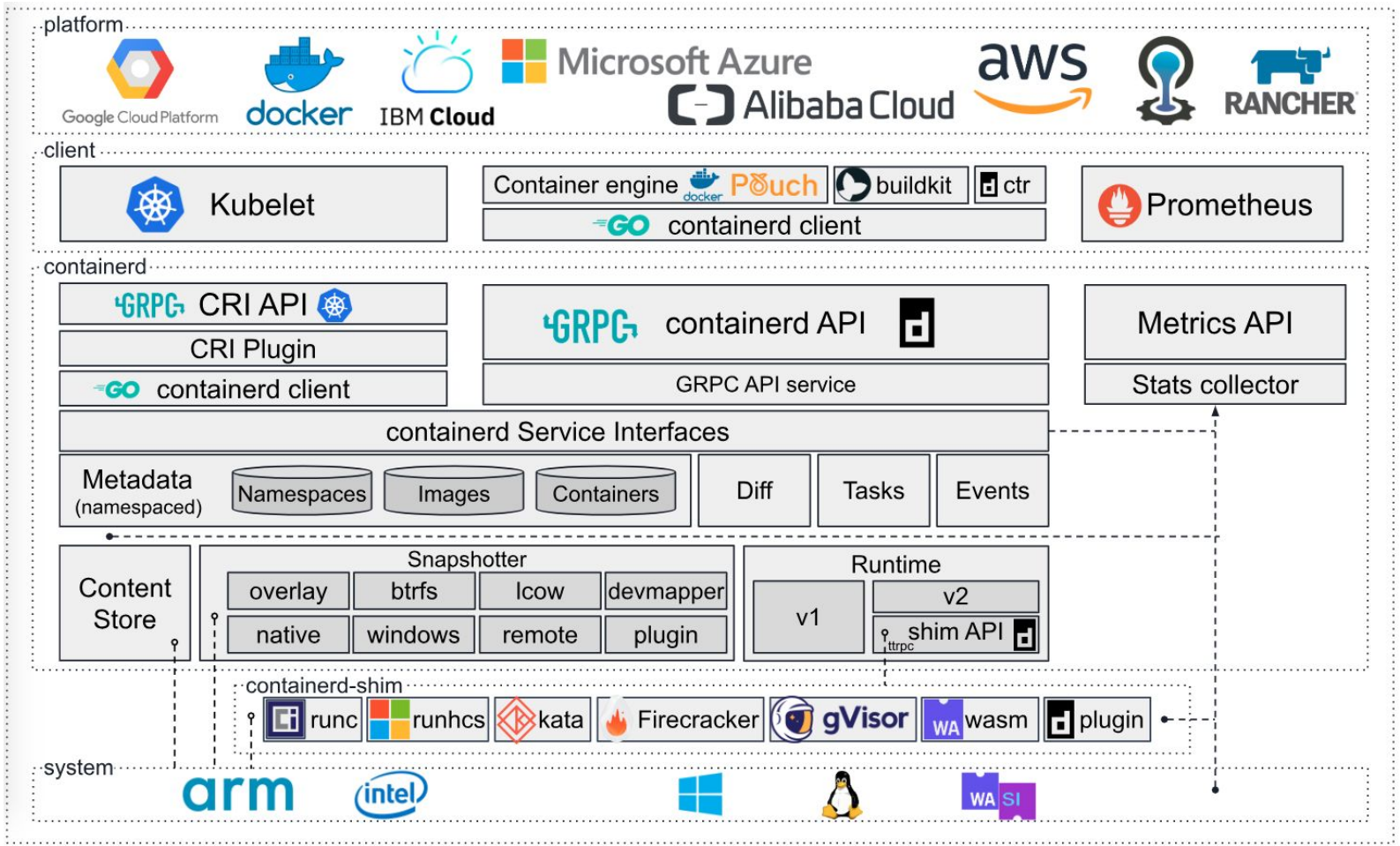
