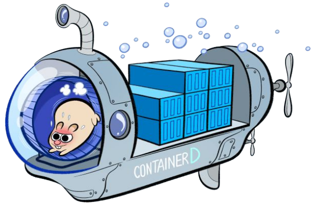
containerd Internals: Building a Core Container Runtime

Stephen Day (@stevvooe), Docker

October 24, 2017
Docker Prague Meetup

A Brief History



APRIL 2016 Containerd “0.2” announced, Docker 1.11

Management/Supervisor for the OCI `runc` executor

Announce expansion of containerd OSS project DECEMBER 2016

containerd

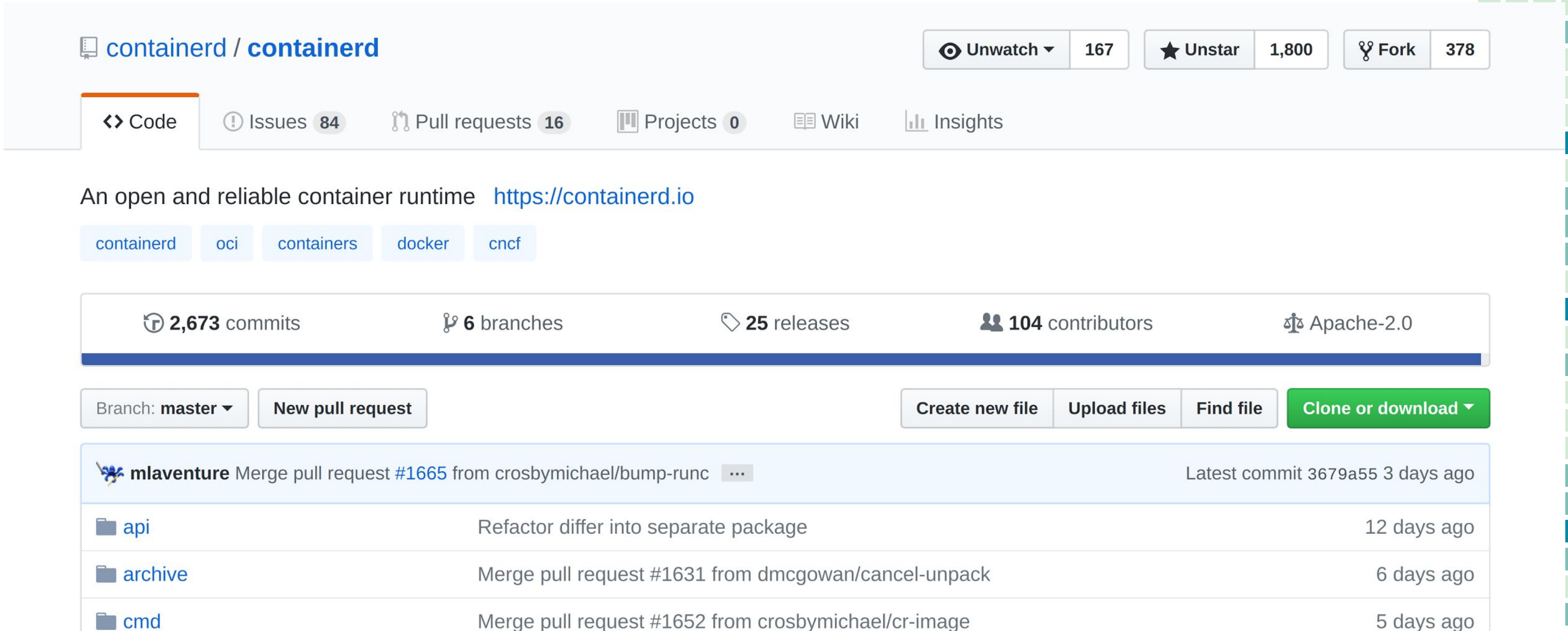
Containerd 1.0: A core container runtime project for the industry

MARCH 2017 Containerd project contributed to CNCF

CLOUD NATIVE COMPUTING FOUNDATION

containerd

https://github.com/containerd/containerd



containerd / containerd

Unwatch 167 | Unstar 1,800 | Fork 378

An open and reliable container runtime https://containerd.io

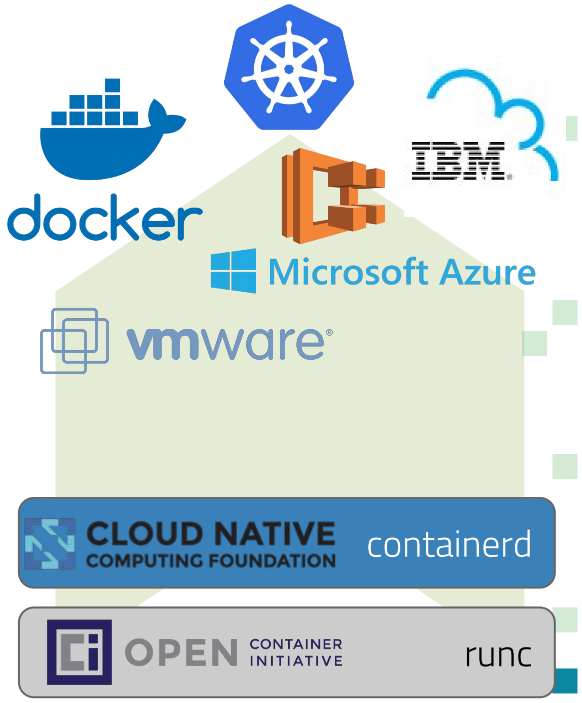
2,673 commits | 6 branches | 25 releases | 104 contributors | Apache-2.0

mlaventure Merge pull request #1665 from crosbymichael/bump-runc … Latest commit 3679a55 3 days ago

api	Refactor differ into separate package	12 days ago
archive	Merge pull request #1631 from dmcgowan/cancel-unpack	6 days ago
cmd	Merge pull request #1652 from crosbymichael/cr-image	5 days ago

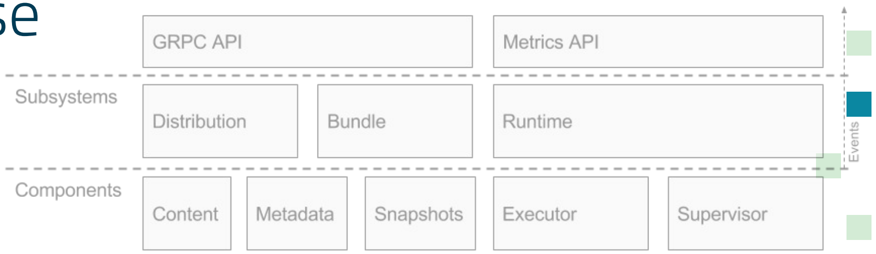
Why Containerd 1.0?

