KubeCon | CloudNativeCon

North America 2021

RESILIENCE
REALIZED

Faster Container Image Distribution on a Variety of Tools with Lazy Pulling

Kohei Tokunaga, NTT Corporation
Tao Peng, Ant Group

Pull is time-consuming

- Large images speeds down cold start of containers
- Build takes long if base images are large
- Not all images are minimizable e.g. language runtimes, frameworks, ..

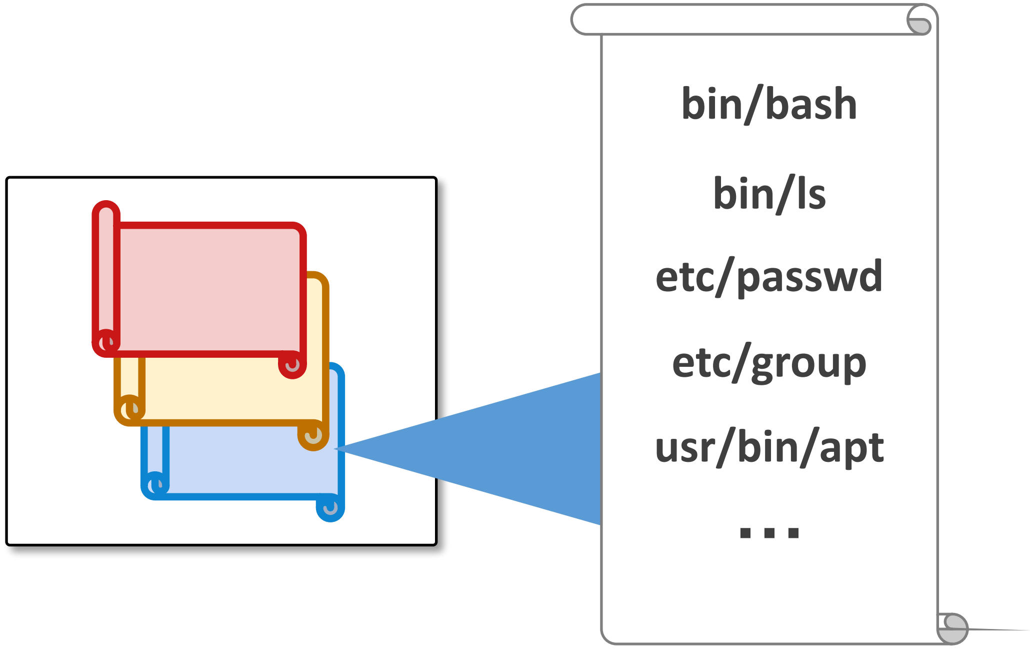
pulling packages accounts for 76% of container start time,
but only 6.4% of that data is read [Harter et al. 2016]

[Harter et al. 2016] Tyler Harter, Brandon Salmon, Rose Liu, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau. "Slacker: Fast Distribution with Lazy Docker Containers". 14th USENIX Conference on File and Storage Technologies (FAST ’16). February 22–25, 2016, Santa Clara, CA, USA

Problem on the current OCI image

container can’t start before the all layers become locally available

Image is a set of tar (+compression) layers



bin/bash

bin/ls

etc/passwd

etc/group

usr/bin/apt

…

- Non seekable
  - Need to scan the entire blob even for extracting single file entry
- No parallel extraction
  - Need to scan sequentially

OCI-alternative *accelerated* images and lazy pulling

Lazy pulling:
Starting up containers without waiting for the pull completion

- **eStargz**
  - Lazy pullable format with prefetch optimization + content verification
  - Proposed as a backward-compatible extension to OCI Image Spec

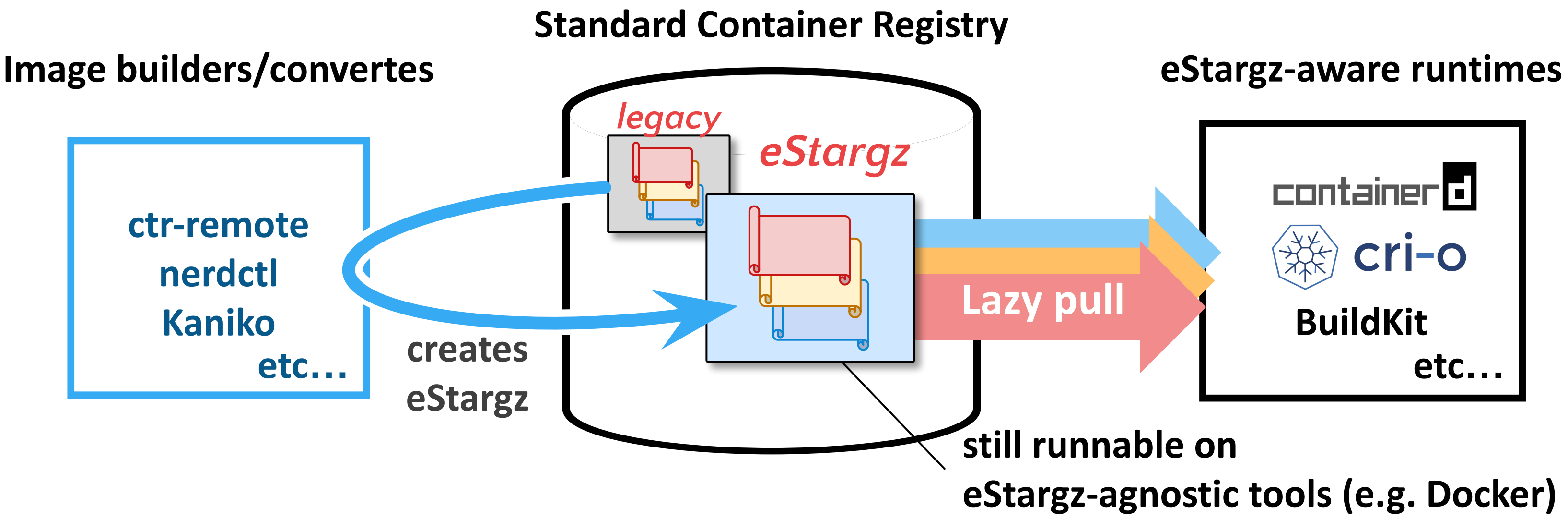
- **Nydus**
  - Lazy pullable format with prefetch, chunk dedup and e2e data integrity
  - Compatible with OCI Distribution Spec and Artifacts Spec
  - Proposed as OCI "v2" format (incompatible to current OCI Image Spec)

eStargz

Kohei Tokunaga, NTT Corporation

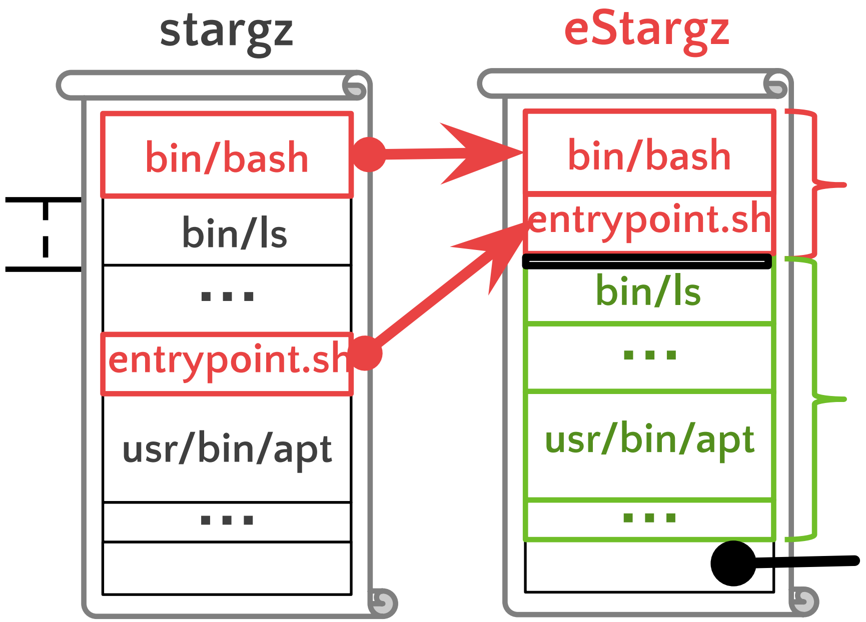
eStargz: Standard-compatible lazy pulling

- **100% OCI-compatible**
  - Lazy pullable from standard registries (ghcr.io, docker.io, …etc)
  - Even legacy (lazy-pulling-agnostic) runtime can run eStargz
- **Usable with a variety of tools** (tracker https://github.com/containerd/stargz-snapshotter/issues/258)
  - Kubernetes, k3s, containerd, CRI-O, Podman, BuildKit, Kaniko, ko, buildpacks.io, go-containerregistry…
- **Performance optimization and content verification**
  - Important files can be prefetched to avoid NW overhead
  - Each chunk comes with checksum



eStargz image layer format

- Discussed in Stargz Snapshotter of containerd: https://github.com/containerd/stargz-snapshotter
- Compatible to gzip = usable as a valid OCI/Docker image layer
- Based on stargz by CRFS (https://github.com/google/crfs)
  - eStargz comes with performance optimization and content verification



gzip member per regular file/chunk
can be extracted per-file using **HTTP Range Request**

