containerd

Take Control of your Filesystem with Snapshotters

Stephen Day
@stevvooe

May 2, 2018
KubeCon EU

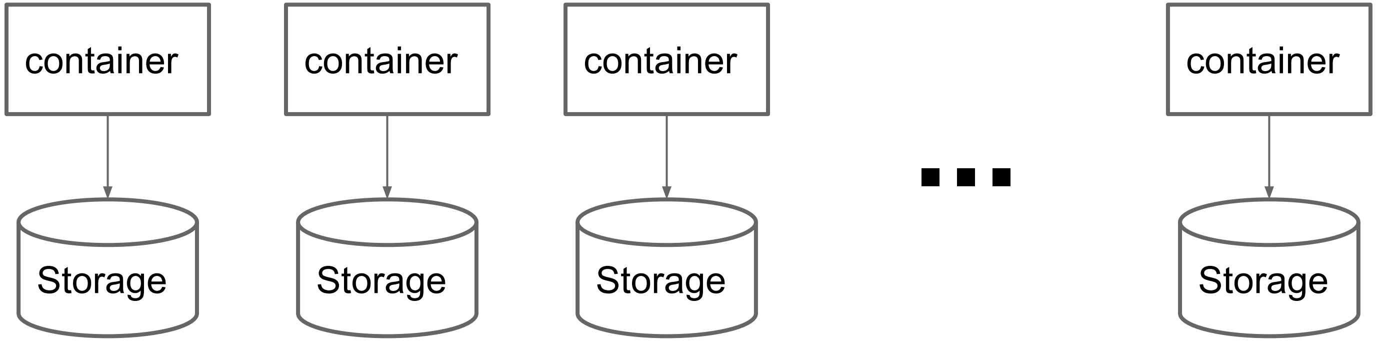
Why this talk?

- Provide rough history of container filesystems
- Introduce snapshotters in more detail
- Inspire new innovation in this area
  - Builders
  - Volume snapshotting

Why the complexity?

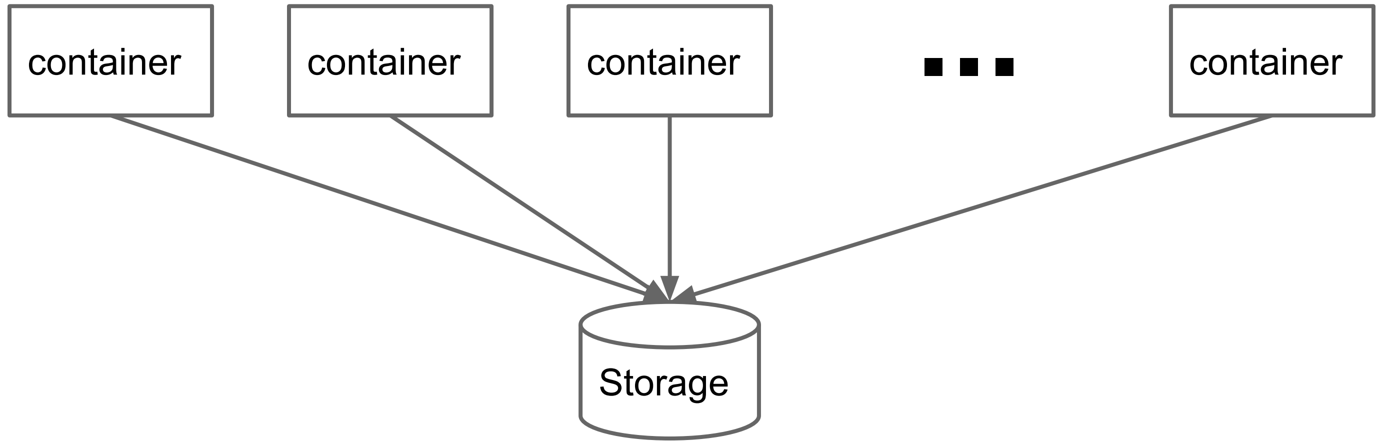
- Build up a root file system for a container
- Reduce storage requirements
  - -> Increase workload density