- Continue projects **spun out** from monolithic Docker engine
- Expected use **beyond** Docker engine (Kubernetes CRI)
- Donation to **foundation** for broad industry collaboration
  - Similar to runc/libcontainer and the OCI



Technical Goals/Intentions

- Clean **gRPC-based** API + client library
- Full **OCI** support (runtime and image spec)
- **Stability** and **performance** with tight, well-defined core of container function
- **Decoupled** systems (image, filesystem, runtime) for pluggability, reuse



Requirements

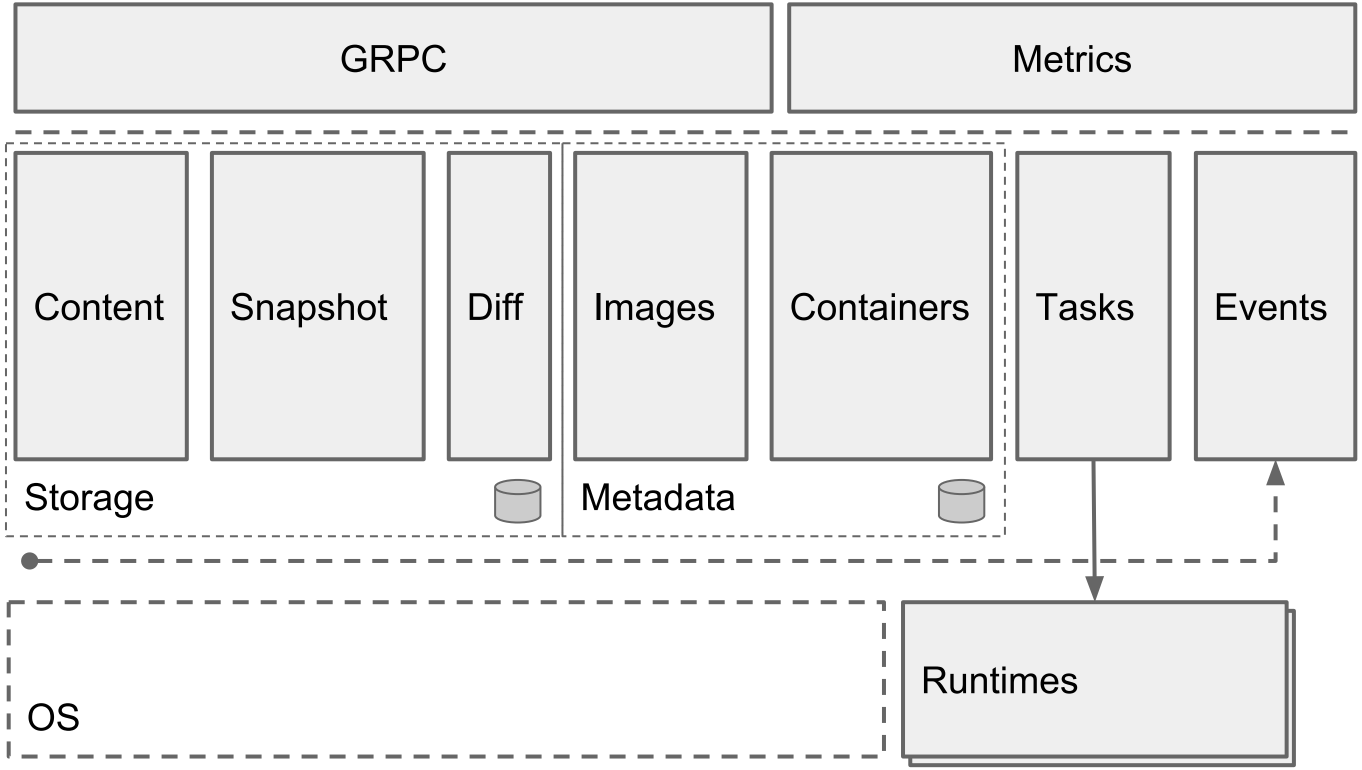
- **A la carte**: use only what is required
- Runtime **agility**: fits into different platforms
  - Pass-through container configuration (direct OCI)
- **Decoupled**
- Use **known-good** technology
  - OCI container runtime and images
  - gRPC for API
  - Prometheus for Metrics

Use cases

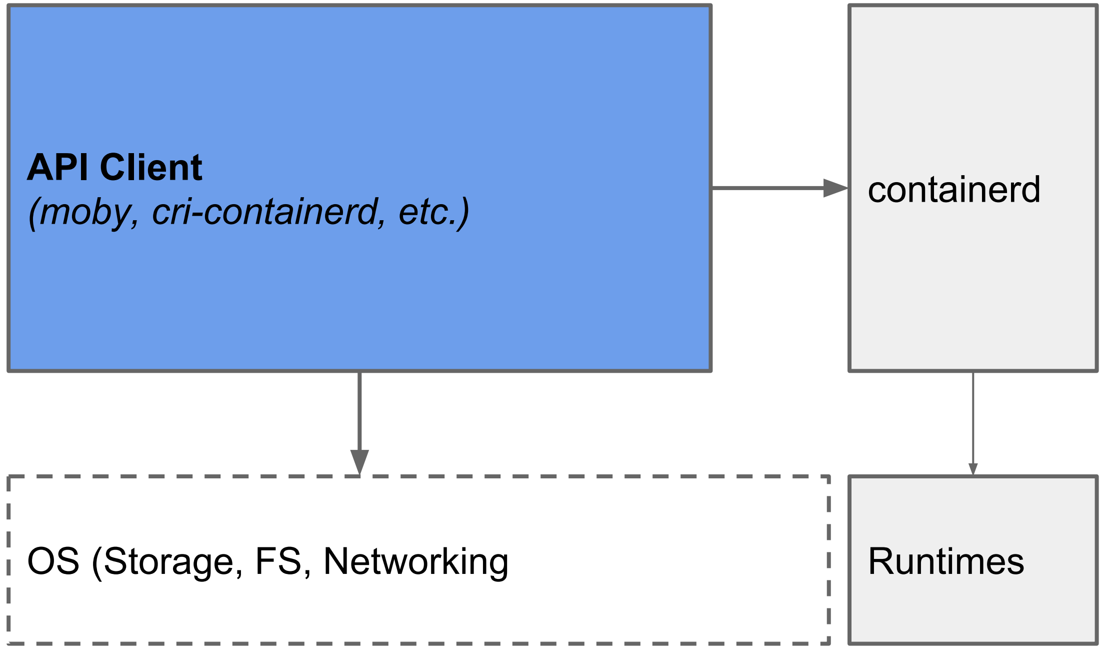
- **CURRENT**
- Docker (moby)
- Kubernetes (cri-containerd)
- SwarmKit (experimental)
- LinuxKit
- BuildKit

- **FUTURE/POTENTIAL**
- IBM Cloud/Bluemix
- OpenFaaS
- Puppet R&D
- *{your project here}*

