containerd Deep Dive

Akihiro Suda (NTT) & Wei Fu (Alibaba Cloud)

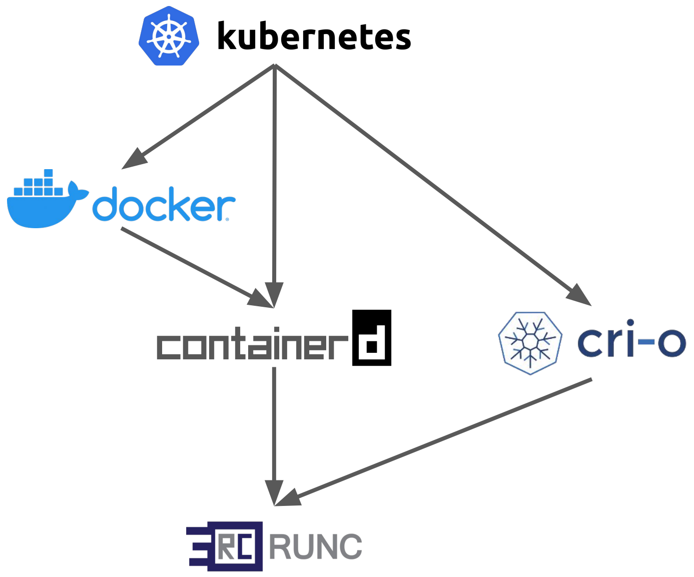
KubeCon EU 2020 Virtual (Aug 19, 2020)

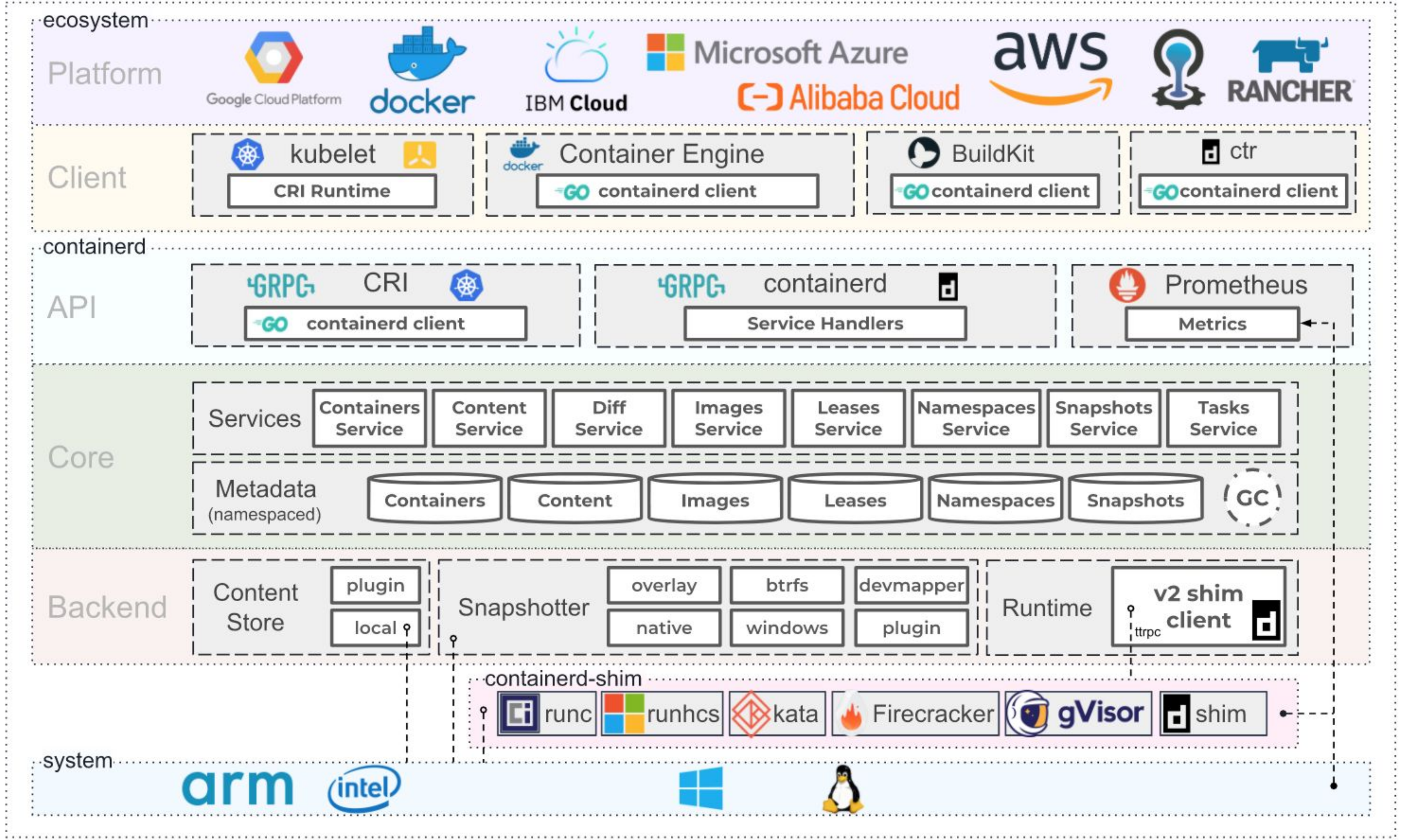
containerd overview

Akihiro Suda (NTT)

What is containerd ?