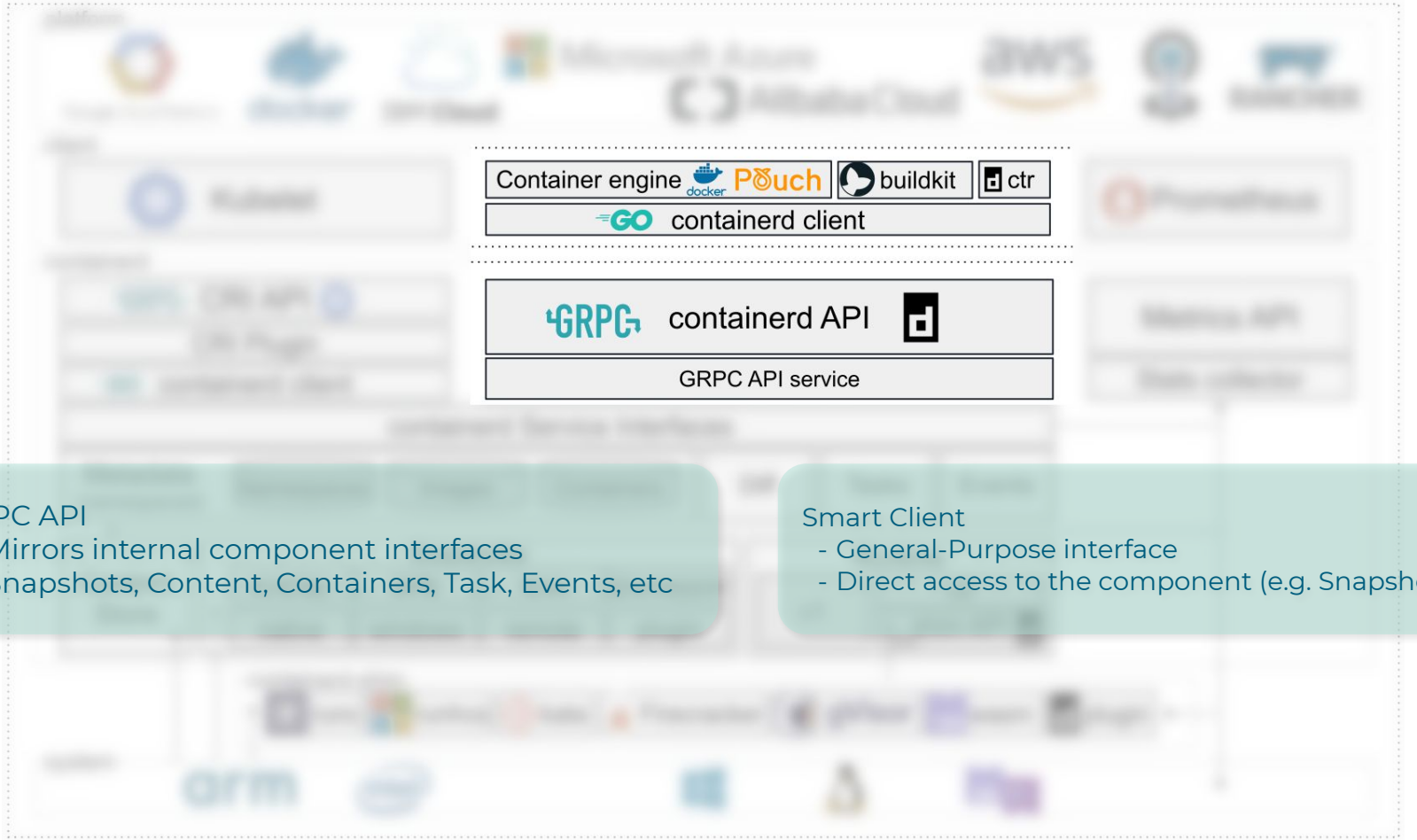
Deep dive into containerd

