containerd intro

kubecon 2019

History of containerd

containerd

containerd early days (early 2016)

- Interfaced with runC
- Provided gRPC API
- Separated container lifecycle from engine lifecycle
- Integrated in Docker 1.11

Container Runtime Interface (late 2016)

- Defines what is Kubernetes Runtime
- containerd scope increased to match CRI requirement, including image

Roadmap to containerd 1.0 (late 2016)

- Runtime already solid, stabilize API
- Create ***Snapshot*** interface
- Build distribution around ***Content Store***

containerd joins CNCF (early 2017)

- CRI implementation started
- New plugin architecture
- Focus on stability and full OCI image support

containerd 1.0 GA (late 2017)

- Released December 2017
- API stabilized and supported
- CRI implementation goes alpha

containerd 1.1 (early 2018)

- Released April 2018
- CRI implementation goes beta
- CRI included as built-in plugin

containerd 1.2 (late 2018)

- Released October 2018
- Runtime shim stabilized

CONGRATS

containerd

CLASS OF 2019

Love,
CNCF

containerd status

containerd matures

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

IBM Cloud

- 75% of production IKS clusters are running containerd
- IBM Cloud Functions running containerd in production

Google Cloud

- containerd 1.1 & 1.2 used in production by GKE customers
- GKE Sandbox using containerd + gVisor

gVisor

Alibaba Group

- containerd in production since containerd 1.0
- 100K+ containers running on containerd 1.2
- Running PouchContainer with fully integrated with containerd

PouchContainer

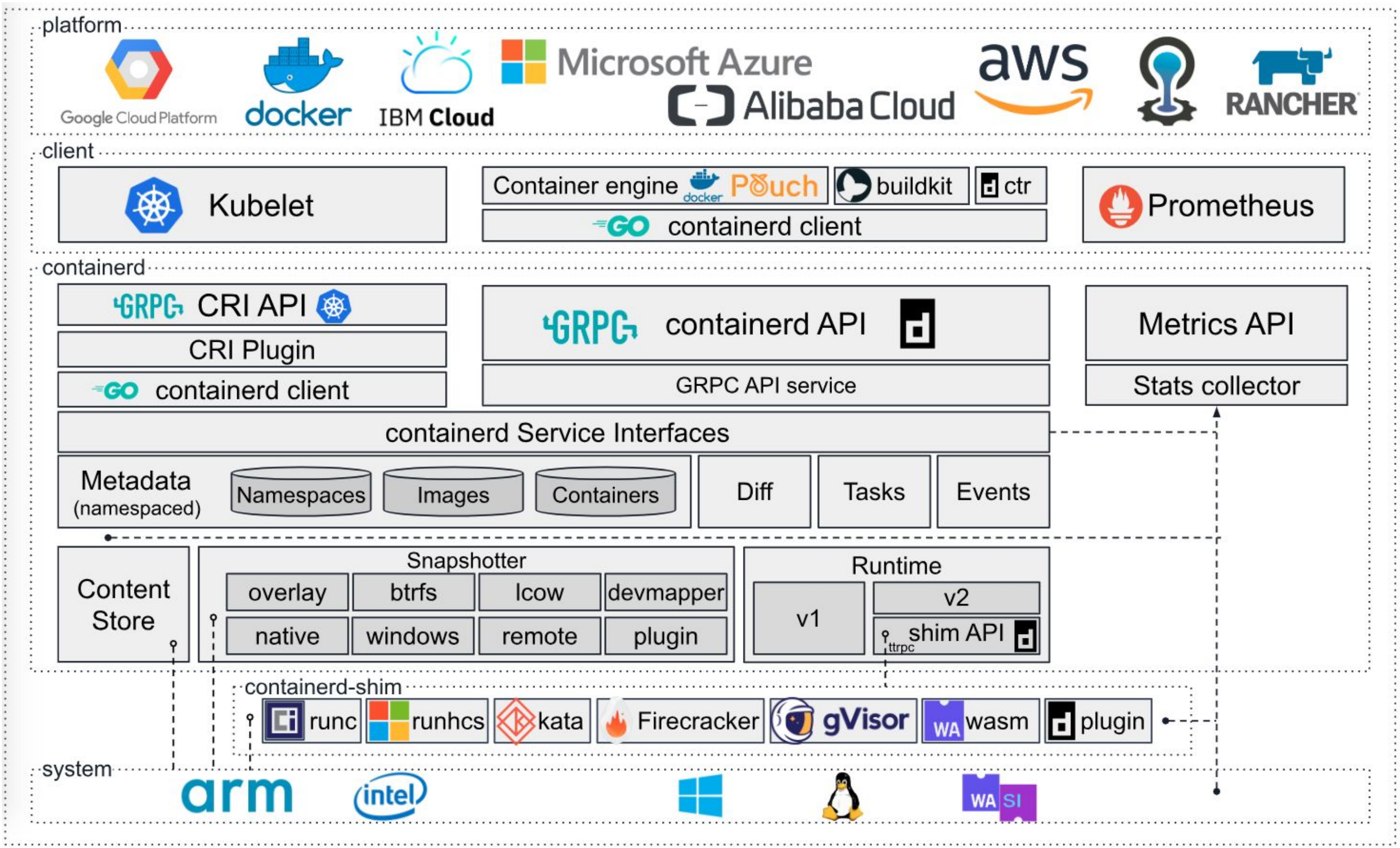
aws

- Contributed devicemapper snapshotter
- Firecracker + containerd in development with working prototype

Firecracker

- containerd 1.0 used since 17.12
- New releases of Docker uses latest containerd release