- "Mid-level Container runtime"
  - Below platforms (Docker, Kubernetes)
  - Above lower level runtimes (runc)
- Resource Manager
  - Container processes
  - Image artifacts
  - Filesystem snapshots
  - Metadata and dependencies
- CNCF graduated project since February 2019
  - Following Kubernetes, Prometheus, Envoy, and CoreDNS




ecosystem
Platform
Google Cloud Platform
docker
IBM Cloud
Microsoft Azure
Alibaba Cloud
aws
RANCHER
Client
kubelet
CRI Runtime
Container Engine
docker
GO containerd client
BuildKit
GO containerd client
ctr
GO containerd client
containerd
API
GRPC
CRI
GO containerd client
GRPC
containerd
Service Handlers
Prometheus
Metrics
Core
Services
Containers Service
Content Service
Diff Service
Images Service
Leases Service
Namespaces Service
Snapshots Service
Tasks Service
Metadata (namespaced)
Containers
Content
Images
Leases
Namespaces
Snapshots
GC
Backend
Content Store
plugin
local
Snapshotter
overlay
btrfs
devmapper
native
windows
plugin
Runtime
v2 shim client
ttrpc
containerd-shim
runc
runhcs
kata
Firecracker
gVisor
shim
system
arm
intel

Highly customizable

- **Runtime plugins**
    - Runc, gVisor, Kata, Firecracker...
- **Snapshotter plugins**
    - OverlayFS, BtrFS, ZFS, ...
- **Content store plugins**
    - Local, IPFS...
- **Stream processor plugins**
    - ImgCrypt, zstd...

Adoption of containerd

- Container engines

Docker & Moby	k3c	PouchContainer

- Kubernetes distributions

k3s	kubespray	microk8s
Charmed Kubernetes	kind	minikube

- Managed Kubernetes Services

Alibaba ACK	Amazon EKS (Fargate nodes)	Azure AKS
Google GKE	IBM IKS	And more...

KubeCon CloudNativeCon Europe 2020 Virtual

Adoption of containerd

- **BuildKit**
  - The modern implementation of `docker build`
- **LinuxKit**
  - Small Linux distro with containerd as the init
- **Faasd**
  - OpenFaaS for containerd
- **VMware Fusion Nautilus**
  - containerd on macOS, using VMware as the runtime plugin

Upcoming features in v1.4

Akihiro Suda (NTT)

Lazy pulling of images

- Run containers before completion of downloading the images

- Use cases:
    - Python/Ruby/Java/dotNET images
    - FaaS
    - Web apps with huge amount of HTML templates and media files
    - Jupyter Notebooks with big data samples included
    - Full GNOME/KDE desktop

Lazy pulling of images: Stargz & eStargz

- The containerd snapshotter plugin for Stargz & eStargz
  https://github.com/containerd/stargz-snapshotter

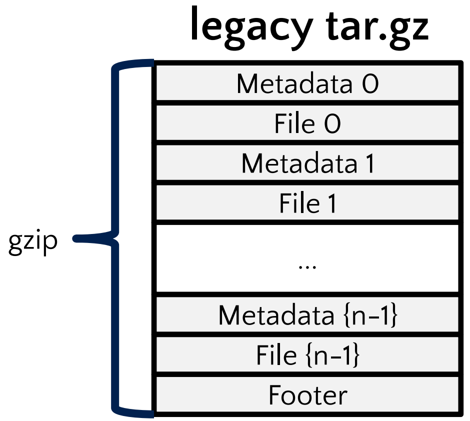
- **Stargz**: seekable tar.gz for lazy-pullable container images

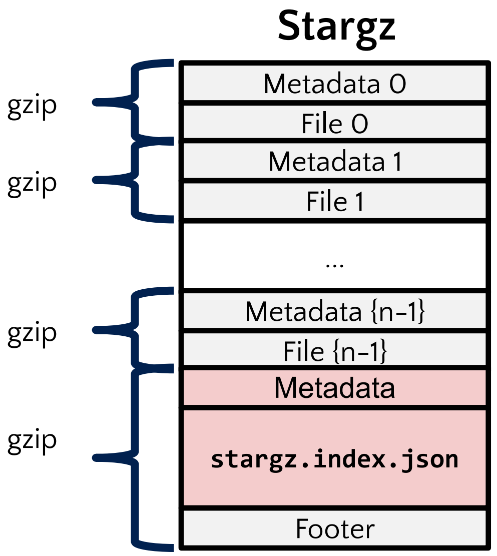
- **eStargz**: extended Stargz for batching frequently used files

