containerd: integration

Lantao Liu (Google)
Wei Fu (Alibaba Cloud)

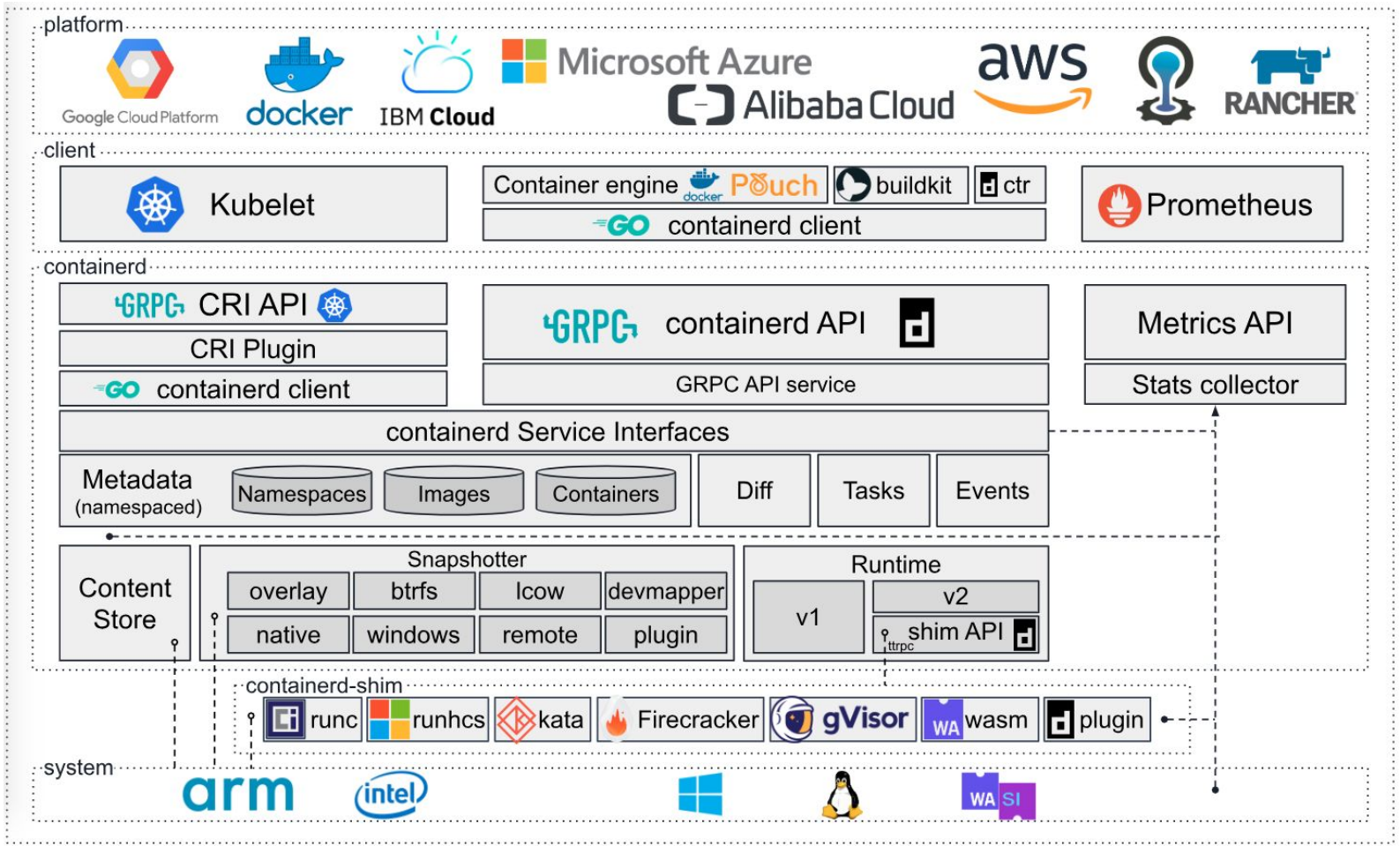
containerd status


CONGRATS
containerd
CLASS OF 2019
Love, CNCF

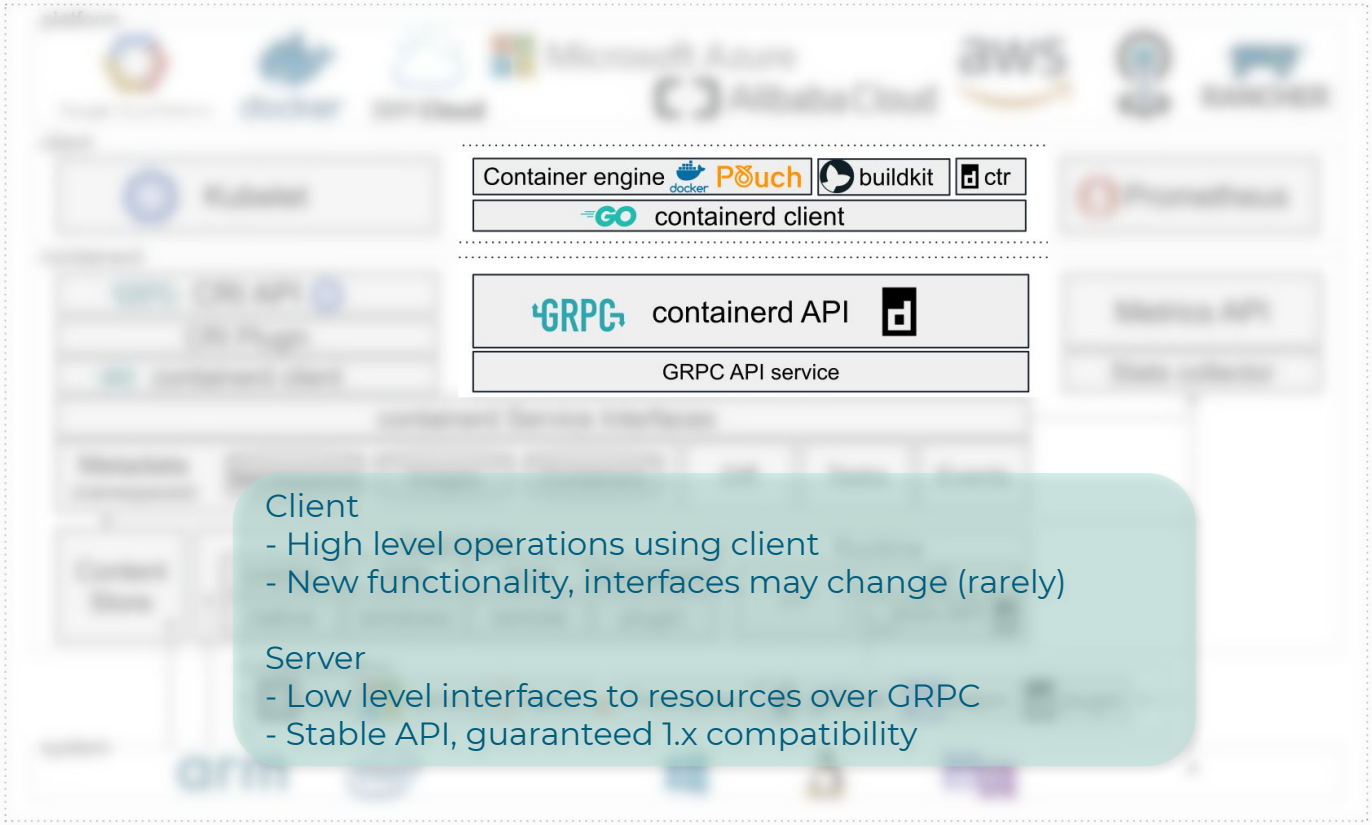
containerd matures

- 5th project to graduate from CNCF
- Broad support from companies
- All major cloud providers using containerd
- Support Linux and Windows platform

Architecture


platform
Google Cloud Platform
docker
IBM Cloud
Microsoft Azure
Alibaba Cloud
aws
RANCHER
client
Kubelet
Container engine
docker
Pouch
buildkit
ctr
containerd client
Prometheus
containerd
GRPC
CRI API
CRI Plugin
containerd client
GRPC
containerd API
GRPC API service
Metrics API
Stats collector
containerd Service Interfaces
Metadata
(namespaced)
Namespaces
Images
Containers
Diff
Tasks
Events
Content
Store
Snapshotter
overlay
btrfs
lcow
devmapper
native
windows
remote
plugin
Runtime
v1
v2
shim API
ttrpc
containerd-shim
runc
runhcs
kata
Firecracker
gVisor
WA
wasm
plugin
system
arm
intel
WA
SI

Client-Server Design

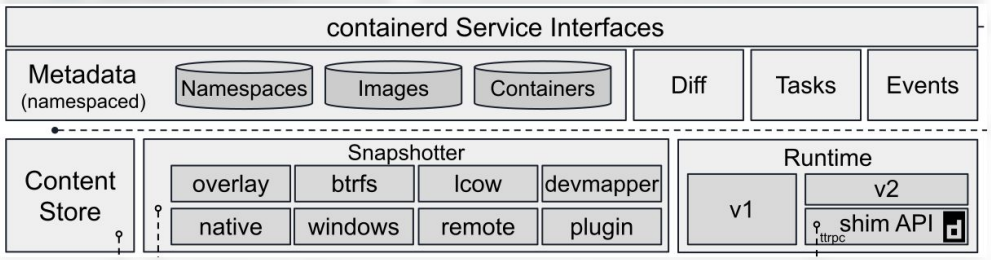


Client
- High level operations using client
- New functionality, interfaces may change (rarely)