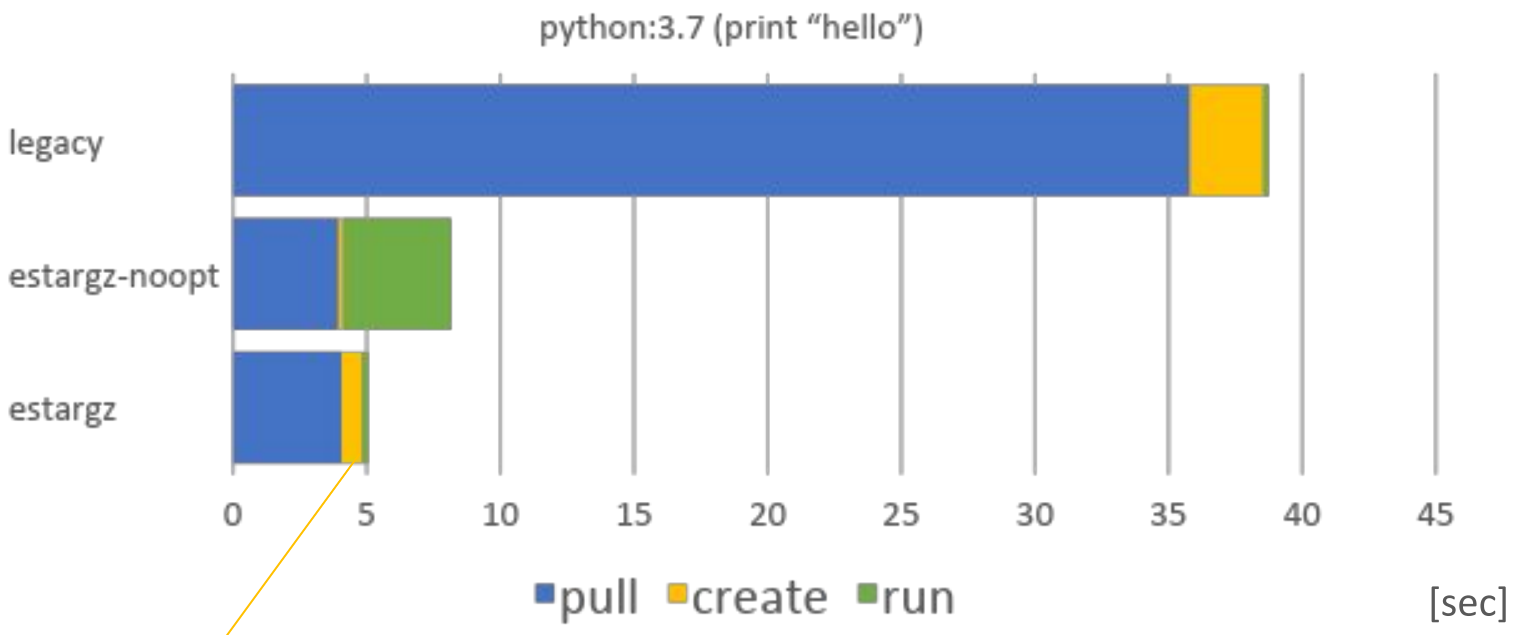
Prioritized files
Prefetched by a single **Range Request**

Files fetched on demand
Downloaded in background as well

TOC(Table Of Contents) and footer
Files metadata, payload offset, etc...

For more details: https://github.com/containerd/stargz-snapshotter/blob/master/docs/stargz-estargz.md

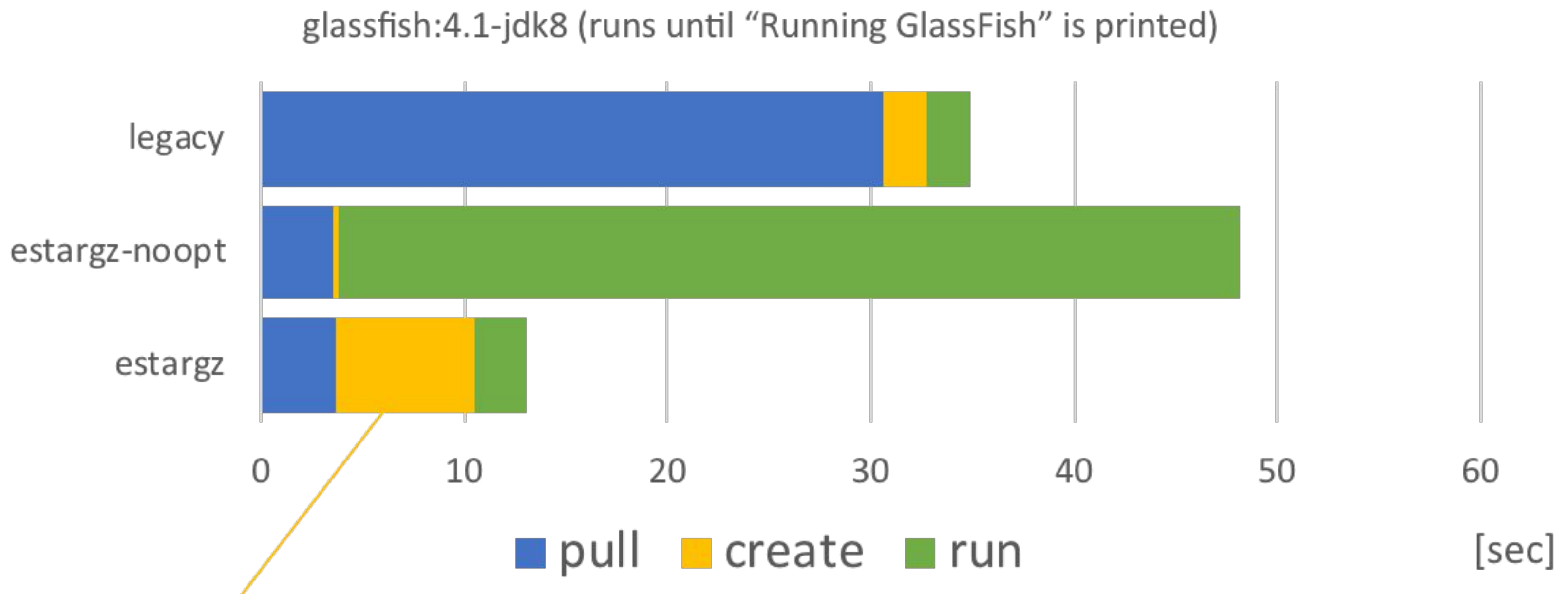
Benchmarking result



Waits for prefetch completion during create

- Method based on Hello Bench [Harter, et al. 2016]
- Takes 95 percentile of 100 operations
- Host: EC2 Oregon (m5.2xlarge, Ubuntu 20.04)
- Registry: GitHub Container Registry (ghcr.io)
- Runtime: containerd
- Stargz Snapshotter commit: 7f45f7438617728dd06bc9853afb5e42c1d3d5a3

Benchmarking result



Waits for prefetch completion

- Benchmarking spec is the same as the previous slide

Lazy pulling on Kubernetes

- CRI runtimes + plugins (discussed later) enable lazy pulling on Kubernetes
  - Containerd + Stargz Snapshotter
  - CRI-O + Stargz Store
- Real-world use-case at CERN for speeding up analysis pipeline [1] (13x faster pull for 5GB image)

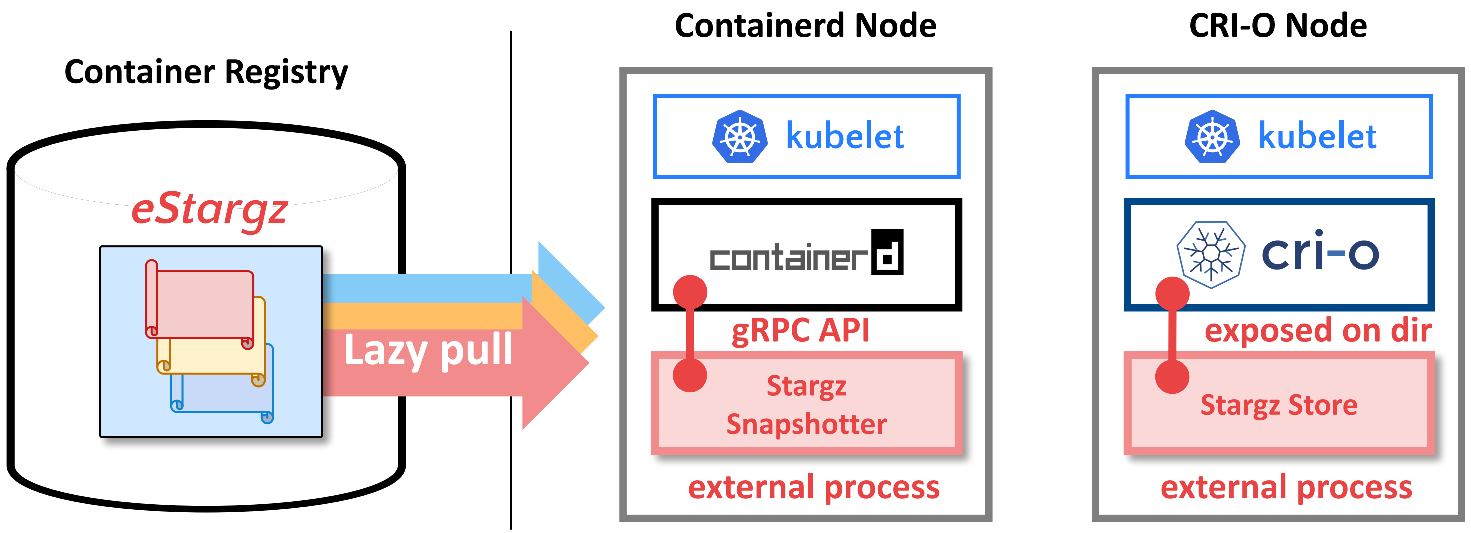
[1] Ricardo Rocha & Spyridon Trigazis, CERN. "Speeding Up Analysis Pipelines with Remote Container Images". KubeCon+CloudNativeCon 2020 NA. https://sched.co/ekDj

- k3s supports lazy pulling of eStargz (merged to the main, will be included in k3s v1.22)

```
$ k3s server --snapshotter=stargz
```

- Lazy-pull-enabled KinD image is available on `ghcr.io/stargz-containers` repo

```
$ kind create cluster --name estargz-demo --image ghcr.io/stargz-containers/estargz-kind-node:0.8.0
```