- Both are fully compatible with legacy OCI tar.gz

KubeCon CloudNativeCon Europe 2020 Virtual

Lazy pulling of images: Stargz & eStargz



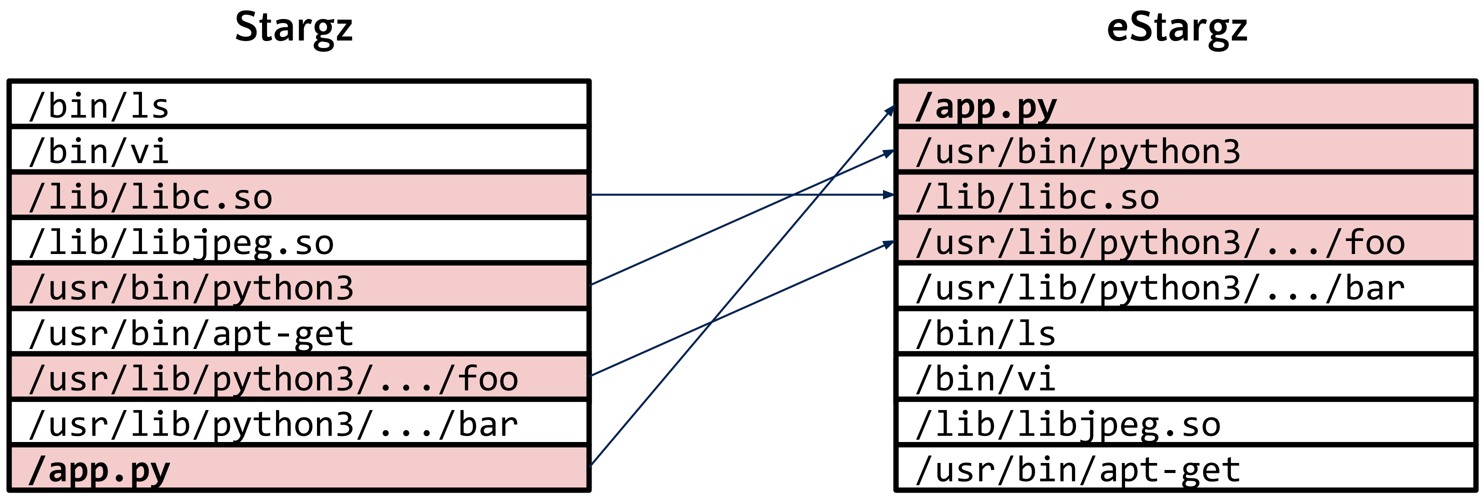


Can't inspect file offsets without reading the whole archive

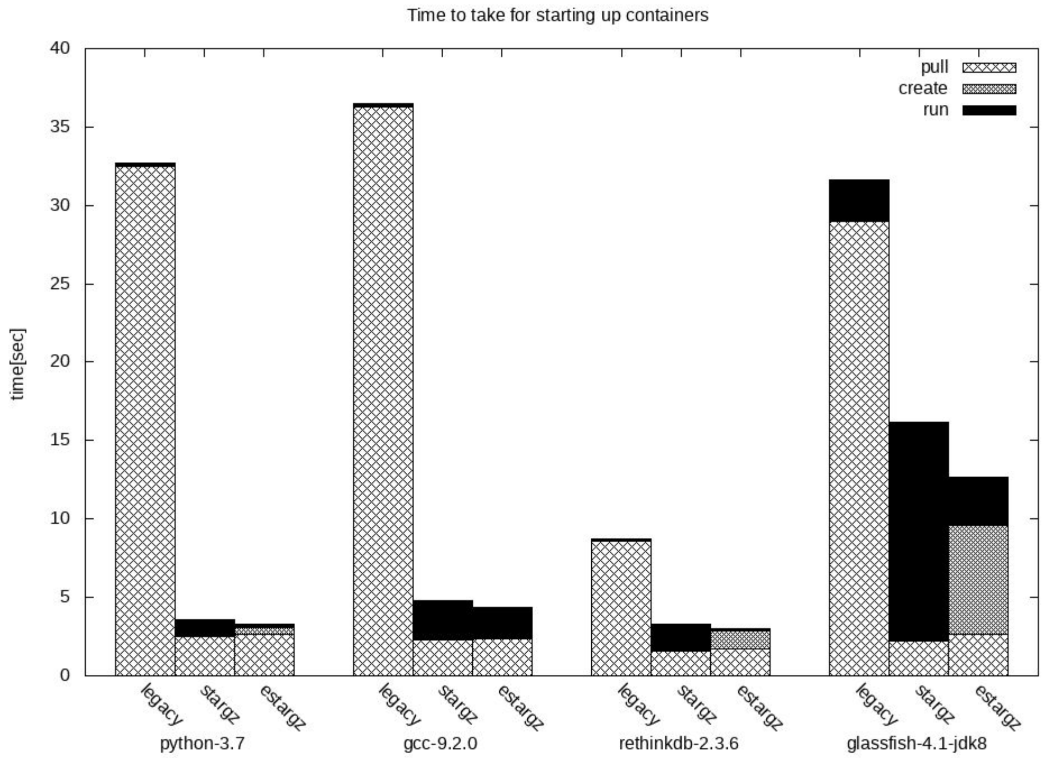
Can inspect the file offsets immediately

Lazy pulling of images: Stargz & eStargz

- eStargz profiles the actual file access pattern and reorders the file entries, so that relevant files can be prefetched in a single HTTP request