Architecture Overview



platform

Google Cloud Platform

docker

IBM Cloud

Microsoft Azure

Alibaba Cloud

aws

RANCHER

client

Kubelet

Container engine docker PØuch

buildkit

ctr

GO containerd client

Prometheus

containerd

GRPC CRI API

CRI Plugin

GO 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

wasm

plugin

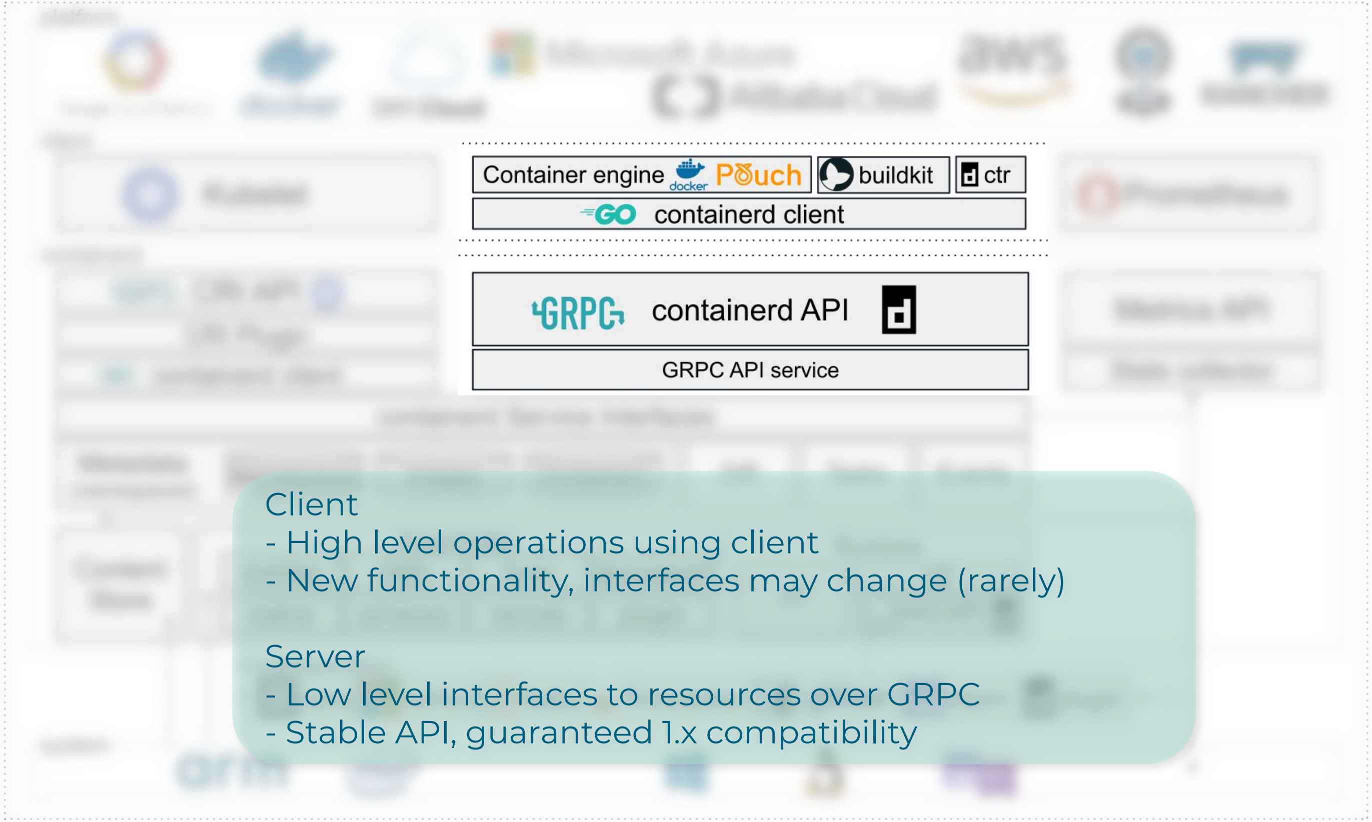
system

arm

intel

WA SI

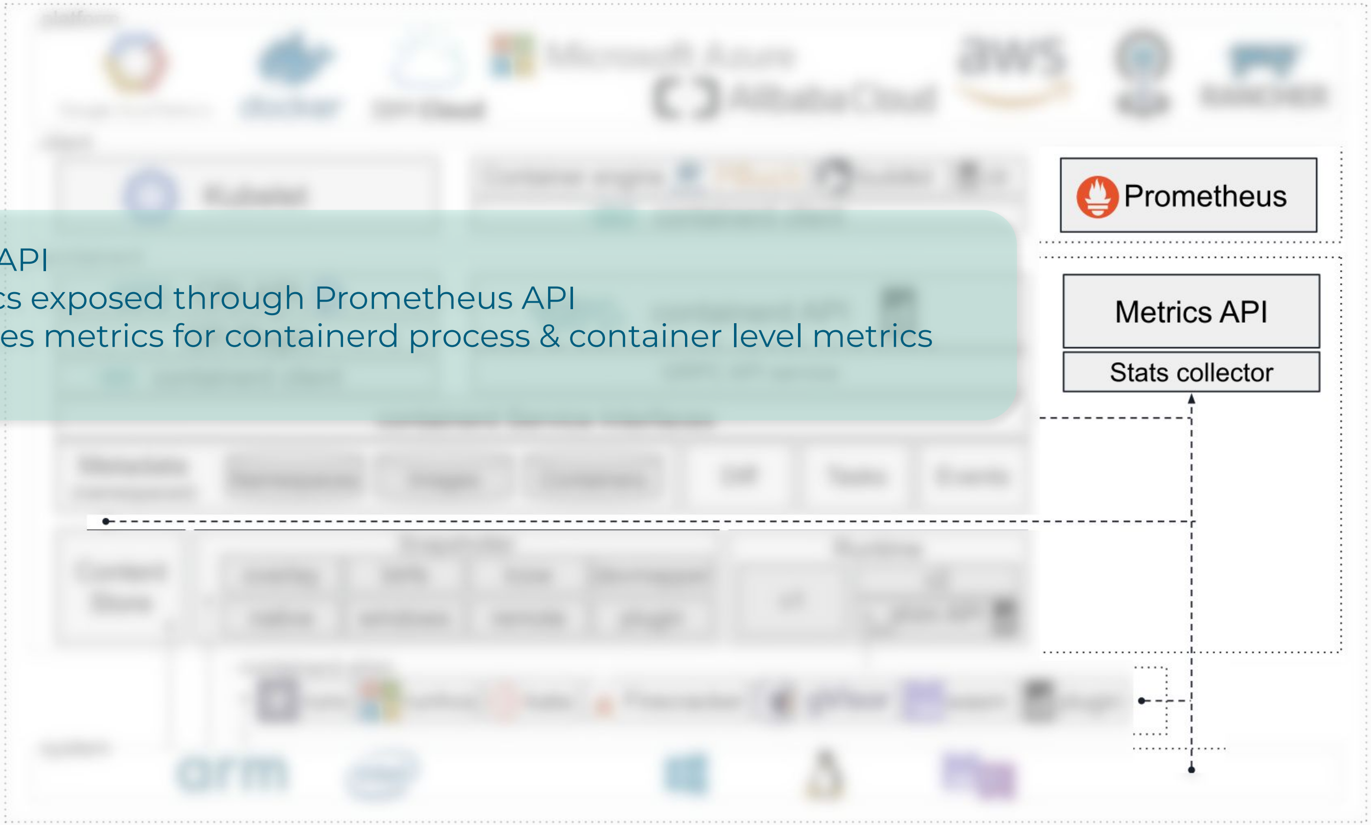
Client-Server Design



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

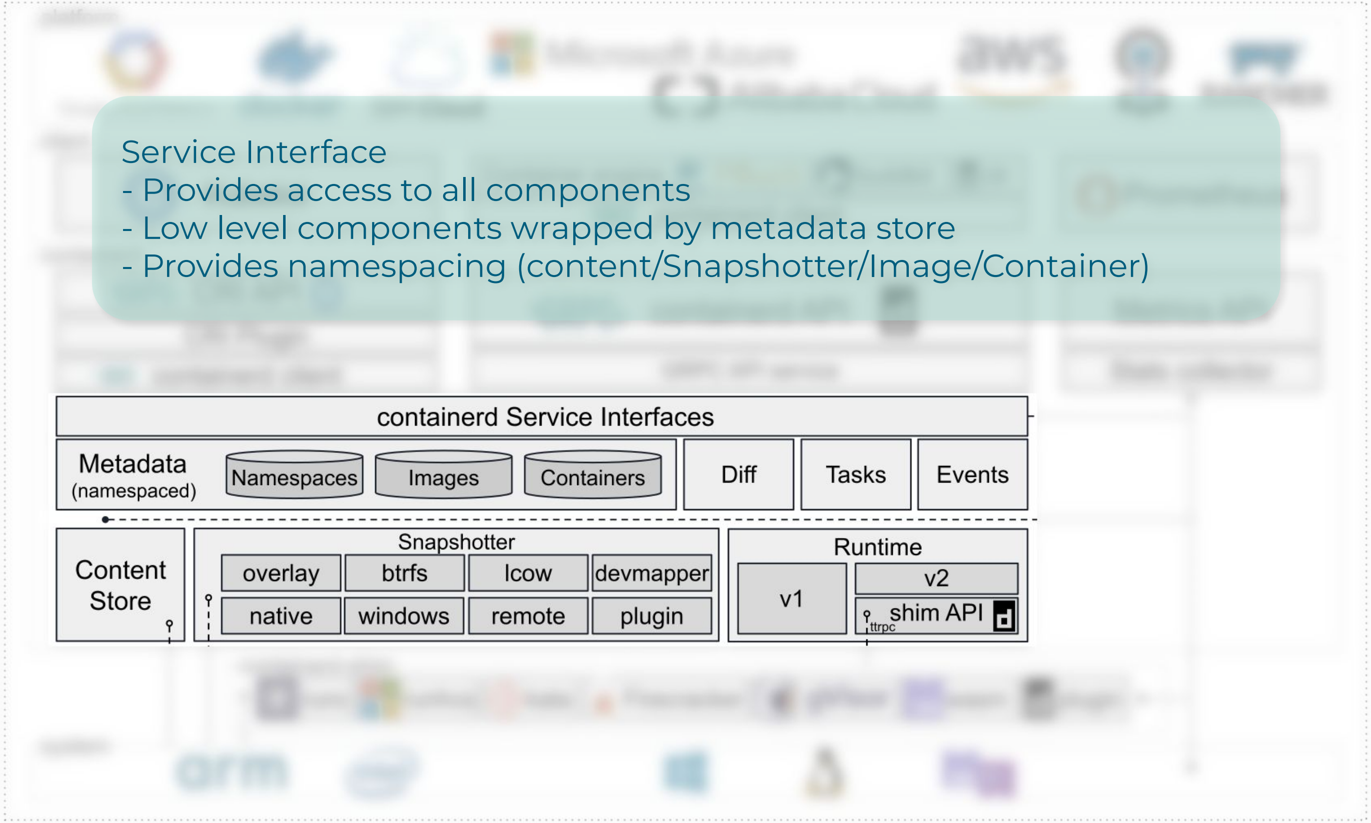
Server
- Low level interfaces to resources over GRPC
- Stable API, guaranteed 1.x compatibility