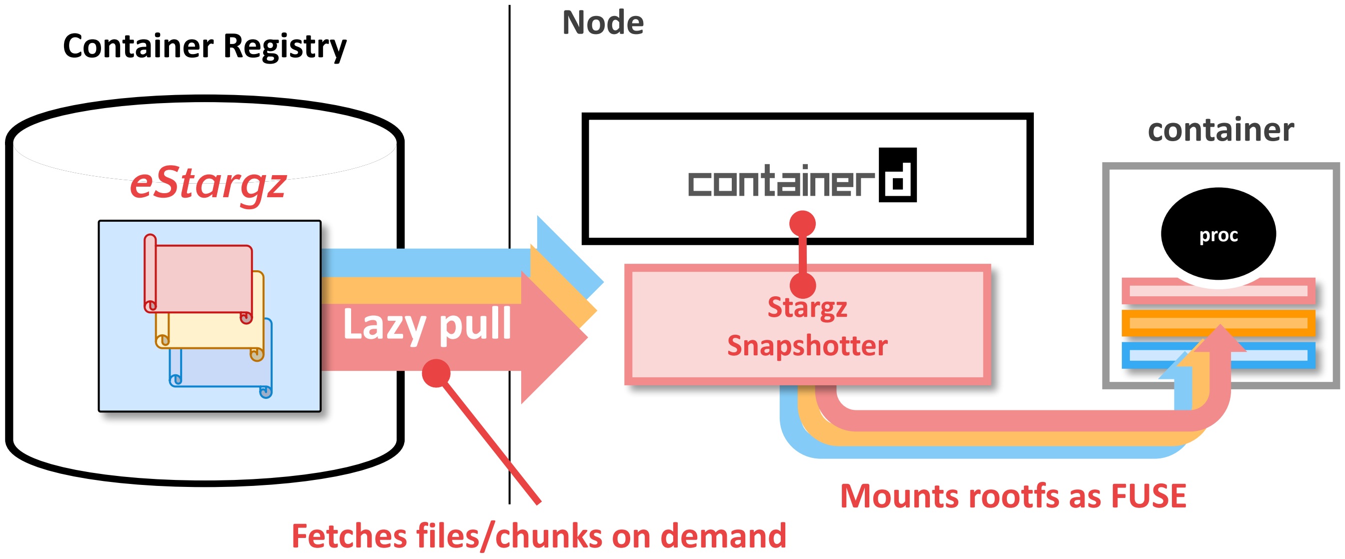


Lazy pulling on containerd

- Stargz Snapshotter plugin : https://github.com/containerd/stargz-snapshotter
  - Implements **remote snapshotter** plugin interface

- **nerdctl** (Docker-compatible CLI for containerd) supports lazy pulling
  - https://github.com/containerd/nerdctl

```
$ nerdctl --snapshotter=stargz run ghcr.io/stargz-containers/python:3.9-esgz
$ nerdctl --snapshotter=stargz compose -f docker-compose.stargz.yaml up
```



Container Registry

Node

eStargz

Lazy pull

containerd

Stargz Snapshotter

container

proc

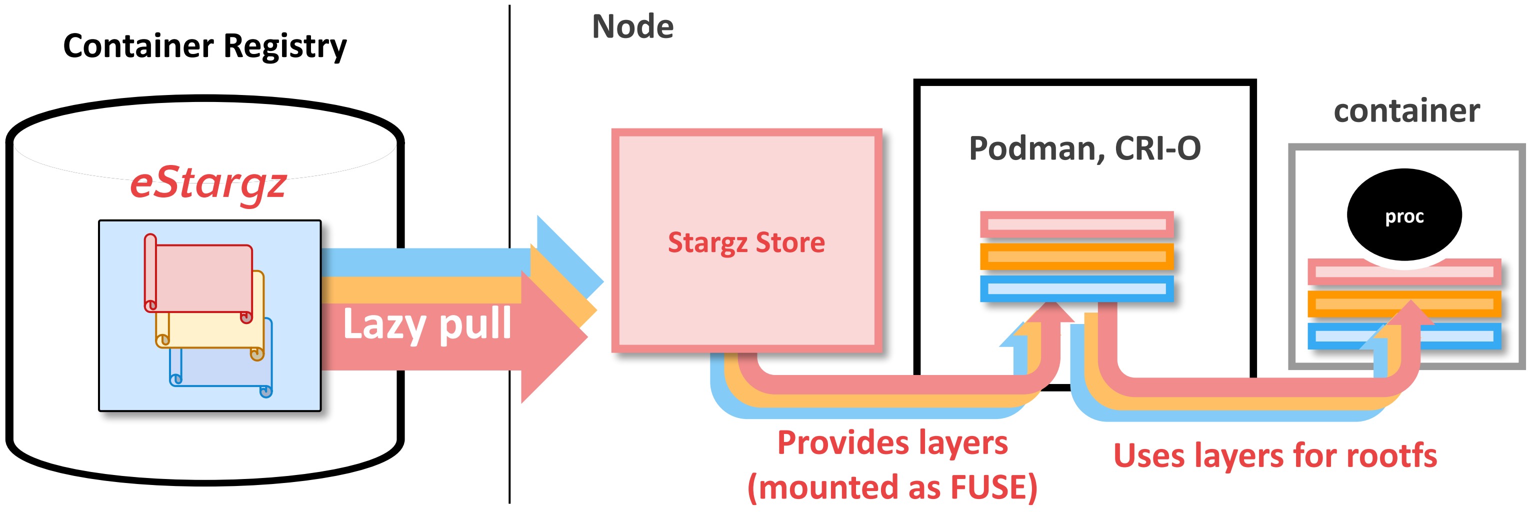
Mounts rootfs as FUSE

Fetches files/chunks on demand

Lazy pulling on Podman/CRI-O

- Stargz Store plugin : https://github.com/containerd/stargz-snapshotter
  - Developped in Stargz Snapshotter project
  - Implements **Additional Layer Store** plugin
  - Podman >= v3.3.0, CRI-O >= v1.22.0

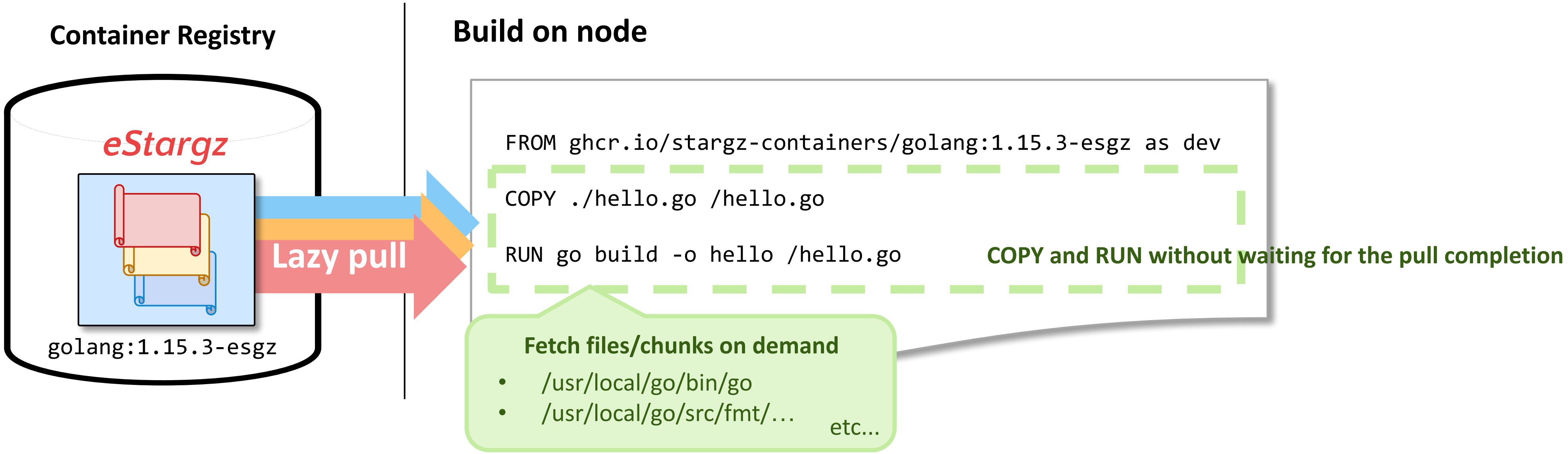
```
$ podman run ghcr.io/stargz-containers/python:3.9-esgz
```



Lazy pulling on BuildKit

- Experimentally supports lazy pulling of base images since v0.8.0
- Can shorten the time of build that require pull
  - e.g. on temporary (and fresh) CI instances

```
$ docker buildx create --use --name lazy-builder \
         --driver docker-container --driver-opt image=moby/buildkit:master \
         --buildkitd-flags '--oci-worker-snapshotter=stargz'
$ docker buildx inspect --bootstrap lazy-builder
$ docker buildx build .
```



```
FROM ghcr.io/stargz-containers/golang:1.15.3-esgz as dev
COPY ./hello.go /hello.go
RUN go build -o hello /hello.go
```

COPY and RUN without waiting for the pull completion

Fetch files/chunks on demand

- /usr/local/go/bin/go
- /usr/local/go/src/fmt/…

etc...