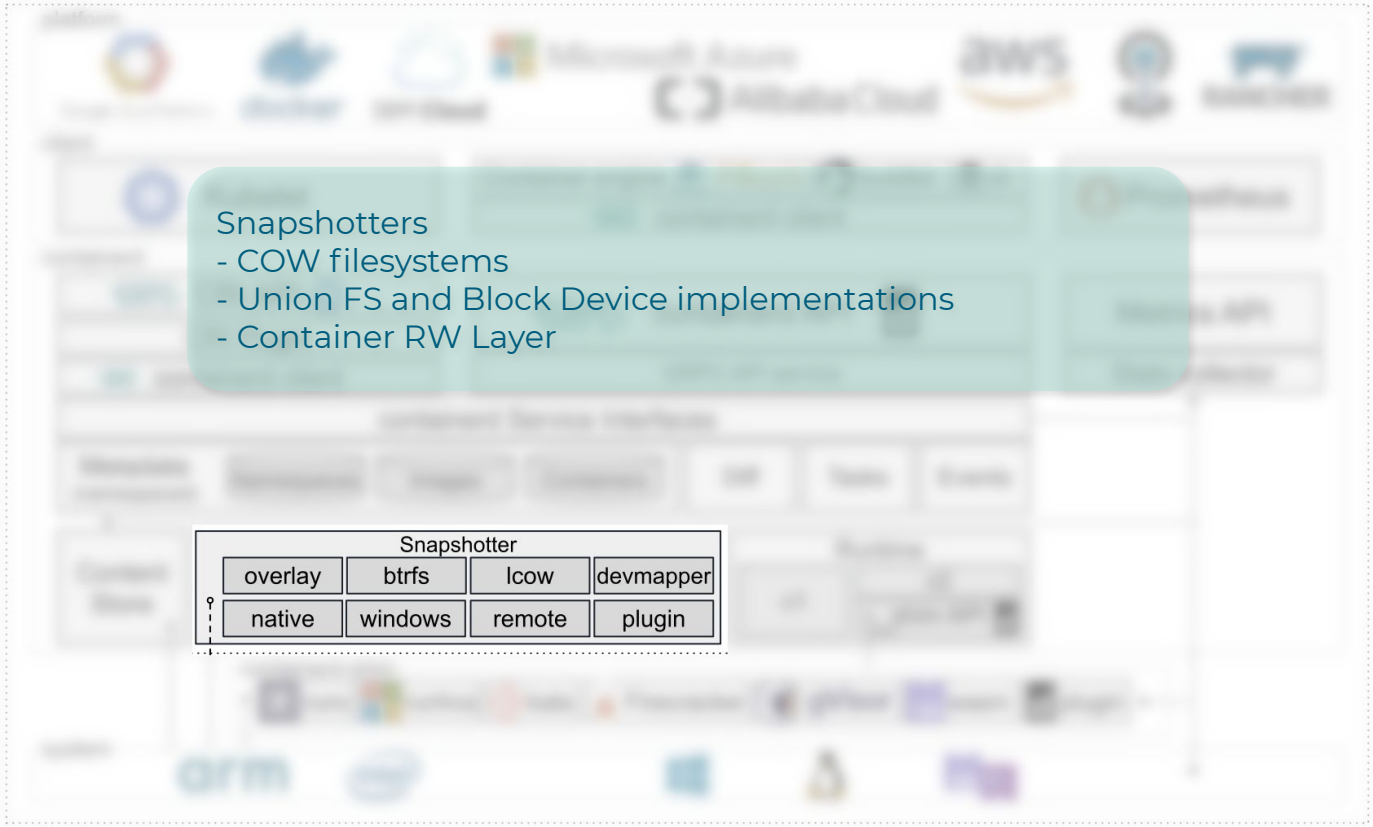
Server
- Low level interfaces to resources over GRPC
- Stable API, guaranteed 1.x compatibility

Backend

Service Interface
- Provides access to all components
- Low level components wrapped by metadata store
- Provides namespacing (content/Snapshotter/Image/Container)

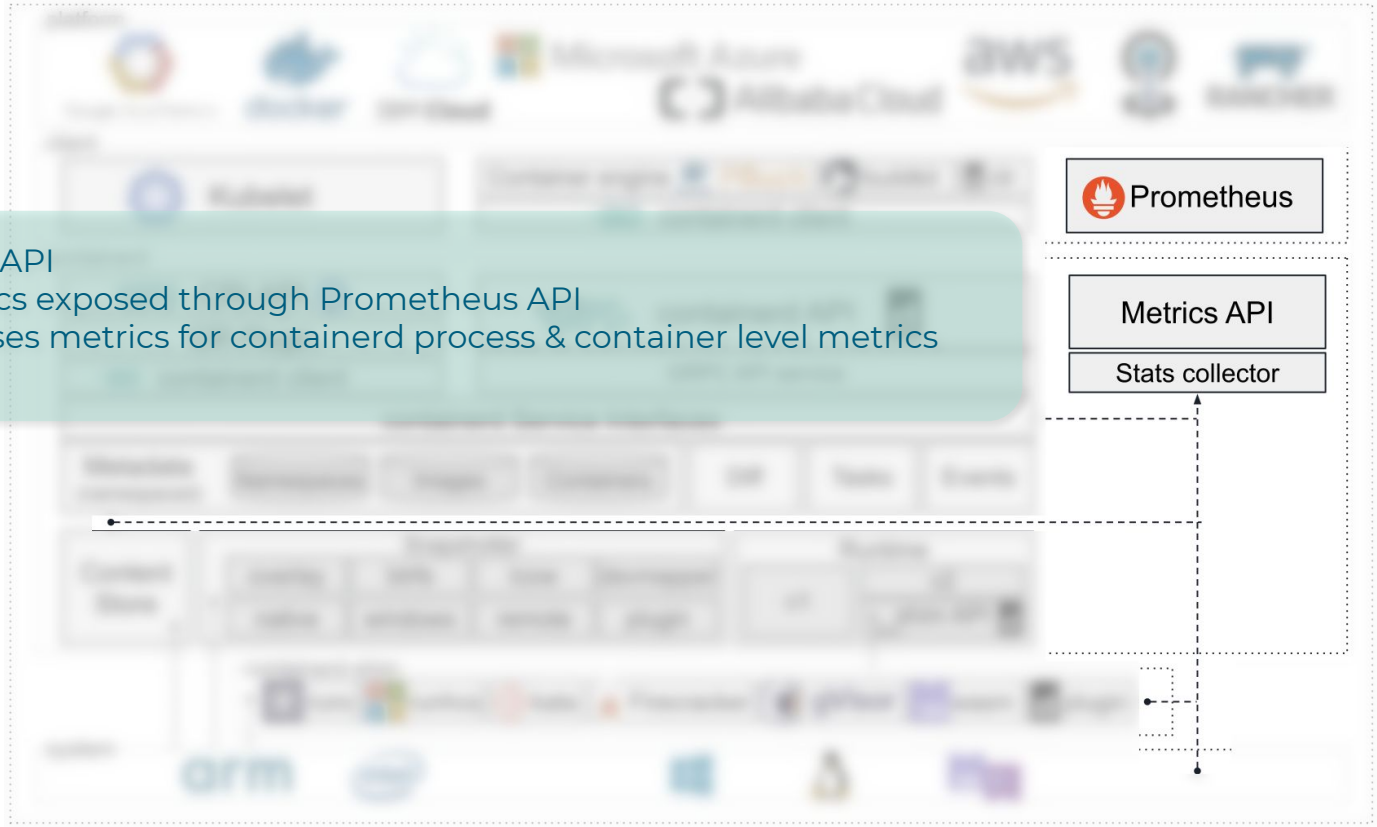


Snapshotter

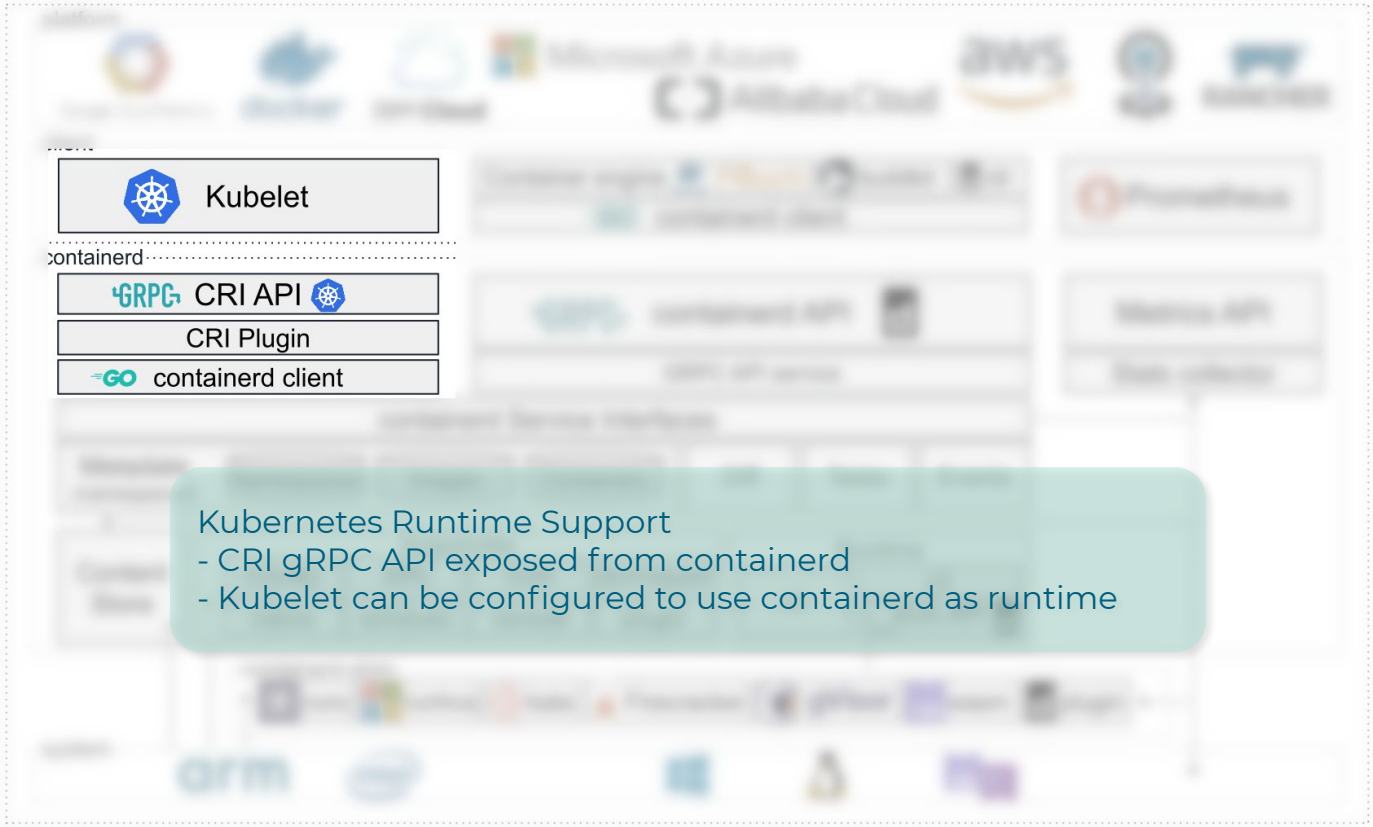


Metrics

Metric API
- Metrics exposed through Prometheus API
- Exposes metrics for containerd process & container level metrics