Metrics



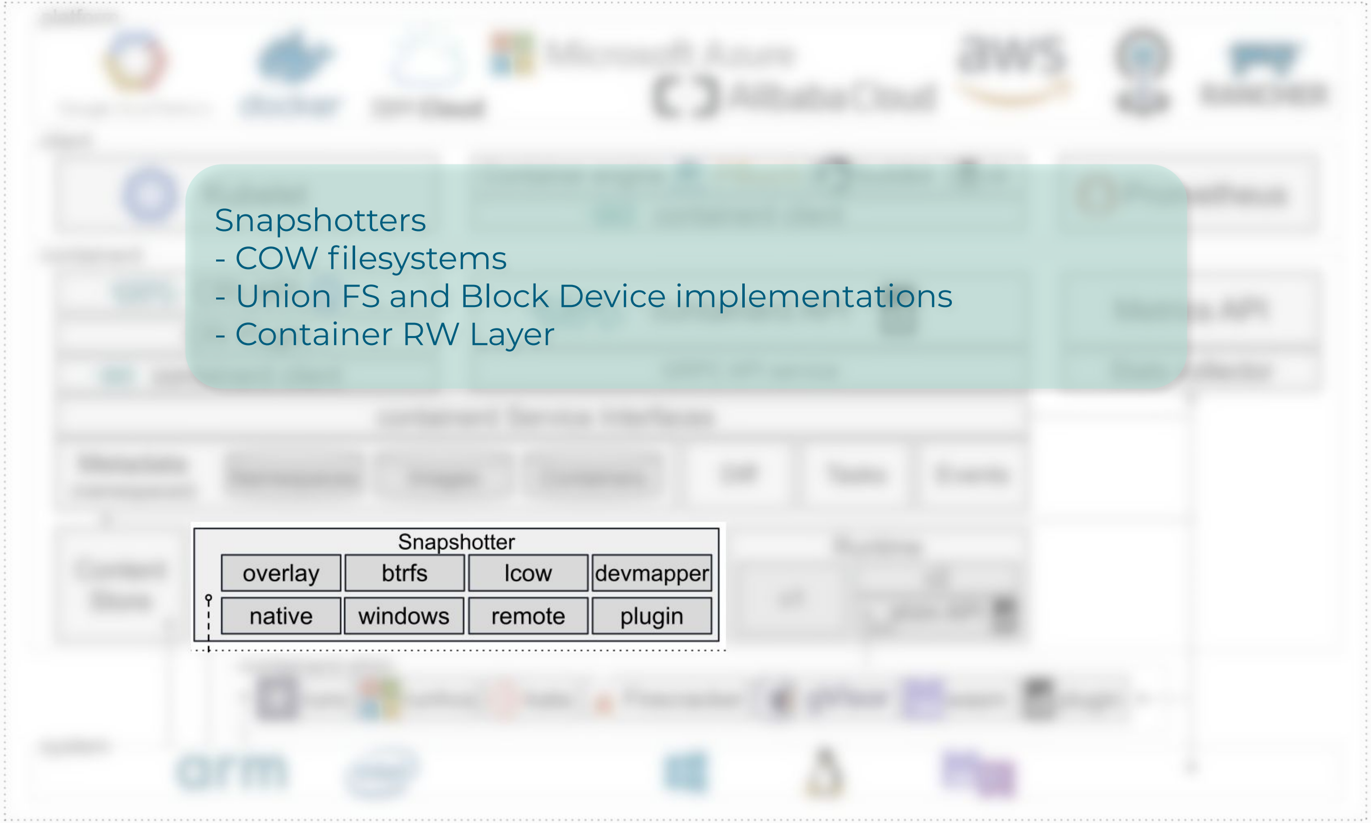
Metric API
- Metrics exposed through Prometheus API
- Exposes metrics for containerd process & container level metrics

Backend



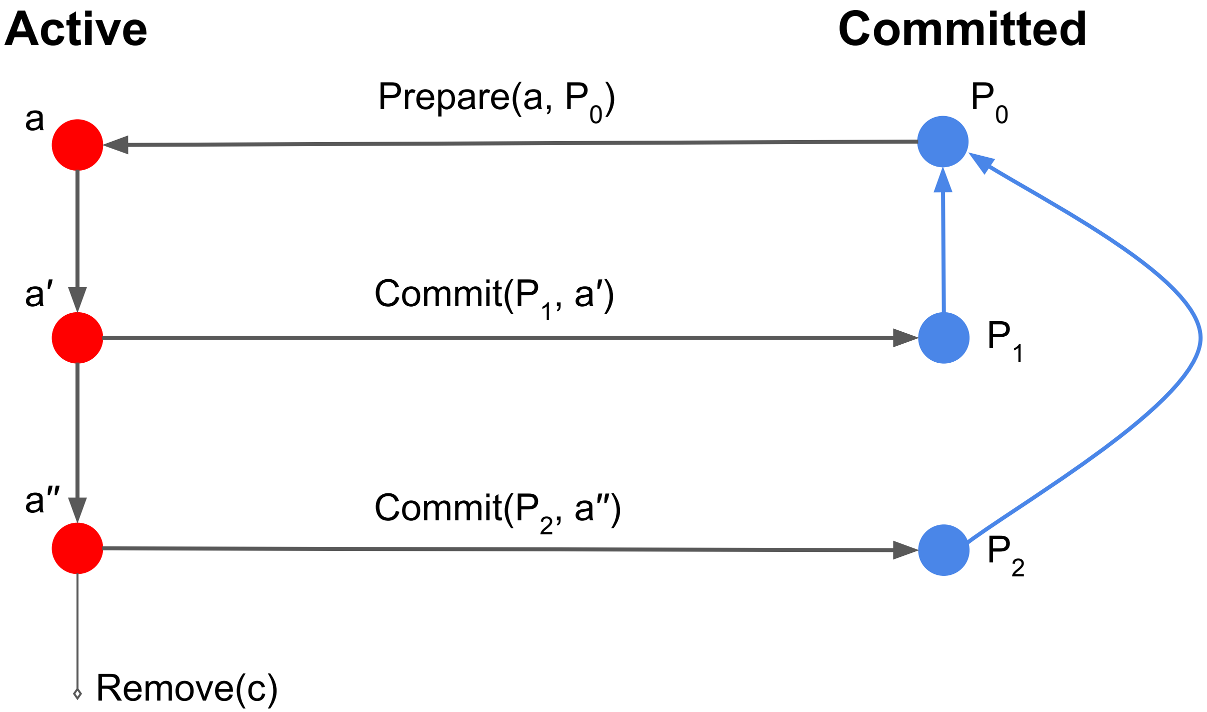
Service Interface
- Provides access to all components
- Low level components wrapped by metadata store
- Provides namespacing (content/Snapshotter/Image/Container)