More details at blog: https://medium.com/nttlabs/buildkit-lazypull-66c37690963f

Building eStargz

BuildKit: https://github.com/moby/buildkit

- Building eStargz supported on the main branch, hope to come in v0.10.x
- eStargz is supported by Buildx or standalone BuildKit (buildctl + buildkitd)
- Docs: https://github.com/moby/buildkit/blob/master/docs/stargz-estargz.md#creating-stargzestargz-images

```
$ docker buildx build \
    -o type=registry,name=ktokunaga/hello:esgz,oci-mediatypes=true,compression=estargz .
```

Kaniko: https://github.com/GoogleContainerTools/kaniko

- Image builder runnable in containers and Kubernetes
- Requires GGCR_EXPERIMENT_ESTARGZ=1
- Base images need to be eStargz

```
$ docker run --rm -e GGCR_EXPERIMENT_ESTARGZ=1 \
    -v /tmp/context:/workspace -v ~/.docker/config.json:/kaniko/.docker/config.json:ro \
    gcr.io/kaniko-project/executor:v1.6.0 --destination "ghcr.io/ktock/sample:esgz"
```

Converting an image into eStargz

ctr-remote: https://github.com/containerd/stargz-snapshotter/blob/v0.7.0/docs/ctr-remote.md

- CLI for containerd provided by Stargz Snapshotter project
- Supports prefetch optimization for eStargz

```
$ ctr-remote image optimize --oci ghcr.io/ktock/foo:1 ghcr.io/ktock/foo:1-esgz
```

nerdctl: https://github.com/containerd/nerdctl/blob/v0.11.1/docs/stargz.md

- Docker-compatible CLI for containerd
- Can be combined with `nerdctl build` command

```
$ nerdctl image convert --estargz --oci ghcr.io/ktock/foo:1 ghcr.io/ktock/foo:1-esgz
```

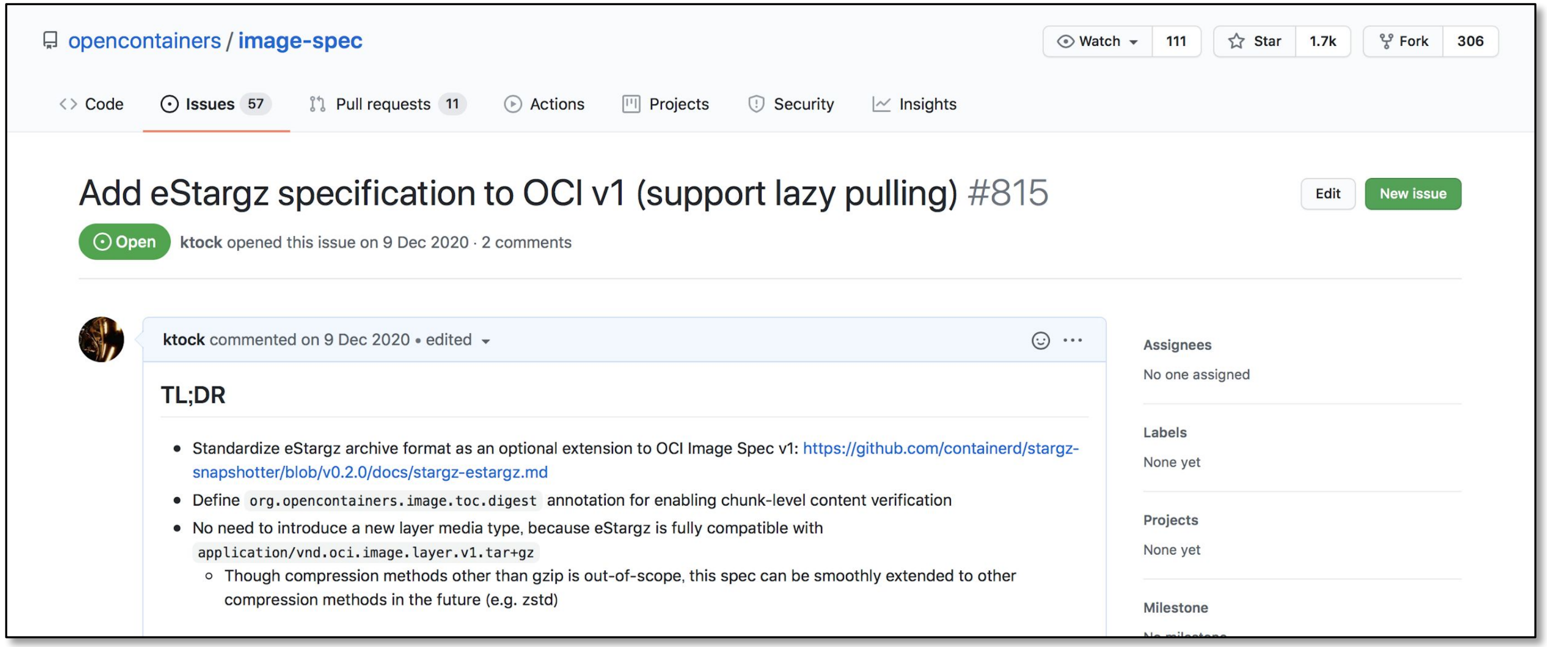
go-containerregistry and crane CLI : https://github.com/google/go-containerregistry

- Library and CLI to interact with registries
- requires `GGCR_EXPERIMENT_ESTARGZ=1`
- downstream tools (e.g.Kaniko, ko and buildpacks.io) supports eStargz creation as well

```
$ go install github.com/google/go-containerregistry/cmd/crane@latest
$ GGCR_EXPERIMENT_ESTARGZ=1 crane optimize ghcr.io/ktock/foo:1 ghcr.io/ktock/foo:1-esgz
```

Discussion toward OCIv1 extension

https://github.com/opencontainers/image-spec/issues/815



Add eStargz specification to OCI v1 (support lazy pulling) #815

ktock opened this issue on 9 Dec 2020 · 2 comments

ktock commented on 9 Dec 2020 • edited

TL;DR

- Standardize eStargz archive format as an optional extension to OCI Image Spec v1: https://github.com/containerd/stargz-snapshotter/blob/v0.2.0/docs/stargz-estargz.md
- Define `org.opencontainers.image.toc.digest` annotation for enabling chunk-level content verification
- No need to introduce a new layer media type, because eStargz is fully compatible with `application/vnd.oci.image.layer.v1.tar+gz`
  - Though compression methods other than gzip is out-of-scope, this spec can be smoothly extended to other compression methods in the future (e.g. zstd)

- eStargz is now widely usable on tools in the community
- Proposed to OCI Image Spec (discussion is on-going)
- Proposed as backward-compatible extensions
  - Extension for the current gzip layer (`application/vnd.oci.image.layer.v1.tar+gz`)
  - Additional annotation for content verification: `org.opencontainers.image.toc.digest`

Nydus Image Service

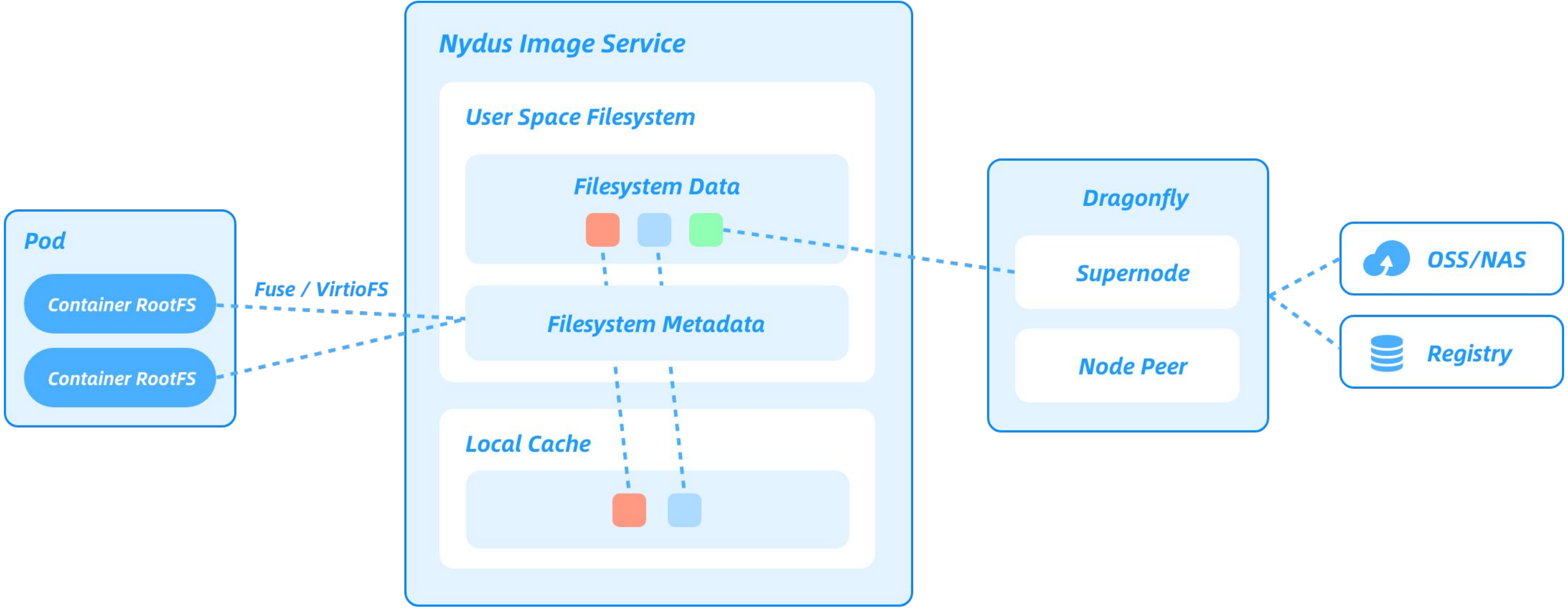
Tao Peng, Ant Group

Nydus: Improved Lazy Pulling and More

- **Standard lazy pulling with prefetch policies**
  - Lazy pullable from standard registries (cloud registries, docker registry, harbor, etc)
  - Semantical prefetch, hinted prefetch, background prefetch all data
- **Chunk level deduplication with layered build cache**
  - Speed up both image conversion and downloading
- **End-to-end data integrity**
  - Runtime data integrity check
- **Reproducible Image Building**
  - Build environment agnostic
- **Compatible with OCI Distribution Spec and Artifacts Spec**
  - Usable with most existing container registry implementations
- **Rich container ecosystem integration**
  - Kubernetes, docker, containerd, buildkit, harbor, dragonfly, runc, kata-containers etc
- **Resource efficient and production ready**
  - Large scale deployment at Ant and available via Alibaba Cloud services