Architecture



Architecture



Metrics

```
# HELP container_blkio_io_service_bytes_recursive_bytes The blkio io service bytes recursive
# TYPE container_blkio_io_service_bytes_recursive_bytes gauge
container_blkio_io_service_bytes_recursive_bytes{container_id="foo4",device="/dev/nvme0n1",major="259",minor="0",namespace="default",op="Async"} 1.07159552e+08
container_blkio_io_service_bytes_recursive_bytes{container_id="foo4",device="/dev/nvme0n1",major="259",minor="0",namespace="default",op="Read"} 0
container_blkio_io_service_bytes_recursive_bytes{container_id="foo4",device="/dev/nvme0n1",major="259",minor="0",namespace="default",op="Sync"} 81920
container_blkio_io_service_bytes_recursive_bytes{container_id="foo4",device="/dev/nvme0n1",major="259",minor="0",namespace="default",op="Total"} 1.07241472e+08
container_blkio_io_service_bytes_recursive_bytes{container_id="foo4",device="/dev/nvme0n1",major="259",minor="0",namespace="default",op="Write"} 1.07241472e+08
# HELP container_blkio_io_serviced_recursive_total The blkio io servied recursive
# TYPE container_blkio_io_serviced_recursive_total gauge
container_blkio_io_serviced_recursive_total{container_id="foo4",device="/dev/nvme0n1",major="259",minor="0",namespace="default",op="Async"} 892
container_blkio_io_serviced_recursive_total{container_id="foo4",device="/dev/nvme0n1",major="259",minor="0",namespace="default",op="Read"} 0
container_blkio_io_serviced_recursive_total{container_id="foo4",device="/dev/nvme0n1",major="259",minor="0",namespace="default",op="Sync"} 888
container_blkio_io_serviced_recursive_total{container_id="foo4",device="/dev/nvme0n1",major="259",minor="0",namespace="default",op="Total"} 1780
container_blkio_io_serviced_recursive_total{container_id="foo4",device="/dev/nvme0n1",major="259",minor="0",namespace="default",op="Write"} 1780
# HELP container_cpu_kernel_nanoseconds The total kernel cpu time
# TYPE container_cpu_kernel_nanoseconds gauge
container_cpu_kernel_nanoseconds{container_id="foo4",namespace="default"} 2.6e+08
# HELP container_cpu_throttle_periods_total The total cpu throttle periods
# TYPE container_cpu_throttle_periods_total gauge
container_cpu_throttle_periods_total{container_id="foo4",namespace="default"} 0
# HELP container_cpu_throttled_periods_total The total cpu throttled periods
# TYPE container_cpu_throttled_periods_total gauge
container_cpu_throttled_periods_total{container_id="foo4",namespace="default"} 0
# HELP container_cpu_throttled_time_nanoseconds The total cpu throttled time
# TYPE container_cpu_throttled_time_nanoseconds gauge
container_cpu_throttled_time_nanoseconds{container_id="foo4",namespace="default"} 0
# HELP container_cpu_total_nanoseconds The total cpu time
# TYPE container_cpu_total_nanoseconds gauge
container_cpu_total_nanoseconds{container_id="foo4",namespace="default"} 1.003301578e+09
# HELP container_cpu_user_nanoseconds The total user cpu time
# TYPE container_cpu_user_nanoseconds gauge
container_cpu_user_nanoseconds{container_id="foo4",namespace="default"} 7e+08
# HELP container_hugetlb_failcnt_total The hugetlb failcnt
# TYPE container_hugetlb_failcnt_total gauge
container_hugetlb_failcnt_total{container_id="foo4",namespace="default",page="1GB"} 0
container_hugetlb_failcnt_total{container_id="foo4",namespace="default",page="2MB"} 0
# HELP container_hugetlb_max_bytes The hugetlb maximum usage
# TYPE container_hugetlb_max_bytes gauge
container_hugetlb_max_bytes{container_id="foo4",namespace="default",page="1GB"} 0
container_hugetlb_max_bytes{container_id="foo4",namespace="default",page="2MB"} 0
# HELP container_hugetlb_usage_bytes The hugetlb usage
# TYPE container_hugetlb_usage_bytes gauge
container_hugetlb_usage_bytes{container_id="foo4",namespace="default",page="1GB"} 0
container_hugetlb_usage_bytes{container_id="foo4",namespace="default",page="2MB"} 0
# HELP container_memory_active_anon_bytes The active_anon amount
# TYPE container_memory_active_anon_bytes gauge
container_memory_active_anon_bytes{container_id="foo4",namespace="default"} 2.658304e+06
# HELP container_memory_active_file_bytes The active_file amount
# TYPE container_memory_active_file_bytes gauge
container_memory_active_file_bytes{container_id="foo4",namespace="default"} 7.319552e+06
# HELP container_memory_cache_bytes The cache amount used
# TYPE container_memory_cache_bytes gauge
container_memory_cache_bytes{container_id="foo4",namespace="default"} 5.0597888e+07
# HELP container_memory_dirty_bytes The dirty amount
```

