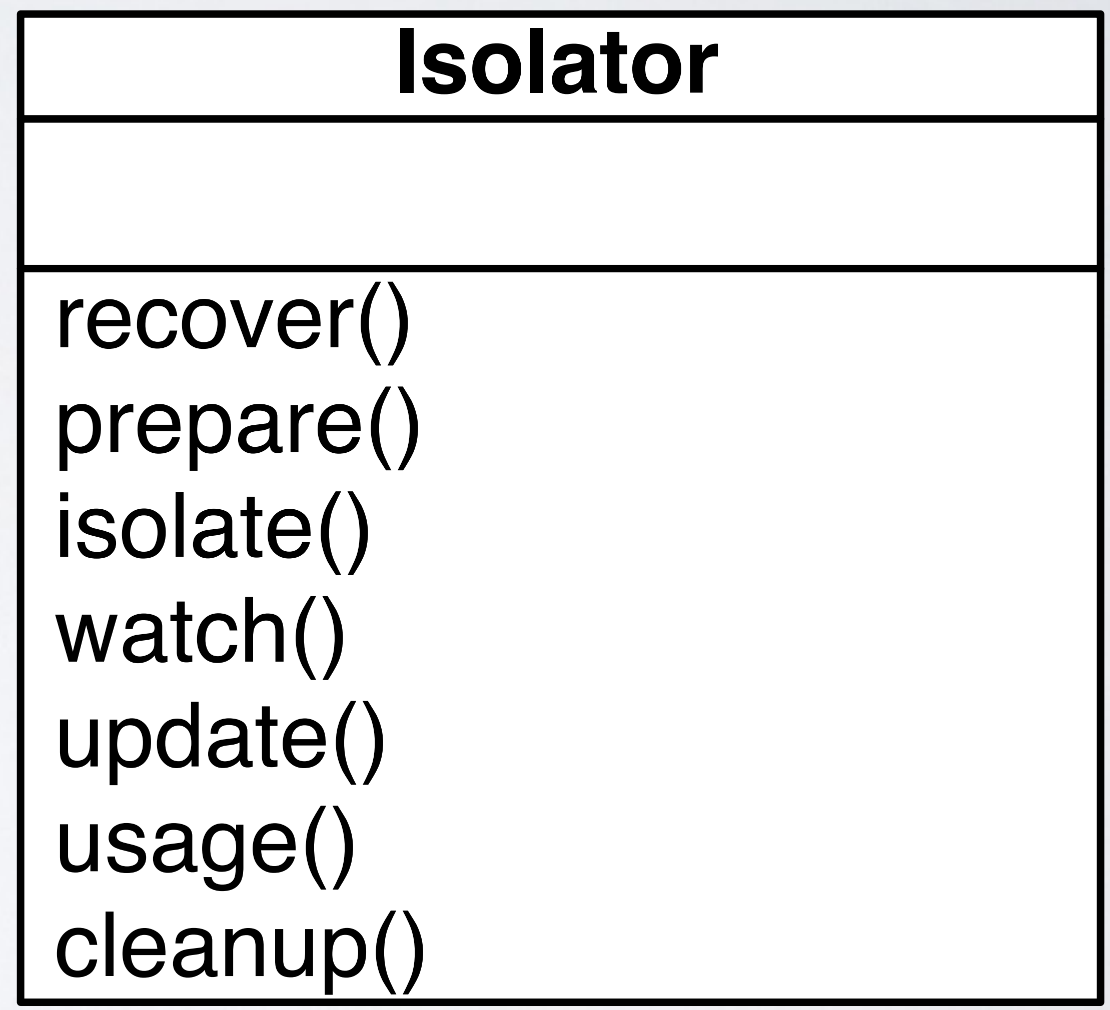
CONTAINERIZER

- Recovery: agent crash tolerance.
- Update: grow and shrink container as needed.
- Usage: container statistics.
- Wait: tied to executor lifecycle.