A naive model



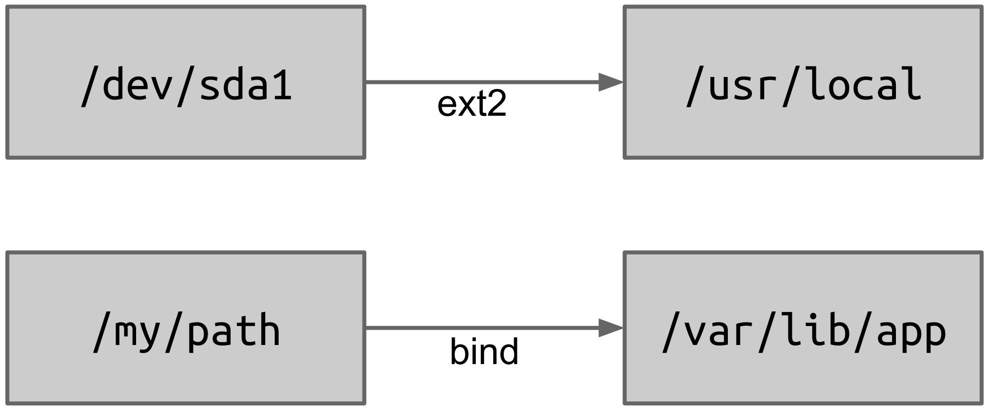
Storage Cost = O(N)

The Goal

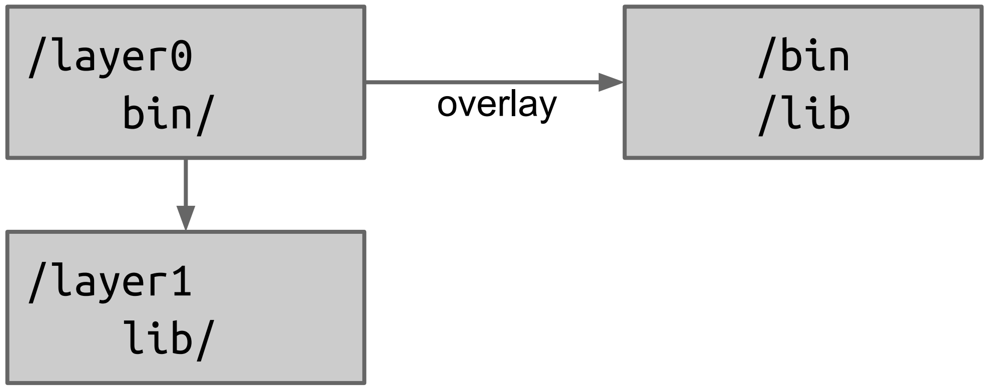


Storage Cost = O(1)