Lantao Liu (Google)
Wei Fu (Alibaba Cloud)

Architecture - Recap


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

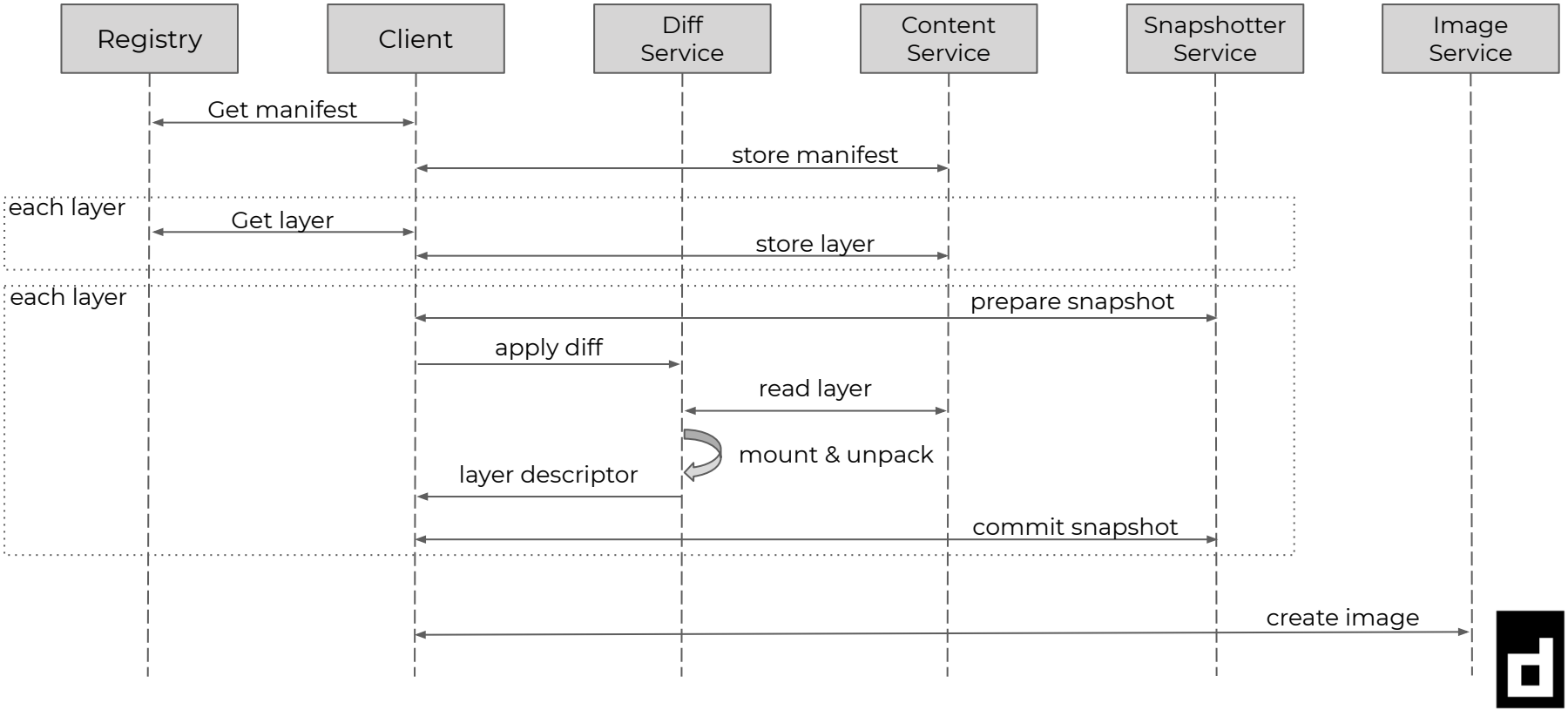
Smart Client Model



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)