Containerizer
recover() launch() update() usage() wait() destroy()

ISOLATOR

- Prepare: set up container isolation feature. e.g., create cgroups.
- Isolate: isolate the process. e.g., write control files.
- Watch: enforce isolation, report violation.



docker

appc

FILESYSTEM PROVISIONING AND ISOLATION

CONTAINER SPECS

What's in it

- Filesystem contents: rootfs(es)
- Manifest / static configuration:
  - Version, dependencies, etc.
  - Mounts points
  - App: env, cmd, args, etc.

CONTAINER SPECS

How to run it

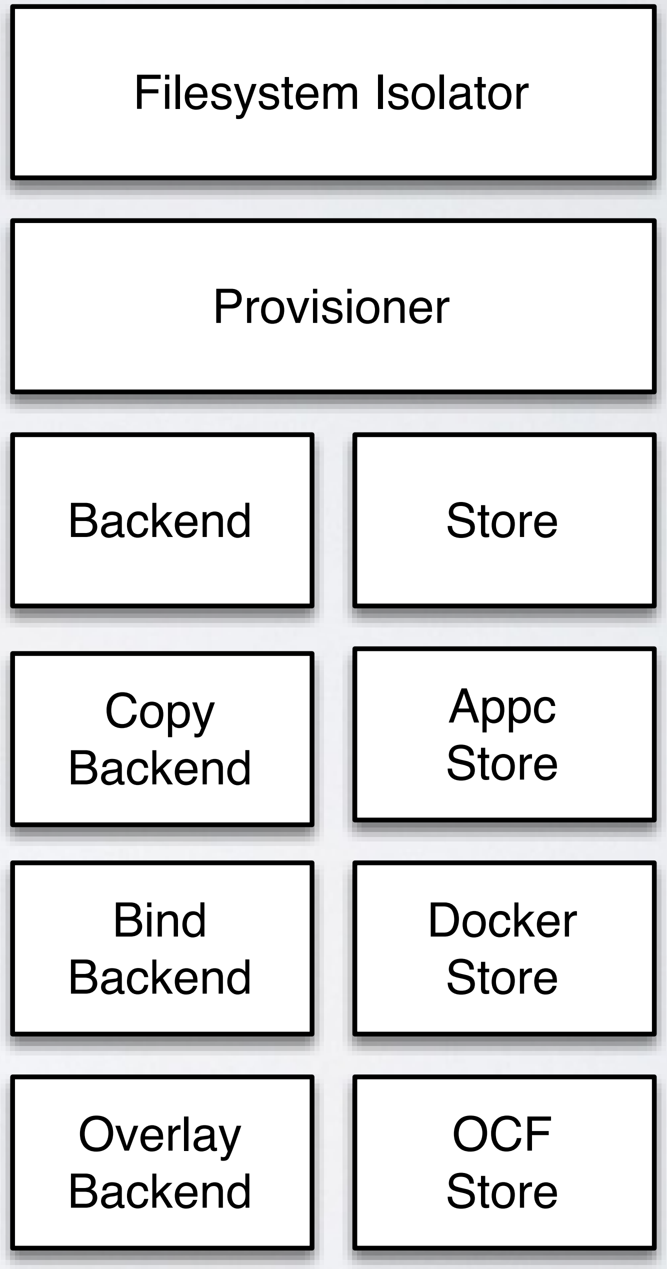
- Runtime configuration
  - hooks
  - mounts (volumes)
  - Resources: cpus, mem, disk, etc.

FILESYSTEM ISOLATION

- With a new rootfs.
  - Decoupling from the host filesystem allow better application portability and infrastructure flexibility.
- Without a new rootfs.
  - Volumes isolated inside the container mount namespace.
  - Mesos allows volume sources to be container images so the framework executor is not jailed but it can isolate its end-user logic inside a container rootfs.
- Other aspects of isolation
  - Mounting <work_dir>/tmp as /tmp.