Kubernetes Runtime Support

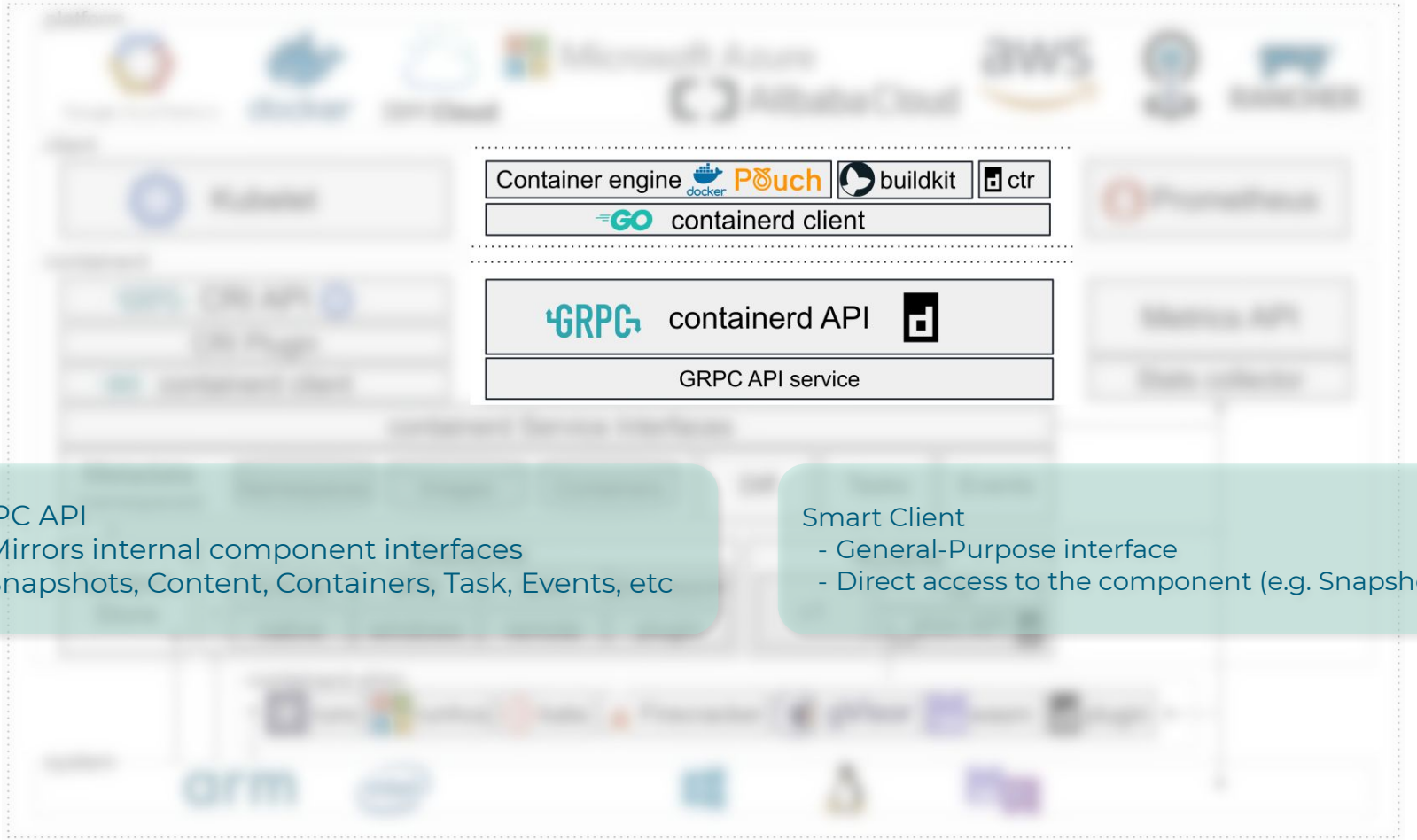


Kubernetes Runtime Support
- CRI gRPC API exposed from containerd
- Kubelet can be configured to use containerd as runtime

Summary

- Stable gRPC interface
- Kubernetes Runtime Support

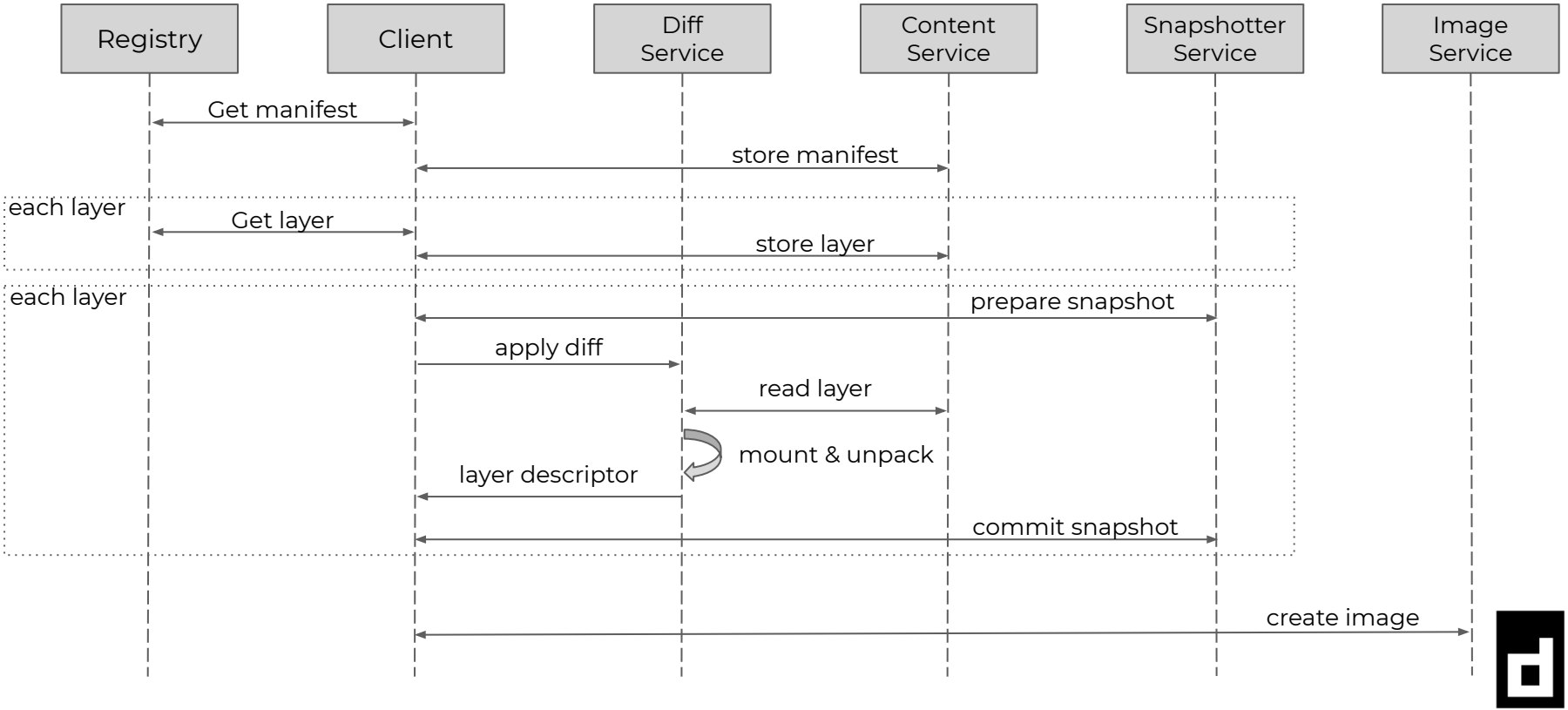
Smart Client Model



gRPC API
- Mirrors internal component interfaces
- Snapshots, Content, Containers, Task, Events, etc

Smart Client
- General-Purpose interface
- Direct access to the component (e.g. Snapshots)

Pull Image


Registry
Client
Diff Service
Content Service
Snapshotter Service
Image Service
Get manifest
store manifest
each layer
Get layer
store layer
each layer
prepare snapshot
apply diff
read layer
mount & unpack
layer descriptor
commit snapshot
create image

Push Image


Registry
Client
Image Service
Content Service
get image
read manifest
each layer
read layer
Push layer
Push manifest