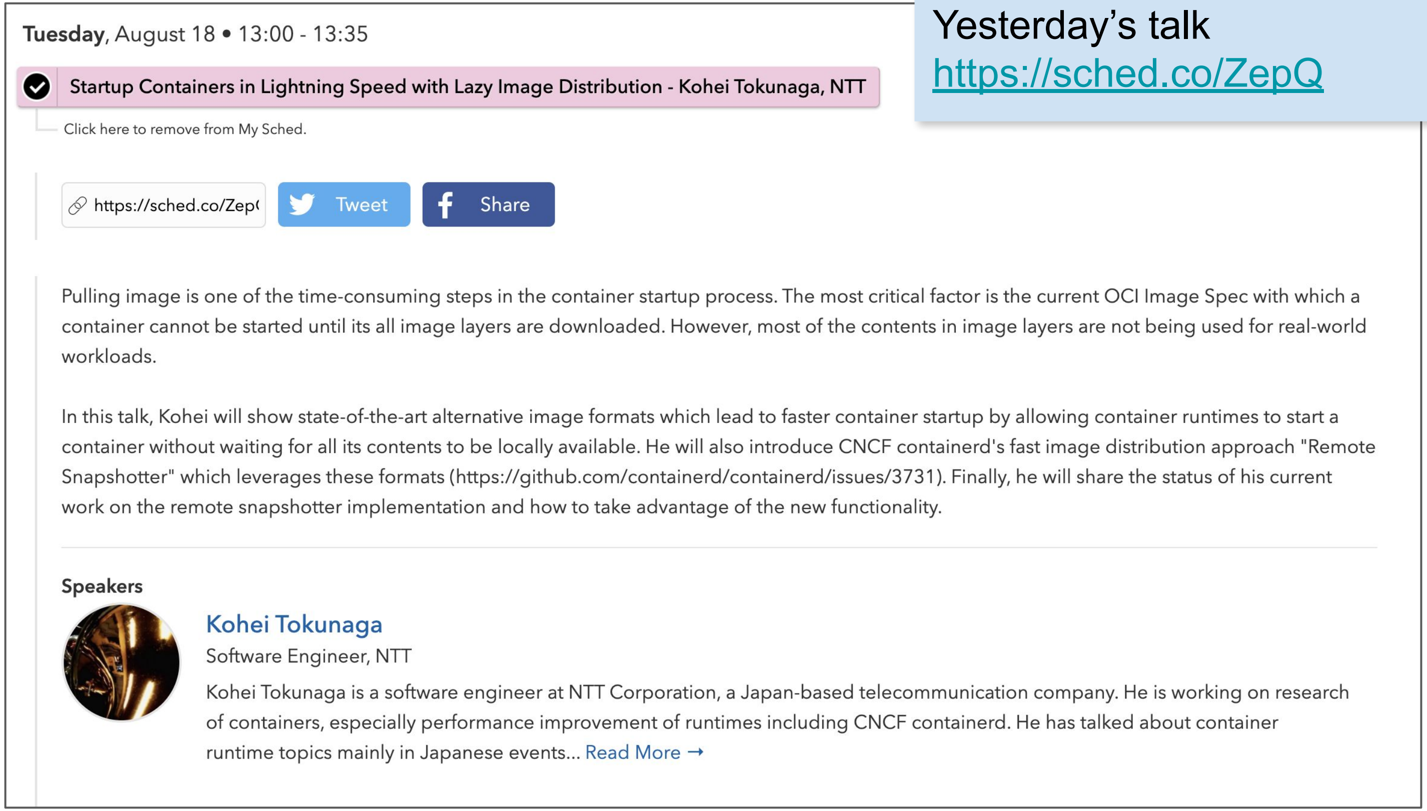


Lazy pulling of images: Stargz & eStargz



KubeCon CloudNativeCon Europe 2020 Virtual

Lazy pulling of images: Stargz & eStargz



Tuesday, August 18 • 13:00 - 13:35

Startup Containers in Lightning Speed with Lazy Image Distribution - Kohei Tokunaga, NTT

Click here to remove from My Sched.

https://sched.co/Zep(Tweet Share

Pulling image is one of the time-consuming steps in the container startup process. The most critical factor is the current OCI Image Spec with which a container cannot be started until its all image layers are downloaded. However, most of the contents in image layers are not being used for real-world workloads.

In this talk, Kohei will show state-of-the-art alternative image formats which lead to faster container startup by allowing container runtimes to start a container without waiting for all its contents to be locally available. He will also introduce CNCF containerd's fast image distribution approach "Remote Snapshotter" which leverages these formats (https://github.com/containerd/containerd/issues/3731). Finally, he will share the status of his current work on the remote snapshotter implementation and how to take advantage of the new functionality.

Speakers

Kohei Tokunaga

Software Engineer, NTT

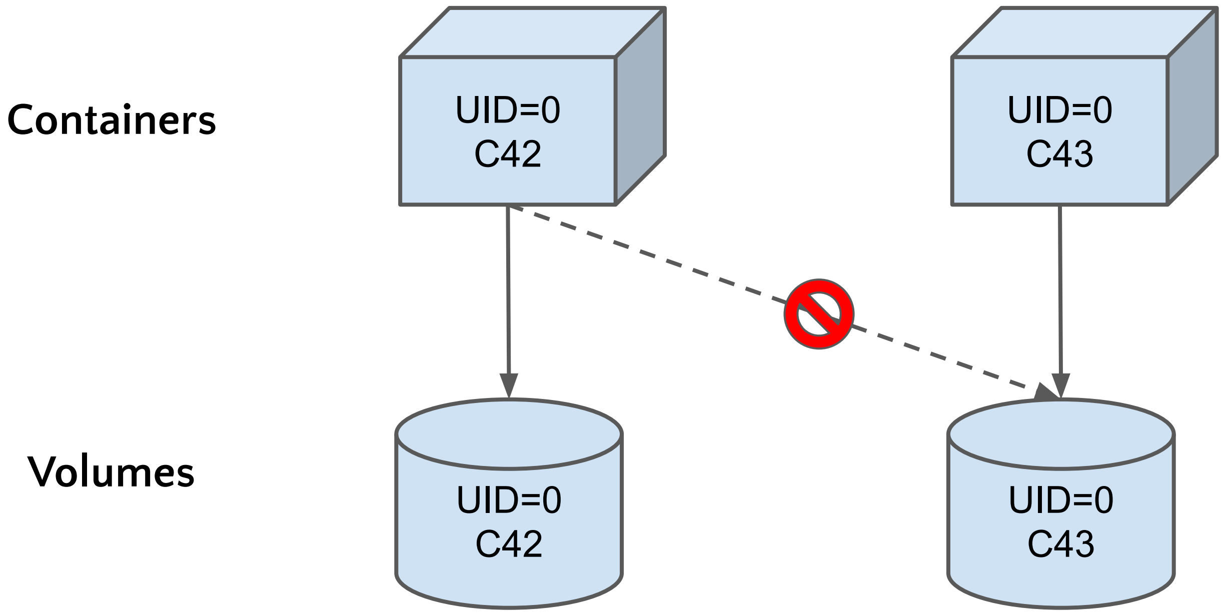
Kohei Tokunaga is a software engineer at NTT Corporation, a Japan-based telecommunication company. He is working on research of containers, especially performance improvement of runtimes including CNCF containerd. He has talked about container runtime topics mainly in Japanese events... Read More →

Yesterday's talk
https://sched.co/ZepQ

KubeCon CloudNativeCon Europe 2020 Virtual

Support for SELinux MCS on CRI mode

- MCS: multi-category security



Support for cgroup v2

- The new cgroup hierarchy, adopted by Fedora (since 31)

- Simpler layout
  - V1: /sys/fs/cgroup/{memory,cpu,devices,pids....}/foo
  - V2: /sys/fs/cgroup/foo

- Supports eBPF integration, pressure metrics, improved OOM control...

