Distribute

5th High Performance Container Workshop - ISC19

Scope and Introduction

This segment focuses on **DISTRIBUTION** aspects

We do not talk about distribute or orchestrate.

We might talk about what runtime support is needed. :)

OCI Image Spec & Distribution

Akihiro Suda (@_AkihiroSuda_)
NTT Software Innovation Center

Open Containers Initiative Specifications

- **OCI Runtime Spec**
  - How to create container from config JSON and rootfs dir
  - Based on Docker libcontainer (now runc)

- **OCI Image Spec**
  - How to represent image layers for OCI runtimes
  - Based on Docker Image Manifest V2, Schema 2

- **OCI Distribution Spec**
  - How to distribute OCI images
  - Based on Docker Registry HTTP API

Image layout

- **Merkle DAG structure ensures reproducibility of** **`docker pull foo@sha256:e692418e…`**

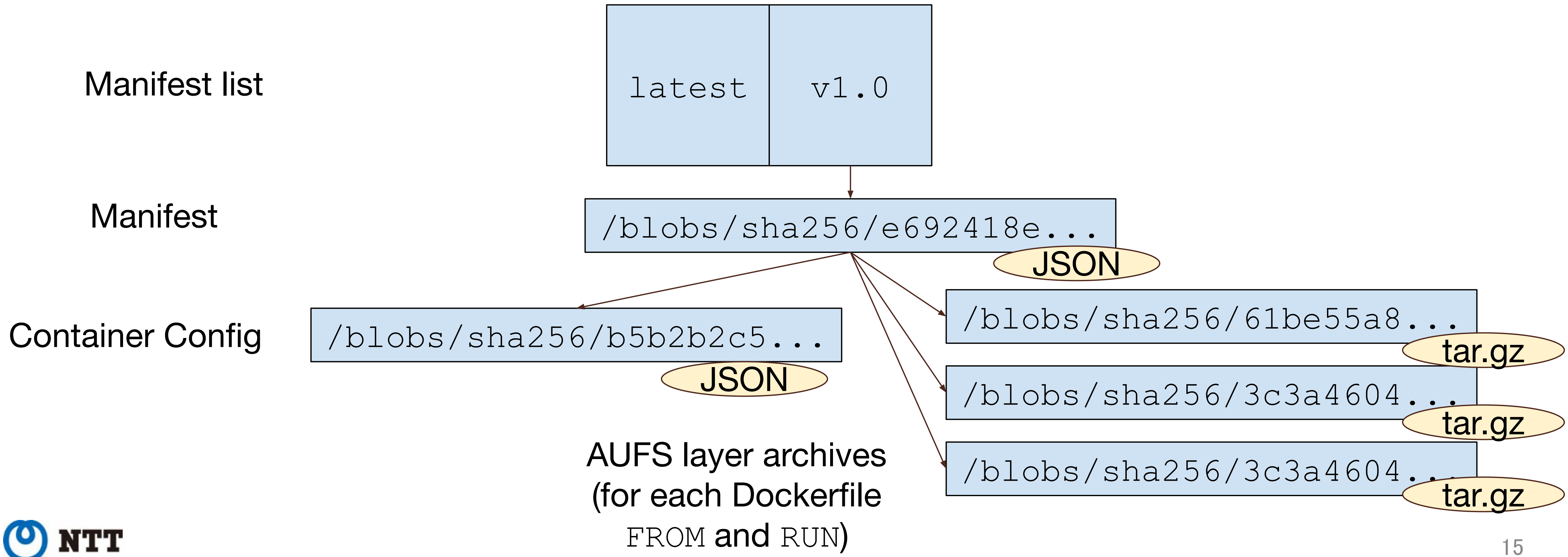


Image layout