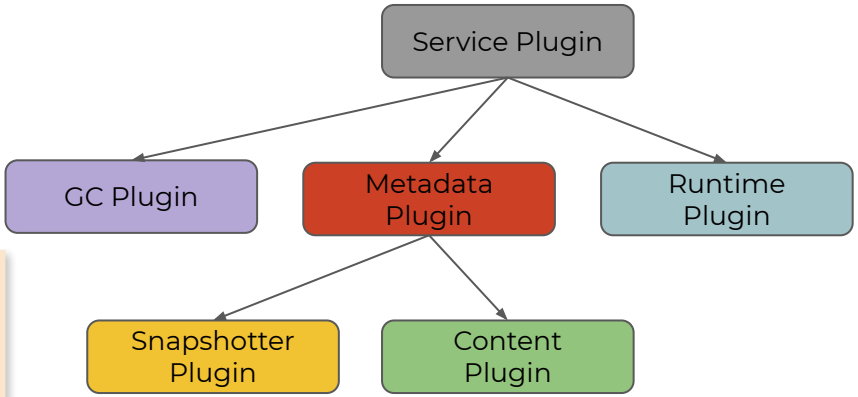
Aimed to

- Loosely coupled components
- Bring decoupled components together into usable toolset
- General Purpose API in client side, not in server side
- Support any custom requirements

Component as Plugin

Plugin Registration

- loose coupling and clear boundaries
- dependency Graph




```
plugin.Register(&plugin.Registration{
        Type: plugin.MetadataPlugin,
        ID:   "bolt",
        Requires: []plugin.Type{
            plugin.ContentPlugin,
            plugin.SnapshotPlugin,
        },
        Config: &srvconfig.BoltConfig{
            ContentSharingPolicy: srvconfig.SharingPolicyShared,
        },
        InitFn: func(ic *plugin.InitContext) (interface{}, error) {
        },
}
```

Recompiled with 3th party plugins

- Provided common entrypoint for server bootstrap
  - containerd/containerd#2131
- Easy to extend one domain by plugin registration
- Build your owner containerd with aufs snapshotter
  - code in gist

External Plugins

Extend without recompiling containerd...

- Proxy to another gRPC service
- Via a runtime binary available in containerd's PATH

Proxy Plugin on gRPC

Support Proxy

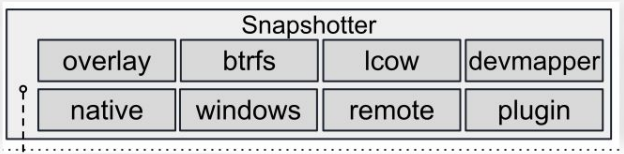
- Create remote plugin as proxy
- Configure it for containerd
- Snapshotter and Content only

```
for name, pp := range config.ProxyPlugins {
        ...
        switch pp.Type {
        case string(plugin.SnapshotPlugin), "snapshot":
                t = plugin.SnapshotPlugin
                f = func(conn *grpc.ClientConn) interface{} {
                        return ssproxy.NewSnapshotter(ssapi.NewSnapshotsClient(conn), ssname)
                }

        case string(plugin.ContentPlugin), "content":
                t = plugin.ContentPlugin
                f = func(conn *grpc.ClientConn) interface{} {
                        return csproxy.NewContentStore(csapi.NewContentClient(conn))
                }
        default:
                log.G(ctx).WithField("type", pp.Type).Warn("unknown proxy plugin type")
        }

        plugin.Register(&plugin.Registration{
                Type: t,
                ID:   name,
                InitFn: func(ic *plugin.InitContext) (interface{}, error) {
                        ...
                        return f(conn), nil
                },
        },
}
```

```
// 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);
}
```



Remote Snapshotter
- implement Snapshotter gRPC API
- containerd as proxy

Remote snapshotter service

- Configure with ***proxy_plugins***
- Build as an external plugin

```
[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 Plugins

Why external runtime plugins?

- 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

Runtime v2 API

- Minimal and scoped to the execution lifecycle of a container
- Binary naming convention
  - Type *io.containerd.runsc.v1* -> Binary *containerd-shim-runsc-v1*
- Host level shim configuration



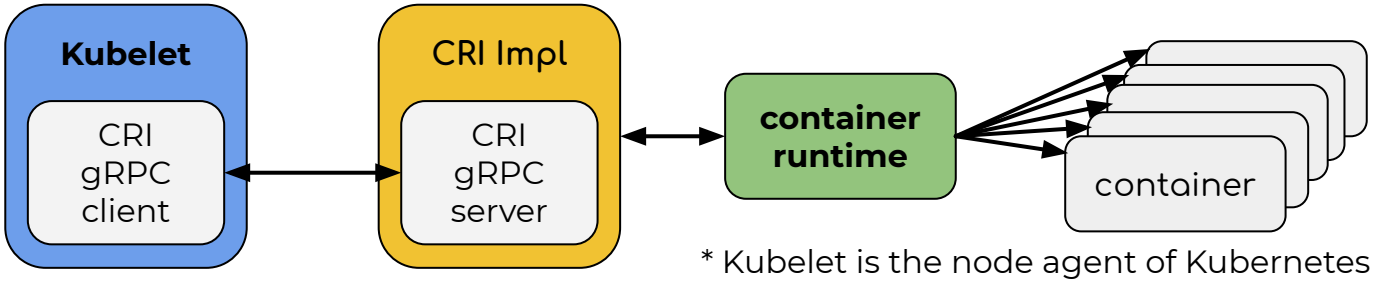