Containerd: Rich Go API

Getting Started

https://github.com/containerd/containerd/blob/master/docs/getting-started.md

GoDoc

https://godoc.org/github.com/containerd/containerd

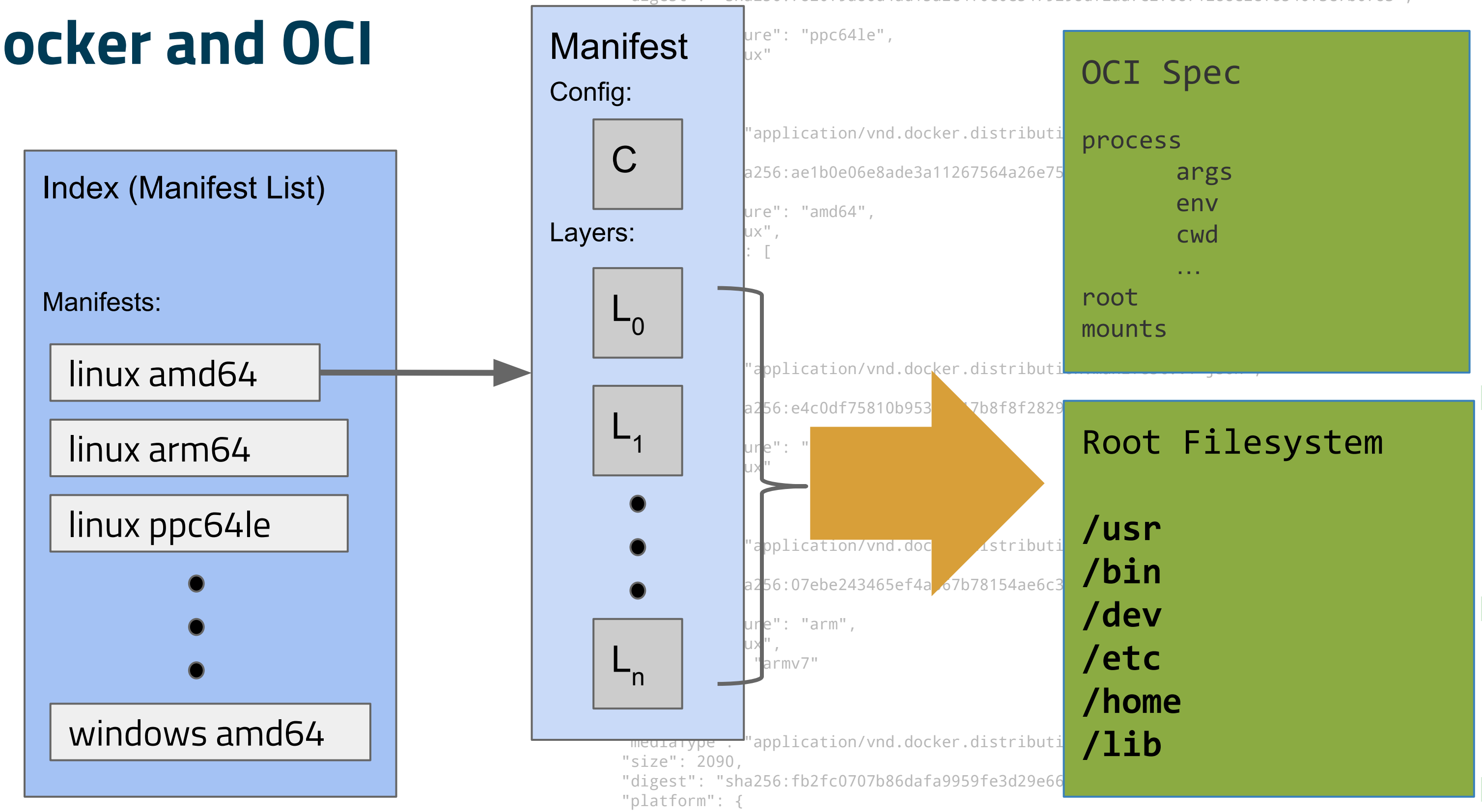
Pulling an Image

What do runtimes need?

Image Formats

Docker and OCI

```
{
   "schemaVersion": 2,
   "mediaType": "application/vnd.docker.distribution.manifest.list.v2+json",
   "manifests": [
      {
         "mediaType": "application/vnd.docker.distribution.manifest.v1+json",
         "size": 2094,
         "digest": "sha256:7820f9a86d4ad15a2c4f0c0e5479298df2aa7c2f6871288e2ef8546f3e7b6783",
```



```
         "size": 2090,
         "digest": "sha256:fb2fc0707b86dafa9959fe3d29e66
         "platform": {
            "architecture": "arm64",
            "os": "linux",
            "variant": "armv8"
         }
```

Content Addressability

```
digest.FromString(“foo”) ->
        “sha256:2c26b46b68ffc68ff99b453c1d30413413422d706483bfa0f98a5e886266e7ae”
digest.FromString(“foo tampered”) ->
        “sha256:51f7f1d1f6bebed72b936c8ea257896cb221b91d303c5b5c44073fce33ab8dd8”
digest.FromString(“bar sha256:2c...”) ->
        “sha256:2e94890c66fbcccca9ad680e1b1c933cc323a5b4bcb14cc8a4bc78bb88d41055”
```

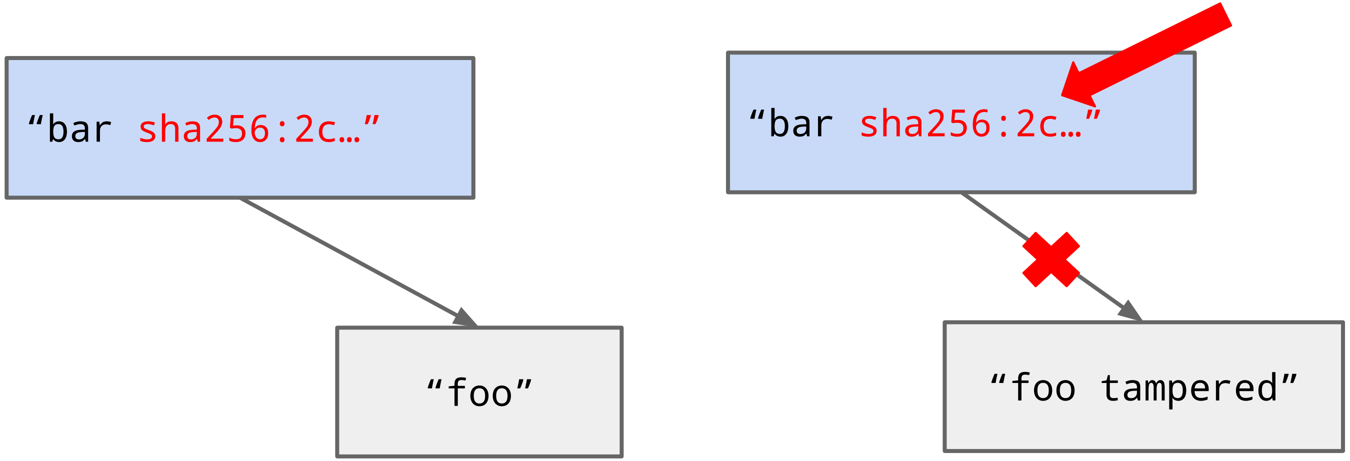
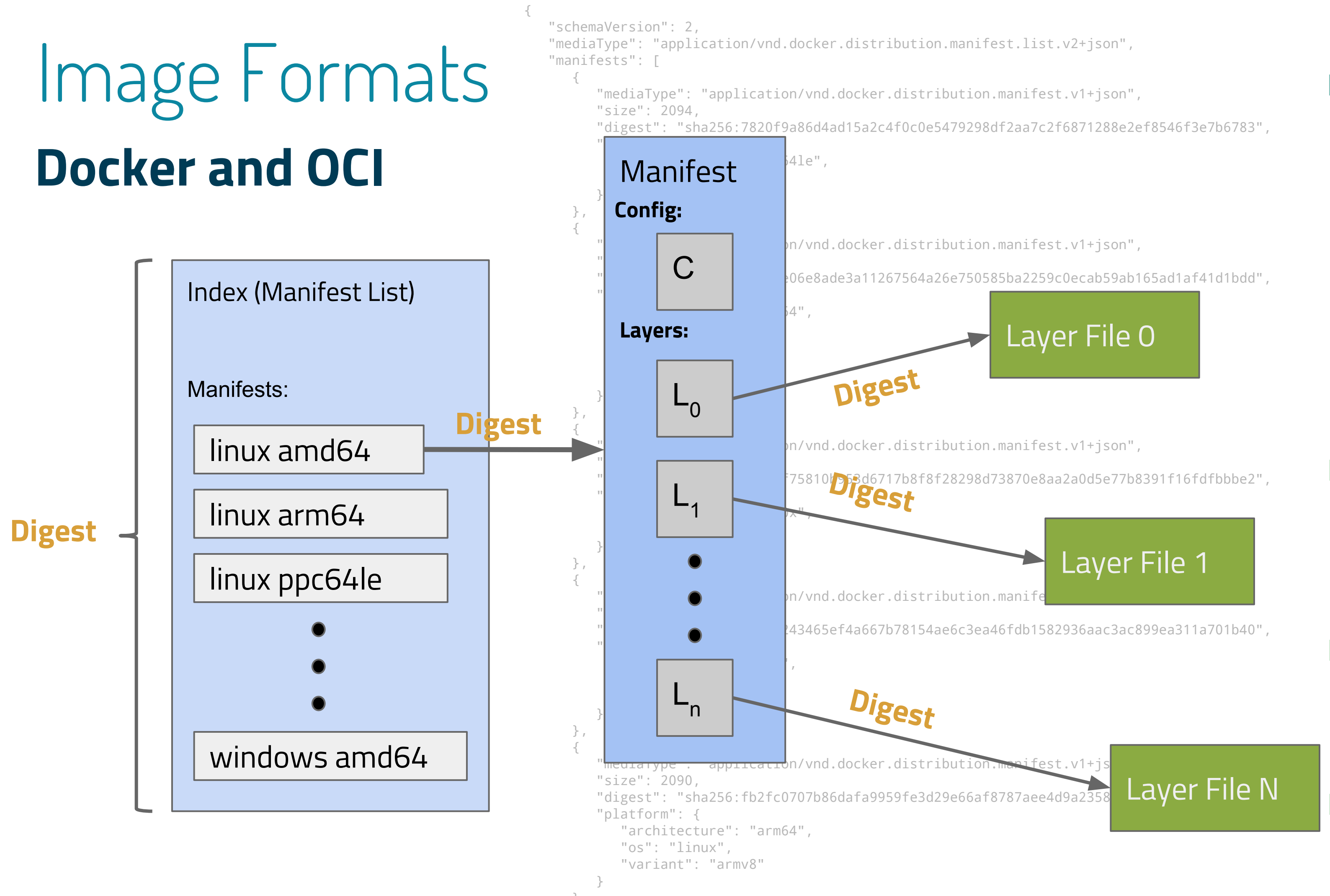