- Friendly to non-root users

Improved support for rootless mode

- Run containerd (and relevant components) as a non-root user
- Protect the host from potential vulnerabilities
- Adoption in containerd-related projects
  - Docker
  - BuildKit
  - k3s
  - k3c (on plan)
  - Kubernetes (on proposal, KEP 1371)

Improved support for rootless mode

- [v1.3] No support for resource limitation (docker run --cpus … --memory …)
  - Because unprivileged users cannot control cgroups

- [v1.3] No support for overlayfs snapshotter
  - Because unprivileged users cannot mount overlayfs (except on Ubuntu/Debian kernels)
  - “Native” snapshotter can be used, but slow and wastes the disk

Improved support for rootless mode

- [v1.3] No support for resource limitation (docker run --cpus … --memory …)
  - Because unprivileged users cannot control cgroups

    → v1.4 supports resource limitation
    (requires cgroup v2 and systemd)

- [v1.3] No support for overlayfs snapshotter
  - Because unprivileged users cannot mount overlayfs

    (except on Ubuntu/Debian kernels)
  - "Native" snapshotter can be used, but slow and wastes the disk

    → v1.4 supports FUSE-OverlayFS snapshotter
    (requires kernel >= 4.18)

Demo: Rootless Kubernetes with Cgroup v2

“Usernetes” https://github.com/rootless-containers/usernetes

https://asciinema.org/a/349859

Other changes in v1.4

- Windows CRI
- systemd NOTIFY_SOCKET
- Support reloading CNI config without restarting the daemon
- Socat binary is no longer needed

Release note: https://github.com/containerd/containerd/releases

v1.5 planning

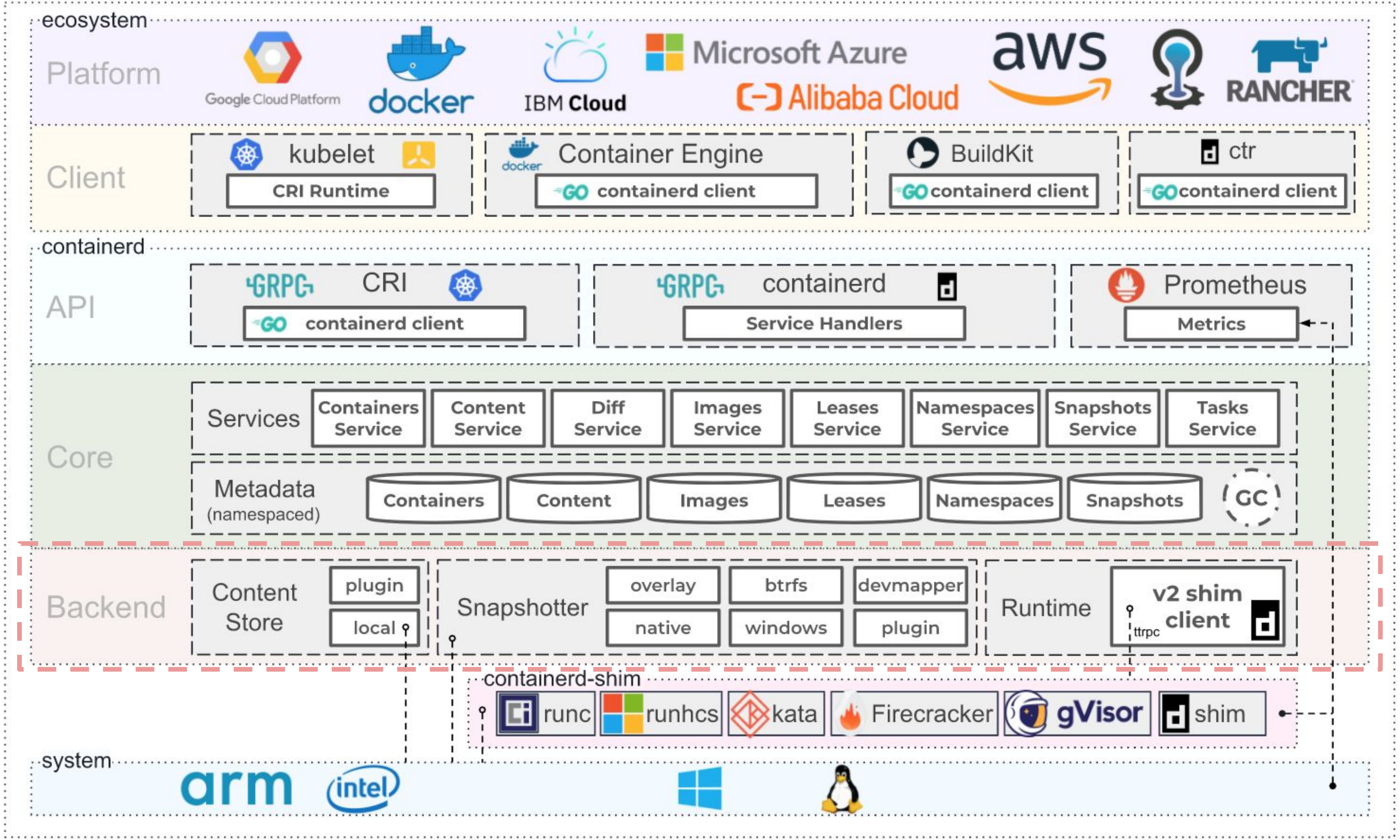
- NRI: Node Resource Interface (#4411)
  - The new common interface for node resources such as cgroup
  - The plugin spec is very similar to CNI

- Sandbox API (#4131)
  - Pod sandbox as a first-class object
  - No “/pause” process

- Filesystem quota (#759)

containerd: external plugins

Wei Fu (Alibaba Cloud)