```
service Task {
  rpc State(StateRequest) returns (StateResponse);
  rpc Create(CreateTaskRequest) returns (CreateTaskResponse);
  rpc Start(StartRequest) returns (StartResponse);
  rpc Delete(DeleteRequest) returns (DeleteResponse);
  rpc Pids(PidsRequest) returns (PidsResponse);
  rpc Pause(PauseRequest) 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);
}
```

Kubernetes CRI

Kubernetes CRI

Kubernetes CRI (Container Runtime Interface) is:

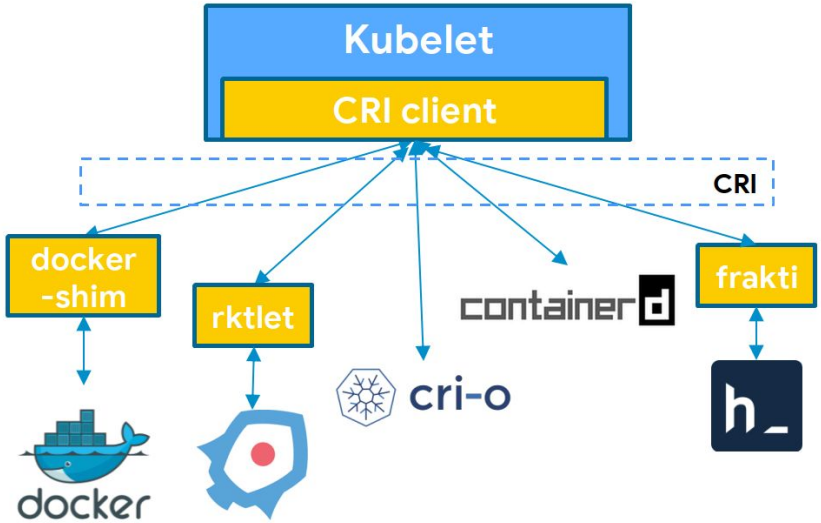
- A gRPC interface and a group of libraries
- Enables Kubernetes to use a wide variety of container runtimes
- Introduced in Kubernetes 1.5



CRI Runtimes

Container runtimes implemented CRI:

- **containerd**
- cri-o
- dockershim (upstream)
- frakti: Hyper, lightweight-VM container
- pouch: Alibaba, based on containerd
- rktlet: CoreOS, rkt container
- virtlet: OpenStack, real VM



Containerd CRI Plugin

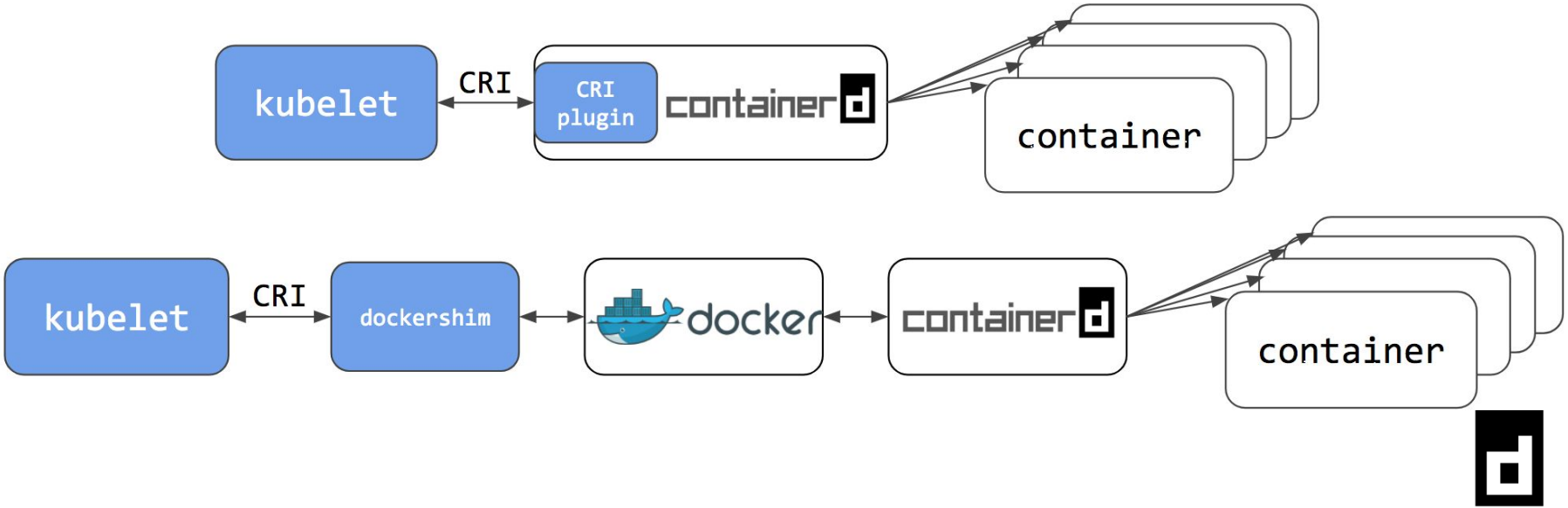
CRI Plugin

cri plugin: A containerd plugin implementation of CRI.

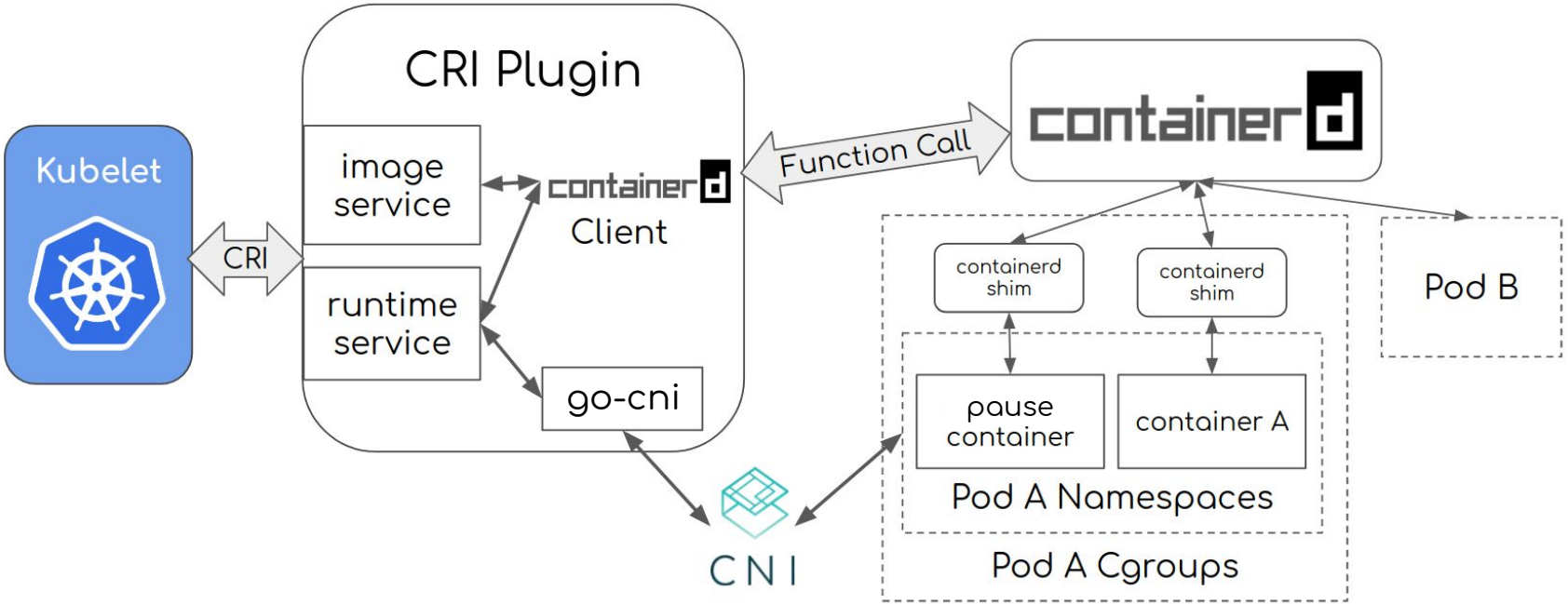
- https://github.com/containerd/cri
- Native plugin since containerd 1.1.
- GA since April 2018. (test dashboard)

CRI Plugin

cri plugin: A containerd plugin implementation of CRI.



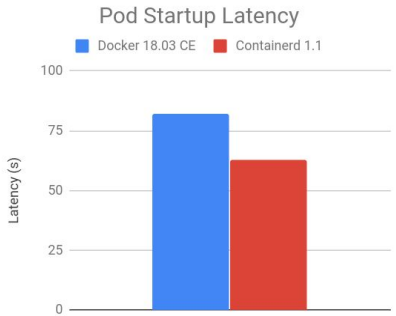
Architecture

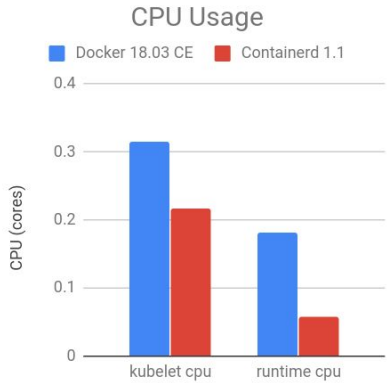


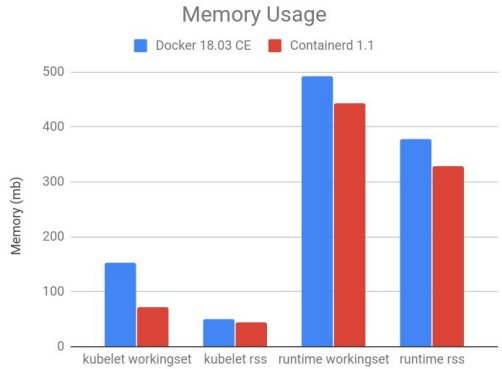
Performance

Dockershim (Docker CE 18.03) vs. CRI Plugin (Containerd 1.1):

- 105 pods batch startup benchmark
- 105 pods management overhead benchmark.



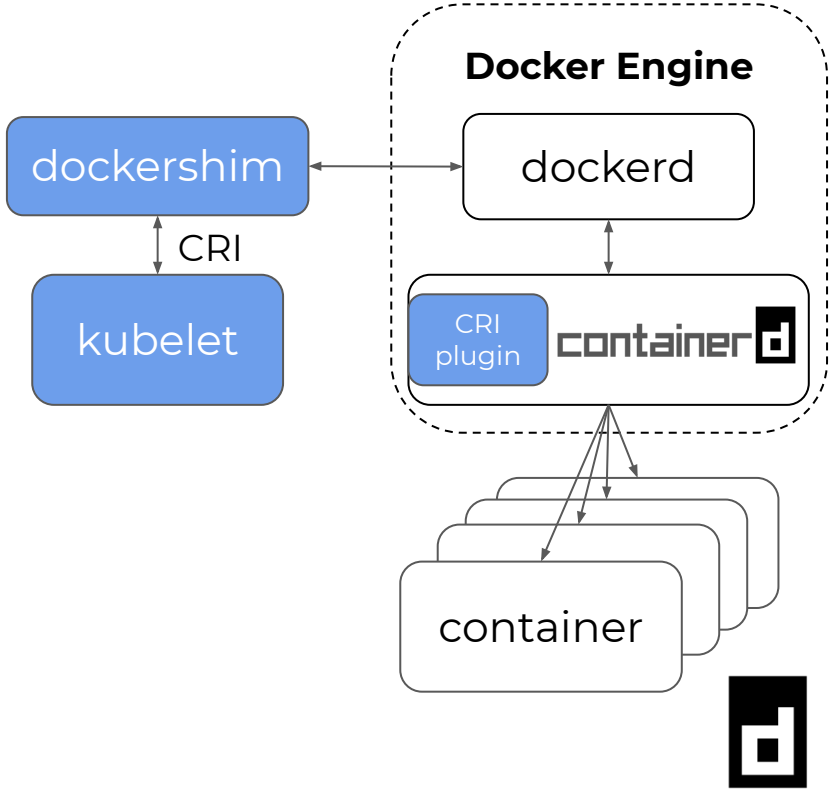




Dockershim → CRI Plugin

Containerd is part of Docker, just:

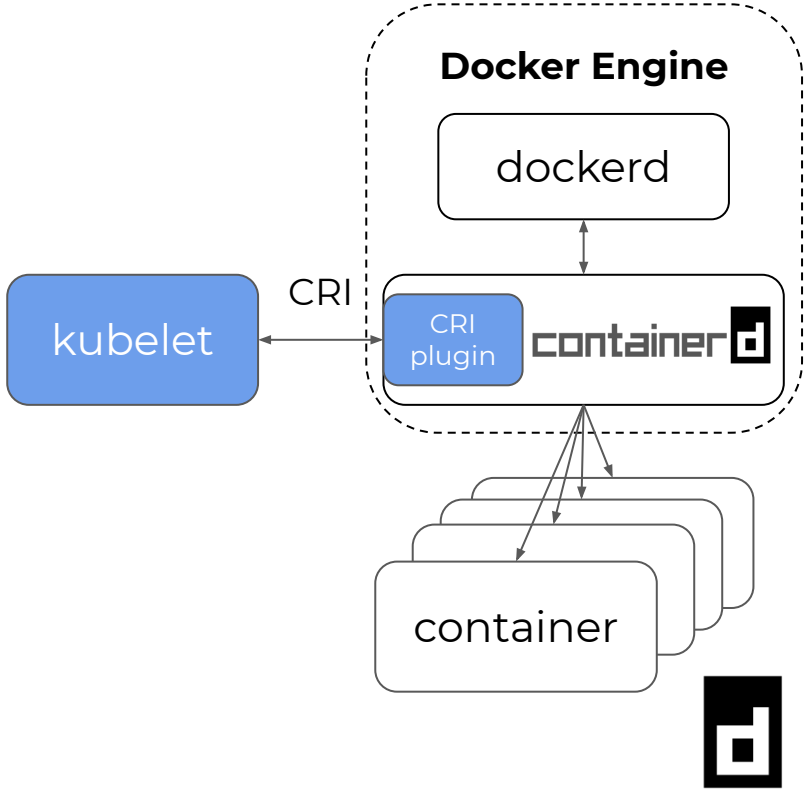
- Upgrade Docker CE to 18.09+ (containerd 1.2+)
- Set dockerd flag:
  - *--cri-containerd*
- Set kubelet flags:
  - *--container-runtime=remote*
  - *--container-runtime-endpoint=unix://*
    /run/containerd/containerd.sock
- Install a CNI plugin, e.g. calico, weaveworks



Dockershim → CRI Plugin

Containerd is part of Docker, just:

- Upgrade Docker CE to 18.09+ (containerd 1.2+)
- Set dockerd flag:
  - *--cri-containerd*
- Set kubelet flags:
  - *--container-runtime=remote*
  - *--container-runtime-endpoint=unix://*
    /run/containerd/containerd.sock
- Install a CNI plugin, e.g. calico, weaveworks



Containerd in GKE

GKE

GKE (Google Kubernetes Engine) is a hosted Kubernetes service provided by Google Cloud.

Containerd in GKE

Containerd in GKE Status:

- GKE 1.11: Beta
- GKE 1.12: Default on Master Nodes
- GKE 1.14: GA
- GKE 1.1x: Default on All Nodes

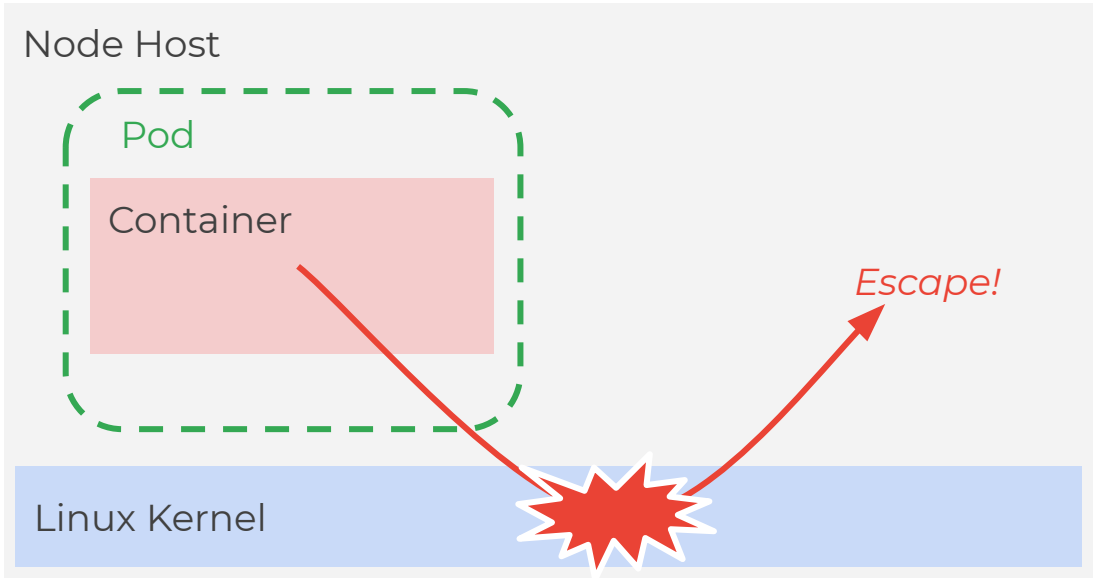
GKE Sandbox

Container Escape

Exploit **bugs** in the **Linux Kernel** via system surface like syscalls and /proc files to **bypass** container mechanisms or **elevate privilege**.

Examples:

- runC container escape CVE-2019-5736
- Dirty Cow CVE-2016-5195



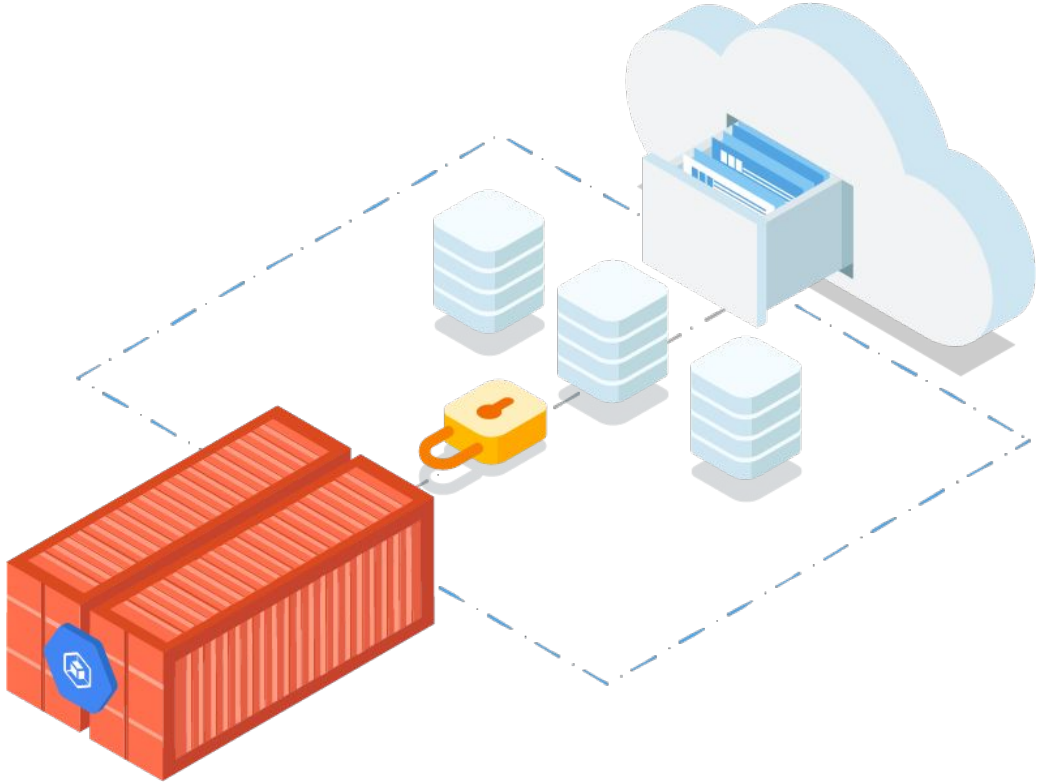
GKE Sandbox

Defense in depth to your pods

Second layer of defense between containerized workloads in GKE.

Defense-in-depth security principles **without application changes**.

Currently focused on **gVisor**, and **other sandbox technologies** can be supported in the future.



gVisor

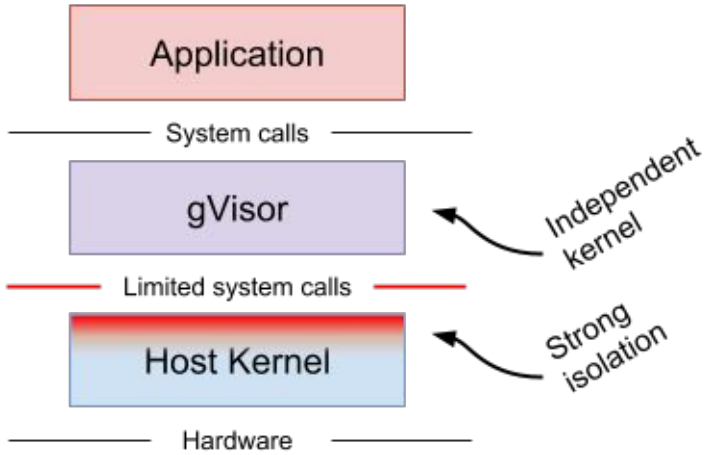
A sandbox technology based on user space kernel written in Go.

Open sourced by Google in May 2018.

OCI conformant: **runsc**

Repo: https://github.com/google/gvisor

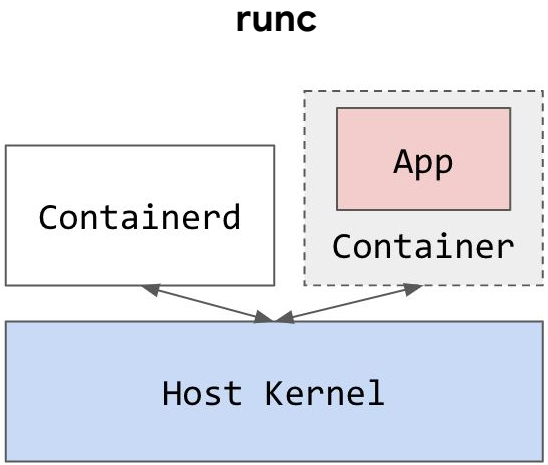
Website: https://gvisor.dev/

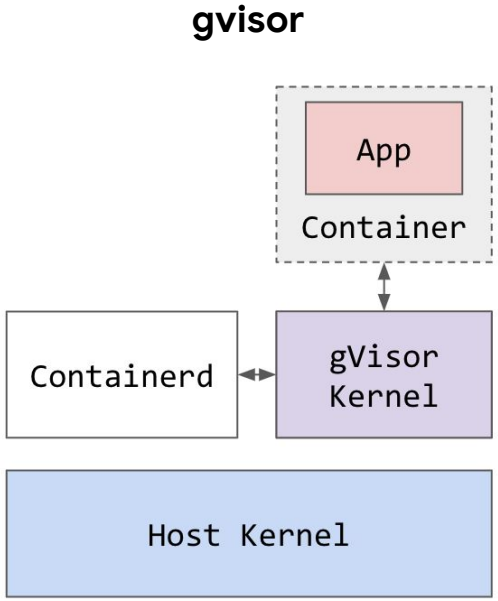


Support gVisor in Containerd

gVisor is different from runc.

- Signals
- Stats
- Terminal
- ...

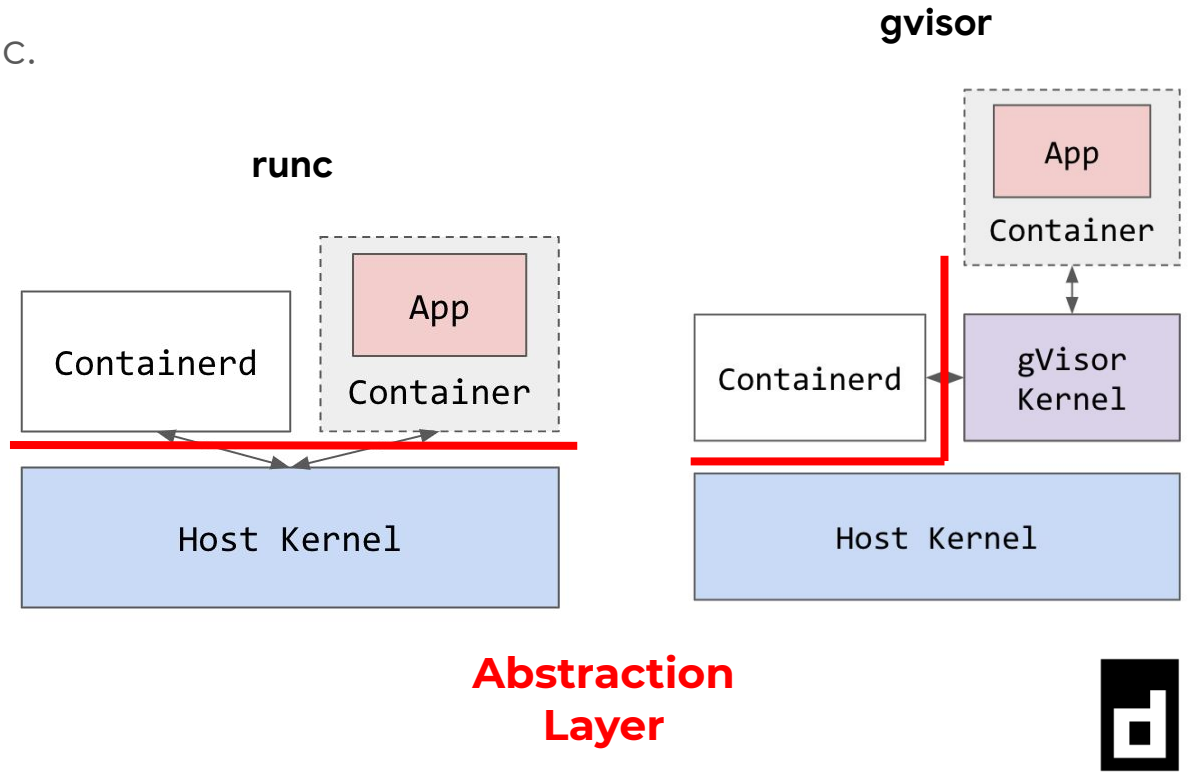




Support gVisor in Containerd

gVisor is different from runc.

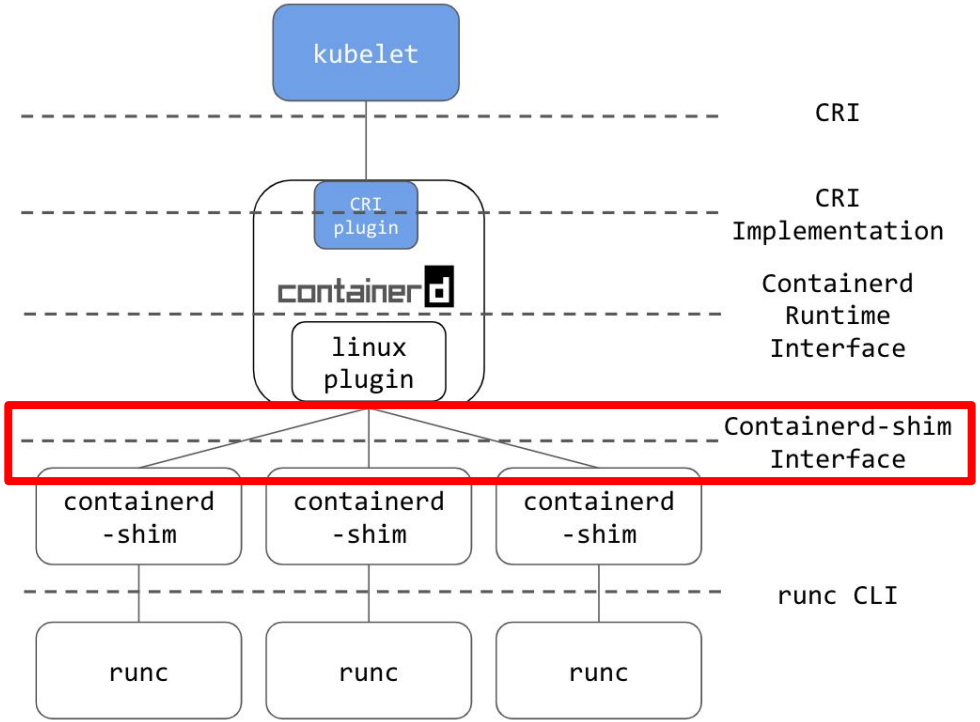
- Signals
- Stats
- Terminal
- ...



Support gVisor in Containerd

gVisor is different from runc.

- Signals
- Stats
- Terminal
- ...
- **The shim interface is just right!**



Containerd Shim 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);
}
```