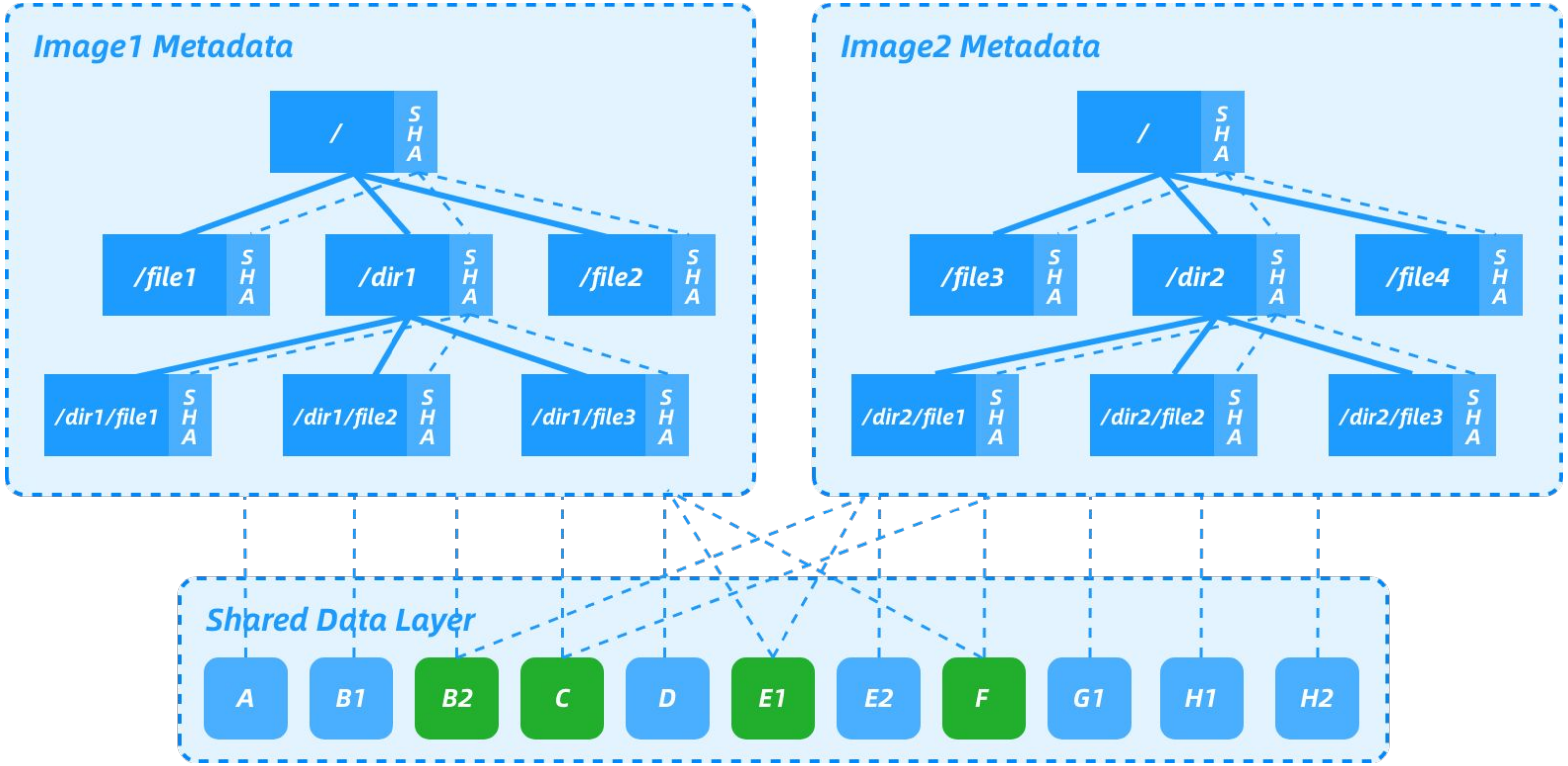
FUSE/virtiofs Unified Architecture

- **Native container runtime support**
  - One translation layer (FUSE or virtiofs) for both runc and kata containers
- **Shared uncompressed local cache**
  - Download/uncompress once and use all the time
- **Singleton mode with extremely low memory footprint**
  - ~5MB per instance



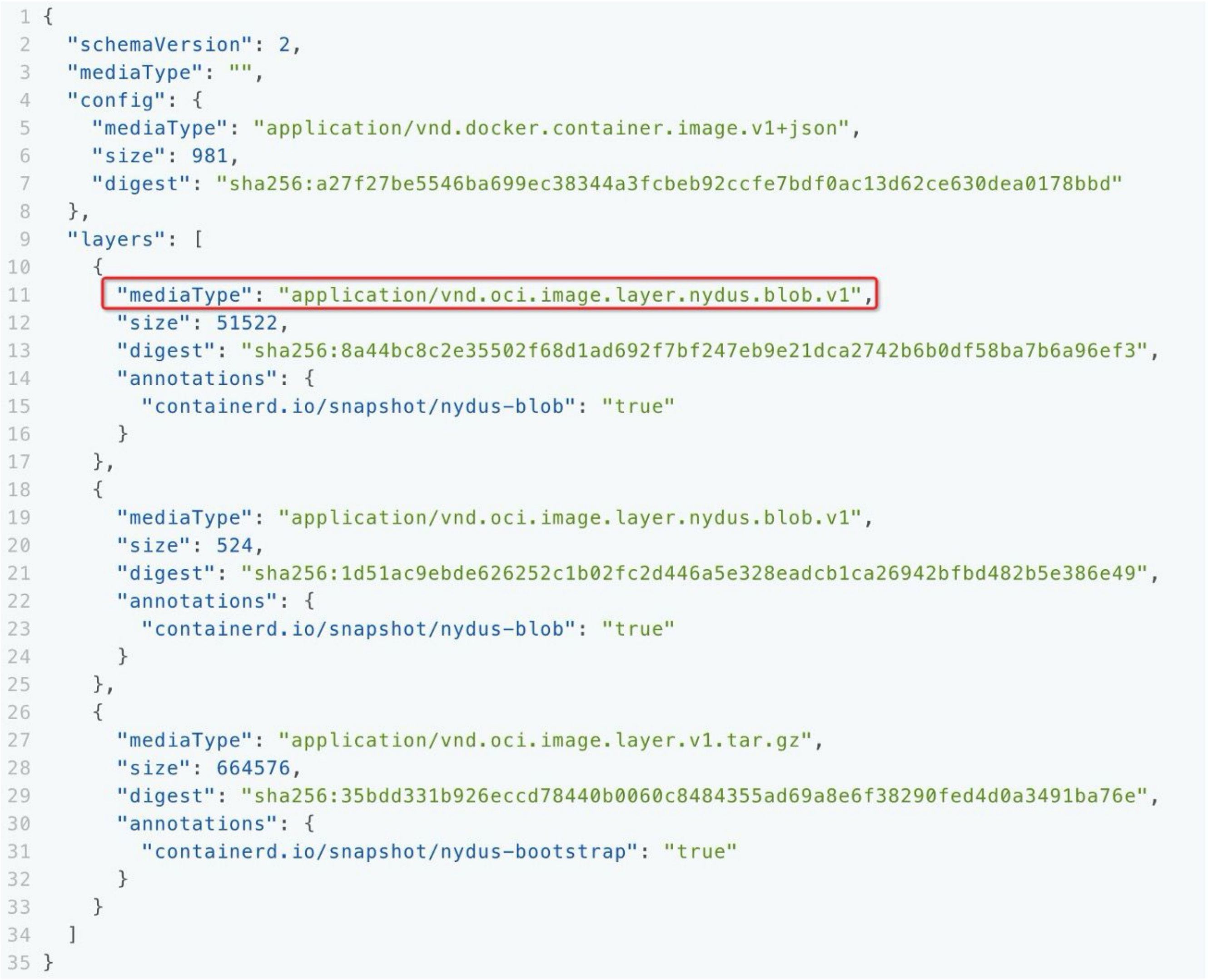
Registry Acceleration File System

- **Merkle tree metadata layer**
  - self-verifiable
- **Chunk-shared data layer**
  - enabling chunk level deduplication