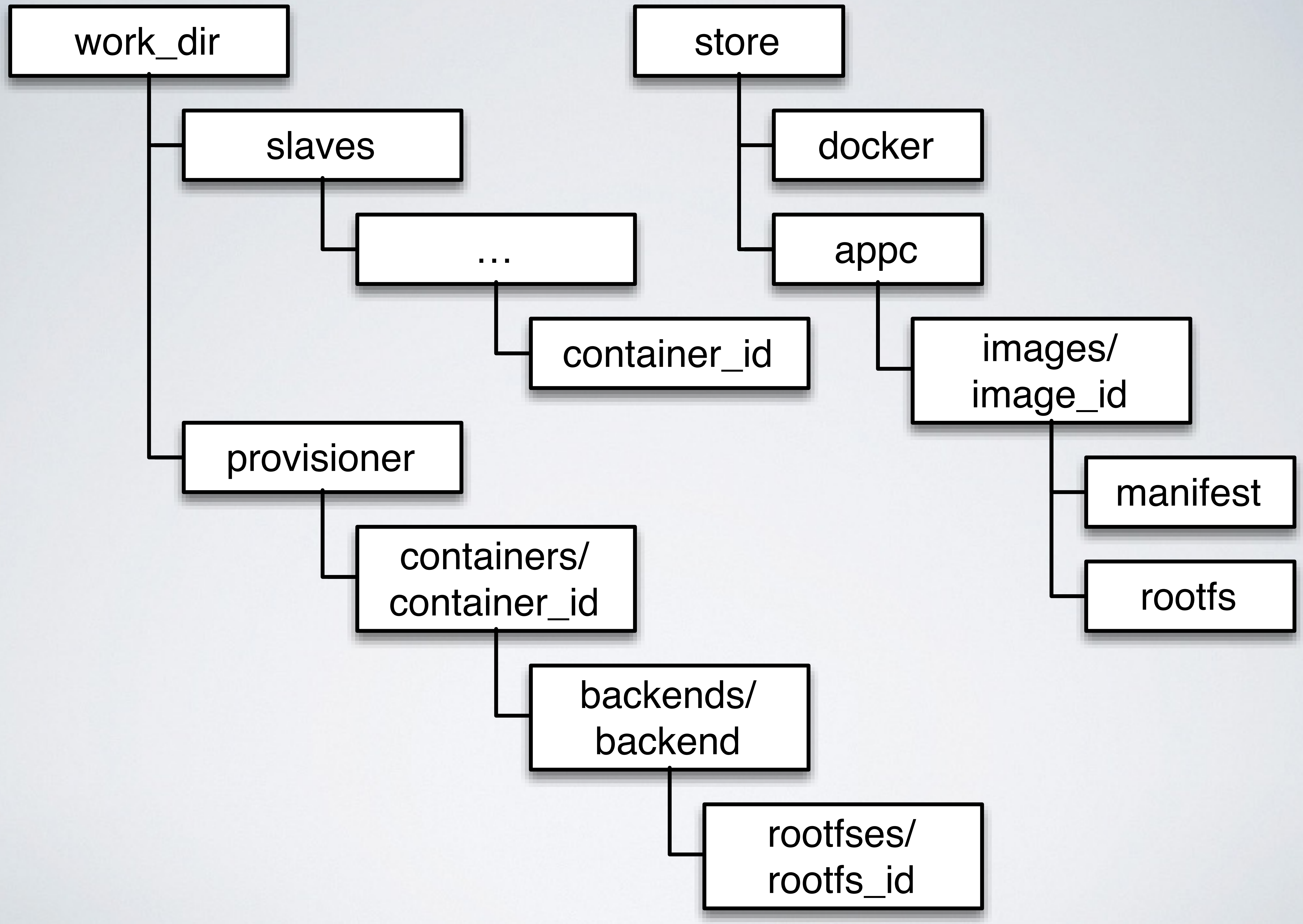
FILESYSTEM PROVISIONING

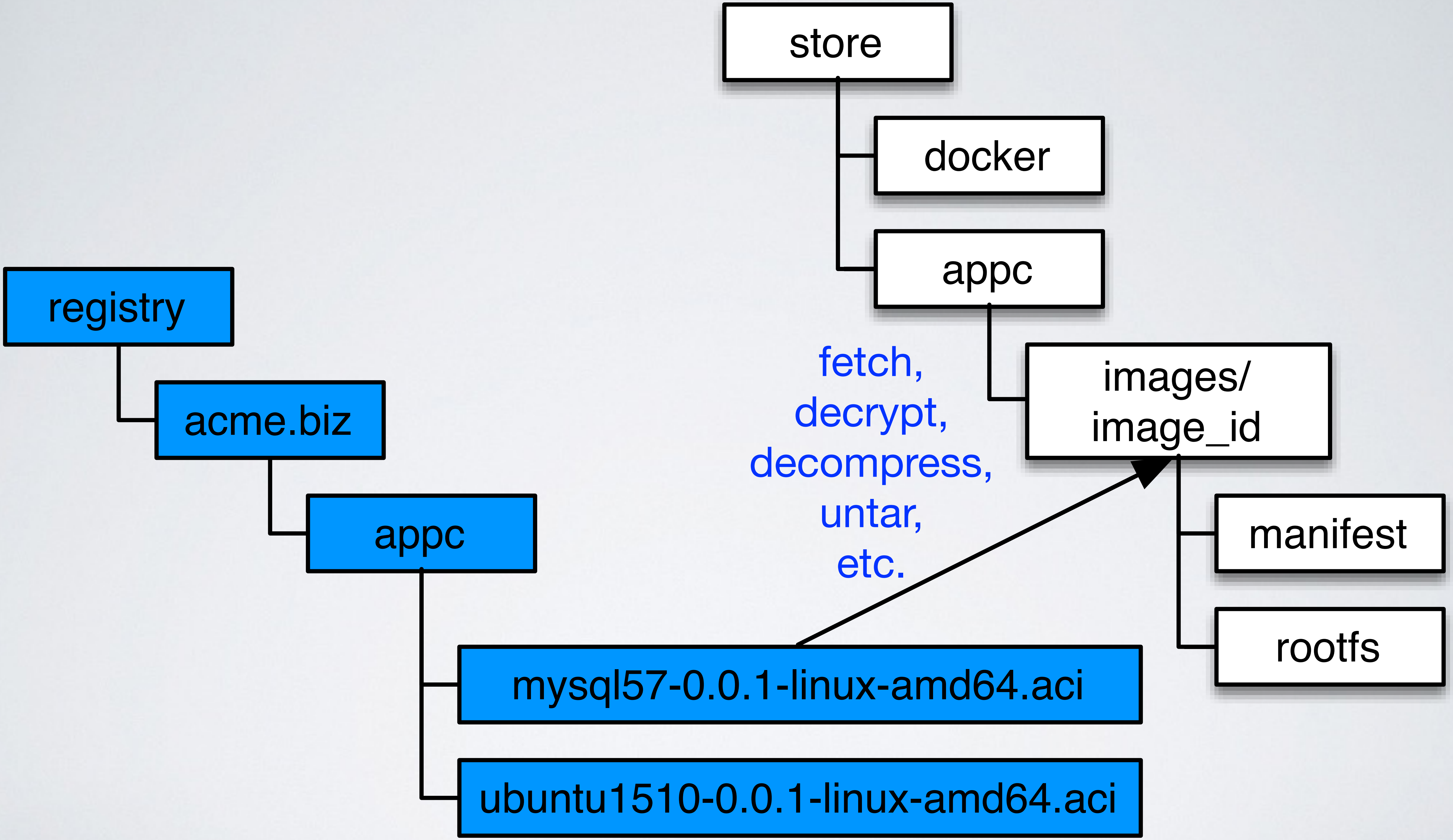
- A universal provisioner for multiple images types.
- Vendor specific store which does discover, fetching and processing.
- Provision rootfs (e.g., via bind mount).