Snapshotter

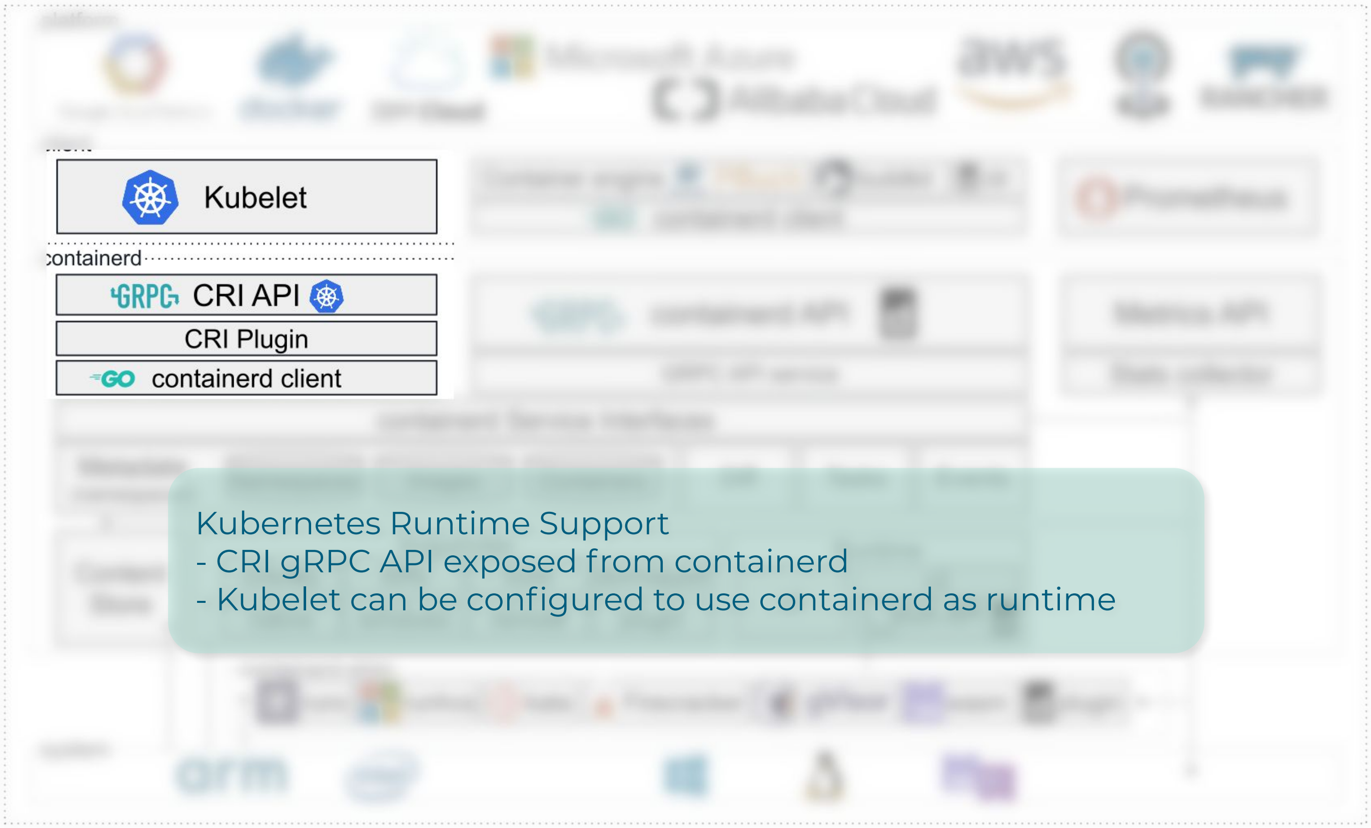


Snapshotters
- COW filesystems
- Union FS and Block Device implementations
- Container RW Layer

Snapshotter



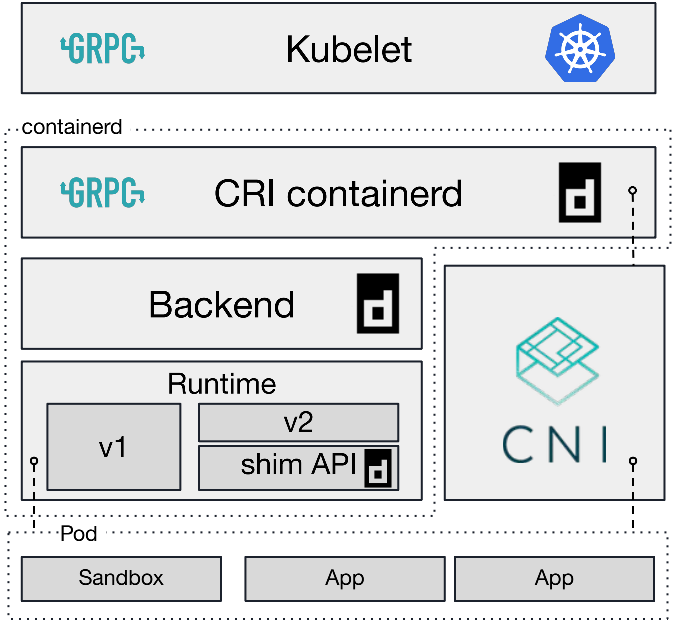
Kubernetes Runtime Support



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

Kubernetes Runtime Support

- Fully support CNI Plugins
- Network handled by CNI
- Pod can share one shim



Summary

- Support for OCI runtime and image specifications
- Stable gRPC interface
- Kubernetes Runtime Support

Demo time

Command line

- ctr
  - Development tool ships with containerd
  - Lower level commands (directly managing snapshots, images, containers)

- crictl
  - CLI for any CRI runtime, more stable (commands less likely to change)
  - Higher level operations (pull, run, pod management)