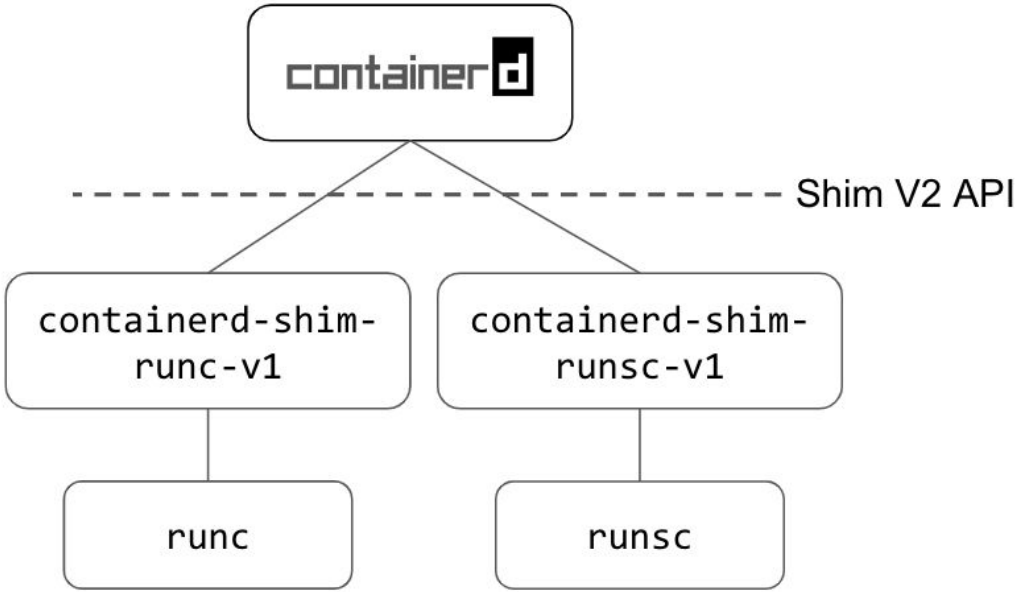
Containerd Shim V2

```
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);
}
```