- **Supports multi-arch (use BuildKit to build)**

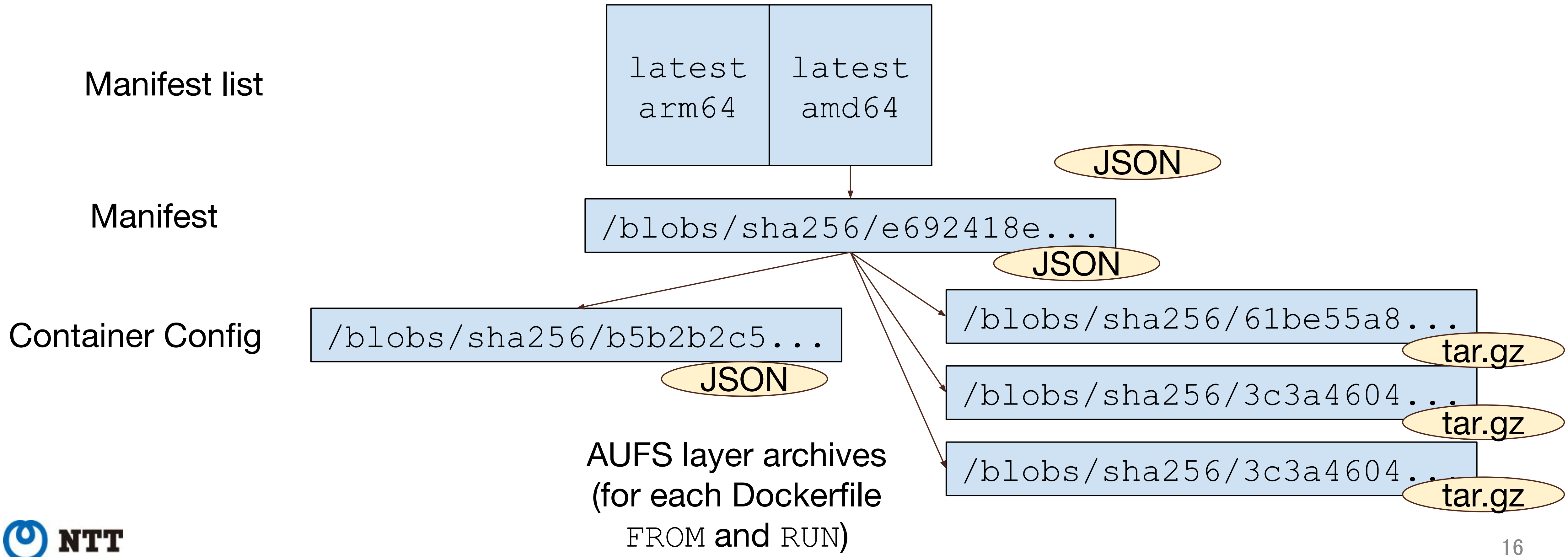
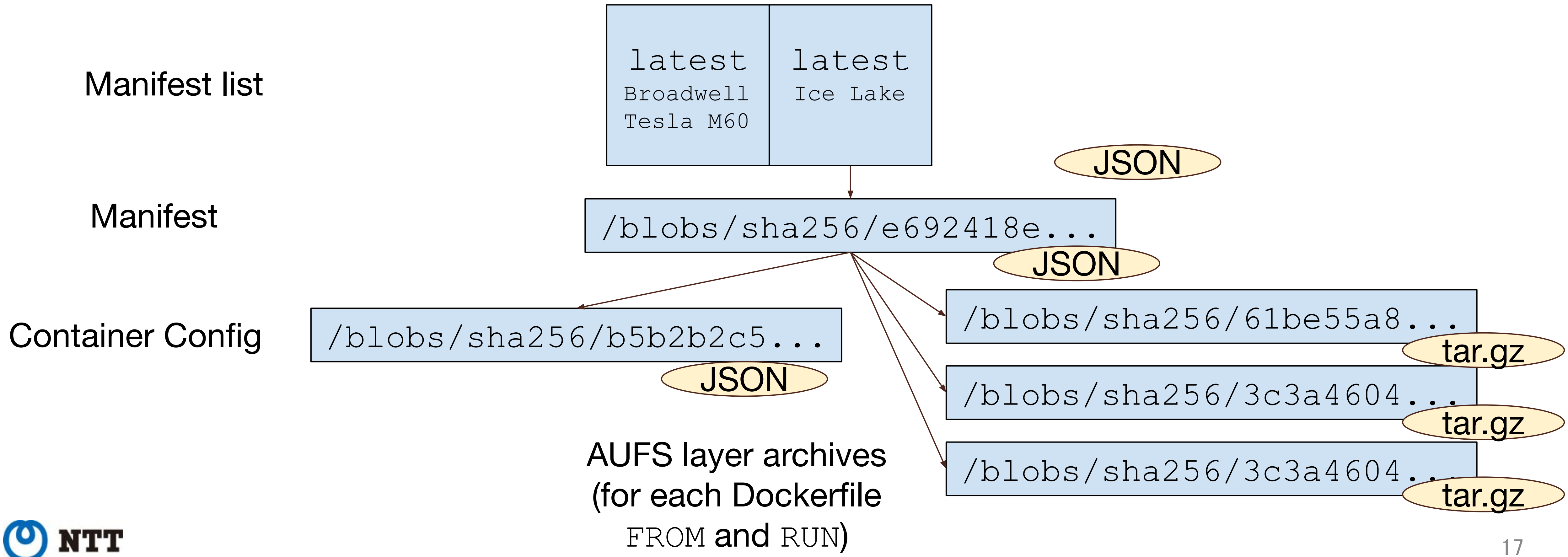


Image layout

- **And even multi-microarchitectures via qnib/metahub**
- **https://metahub.qnib.org**



Post-OCI image format?

- **Issues of current OCI v1**
  - Too coarse deduplication granularity
  - Containers cannot be started until the entire image is pulled

- **An alternative: CernVM-FS**
  - Supports file-level deduplication rather than layer-level
  - Files are lazy-pulled on demand using FUSE

- **Integrating CernVM-FS to containerd is under discussion**
  https://github.com/containerd/containerd/issues/2943

Post-OCI image format?

- **"OCI v2"** https://github.com/openSUSE/umoci/issues/256
  - Much finer deduplication granularity
  - No implementation yet

- **Container Registry Filesystem** https://github.com/google/crfs
  - Focus on lazy-pulling CI images