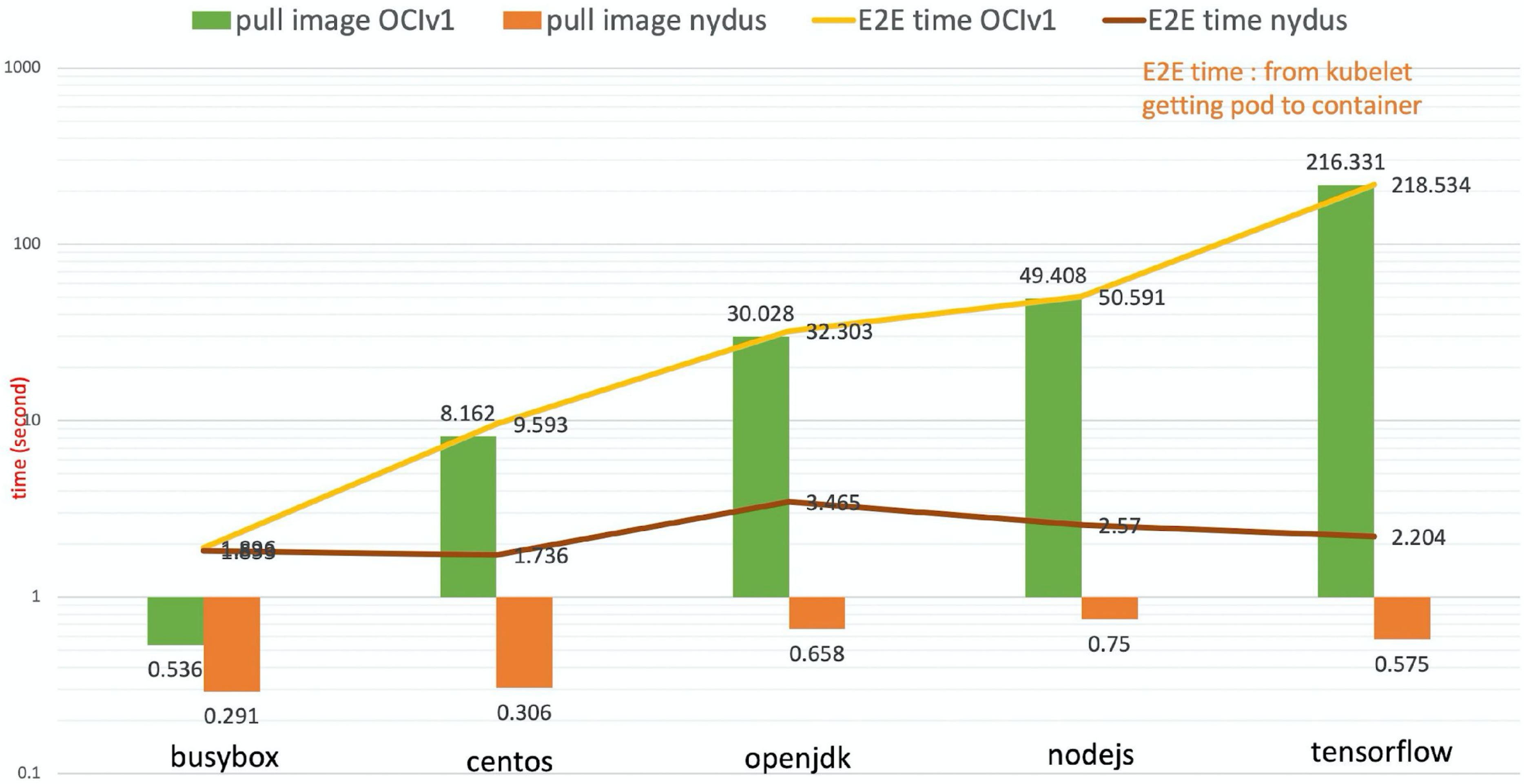
OCI Spec Compatibility

- Metadata/data layers are OCI manifest layers
- New media type in the image manifest spec
- Compatible with OCI artifacts/distribution spec
- Compatible registry GC functionality
- Widely working with container registries
  - cloud registries
  - docker registry
  - harbor
  - etc.



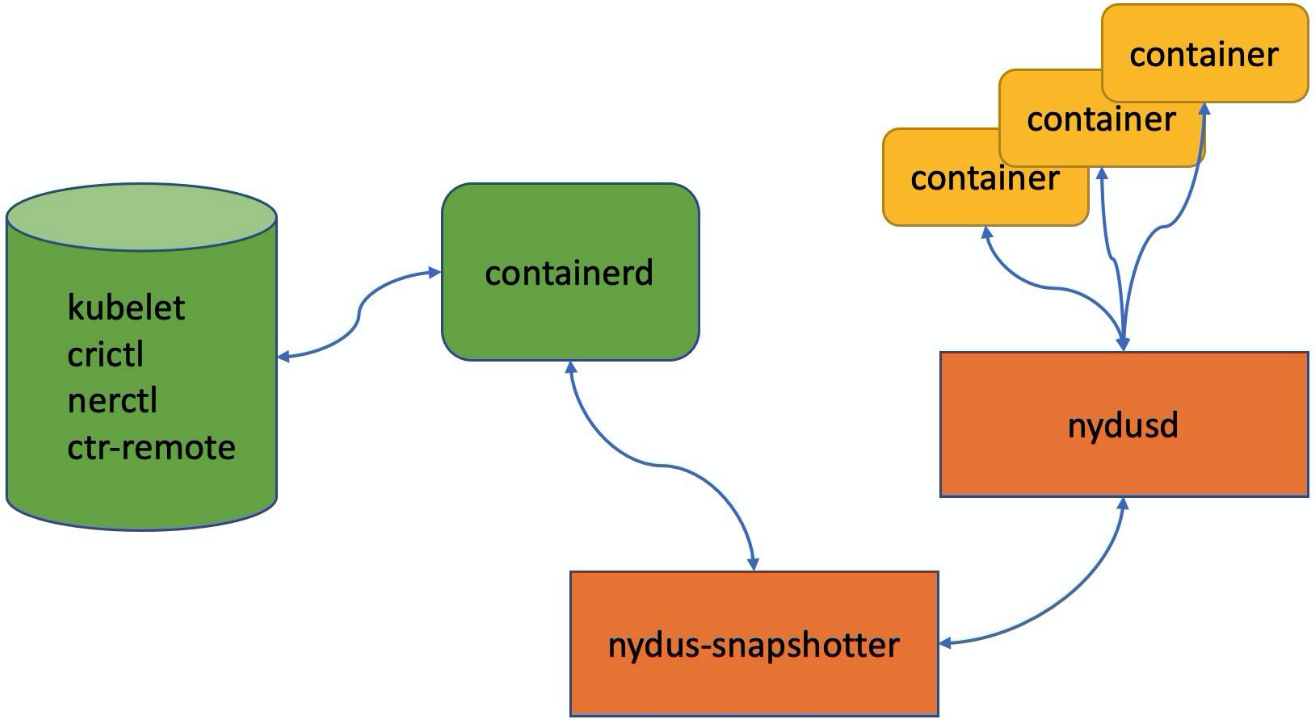
```json
{
  "schemaVersion": 2,
  "mediaType": "",
  "config": {
    "mediaType": "application/vnd.docker.container.image.v1+json",
    "size": 981,
    "digest": "sha256:a27f27be5546ba699ec38344a3fcbeb92ccfe7bdf0ac13d62ce630dea0178bbd"
  },
  "layers": [
    {
      "mediaType": "application/vnd.oci.image.layer.nydus.blob.v1",
      "size": 51522,
      "digest": "sha256:8a44bc8c2e35502f68d1ad692f7bf247eb9e21dca2742b6b0df58ba7b6a96ef3",
      "annotations": {
        "containerd.io/snapshot/nydus-blob": "true"
      }
    },
    {
      "mediaType": "application/vnd.oci.image.layer.nydus.blob.v1",
      "size": 524,
      "digest": "sha256:1d51ac9ebde626252c1b02fc2d446a5e328eadcb1ca26942bfbd482b5e386e49",
      "annotations": {
        "containerd.io/snapshot/nydus-blob": "true"
      }
    },
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar.gz",
      "size": 664576,
      "digest": "sha256:35bdd331b926eccd78440b0060c8484355ad69a8e6f38290fed4d0a3491ba76e",
      "annotations": {
        "containerd.io/snapshot/nydus-bootstrap": "true"
      }
    }
  ]
}
```

Lazy Pulling Benchmark



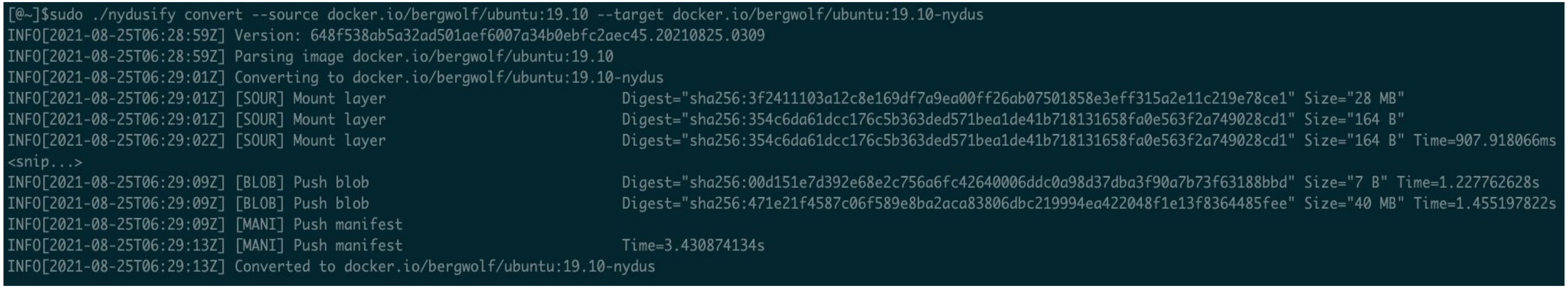
Lazy Pulling with Variety of Tools

- Nydus Snapshotter plugin
  - https://github.com/dragonflyoss/image-service/tree/master/contrib/nydus-snapshotter
- Usable for variety of container management tools
  - kubernetes (kubelet)
  - crictl
  - nerdctl
  - ctr-remote