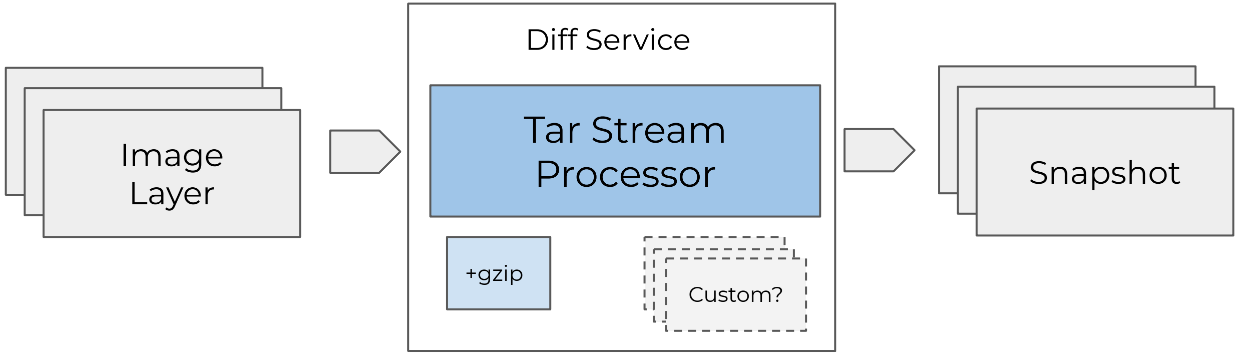

ecosystem
Platform
Google Cloud Platform
docker
IBM Cloud
Microsoft Azure
Alibaba Cloud
aws
RANCHER
Client
kubelet
CRI Runtime
Container Engine
docker
containerd client
BuildKit
containerd client
ctr
containerd client
containerd
API
GRPC
CRI
containerd client
GRPC
containerd
Service Handlers
Prometheus
Metrics
Core
Services
Containers Service
Content Service
Diff Service
Images Service
Leases Service
Namespaces Service
Snapshots Service
Tasks Service
Metadata (namespaced)
Containers
Content
Images
Leases
Namespaces
Snapshots
GC
Backend
Content Store
plugin
local
Snapshotter
overlay
btrfs
devmapper
native
windows
plugin
Runtime
v2 shim client
ttrpc
containerd-shim
runc
runhcs
kata
Firecracker
gVisor
shim
system
arm
intel

Backend as external plugins

- **Big goal - no re-compilation required!!!**
- Stream processors
- gRPC proxy plugin for image storage
- RuntimeV2 proto for OCI Runtime

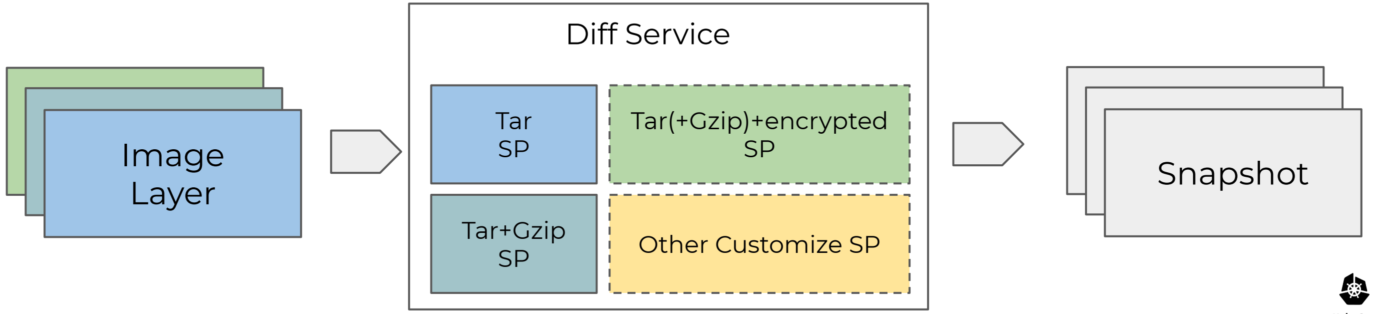
Stream processor

- OCI Image layer data packaged in tar archive
- OCI image spec only supports few compression algorithms
    - +gzip/+zstd, but +gzip is more common
- How to handle experimental media-type stream?
    - Or encryption purpose?



Stream processor

- Stream processor(SP) is binary plugin handling media-type stream
  - Accepts customize media-types, returns other one
  - Call binary for media-type converter
- Example
  - containerd/imgcrypt



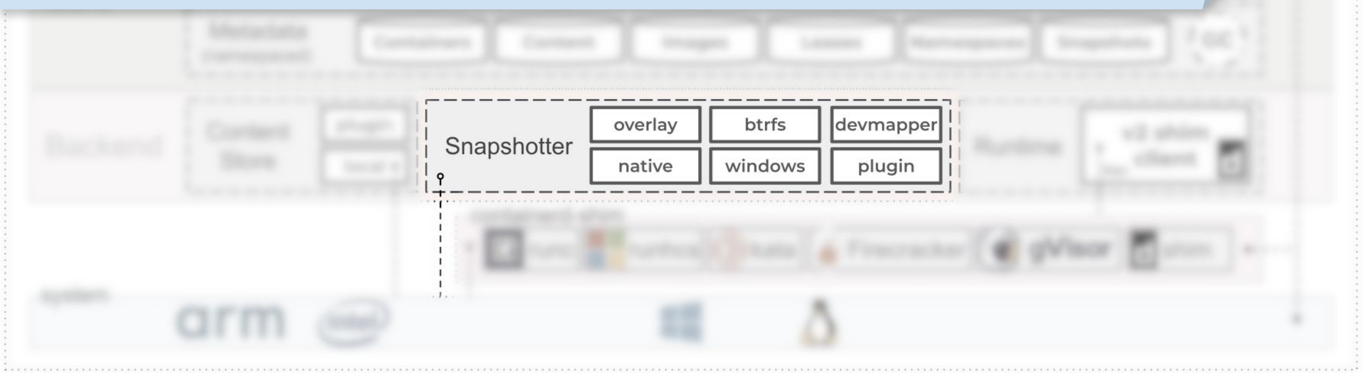
Stream processor - Demo

- Integrate with +zstd media-type
- asciinema link

```
[stream_processors]
    [stream_processors."zstd"]
        accepts = ["application/vnd.oci.image.layer.v1.tar+zstd"]
        returns = "application/vnd.oci.image.layer.v1.tar"
        path = "zstd"
        args = ["-dcf"]
```