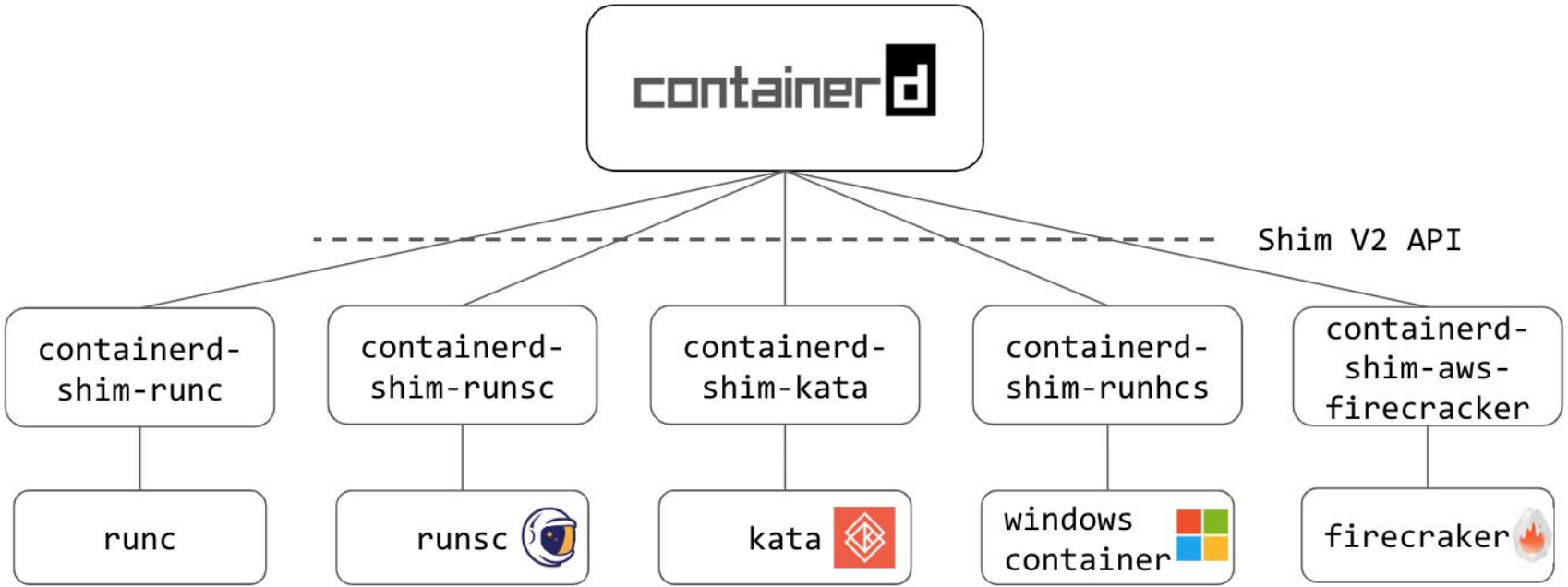
Containerd Shim for gVisor

containerd-shim-runsc-v1

- A shim v2 implementation for gvisor.
- Repo: https://github.com/google/gvisc-containerd-shim



Shim V2 is Now a Standard



Support gVisor in Kubernetes - RuntimeClass

Kubernetes RuntimeClass API

```
apiVersion:
node.k8s.io/v1beta1
kind: RuntimeClass
metadata:
  name: gvisor
spec:
  runtimeHandler: gvisor
  ...
```