Image Formats

Docker and OCI



```
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.docker.distribution.manifest.list.v2+json",
  "manifests": [
    {
      "mediaType": "application/vnd.docker.distribution.manifest.v1+json",
      "size": 2094,
      "digest": "sha256:7820f9a86d4ad15a2c4f0c0e5479298df2aa7c2f6871288e2ef8546f3e7b6783",
                                    54le",
      }
    },
    {
                                    on/vnd.docker.distribution.manifest.v1+json",
                                    e06e8ade3a11267564a26e750585ba2259c0ecab59ab165ad1af41d1bdd",
                                    54",
      }
    },
    {
                                    on/vnd.docker.distribution.manifest.v1+json",
                                    f75810b     d6717b8f8f28298d73870e8aa2a0d5e77b8391f16fdfbbbe2",
                                    0x",
      }
    },
    {
                                    on/vnd.docker.distribution.manife
                                    243465ef4a667b78154ae6c3ea46fdb1582936aac3ac899ea311a701b40",
                                    ",
      }
    },
    {
      "mediaType": "application/vnd.docker.distribution.manifest.v1+js
      "size": 2090,
      "digest": "sha256:fb2fc0707b86dafa9959fe3d29e66af8787aee4d9a2358
      "platform": {
        "architecture": "arm64",
        "os": "linux",
        "variant": "armv8"
      }
```

Resolution

Getting a digest from a name:

`ubuntu`

`sha256:71cd81252a3563a03ad8daee81047b62ab5d892ebbfbf71cf53415f29c130950`

Image Names in Docker

Reference Type	CLI	Canonical
Repository	ubuntu	docker.io/library/ubuntu
Untagged	ubuntu	docker.io/libary/ubuntu:latest
Tagged	ubuntu:16.04	docker.io/library/ubuntu:16.04
Content Trust	ubuntu:latest	docker.io/library/ubuntu@sha256:...
By digest	ubuntu@sha256:....	docker.io/library/ubuntu@sha256:...
Unofficial tagged	stevvooe/ubuntu:latest	docker.io/stevvooe/ubuntu:latest
Private registry tagged	myregistry.com/repo:latest	myregistry.com/repo:latest

Other Approaches to Naming

- **Self Describing**
  - Massive collisions
  - Complex trust scenarios
- **URI Schemes: `docker://docker.io/library/ubuntu`**
  - Redundant
  - Confuses protocols and formats
  - Operationally Limiting
    - let configuration choose protocol *and* format

Locators

Schema-less URIs

`ubuntu` (Docker name)

$\hookrightarrow$ `docker.io/library/ubuntu:latest` (Docker canonical)

$\hookrightarrow$ $(\underbrace{\texttt{docker.io/library/ubuntu}}_{\textbf{locator}},\ \underbrace{\texttt{latest}}_{\textbf{object}})$

Remotes *(hint: think git remotes)*

Locators and Resolution

```
fetcher := resolver.Resolve("docker.io/library/ubuntu")
```

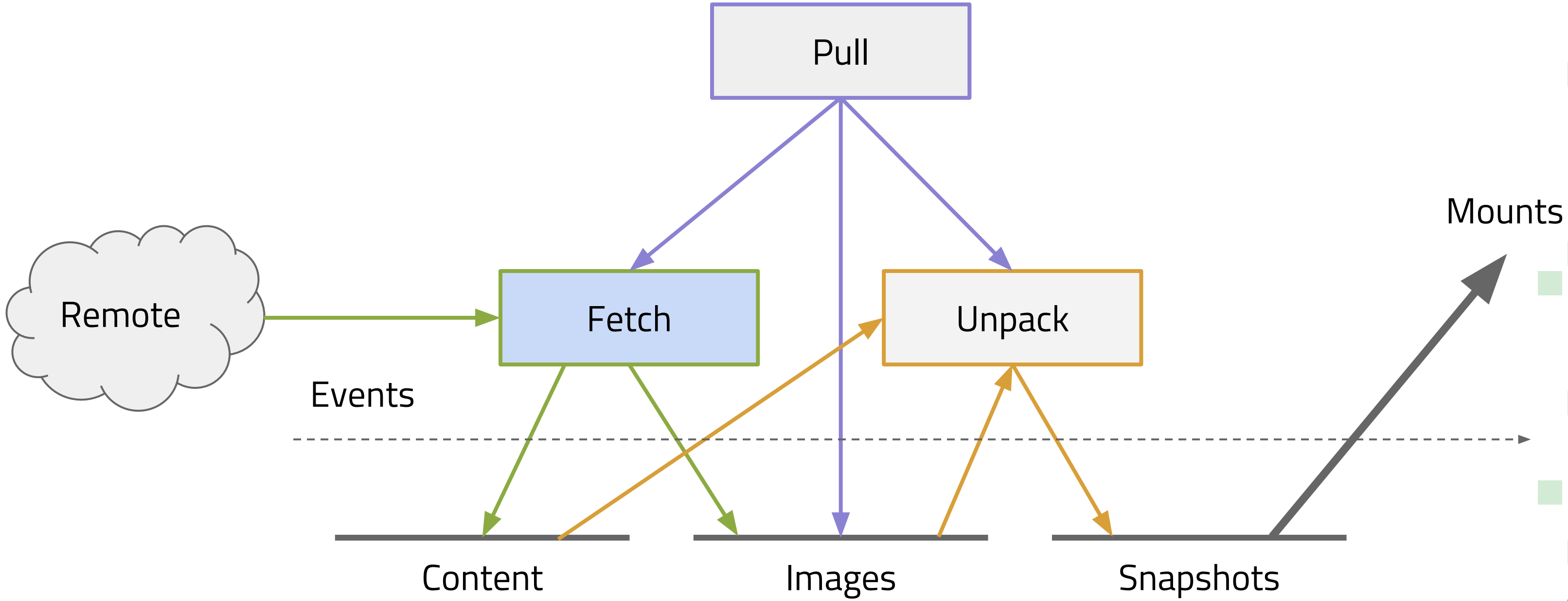
Endlessly Configurable!

```
type Resolver interface {
      Resolve(ctx context.Context, locator string) (Fetcher, error)
}
```

```
type Fetcher interface {
      Fetch(ctx context.Context, id string, hints ...string) (io.ReadCloser, error)
}
```