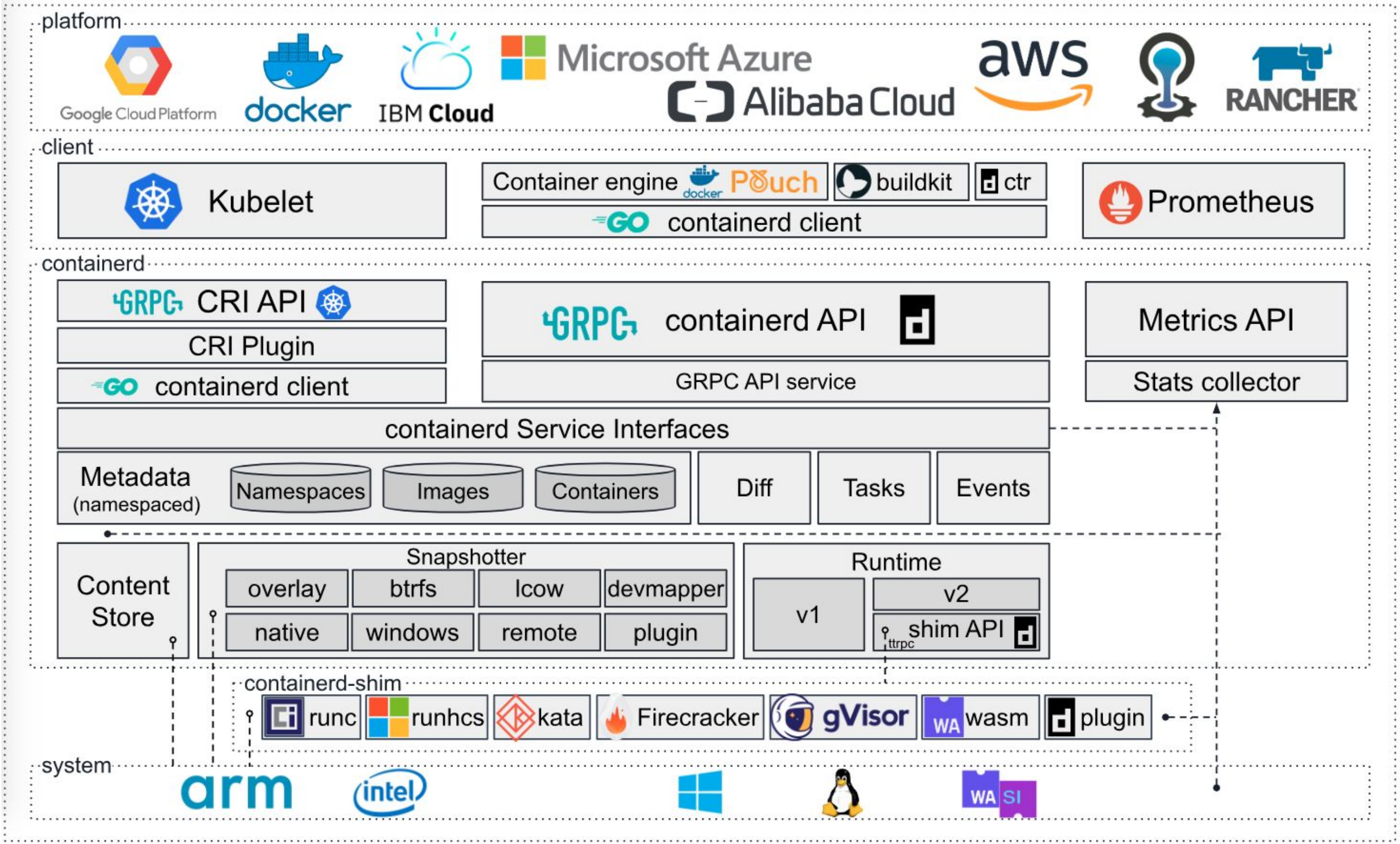
Thank You
Gracias

containerd

Deep dive into containerd

kubecon 2019

Architecture - Recap



platform

Google Cloud Platform

docker

IBM Cloud

Microsoft Azure

Alibaba Cloud

aws

RANCHER

client

Kubelet

Container engine docker PØuch

buildkit

ctr

GO containerd client

Prometheus

containerd

GRPC CRI API