- **IPCS** https://github.com/hinshun/ipcs
  - IPFS integration for containerd

Singularity Image Format - 5 min

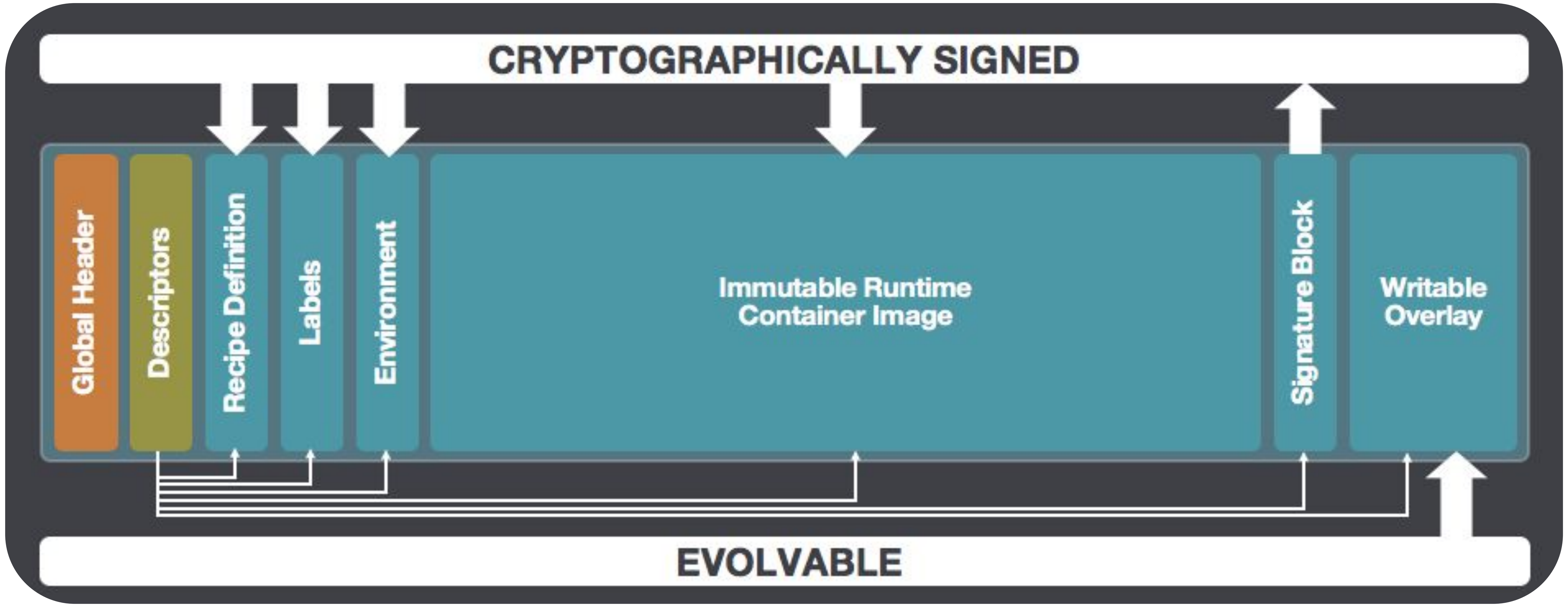
Michael Bauer - HPC Container Workshop ISC19

20 June 2019

The Singularity Image Format (SIF):

- Single file runtime container
- Fully reproducible containers
- Optimized with shared file systems
- Immutable and Regulatory controls compliant
- Cryptographically signed and validated