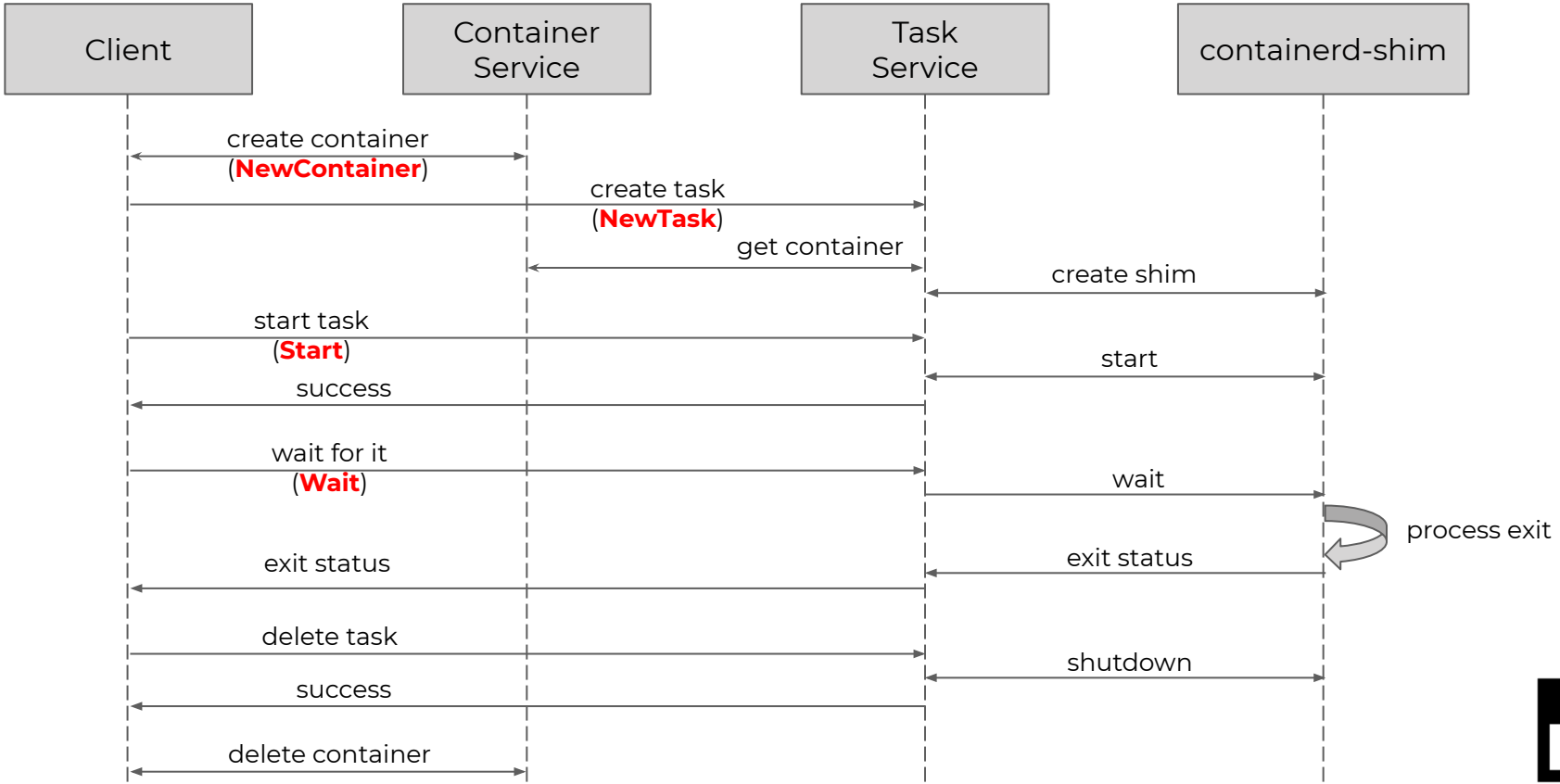
Pull Image


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

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
```

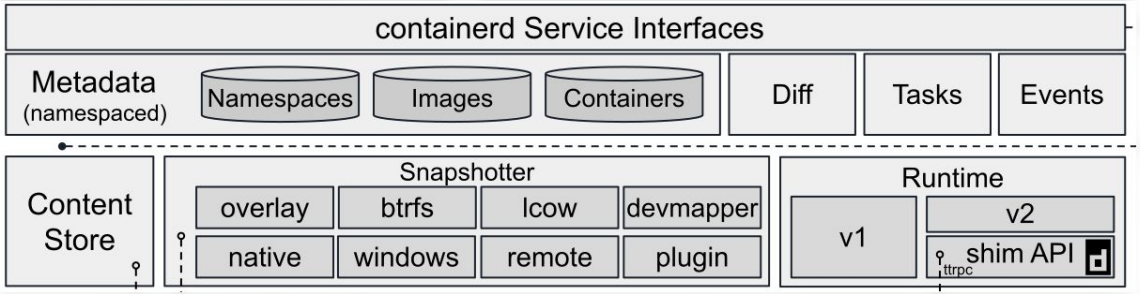
```
type RemoteOpt
    func WithImageHandler(h images.Handler) RemoteOpt
    func WithImageHandlerWrapper(w func(images.Handler) images.Handler) RemoteOpt
    func WithResolver(resolver remotes.Resolver) RemoteOpt
```

Aimed to

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

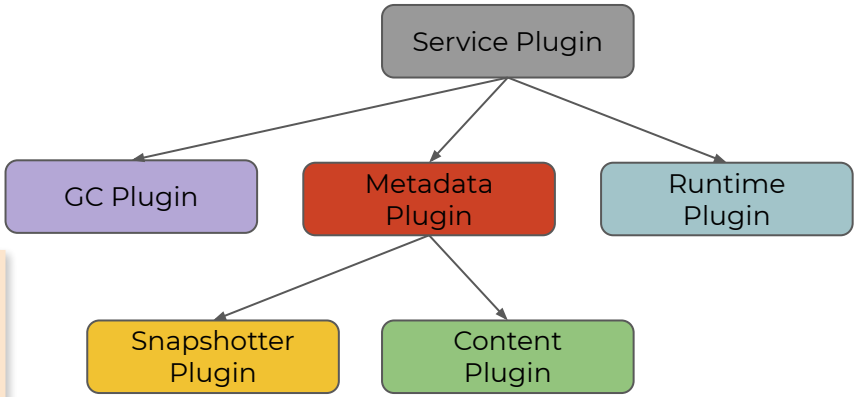
Component as Plugin

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

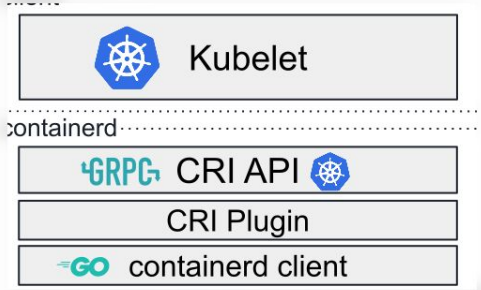


Plugin Registration

- loose coupling and clear boundaries
- dependency Graph