Manual Image Conversion -- nydusify

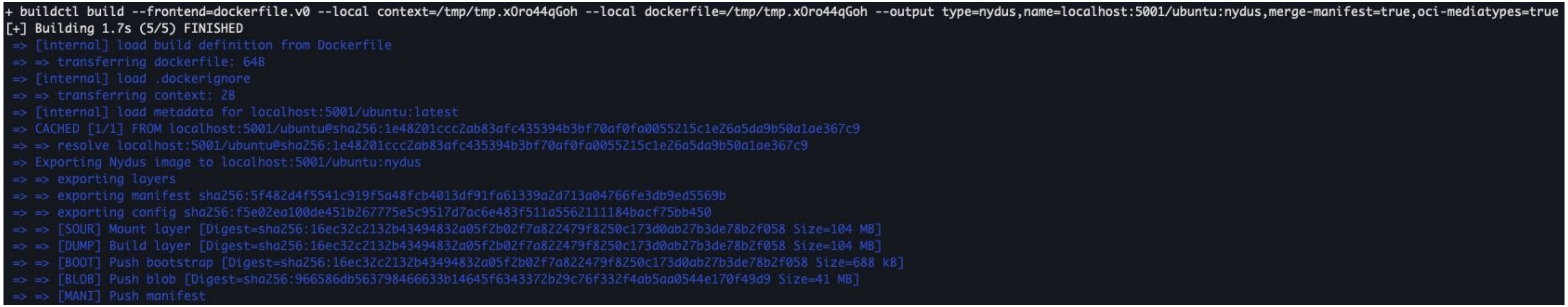
- nydusify
  - https://github.com/dragonflyoss/image-service/tree/master/contrib/nydusify
  - download image from remote registry, convert it to nydus image, and push nydus image to remote registry
  - nydusify convert --source <registry>/<repo>:<tag> \
    --target <registry>/<repo>:<nydus-tag>



```
[@~]$sudo ./nydusify convert --source docker.io/bergwolf/ubuntu:19.10 --target docker.io/bergwolf/ubuntu:19.10-nydus
INFO[2021-08-25T06:28:59Z] Version: 648f538ab5a32ad501aef6007a34b0ebfc2aec45.20210825.0309
INFO[2021-08-25T06:28:59Z] Parsing image docker.io/bergwolf/ubuntu:19.10
INFO[2021-08-25T06:29:01Z] Converting to docker.io/bergwolf/ubuntu:19.10-nydus
INFO[2021-08-25T06:29:01Z] [SOUR] Mount layer                            Digest="sha256:3f2411103a12c8e169df7a9ea00ff26ab07501858e3eff315a2e11c219e78ce1" Size="28 MB"
INFO[2021-08-25T06:29:01Z] [SOUR] Mount layer                            Digest="sha256:354c6da61dcc176c5b363ded571bea1de41b718131658fa0e563f2a749028cd1" Size="164 B"
INFO[2021-08-25T06:29:02Z] [SOUR] Mount layer                            Digest="sha256:354c6da61dcc176c5b363ded571bea1de41b718131658fa0e563f2a749028cd1" Size="164 B" Time=907.918066ms
<snip...>
INFO[2021-08-25T06:29:09Z] [BLOB] Push blob                              Digest="sha256:00d151e7d392e68e2c756a6fc42640006ddc0a98d37dba3f90a7b73f63188bbd" Size="7 B" Time=1.227762628s
INFO[2021-08-25T06:29:09Z] [BLOB] Push blob                              Digest="sha256:471e21f4587c06f589e8ba2aca83806dbc219994ea422048f1e13f8364485fee" Size="40 MB" Time=1.455197822s
INFO[2021-08-25T06:29:09Z] [MANI] Push manifest
INFO[2021-08-25T06:29:13Z] [MANI] Push manifest                          Time=3.430874134s
INFO[2021-08-25T06:29:13Z] Converted to docker.io/bergwolf/ubuntu:19.10-nydus
```

Manual Image Conversion -- buildkit

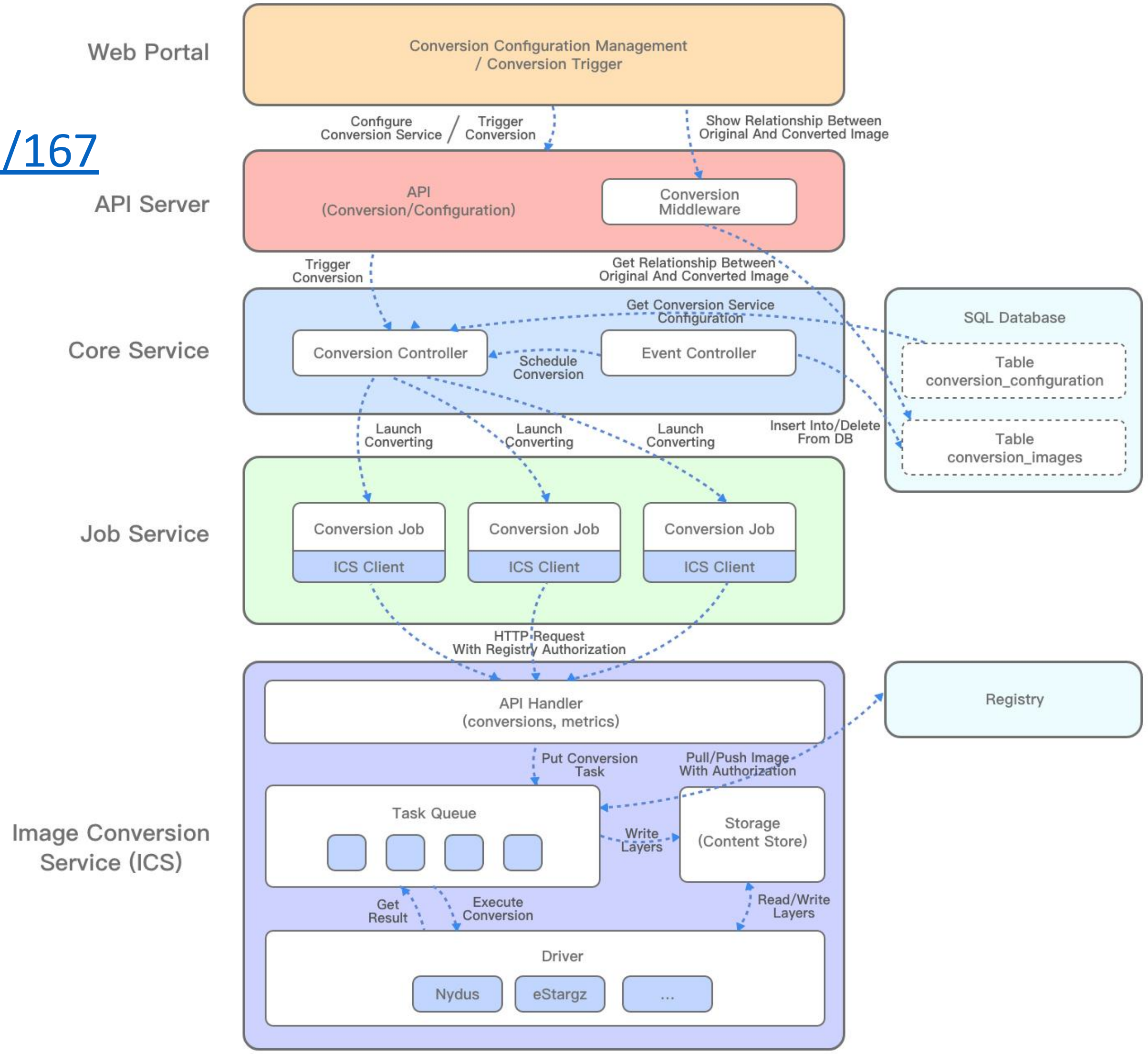
- buildkit
  - https://github.com/moby/buildkit/pull/2045
  - build nydus image directly from dockerfile and push it to remote registry
  - buildctl build --frontend=dockerfile.v0 \
    --local context=/build/dir \
    --local dockerfile=/build/DockerfileDir \
    --output type=nydus,name=<registry>/<repo>:<tag>



```
+ buildctl build --frontend=dockerfile.v0 --local context=/tmp/tmp.xOro44qGoh --local dockerfile=/tmp/tmp.xOro44qGoh --output type=nydus,name=localhost:5001/ubuntu:nydus,merge-manifest=true,oci-mediatypes=true
[+] Building 1.7s (5/5) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 64B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for localhost:5001/ubuntu:latest
 => CACHED [1/1] FROM localhost:5001/ubuntu@sha256:1e48201ccc2ab83afc435394b3bf70af0fa0055215c1e26a5da9b50a1ae367c9
 => => resolve localhost:5001/ubuntu@sha256:1e48201ccc2ab83afc435394b3bf70af0fa0055215c1e26a5da9b50a1ae367c9
 => Exporting Nydus image to localhost:5001/ubuntu:nydus
 => => exporting layers
 => => exporting manifest sha256:5f482d4f5541c919f5a48fcb4013df91fa61339a2d713a04766fe3db9ed5569b
 => => exporting config sha256:f5e02ea100de451b267775e5c9517d7ac6e483f511a5562111184bacf75bb450
 => => [SOUR] Mount layer [Digest=sha256:16ec32c2132b43494832a05f2b02f7a822479f8250c173d0ab27b3de78b2f058 Size=104 MB]
 => => [DUMP] Build layer [Digest=sha256:16ec32c2132b43494832a05f2b02f7a822479f8250c173d0ab27b3de78b2f058 Size=104 MB]
 => => [BOOT] Push bootstrap [Digest=sha256:16ec32c2132b43494832a05f2b02f7a822479f8250c173d0ab27b3de78b2f058 Size=688 kB]
 => => [BLOB] Push blob [Digest=sha256:966586db563798466633b14645f6343372b29c76f332f4ab5aa0544e170f49d9 Size=41 MB]
 => => [MANI] Push manifest
```