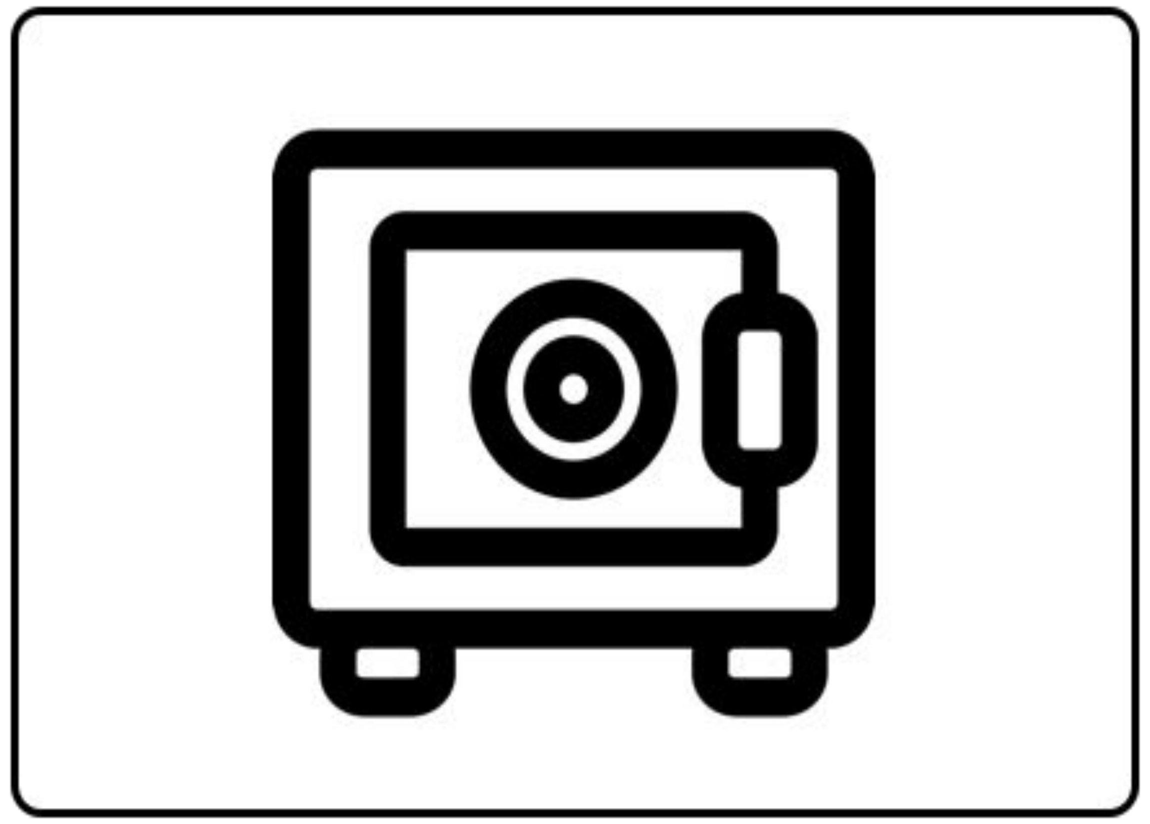
Singularity Container Format Features



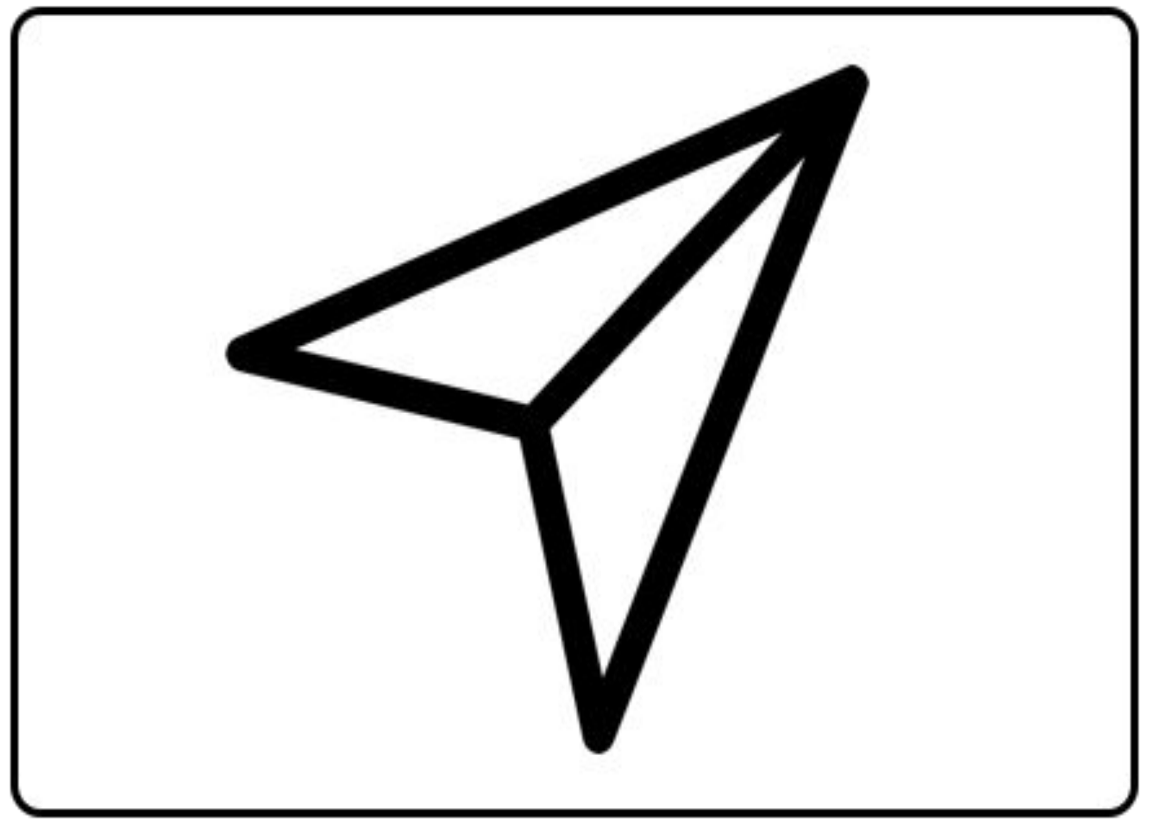
Guaranteed Reproducible

X = X

Encrypted



Mobile

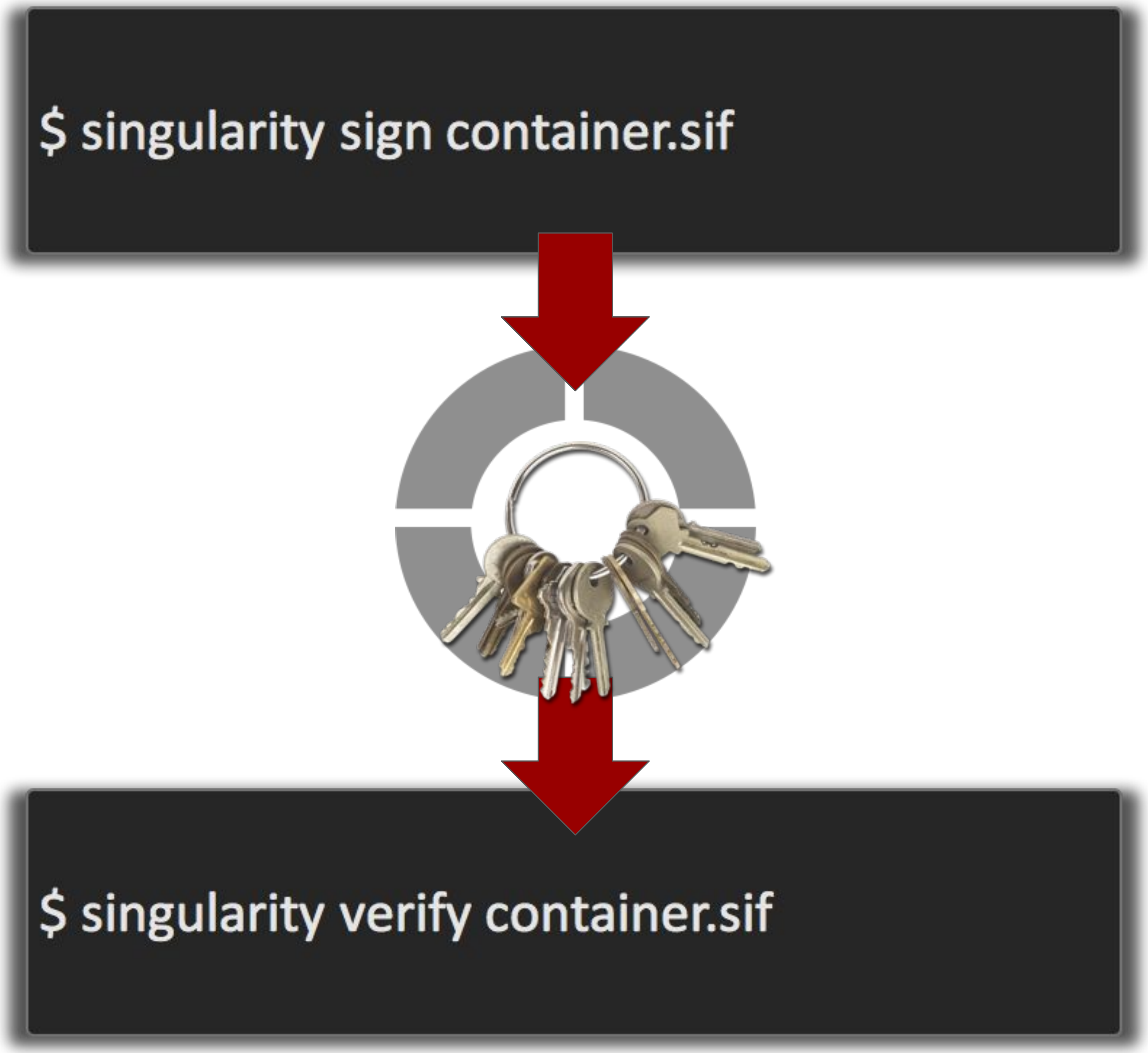


Controls Compliant

Validity

Singularity container images are **immutable**, **cryptographically signed,** and **verifiable**, ensuring absolute trust and bit for bit reproducibility of the container environment.

Note: Singularity uses no tarballs and thus no unsigned intermediate data



Valentin Rothberg <rothberg@redhat.com>
@vlntnrthbrg

What is Skopeo?

- Skopeo is a tool for managing and distributing container images
- The first tool of the *github.com/containers* family
- Used in many non-Docker pipelines to push images (e.g. Open Build Service)
- Developed at *github.com/containers/skopeo*
    - *github.com/containers/image* for image management
    - *github.com/containers/storage* for local storage (overlay, btrfs, vfs, etc.)

Skopeo - born by the desire to inspect remote images

```
$ skopeo inspect docker://fedora:latest
{
    "Name": "docker.io/library/fedora",
    "Digest": "sha256:9c78c69f748953ba8fdb6eb9982e1abefe281d9b931a13f251eb8aec988353de",
    "RepoTags": [...],
    "Created": "2019-06-10T23:20:17.083110434Z",
    ...
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
        "sha256:8f6ac7ed4a91c9630083524efcef2f59f27404320bfee44397f544c252ad4bd4"