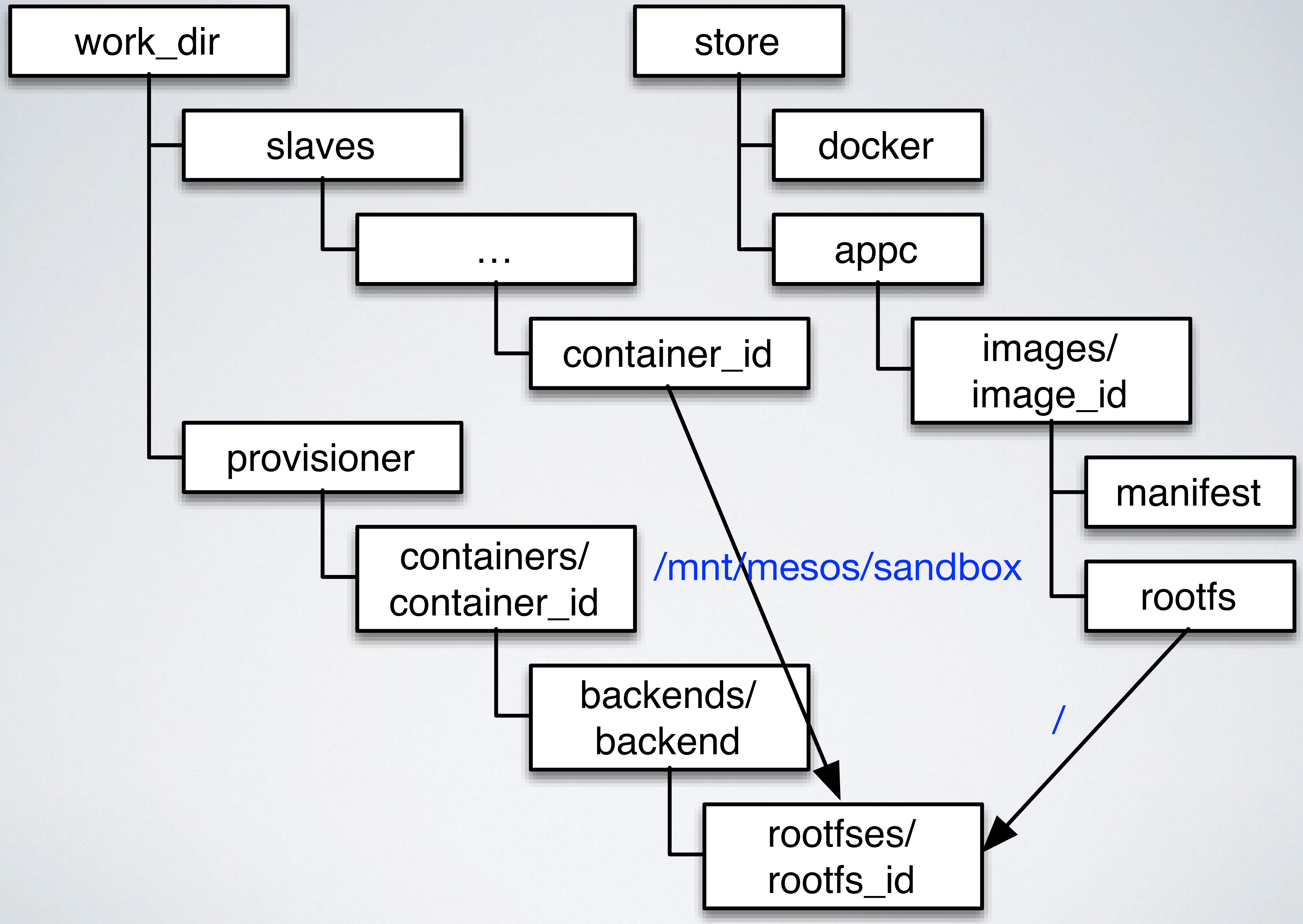


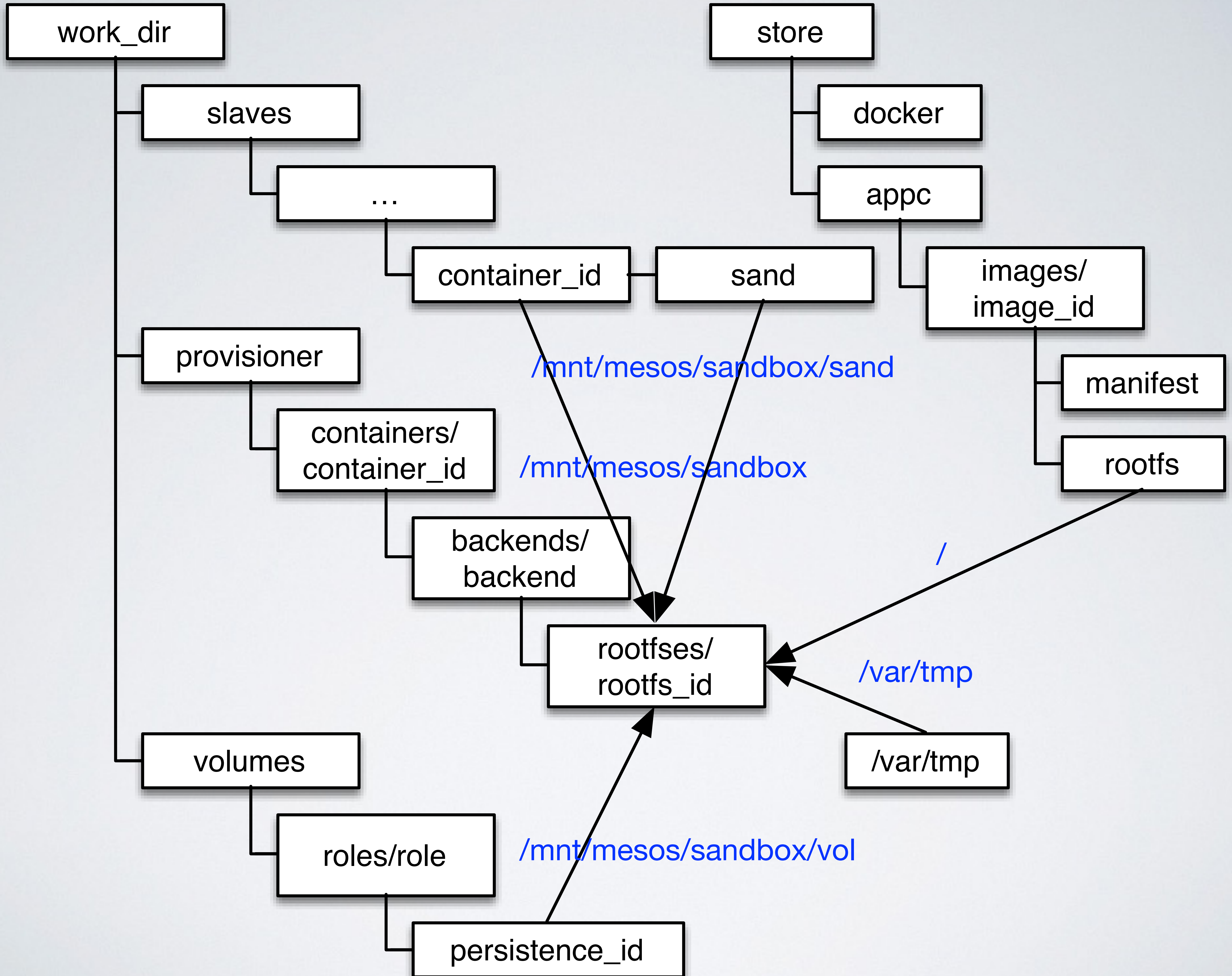
SAMPLE CONTAINER INFO


```
{
  "type" : "MESOS",
  "mesos" : {
    "image" : {
      "type" : "APPC",
      "appc" : {
        "name" : "acme.biz/appc/ubuntu1510",
        "labels" : {
          "labels": [{"key" : "version", "value" : "0.0.1"}]
        }
      }
    }
  },
  "volumes": [
    {"container_path" : "/tmp",  "host_path" : "tmp",  "mode" : "RW"},
    {"container_path" : "/root", "host_path" : "/root",  "mode" : "RW"},
    {"container_path" : "/etc",  "host_path" : "/etc",  "mode" : "RO"},
    {"container_path" : "/var/run",  "host_path" : "/var/run",  "mode" : "RW"},
    {"container_path" : "/var/tmp",  "host_path" : "/var/tmp",  "mode" : "RW"}
  ]
}
```


work_dir
slaves
...
container_id
provisioner
containers/
container_id
backends/
backend
rootfses/
rootfs_id
store
docker
appc
images/
image_id
manifest
rootfs


store
docker
appc
images/
image_id
manifest
rootfs
registry
acme.biz
appc
mysql57-0.0.1-linux-amd64.aci
ubuntu1510-0.0.1-linux-amd64.aci
fetch,
decrypt,
decompress,
untar,
etc.