Pulling an Image

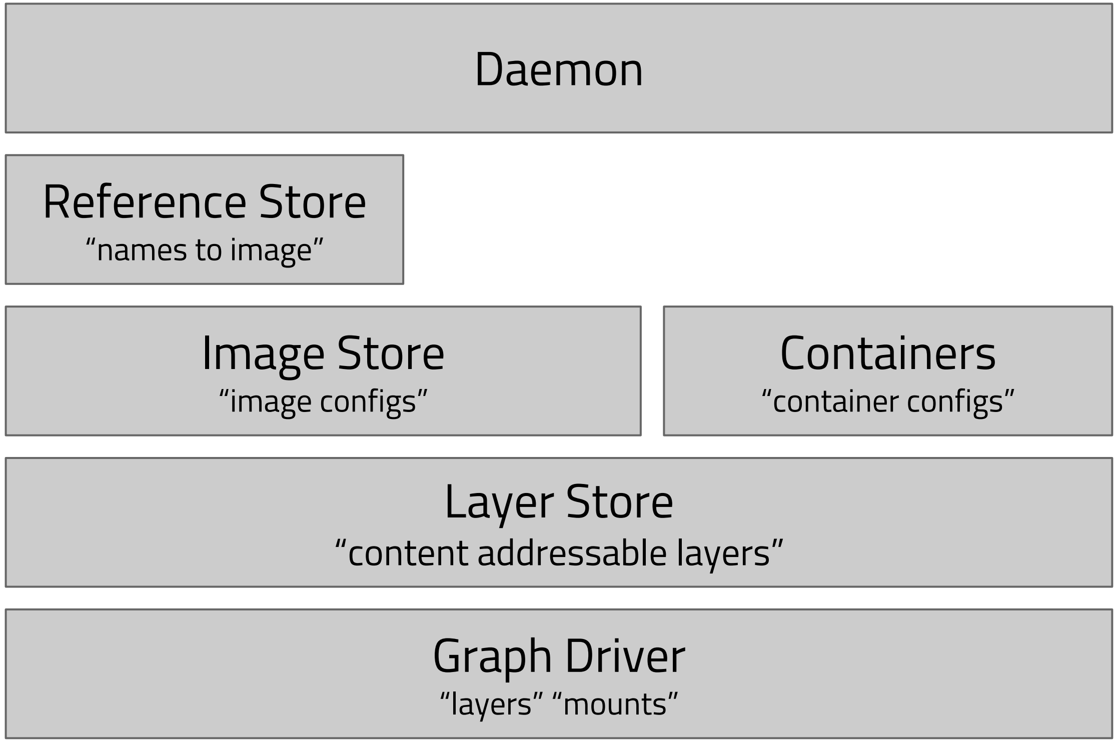
Data Flow



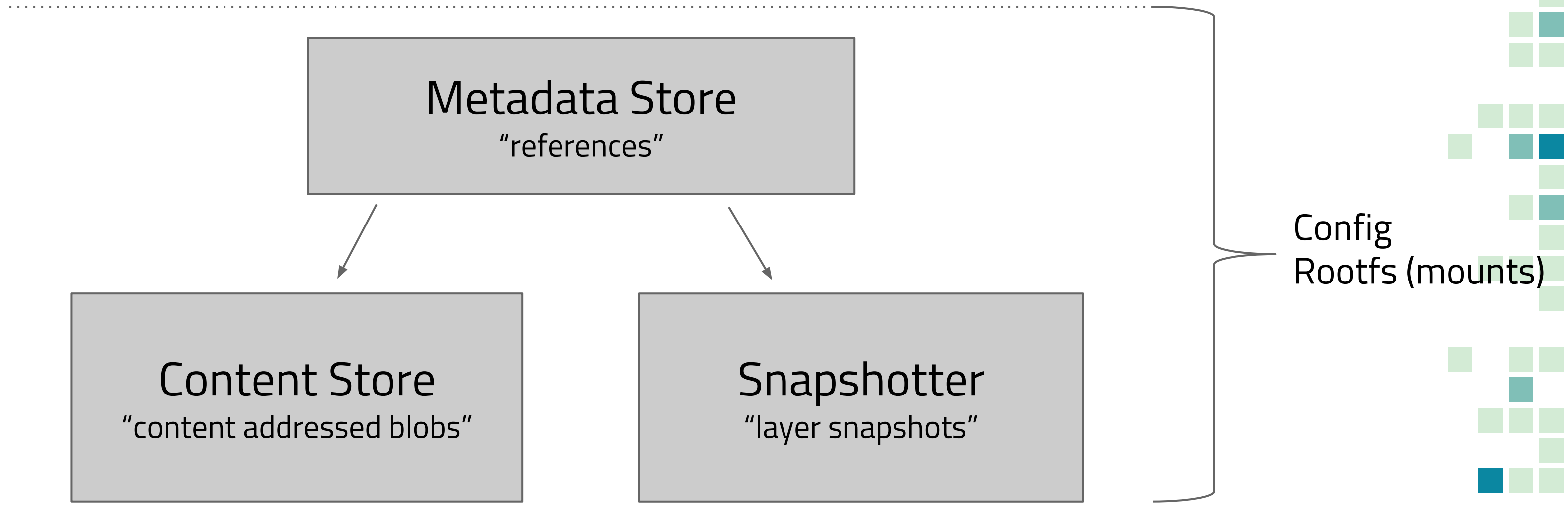
Snapshotters

How do you build a container root filesystem?

Docker Storage Architecture



containerd Storage Architecture



Snapshots

- No mounting, just returns mounts!
- Explicit active (rw) and committed (ro)
- Commands represent lifecycle
- Reference key chosen by caller (allows using content addresses)
- No tars and no diffs