    ]
}
```

Skopeo Copy - supported transports

- Containers-storage
  - Local container storage (e.g., overlay or btrfs)
- Directory
  - Non-standardized format to “explode” an image to a specified path
- Docker
  - Image on a registry (e.g., docker.io/library/fedora:latest)
  - Archive in the docker-save(1) format
  - From a local docker-daemon
- OCI
  - As specified by the OCI image spec
  - Can also be compressed as a tar(1) archive
- OSTree

Skopeo ABC

- The different transports give a lot of flexibility
  - Works rootless where possible (docker-daemon transport usually requires root)
- Non-opinionated way of managing images
  - Copy, inspect and delete
- A podman pull fedora:latest is actually a ...
  - $ skopeo copy docker://docker.io/library/fedora:latest containers-storage:fedora:latest
  - Note: all tools share the same libraries (i.e., containers/image and containers/storage)
- Easy to integrate into toolchains
  - $ skopeo inspect docker://docker.io/fedora:rawhide | jq '.Digest'

    "sha256:905b4846938c8aef94f52f3e41a11398ae5b40f5855fb0e40ed9c157e721d7f8"

Skopeo Resources

- Upstream development and community
  - github.com/containers/skopeo
  - No dedicated IRC channel
    - #podman on Freenode

- Available on *most* Linux distributions
  - Red Hat Enterprise Linux, Fedora
  - openSUSE, Manjaro, Gentoo
  - Archlinux, Ubuntu, Debian (soon)