Automatically created on GKE with GKE Sandbox

Support gVisor in Kubernetes - RuntimeClass

Kubernetes RuntimeClass API

Specify the **gvisor** runtime class name as part of your pod.

```
apiVersion:
node.k8s.io/v1beta1
kind: RuntimeClass
metadata:
  name: gvisor
spec:
  runtimeHandler: gvisor
  ...
```

Automatically created on GKE with GKE Sandbox

```
apiVersion: v1
kind: Pod
...
spec:
  ...
  runtimeClassName: gvisor
```

Support RuntimeClass in CRI

When creating the pod, Kubelet passes **gvisor** runtime handler to containerd through the new **runtime_handler** field in CRI

```
message RunPodSandboxRequest {
    PodSandboxConfig config = 1;
    string runtime_handler = 2;
}
```

Support RuntimeClass in Containerd

When creating the pod, Kubelet passes **gvisor** runtime handler to containerd through the new **runtime_handler** field in CRI

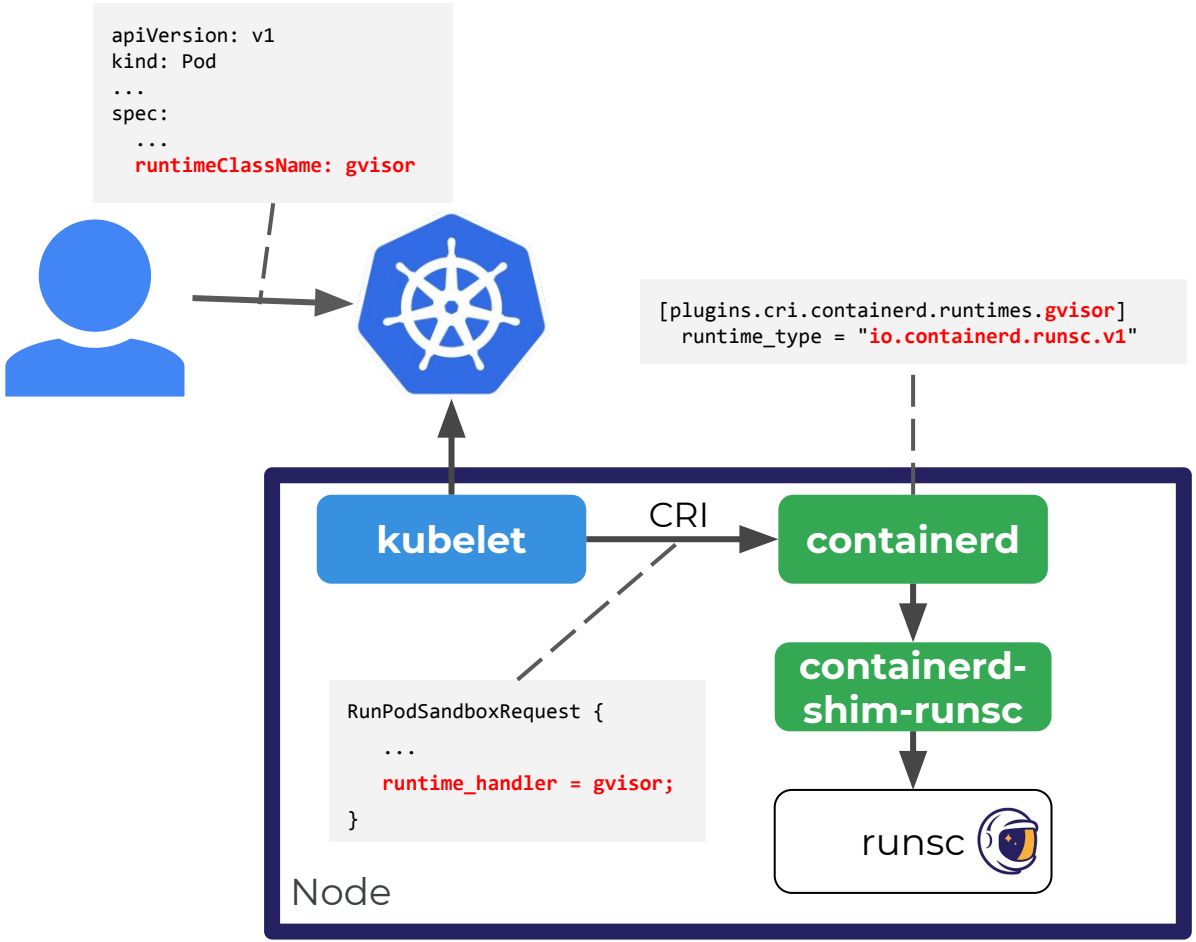
```
message RunPodSandboxRequest {
    PodSandboxConfig config = 1;
    string runtime_handler = 2;
}
```

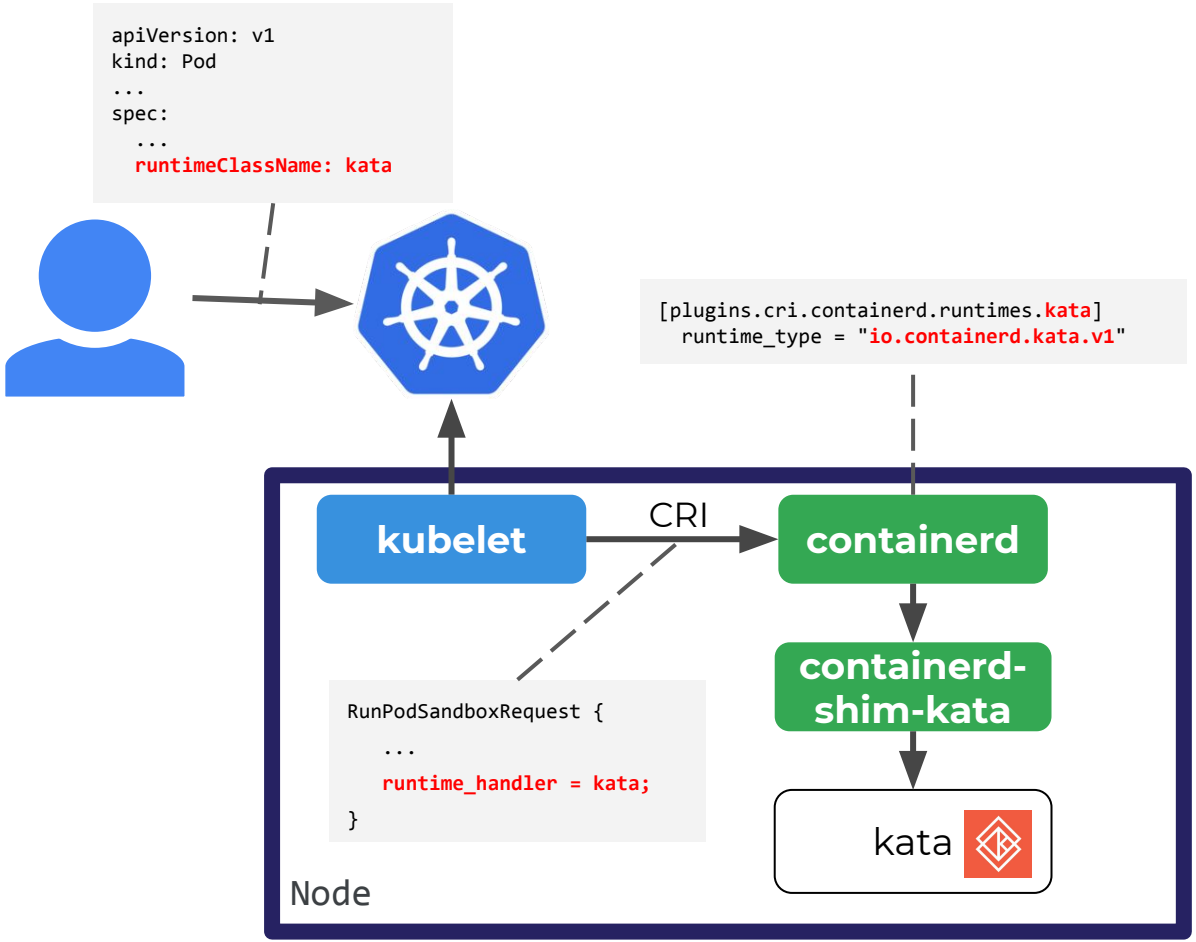
Configure **gvisor** runtime handler in containerd config /etc/containerd/config.toml

```
[plugins.cri.containerd.runtimes.runc]
  runtime_type = "io.containerd.runc.v1"

[plugins.cri.containerd.runtimes.gvisor]
  runtime_type = "io.containerd.runsc.v1"
```

Automatically configured on GKE with GKE Sandbox


apiVersion: v1
kind: Pod
...
spec:
...
runtimeClassName: gvisor
[plugins.cri.containerd.runtimes.gvisor]
runtime_type = "io.containerd.runsc.v1"
kubelet
CRI
containerd
containerd-shim-runsc
runsc
RunPodSandboxRequest {
...
runtime_handler = gvisor;
}
Node


apiVersion: v1
kind: Pod
...
spec:
...
runtimeClassName: kata
[plugins.cri.containerd.runtimes.kata]
runtime_type = "io.containerd.kata.v1"
kubelet
CRI
containerd
containerd-shim-kata
kata
RunPodSandboxRequest {
...
runtime_handler = kata;
}
Node

GKE Sandbox Status

Alpha: 2018/9

Beta: 2019/5

To try it out:

```
gcloud beta container node-pools create NODE_POOL_NAME
--cluster=CLUSTER_NAME --image-type=cos_containerd --sandbox
type=gvisor
```

Recap

Kubernetes Containerd Integration is ready for Production Use

GKE containerd support is Beta

GKE Sandbox is Beta built on Kubernetes + Containerd + gVisor

Containerd is super powerful!

Thank You