```
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 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

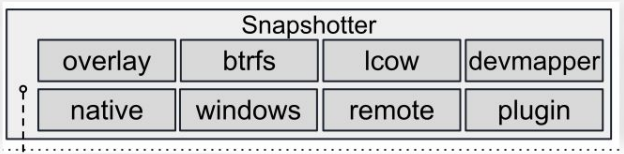
- Create remote plugin as proxy
- Configure it for containerd
- Snapshotter and Content only

```
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

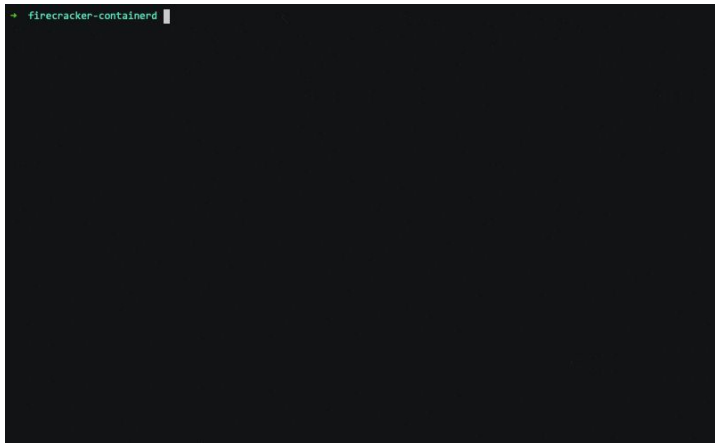



```
service Task {
  rpc State(StateRequest) returns (StateResponse);
  rpc Create(CreateTaskRequest) returns (CreateTaskResponse);
  rpc Start(StartRequest) returns (StartResponse);
  rpc Delete(DeleteRequest) returns (DeleteResponse);
  rpc Pids(PidsRequest) returns (PidsResponse);
  rpc Pause(PauseRequest)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

External Plugin Demo

containerd + Firecracker

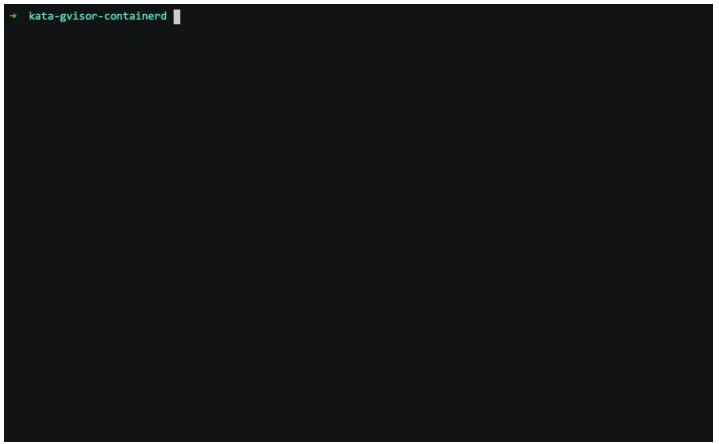
- Demo - integrate with Firecracker runtime
  - using version for
    - containerd@v1.2.6
    - firecracker-containerd@071b8d3
    - firecracker@0.15.2
  - naive is external snapshotter plugin
  - binary convention
    - awk.firecracker -> containerd-shim-awk-firecracker
  - click me



Firecracker

cri-containerd + {gVisor, katacontainer}

- Demo - integrate with gVisor and katacontainer runtime
  - using version for
    - containerd@v1.2.6
    - gVisor@release-20190304.1-184-gebe2f78d9bc8
    - katacontainer@1.7.0-rc1
    - kubeadm@v1.14.1
  - runtime class resource
    - class name -> handler name -> runtime plugin name
  - click me





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