CRI Plugin

GO 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

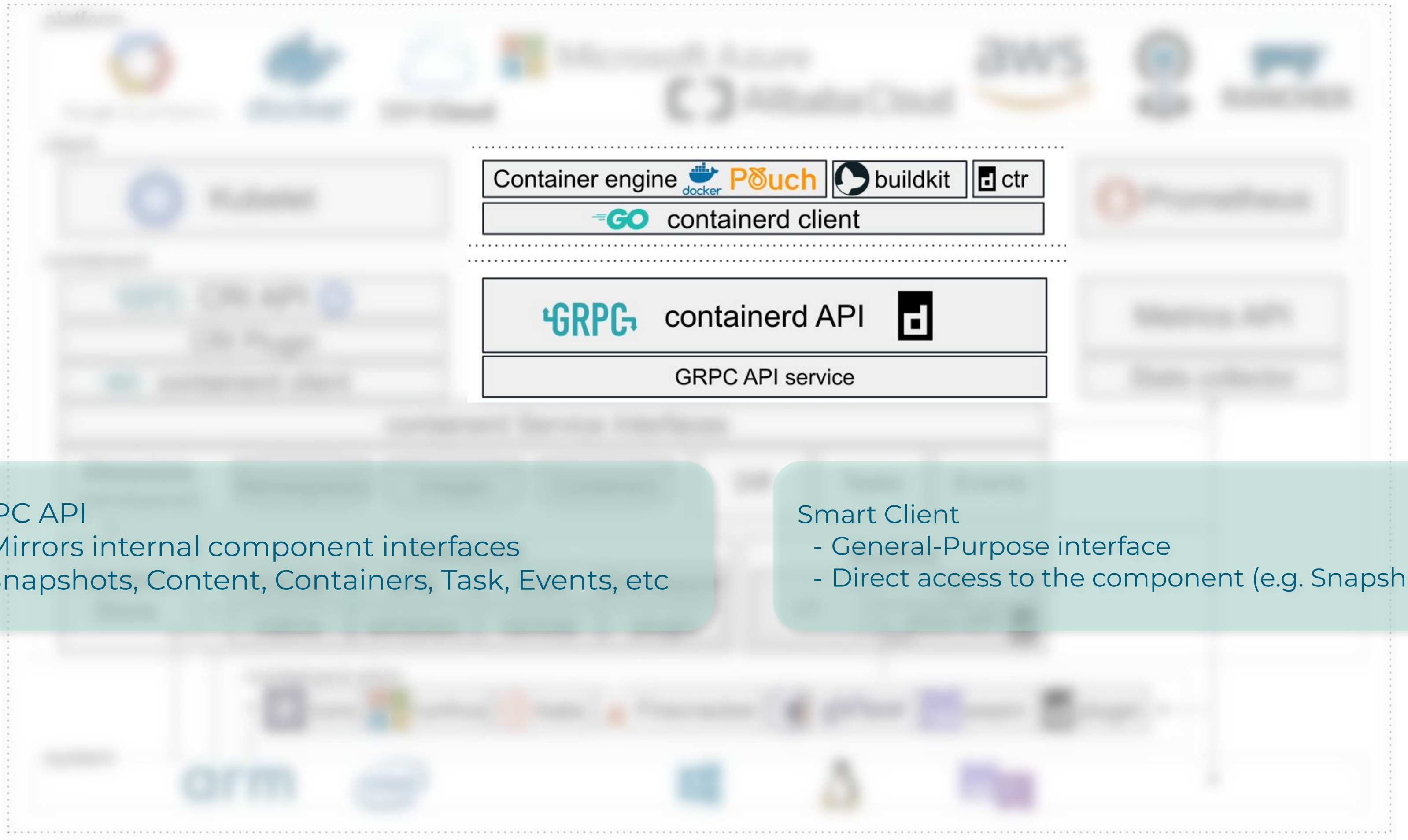
arm

intel

WA SI

Smart Client Model

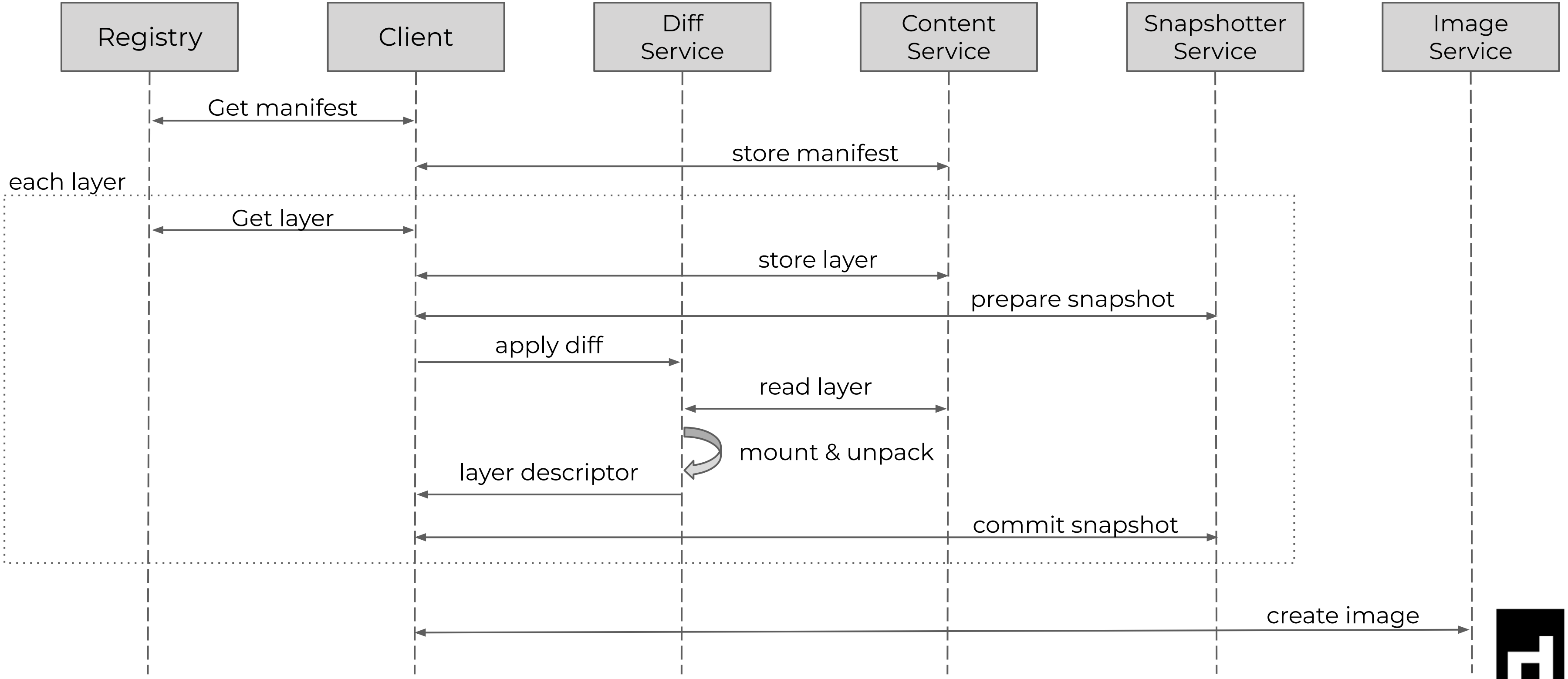
containerd



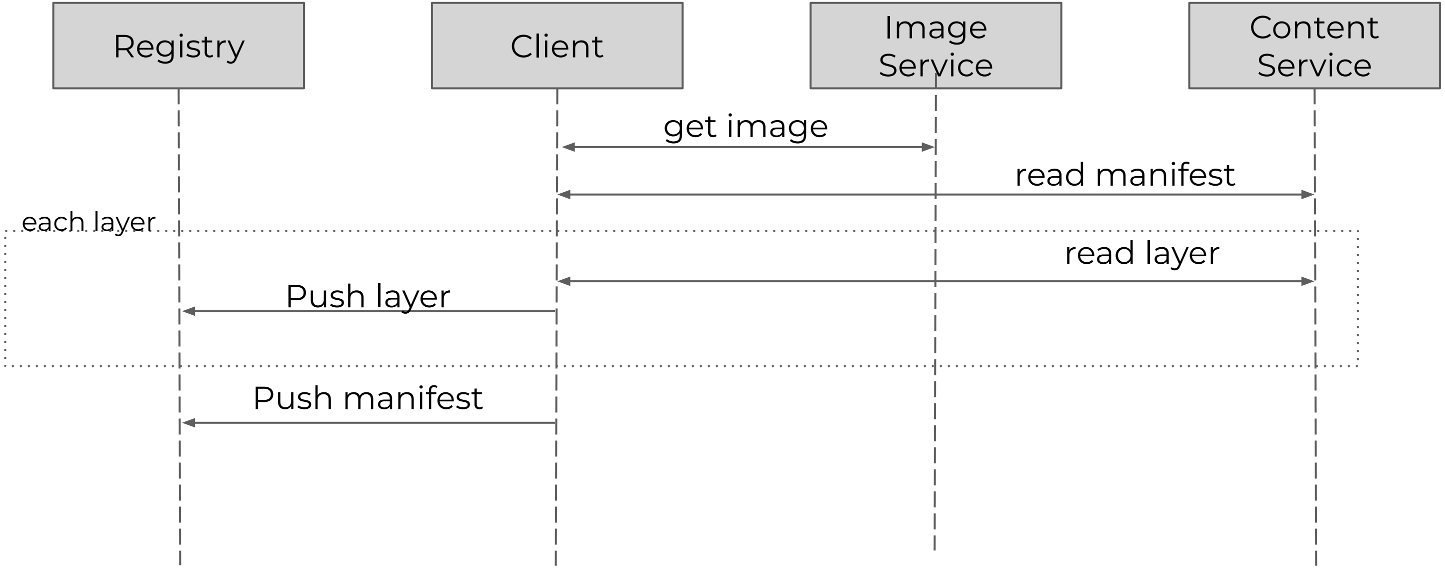
gRPC API
- Mirrors internal component interfaces
- Snapshots, Content, Containers, Task, Events, etc