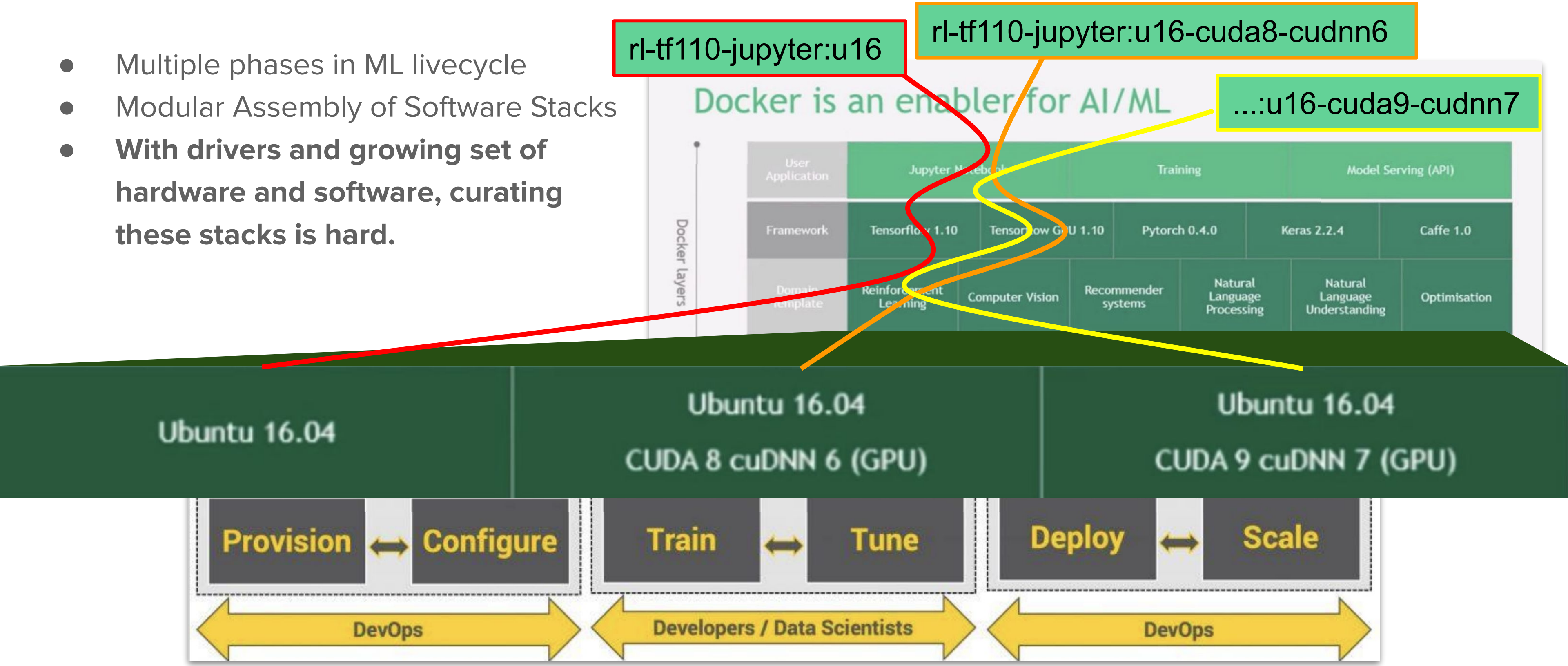
Hardware Optimized OCI Images via MetaHub Registry Proxy

High Performance Container Workshop - ISC19

Christian Kniep, QNIB Solutions
christian@qnib.org

Image Lifecycle needs Composability and Workflow

- Multiple phases in ML livecycle
- Modular Assembly of Software Stacks
- **With drivers and growing set of hardware and software, curating these stacks is hard.**



source: https://thenewstack.io/an-introduction-to-the-machine-learning-platform-as-a-service/

Current State of Affairs [cont]

Images optimized for different CPU µ-arch are picked by name.

As described in the previous slides.


```
$ docker run -ti --rm qnib/bench:cpu-broadwell
>> This container is optimized for: cpu:broadwell
```

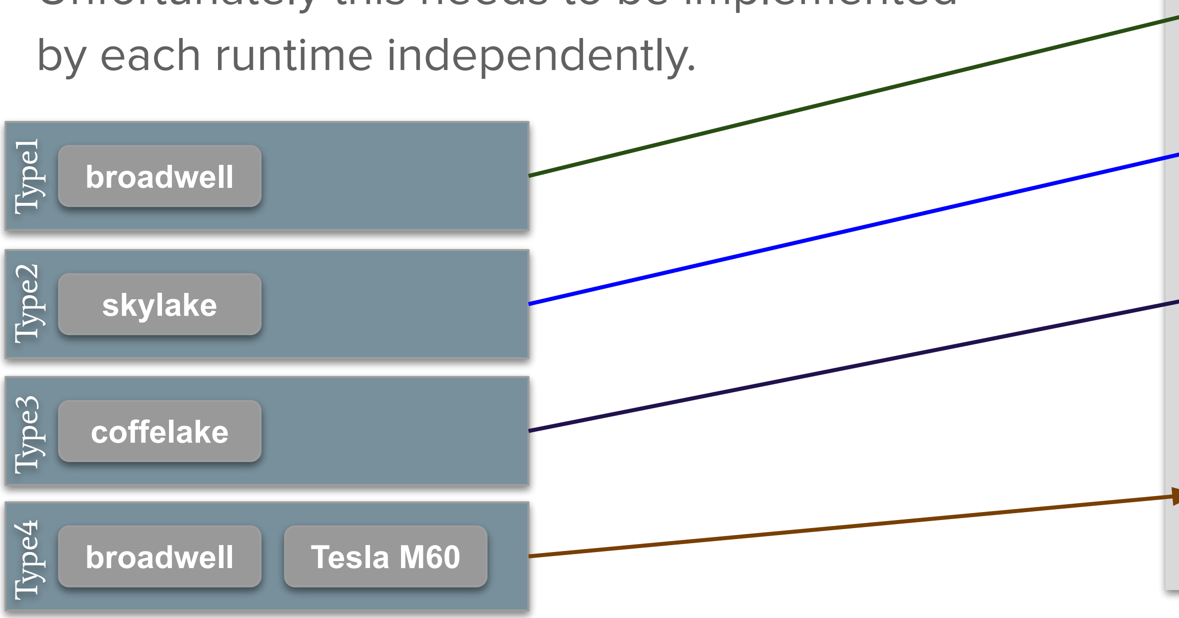
```
$ docker run -ti --rm qnib/bench:cpu-skylake
>> This container is optimized for: cpu:skylake
```

```
$ docker run -ti --rm qnib/bench:generic
>> This container is not optimized for a specific microarchitecture
```

```
$ docker run -ti --rm qnib/bench:cpu_broadwell-nvcap_52
>> This container is optimized for: cpu:broadwell,nvcap:5.2
```

Dynamic Manifest List FTW!