Snapshot proxy plugin

```
// Snapshot service manages snapshots
service Snapshots {
   rpc Prepare(PrepareSnapshotRequest) returns (PrepareSnapshotResponse);
   rpc View(ViewSnapshotRequest) returns (ViewSnapshotResponse);
   rpc Mounts(MountsRequest) returns (MountsResponse);
   rpc Commit(CommitSnapshotRequest) returns (google.protobuf.Empty);
   rpc Remove(RemoveSnapshotRequest) returns (google.protobuf.Empty);
   rpc Stat(StatSnapshotRequest) returns (StatSnapshotResponse);
   rpc Update(UpdateSnapshotRequest) returns (UpdateSnapshotResponse);
   rpc List(ListSnapshotsRequest) returns (stream ListSnapshotsResponse);
   rpc Usage(UsageRequest) returns (UsageResponse);
}
```



Snapshot proxy plugin

- Configure with proxy_plugins
- Example
  - stargz-snapshotter
  - CVMFS Containerd Snapshotter

```
[proxy_plugins]
  [proxy_plugins.customsnapshot]
    type = "snapshot"
    address = "/var/run/mysnapshotter.sock"
```

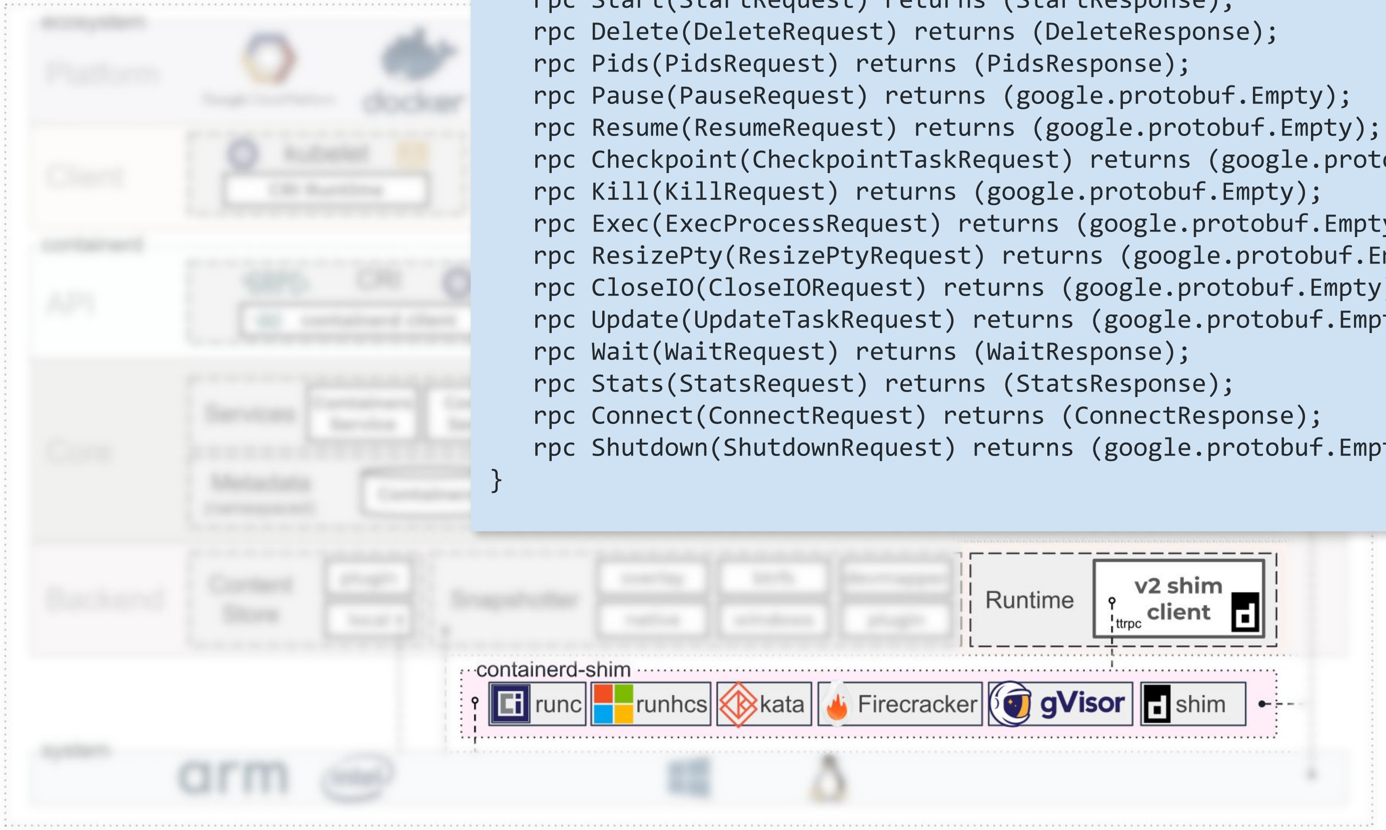
```
package main

import(
    "net"
    "log"


    "github.com/containerd/containerd/api/services/snapshots/v1"
    "github.com/containerd/containerd/contrib/snapshotservice"
)

func main() {
  rpc := grpc.NewServer()
  sn := CustomSnapshotter()
  service := snapshotservice.FromSnapshotter(sn)
  snapshots.RegisterSnapshotsServer(rpc, service)

  // Listen and serve
  l, err := net.Listen("unix", "/var/run/mysnapshotter.sock")
  if err != nil {
    log.Fatalf("error: %v\n", err)
  }

  if err := rpc.Serve(l); err != nil {
    log.Fatalf("error: %v\n", err)
  }
}
```