Evolved from Graph Drivers

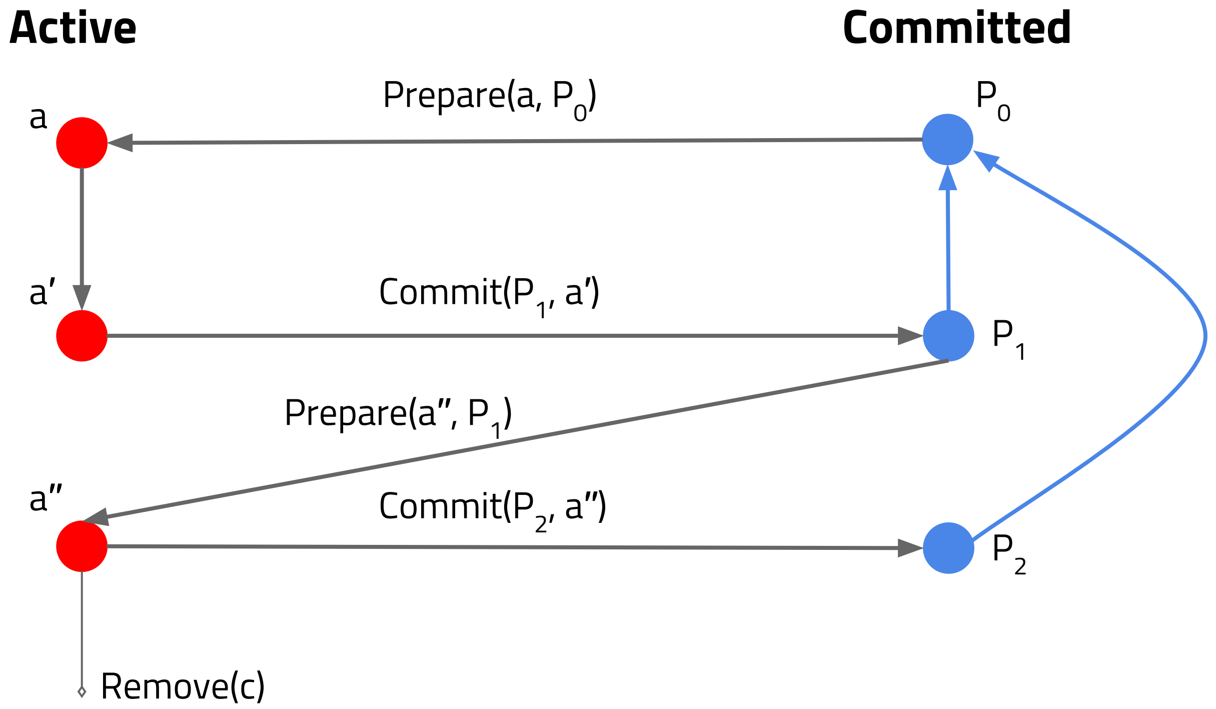
- Simple layer relationships
- Small and focused interface
- Non-opinionated string keys
- External Mount Lifecycle

```go
type Snapshotter interface {
	Stat(key string) (Info, error)
	Mounts(key string) ([]containerd.Mount, error)
	Prepare(key, parent string) ([]containerd.Mount, error)
	View(key, parent string) ([]containerd.Mount, error)
	Commit(name, key string) error
	Remove(key string) error
	Walk(fn func(Info) error) error
}

type Info struct {
	Name    string // name or key of snapshot
	Parent  string
	Kind    Kind
}

type Kind int
const (
	KindView
	KindActive
	KindCommitted
)
```

Snapshot Model



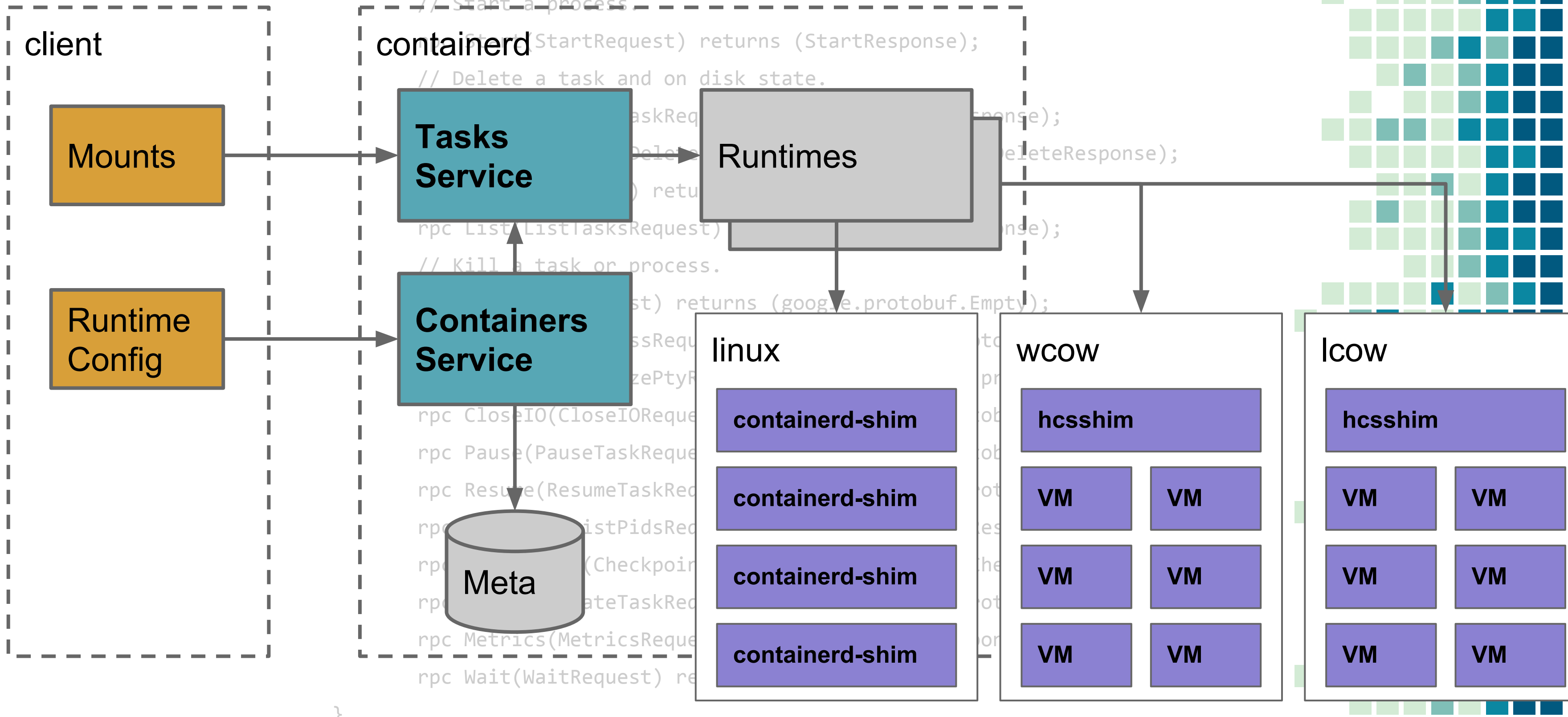
Example: Investigating Root Filesystem

```
$ ctr snapshot ls
...
$ ctr snapshot tree
...
$ ctr snapshot mounts <target> <id>
```