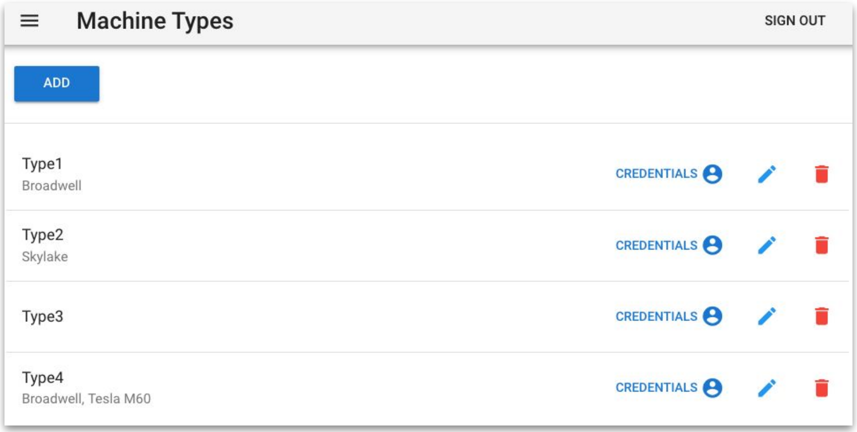
Manifest List provides choice via OCI "platform.features".
Unfortunately this needs to be implemented by each runtime independently.

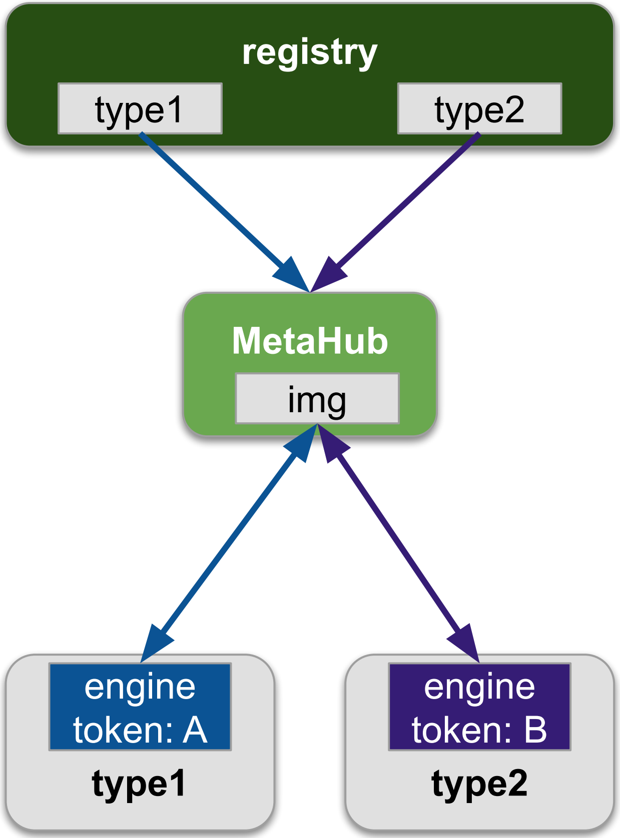



```
image: qnib/bench
manifests:
  - image: qnib/bench:cpu-broadwell
    platform:
      architecture: amd64
      features: [“cpu:broadwell”]
  - image: qnib/bench:cpu-skylake
    platform:
      architecture: amd64
      features: [“cpu:skylake”]
  - image: qnib/bench:cpu-coffelake
    platform:
      architecture: amd64
      features: [“cpu-coffelake”]
  - image:
qnib/bench:cpu_broadwell-nvcap_52
    platform:
      architecture: amd64
      features:
        - cpu-broadwell
        - nvcap-5.2
```