Runtime V2

- A first class shim API for runtime authors to integrate with containerd
  - More VM like runtimes have internal state and more abstract actions
  - A CLI approach introduces issues with state management
  - Each runtimes has its own values, but keep containerd in solid core scope
- Example
  - gVisor
  - KataContainer
  - Firecracker

Runtime V2

```
service Task {
    rpc State(StateRequest) returns (StateResponse);
    rpc Create(CreateTaskRequest) returns (CreateTaskResponse);
    rpc Start(StartRequest) returns (StartResponse);
    rpc Delete(DeleteRequest) returns (DeleteResponse);
    rpc Pids(PidsRequest) returns (PidsResponse);
    rpc Pause(PauseRequest) returns (google.protobuf.Empty);
    rpc Resume(ResumeRequest) returns (google.protobuf.Empty);
    rpc Checkpoint(CheckpointTaskRequest) returns (google.protobuf.Empty);
    rpc Kill(KillRequest) returns (google.protobuf.Empty);
    rpc Exec(ExecProcessRequest) returns (google.protobuf.Empty);
    rpc ResizePty(ResizePtyRequest) returns (google.protobuf.Empty);
    rpc CloseIO(CloseIORequest) returns (google.protobuf.Empty);
    rpc Update(UpdateTaskRequest) returns (google.protobuf.Empty);
    rpc Wait(WaitRequest) returns (WaitResponse);
    rpc Stats(StatsRequest) returns (StatsResponse);
    rpc Connect(ConnectRequest) returns (ConnectResponse);
    rpc Shutdown(ShutdownRequest) returns (google.protobuf.Empty);
}
```

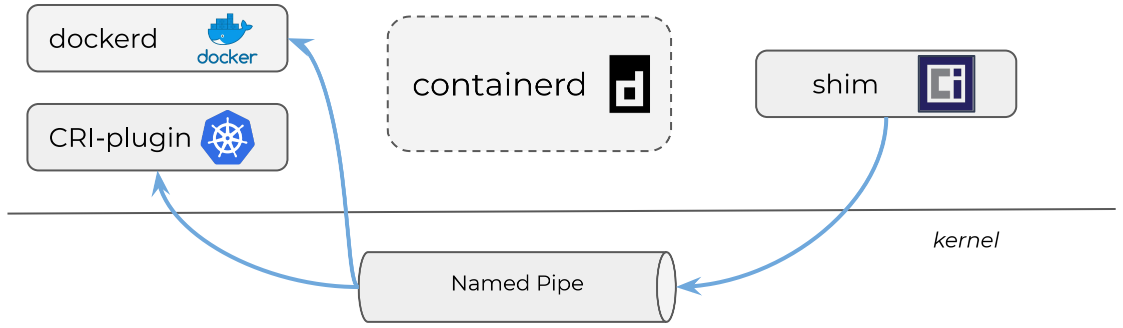


Runtime V2 - Binary

- Binary naming convention
  - Name io.containerd.**runc**.**v2** --> Binary containerd-shim-**runc**-**v2**
    - So both io.containerd.runc.v1 and io.containerd.runc.v2 are runtime V2
    - runc.v2 supports grouping several containers with less resource
    - runc.v2 as CRI plugin's default runtime
  - Via a runtime binary available in containerd's PATH
- Required start/delete sub-commands
  - Resources created by container will be cleanup by delete sub-command

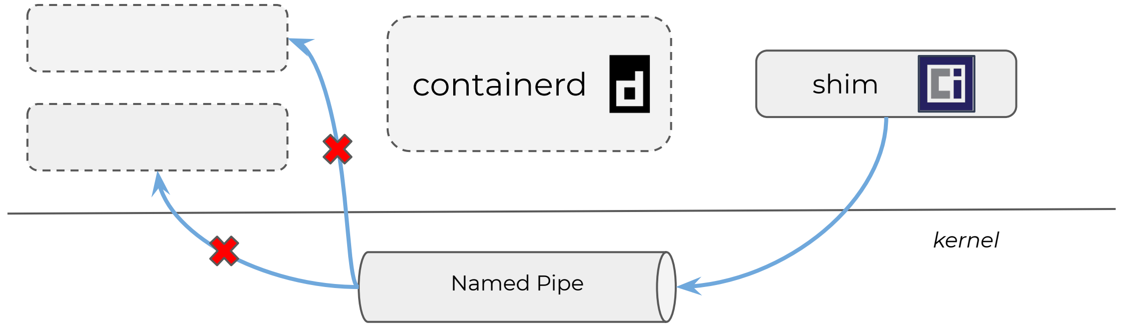
Runtime V2 - Logging

- fifo/npipe as default channel
  - Receiver consumes more resources to handle log output.



Runtime V2 - Logging

- fifo/npipe as default channel
    - Receiver consumes more resources to handle log output.
    - And it requires that **receiver must be alive!!!**
    - Impact running containers if receiver is down too long.



Runtime V2 - Logging

- Support pluggable logging via STDIO URIs
    - fifo - Linux (default)
    - npipe - Windows (default)
    - binary - Linux & Windows
    - file - Linux & Windows

STDIO URI	**schema**	://	**path**	?key=value
file	**file**	://	**/var/log/cntr/hi**	?maxSize=100MB
binary	**binary**	://	**/usr/bin/syslog**	?addr=192.168.0.3

KubeCon CloudNativeCon Europe 2020 Virtual

Thank you