Mounts



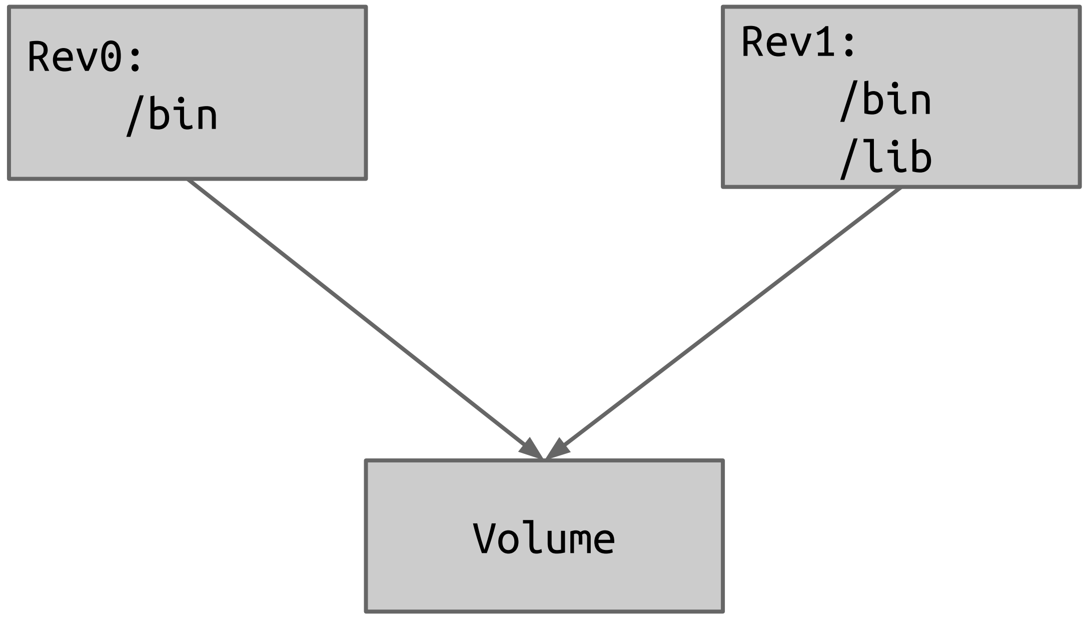
Union File Systems



Union File Systems: Examples

- Plan9
- ufs
- AUFS
- overlay

Snapshot File Systems



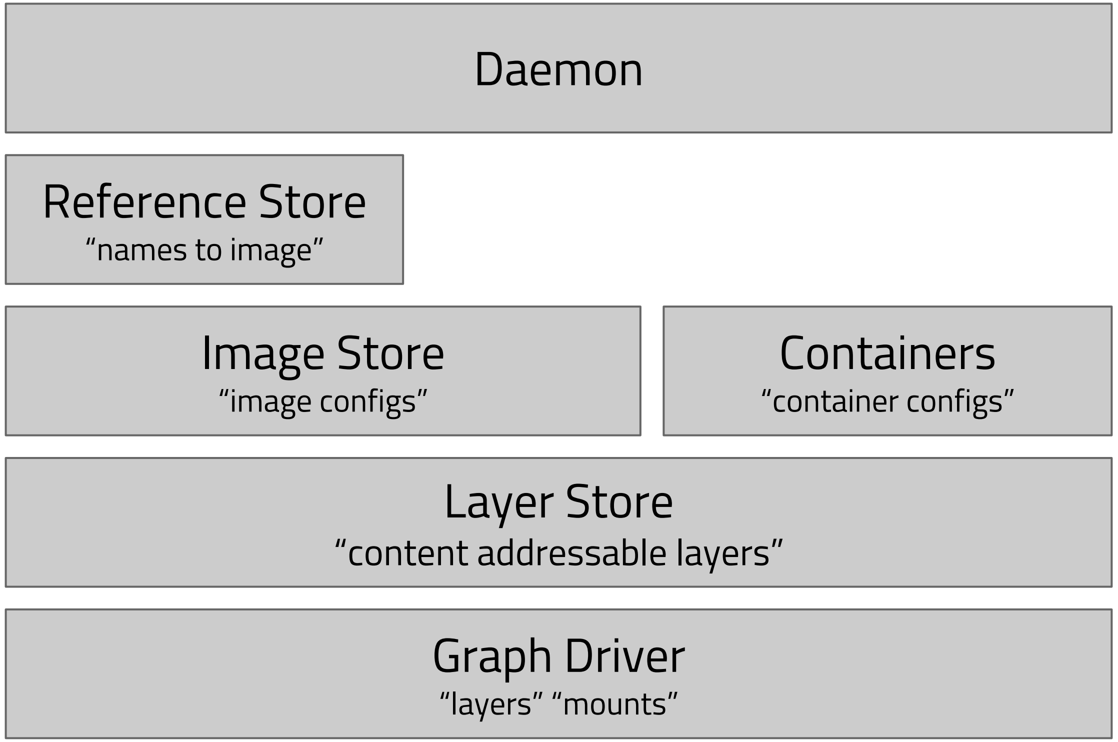
Snapshot File Systems: Examples

- fossil
- NTFS
- Zfs
- Btrfs
- Git (not a filesystem, but similar concept)