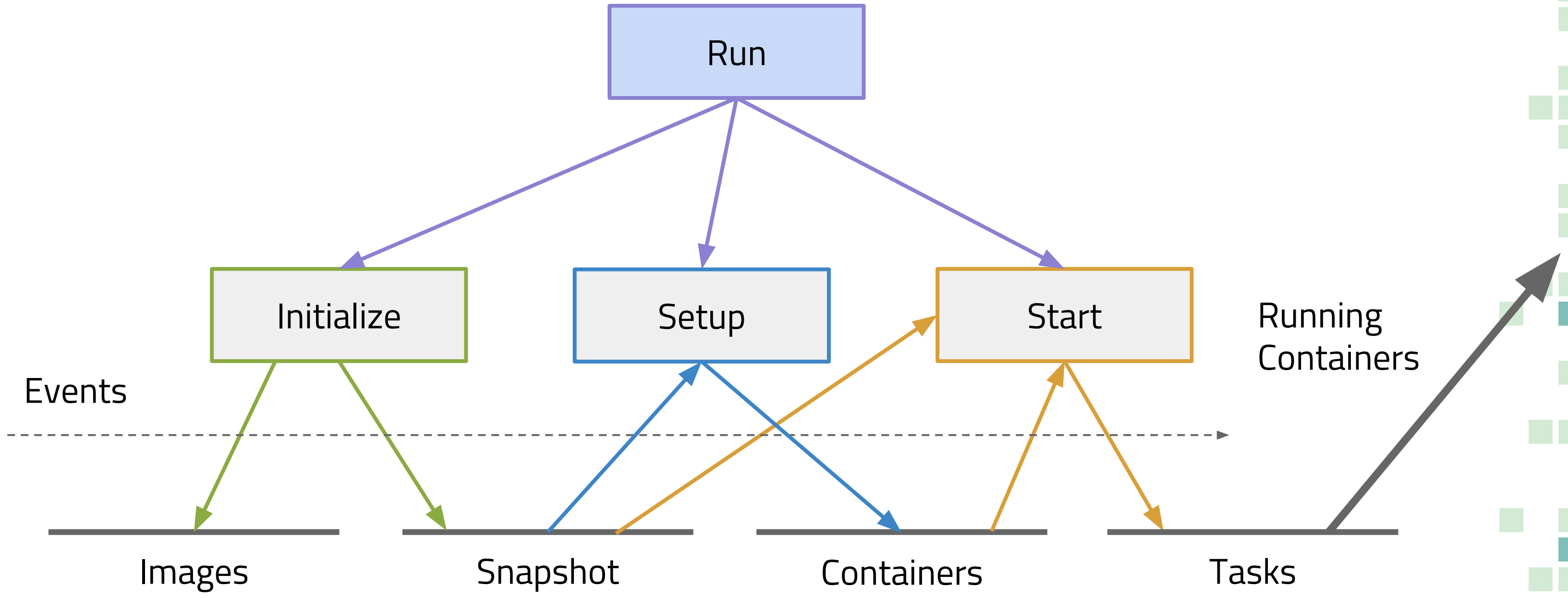
Running a container

Tasks and Runtime

```
service Tasks {
    // Create a task.
    rpc Create(CreateTaskRequest) returns (CreateTaskResponse);
    // Start a process.
              StartRequest) returns (StartResponse);
    // Delete a task and on disk state.
                  askReq                 ponse);
                  Dele                   eleteResponse);
                  ) retu
    rpc List(ListTasksRequest)           nse);
    // Kill a task or process.
                  st) returns (google.protobuf.Empty);
                  ssRequ                 oto
                  zePtyR                 pr
    rpc CloseIO(CloseIOReque             ob
    rpc Pause(PauseTaskReque             ob
    rpc Resume(ResumeTaskReq             ot
    rp            istPidsReq             es
    rp            (Checkpoin             he
    rp            ateTaskReq             ot
    rpc Metrics(MetricsReque             on
    rpc Wait(WaitRequest) re
}
```



Starting a Container



Demo

Example: Pull an Image

Via **ctr** client:

```
$ export \
  CONTAINERD_NAMESPACE=example

$ ctr pull \
  docker.io/library/redis:alpine

$ ctr image ls
...
```

```
import (
        "context"

        "github.com/containerd/containerd"
        "github.com/containerd/containerd/namespaces"
)

// connect to our containerd daemon
client, err := containerd.New("/run/containerd/containerd.sock")
defer client.Close()

// set our namespace to “example”:
ctx := namespaces.WithNamespace(context.Background(), "example")

// pull the alpine-based redis image from DockerHub:
image, err := client.Pull(ctx,
                "docker.io/library/redis:alpine",
                containerd.WithPullUnpack)
```

Example: Run a Container

Via **ctr** client:

```
$ export \
  CONTAINERD_NAMESPACE=example

$ ctr run -t \
  docker.io/library/redis:alpine \
  redis-server

$ ctr c
...
```

```
// create our container object and config
container, err := client.NewContainer(ctx,
      "redis-server",
      containerd.WithImage(image),
      containerd.WithNewSpec(containerd.WithImageConfig(image)),
      )
defer container.Delete()

// create a task from the container
task, err := container.NewTask(ctx, containerd.Stdio)
defer task.Delete(ctx)

// make sure we wait before calling start
exitStatusC, err := task.Wait(ctx)
// call start on the task to execute the redis server
if err := task.Start(ctx); err != nil {
      return err
}
```

Example: Kill a Task

Via **ctr** client:

```
$ export \
  CONTAINERD_NAMESPACE=example

$ ctr t kill redis-server

$ ctr t ls
...
```

```go
// make sure we wait before calling start
exitStatusC, err := task.Wait(ctx)

time.Sleep(3 * time.Second)

if err := task.Kill(ctx, syscall.SIGTERM); err != nil {
	return err
}

// retrieve the process exit status from the channel
status := <-exitStatusC
code, exitedAt, err := status.Result()
if err != nil {
	return err
}
// print out the exit code from the process
fmt.Printf("redis-server exited with status: %d\n", code)
```

Example: Customize OCI Configuration

```go
// WithHtop configures a container to monitor the host via `htop`
func WithHtop(s *specs.Spec) error {
	// make sure we are in the host pid namespace
	if err := containerd.WithHostNamespace(specs.PIDNamespace)(s); err != nil {
		return err
	}
	// make sure we set htop as our arg
	s.Process.Args = []string{"htop"}
	// make sure we have a tty set for htop
	if err := containerd.WithTTY(s); err != nil {
		return err
	}
	return nil
}
```

With{func} functions cleanly separate modifiers

Release

https://github.com/containerd/containerd/blob/master/RELEASES.md

Supported Components

Component	Status	Stablized Version	Links
GRPC API	Beta	1.0	api/
Metrics API	Beta	1.0	
Go client API	Unstable	1.1 tentative	godoc
`ctr` tool	Unstable	Out of scope	-

Support Horizon

Release	Status	Start	End of Life
0.0	End of Life	Dec 4, 2015	-
0.1	End of Life	Mar 21, 2016	-
0.2	Active	Apr 21, 2016	Upon 1.0 release
1.0	Next	TBD	max(TBD+1 year, release of 1.1.0)

1.0.0-beta.2

https://github.com/containerd/containerd/releases/tag/v1.0.0-beta.2

One point Ohhhhhhhhhh!

https://github.com/containerd/containerd/milestone/13

Going further with **containerd**

- **Contributing**: https://github.com/containerd/containerd
  - Bug fixes, adding tests, improving docs, validation
- **Using**: See the getting started documentation in the `docs` folder of the repo
- **Porting/testing**: Other architectures & OSs, stress testing (see bucketbench, containerd-stress):
  - `git clone <repo>, make binaries, sudo make install`
- **K8s CRI**: incubation project to use containerd as CRI
  - In alpha today; e2e tests, validation, contributing

Thank You! Questions?

- **Stephen Day**
  - https://github.com/stevvooe
  - stephen@docker.com
  - Twitter: @stevvooe