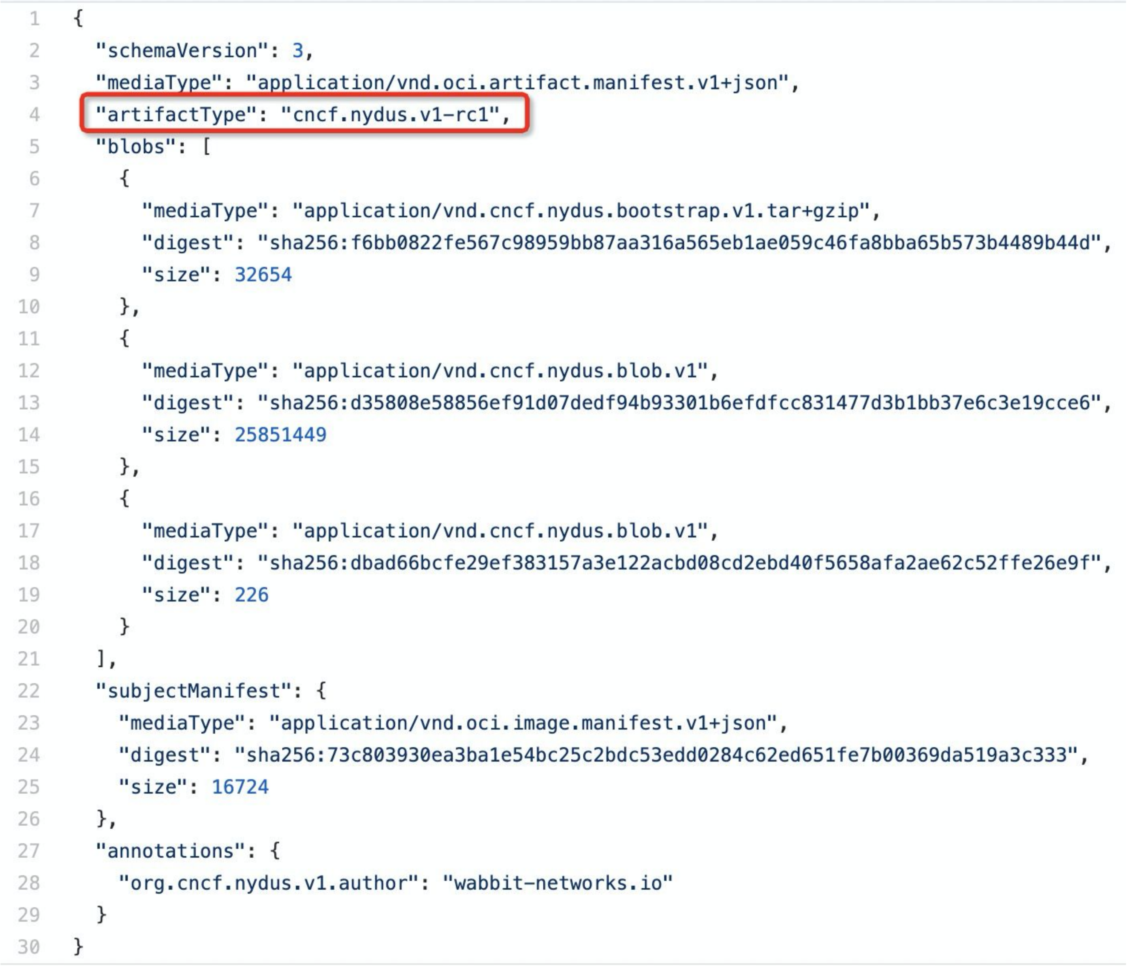
Automatic Conversion with Harbor

- https://github.com/goharbor/community/pull/167
- New image conversion service in harbor
- Started image acceleration working group
- Image format agnostic
  - nydus
  - estargz
  - future more format



OCI Artifacts Manifest

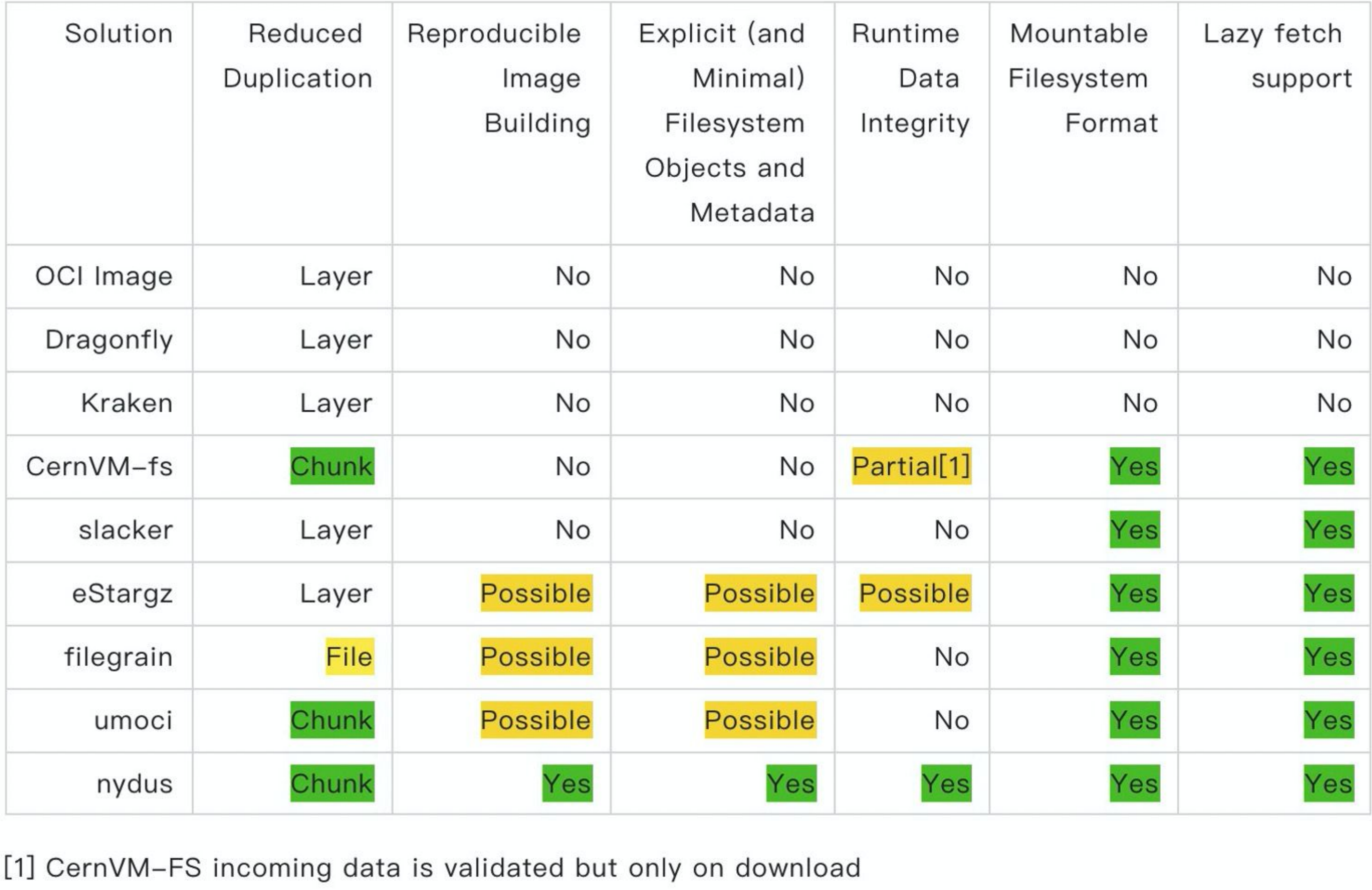
- https://github.com/opencontainers/artifacts/pull/29
- Map OCI image manifest to artifact manifest, to connect artifacts to images
- Beneficial to SBOM, Signatures, Nydus, Scan Results etc.
- Nydus image as an artifact type
    - cncf.nydus.v1-rc1



```json
{
  "schemaVersion": 3,
  "mediaType": "application/vnd.oci.artifact.manifest.v1+json",
  "artifactType": "cncf.nydus.v1-rc1",
  "blobs": [
    {
      "mediaType": "application/vnd.cncf.nydus.bootstrap.v1.tar+gzip",
      "digest": "sha256:f6bb0822fe567c98959bb87aa316a565eb1ae059c46fa8bba65b573b4489b44d",
      "size": 32654
    },
    {
      "mediaType": "application/vnd.cncf.nydus.blob.v1",
      "digest": "sha256:d35808e58856ef91d07dedf94b93301b6efdfcc831477d3b1bb37e6c3e19cce6",
      "size": 25851449
    },
    {
      "mediaType": "application/vnd.cncf.nydus.blob.v1",
      "digest": "sha256:dbad66bcfe29ef383157a3e122acbd08cd2ebd40f5658afa2ae62c52ffe26e9f",
      "size": 226
    }
  ],
  "subjectManifest": {
    "mediaType": "application/vnd.oci.image.manifest.v1+json",
    "digest": "sha256:73c803930ea3ba1e54bc25c2bdc53edd0284c62ed651fe7b00369da519a3c333",
    "size": 16724
  },
  "annotations": {
    "org.cncf.nydus.v1.author": "wabbit-networks.io"
  }
}
```

Community V2 Image Spec Requirements

- OCI community requirements on V2 Image format
  - https://hackmd.io/@cyphar/ociv2-brainstorm



Solution	Reduced Duplication	Reproducible Image Building	Explicit (and Minimal) Filesystem Objects and Metadata	Runtime Data Integrity	Mountable Filesystem Format	Lazy fetch support
OCI Image	Layer	No	No	No	No	No
Dragonfly	Layer	No	No	No	No	No
Kraken	Layer	No	No	No	No	No
CernVM-fs	Chunk	No	No	Partial[1]	Yes	Yes
slacker	Layer	No	No	No	Yes	Yes
eStargz	Layer	Possible	Possible	Possible	Yes	Yes
filegrain	File	Possible	Possible	No	Yes	Yes
umoci	Chunk	Possible	Possible	No	Yes	Yes
nydus	Chunk	Yes	Yes	Yes	Yes	Yes

[1] CernVM-FS incoming data is validated but only on download

Recap

- OCI images are large and slow
- Image lazy pulling is important for starting containers quickly
- Prefetching is a good friend to lazy pulling
- Ecosystem adoption is crucial for new image formats

- eStargz
  - Backward compatibility
  - Extension to the existing OCI image spec

- Nydus
  - Future looking
  - Proposal to the next generation OCI image spec

eStargz Project Information

- 100% OCI-compatible image format for lazy pulling
- Subproject of CNCF graduated containerd
- github : https://github.com/containerd/stargz-snapshotter
- slack : https://slack.containerd.io/ (#containerd-dev channel at CNCF slack)
- Pre-converted images are available on ghcr.io/stargz-containers

Nydus Project Information

NYDUS

- Image acceleration service with various improvements
- CNCF incubator dragonfly sub-project
- github : https://github.com/dragonflyoss/image-service
- slack : https://tinyurl.com/nydus-slack
- tutorial : https://tinyurl.com/nydus-tutorial

Thank You
&
Questions!