MetaHub to proxy normal Registry

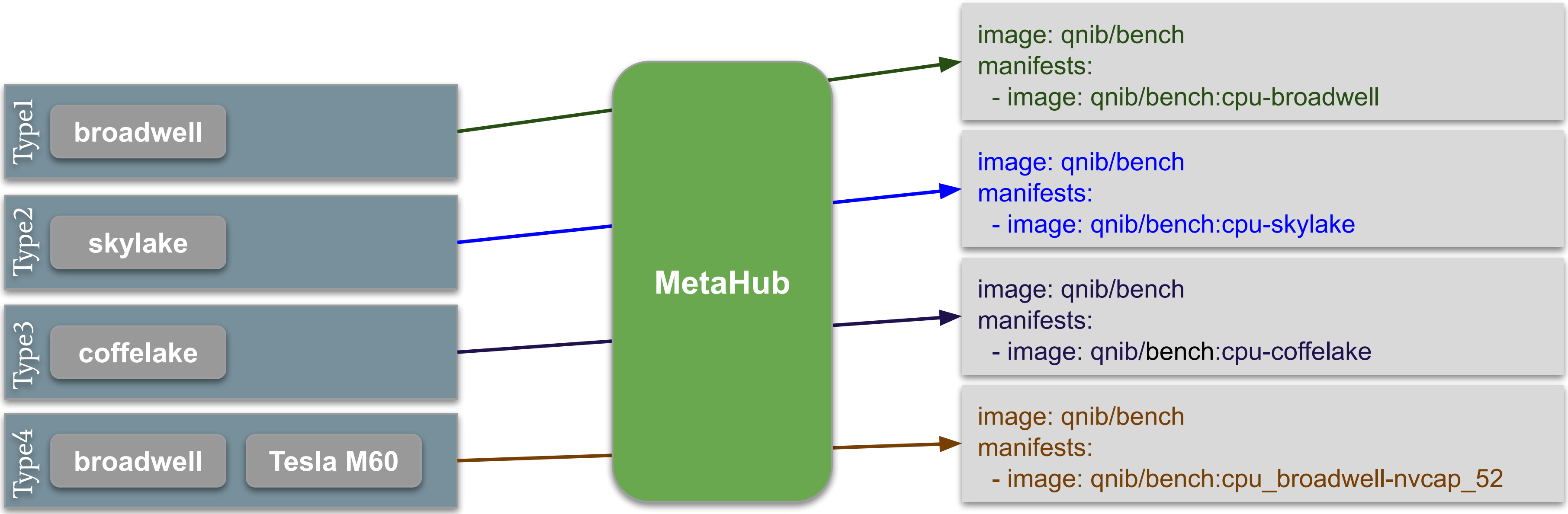
To be agnostic in terms of the runtime and supervision engine, MetaHub will dynamically create a Manifest List depending on the token used to authenticate.





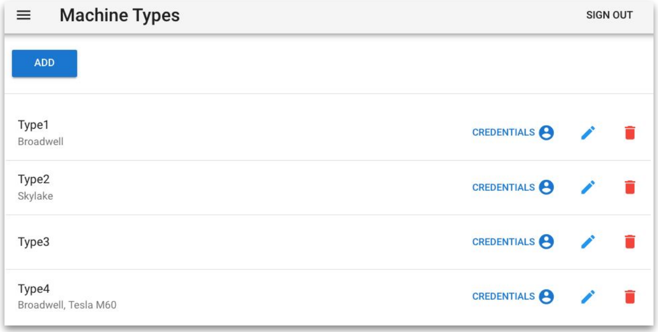
MetaHub to proxy normal Registry [cont #1]

Metahub reduces the Manifest List down to what the runtime ‘needs’.



MetaHub to proxy normal Registry [cont #2]

Each machine type has a special credentials to login to MetaHub.



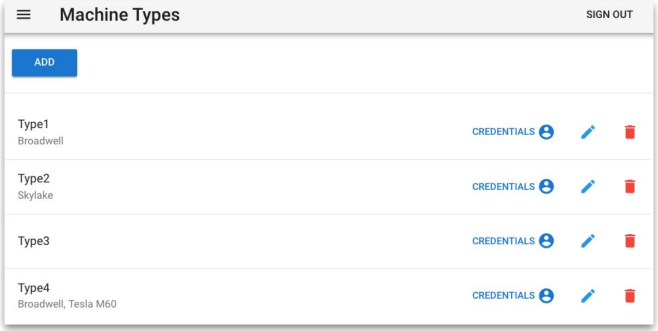

```
$ docker login -u qnib-type1 -p $PASSWD1 metahub.qnib.org
Login Succeeded
```

```
$ docker login -u qnib-type2 -p $PASSWD2 metahub.qnib.org
Login Succeeded
```

```
$ docker login -u qnib-type3 -p $PASSWD3 metahub.qnib.org
Login Succeeded
```

```
$ docker login -u qnib-type4 -p $PASSWD4 metahub.qnib.org
Login Succeeded
```

MetaHub to proxy normal Registry [cont #3]



Thus, the runtime will download the correct choice.

$ docker run -ti --rm metahub.qnib.org/qnib/bench
\>> This container is optimized for: **cpu:broadwell**

$ docker run -ti --rm metahub.qnib.org/qnib/bench
\>> This container is optimized for: **cpu:skylake**

$ docker run -ti --rm metahub.qnib.org/qnib/bench
\>> This container is **not optimized** for a specific microarchitecture

$ docker run -ti --rm metahub.qnib.org/qnib/bench
\>> This container is optimized for: **cpu:broadwell**,**nvcap:5.2**

FOOS FTW! MetaHub released.

https://qnib.org/2019/06/12/metahub/

Metahub: Dynamic Registry Proxy

Jun 12, 2019 • Christian Kniep

I won't say "Long time, no post" - but...

As I had some time at my hands the last couple of month, I was iterating on my idea on hardware optimization using Manifest List from the last post Match Node-Specific Needs Using Manifest Lists.

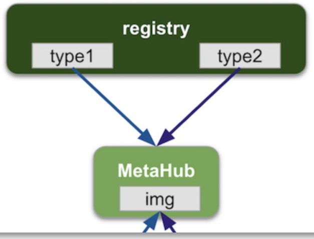
https://github.com/qnib/metahub

MetaHub - Dynamic OCI Registry Proxy

Announcement: qnib.org

The MetaHub project is meta-data registry, which serves images filtered via login so that a machine gets the image that fits the specifics of the host the image is going to run on.

That could be picking an image that not only fits the CPU Architecture (x86-64, ppcle, arm) but is optimized for the microarchitecture of the host (Broadwell, Skylake, ...). And it does not stop there - any host specific attribute can be use: Accelerators, network, configuration or the full depth of gcc options.



Thank you!