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

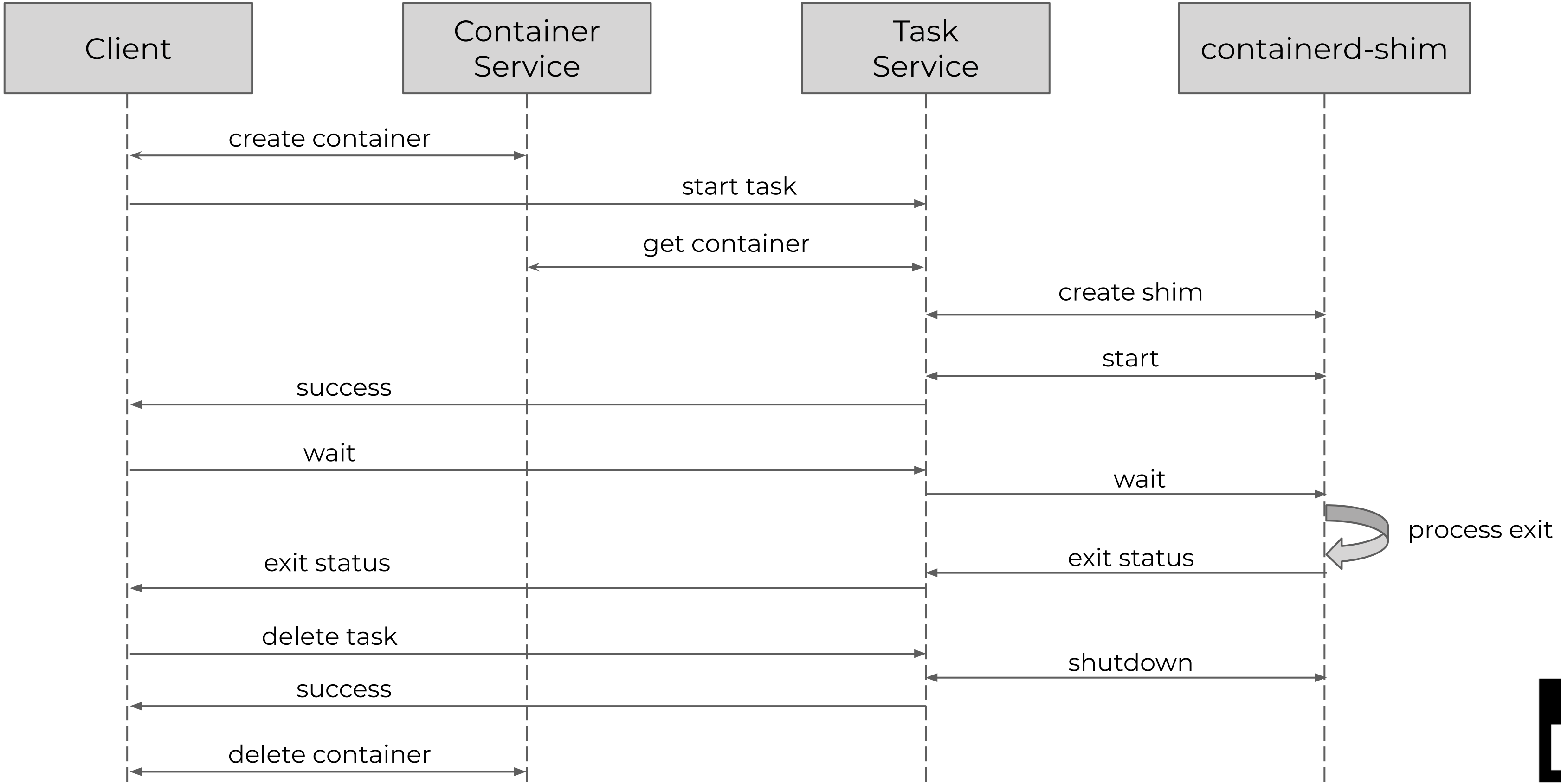
Pull Image


Registry
Client
Diff Service
Content Service
Snapshotter Service
Image Service
Get manifest
store manifest
each layer
Get layer
store layer
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

Run Container



Client Extensibility

- Override services with service options
- Customize push and pull with remote options

```
type ServicesOpt
        func WithContainerService(containerService containersapi.ContainersClient) ServicesOpt
        func WithContentStore(contentStore content.Store) ServicesOpt
        func WithDiffService(diffService diff.DiffClient) ServicesOpt
        func WithEventService(eventService EventService) ServicesOpt
        func WithImageService(imageService imagesapi.ImagesClient) ServicesOpt
        func WithLeasesService(leasesService leases.Manager) ServicesOpt
        func WithNamespaceService(namespaceService namespacesapi.NamespacesClient) ServicesOpt
        func WithSnapshotters(snapshotters map[string]snapshots.Snapshotter) ServicesOpt
        func WithTaskService(taskService tasks.TasksClient) ServicesOpt
```

```
type RemoteOpt
        func WithImageHandler(h images.Handler) RemoteOpt
        func WithImageHandlerWrapper(w func(images.Handler) images.Handler) RemoteOpt
        func WithResolver(resolver remotes.Resolver) RemoteOpt
```

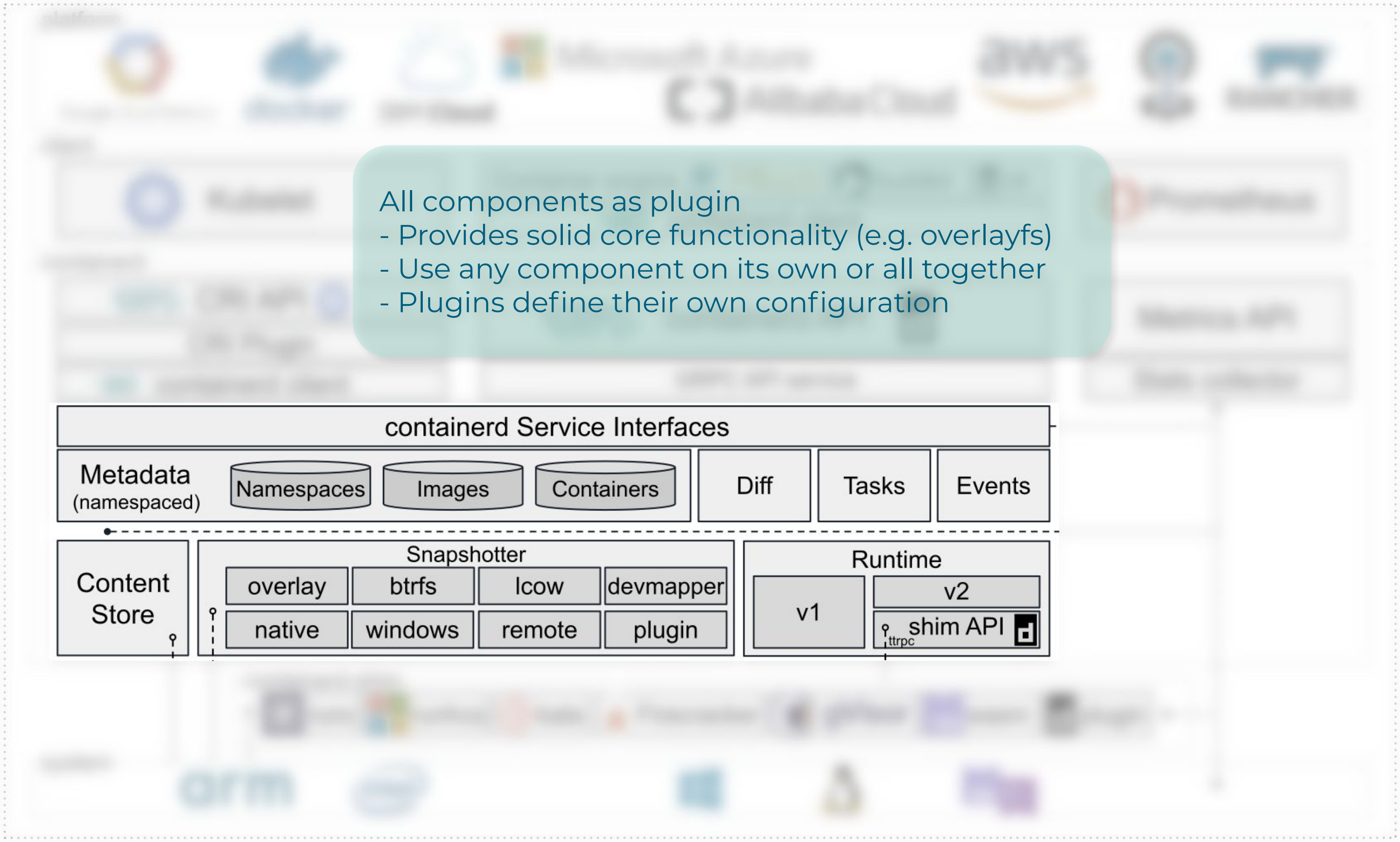
Aimed to

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

Component as Plugin

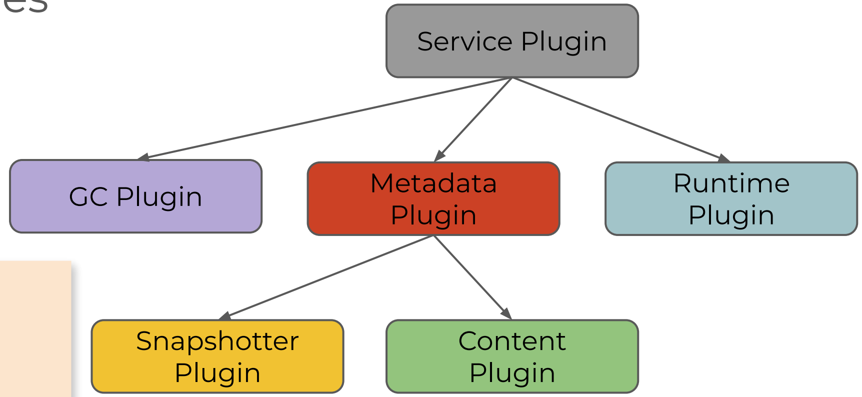
All components as plugin

- Provides solid core functionality (e.g. overlayfs)
- Use any component on its own or all together
- Plugins define their own configuration