work_dir
slaves
...
container_id
provisioner
containers/
container_id
backends/
backend
rootfses/
rootfs_id
store
docker
appc
images/
image_id
manifest
rootfs
/mnt/mesos/sandbox
/



work_dir
slaves
...
container_id
sand
provisioner
containers/
container_id
backends/
backend
rootfses/
rootfs_id
volumes
roles/role
persistence_id
store
docker
appc
images/
image_id
manifest
rootfs
/var/tmp
/mnt/mesos/sandbox/sand
/mnt/mesos/sandbox
/
/var/tmp
/mnt/mesos/sandbox/vol

CONTAINERIZE A LARGE FLEET

CONTAINERIZE YOUR EXISTING CLUSTERS

- Tight coupling with the host accumulated over time.
- Start with a default container image identical to the host environment: fat images.
- Decouple tasks from the host environment: shrink the images; make tasks self-sufficient.
- Update the host environment independently from the containers.
- Separate environment into (a limited number of) image layers.

DECOUPLING DEPENDENCIES

- Software binary dependencies
  - Ideally containers are self-sufficient.
- Configuration dependencies
  - Ideally configuration are pulled from a service and not the host, but may have to bind mount from the host as a compromise.
  - How to push realtime configuration change down to each container without mounting in host config?
- How many layers should there be?
  - Ideally as few as possible and different logical layers managed by teams who own them.

PITFALLS DURING MIGRATION

- Applications rely on host environment (other than aforementioned binaries and configs), e.g., working directory path.
- Host services rely on information from "the contained application's view", e.g., `/proc/<pid>/cwd`, etc.
- Software binaries in the container don't match configuration from the host.

IMAGE IDENTIFICATION & VERIFICATION

- The curse of the ‘latest’ tag/version: is ‘latest’ latest?
- You don’t know if the image has changed until you’ve pulled it down (ETag helps).
- Use image ID for preciseness and immutability.
- Scenario: Emergency release of base image after fixing a zero-day vulnerability.

IMAGE PROVISIONING SCALABILITY

- Upgrade default image for O(10000) hosts.
- Images of GBs in size.
- Network bandwidth.
- What to do about tasks when the default image is still being fetched?

WHERE TO GO FROM HERE

- Persistent container filesystems.
- What are the high-level abstractions for managing and utilizing containers? Pods?
- Support OCF standard.
- Make sure containerization work with Mesos features: oversubscription, IP per container, etc.

EPHEMERAL VS. PERSISTENT CONTAINERS

- Copy-on-write filesystem: overlays
- Ephemeral read-only container filesystem: no top-layer; read-only rootfs with sandbox mounted in.
- Ephemeral writable container filesystem: top layer from sandbox.
- Persistent writable container filesystem: top layer from persistent volumes.

CONCLUSION

- Mesos is by far and away the most proven scalable and production-ready way to manage your containers.
- Filesystem isolation is only one element of it and there is cost and benefits with it.
- Not everything needs to run inside a new rootfs and you can still reap the benefits of other types of containerization even if you don’t.

CONCLUSION

- Still, migrating towards separate container filesystems is a good strategy for many organizations.
- Filesystem provisioning and isolation is WIP, will be released in the next couple of months.
- Mesos is not a container scheduler; it provides high-level cluster APIs and abstracts resources from hosts. Containerization serves this goal.

ACKNOWLEDGEMENTS

Contributors of the native filesystem isolation feature: Lily Chen, Tim Chen, Ian Downes, Jojy Varghese, Mei Wan, Yan Xu, Jie Yu, Chi Zhang.

QUESTIONS?