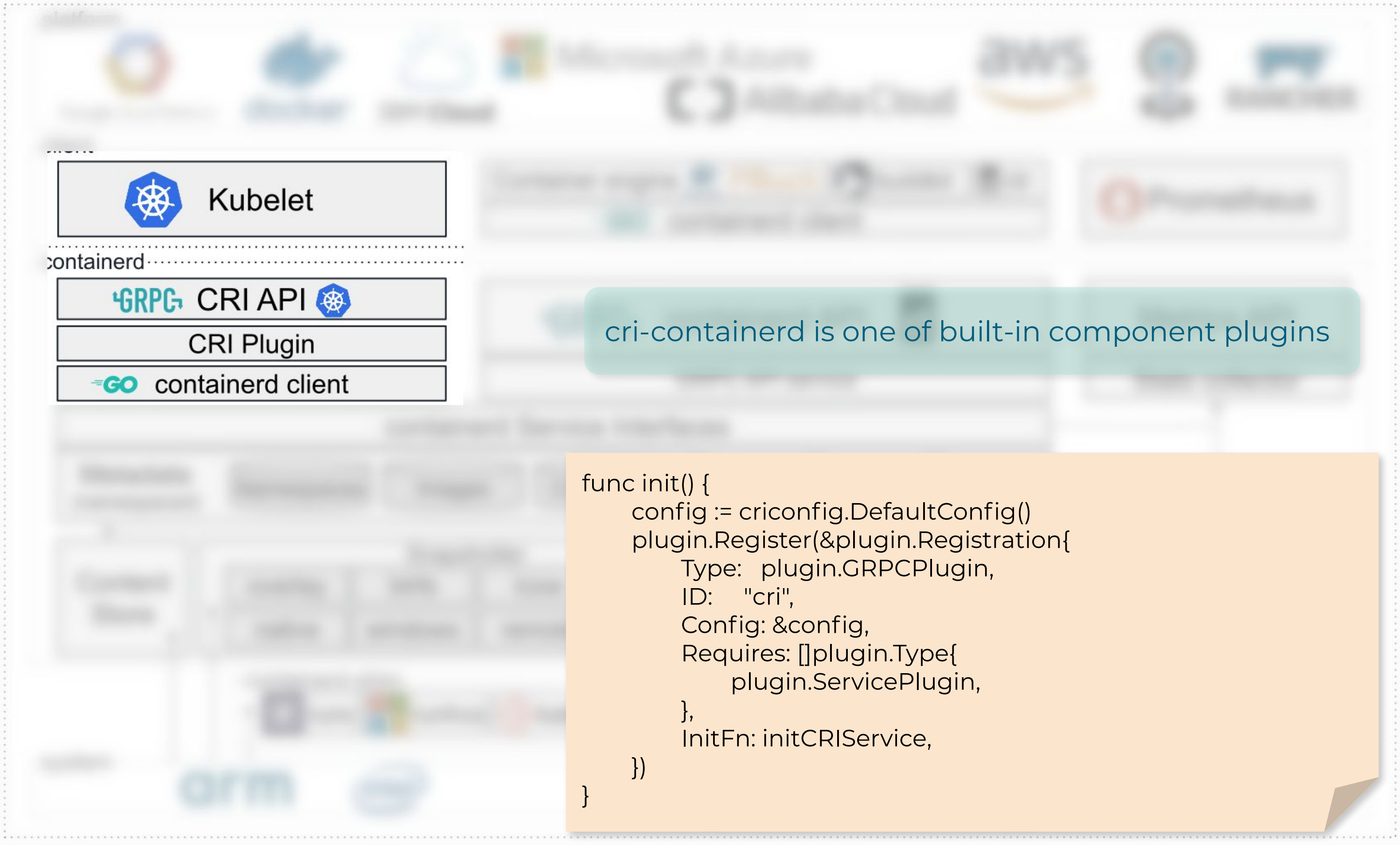


Plugin Registration

- Loose coupling and clear boundaries
- Dependency Graph
- Isolated bootstrap



```
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) {
        },
}
```



cri-containerd is one of built-in component plugins

```
func init() {
        config := criconfig.DefaultConfig()
        plugin.Register(&plugin.Registration{
                Type:   plugin.GRPCPlugin,
                ID:     "cri",
                Config: &config,
                Requires: []plugin.Type{
                        plugin.ServicePlugin,
                },
                InitFn: initCRIService,
        })
}
```

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 zfs/aufs

```
// https://github.com/AkihiroSuda/containerd-example-custom-daemon
package main

import (
	"github.com/containerd/containerd/cmd/containerd/app"
	_ "github.com/containerd/containerd/cmd/containerd/builtins"

	// custom plugins: aufs, zfs
	_ "github.com/containerd/aufs"
	_ "github.com/containerd/zfs"
)

func main() {
	app.Main()
}
```

External Plugins

Extend without recompiling containerd...

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

Proxy Plugin on gRPC

Support Proxy

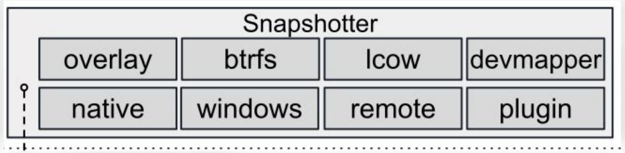
- Create remote plugin as proxy
- Configure it for containerd

```
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);
}
```



Snapshotter

overlay btrfs lcow devmapper

native windows remote plugin

Remote Snapshotter
- implement Snapshotter gRPC API
- containerd as proxy

Remote snapshotter service

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

```
[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 API

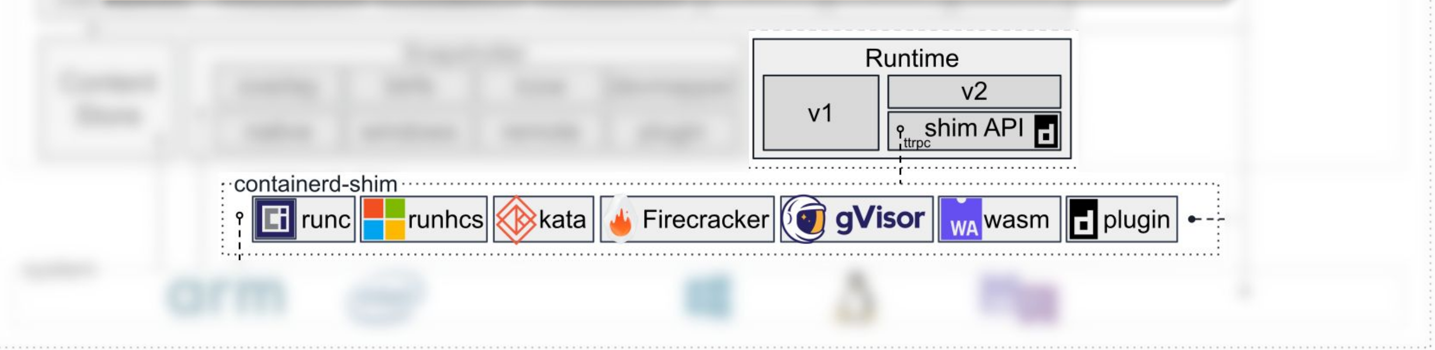
Why external runtime plugins?

- 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 common API

- Minimal and scoped to the execution lifecycle of a container
- Binary naming system
  - 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);
}
```



Runtime Plugin Demo

cri-containerd + gVisor

Demo - integrate with gVisor runtime

cri-containerd + Firecracker

Demo - integrate with Firecracker runtime

containerd v1.3 is coming…

Coming up in containerd

- Growth of plugin ecosystem
- Better support for cluster resources
- Supported CLI
- New ideas around images (encrypted, non-layered)

Thank You
Gracias