Union vs Snapshot

- Union: allows modification of underlying data
- Snapshot: can handle more revisions
- Both: copy-on-write
- Both: **shared data model**

Docker Storage Architecture



Graph Driver Problems

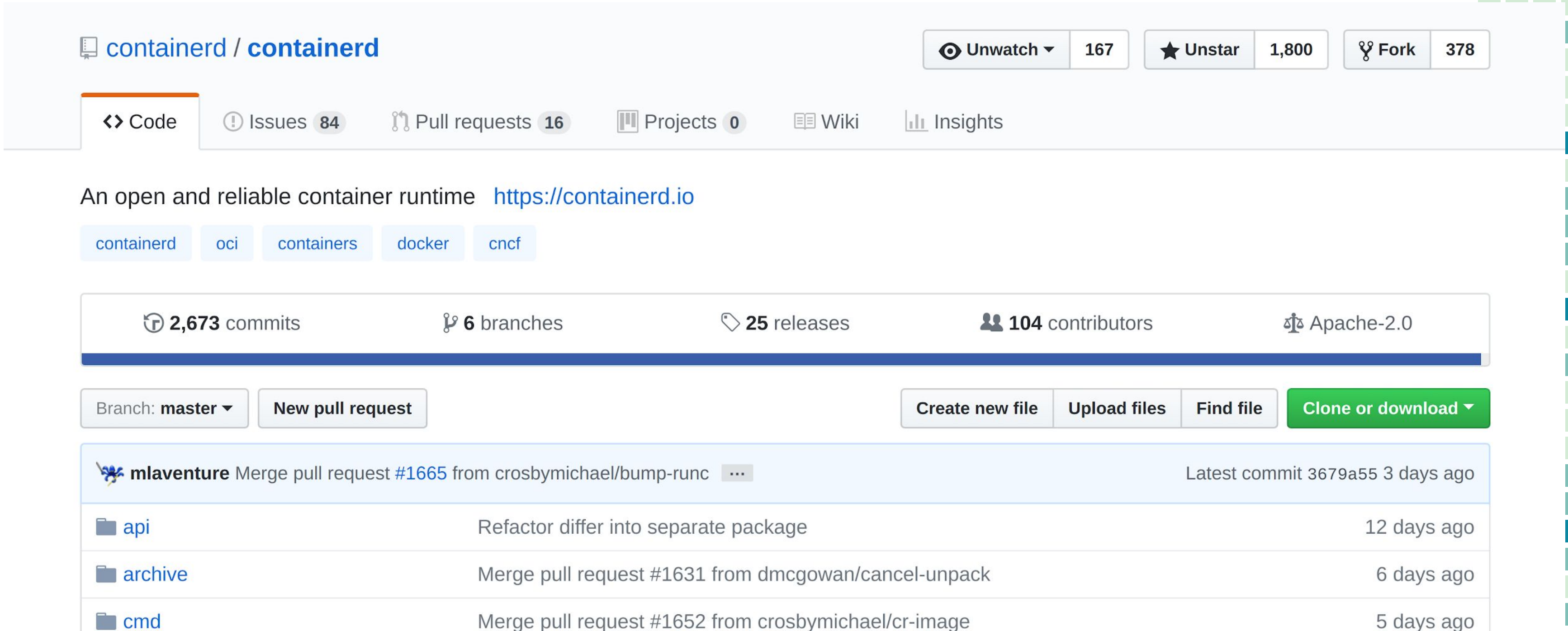
- Inflexible
  - Hard to experiment
- Tightly coupled:
  - container lifecycle
  - Image format
- Primitives can’t be used outside containers

What is containerd?

- A container runtime manager
- Powers Docker and Kubernetes
- Provides primitives to implement containers
- Increments on the internals of Docker

containerd

https://github.com/containerd/containerd



containerd / containerd

Unwatch 167 | Unstar 1,800 | Fork 378

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

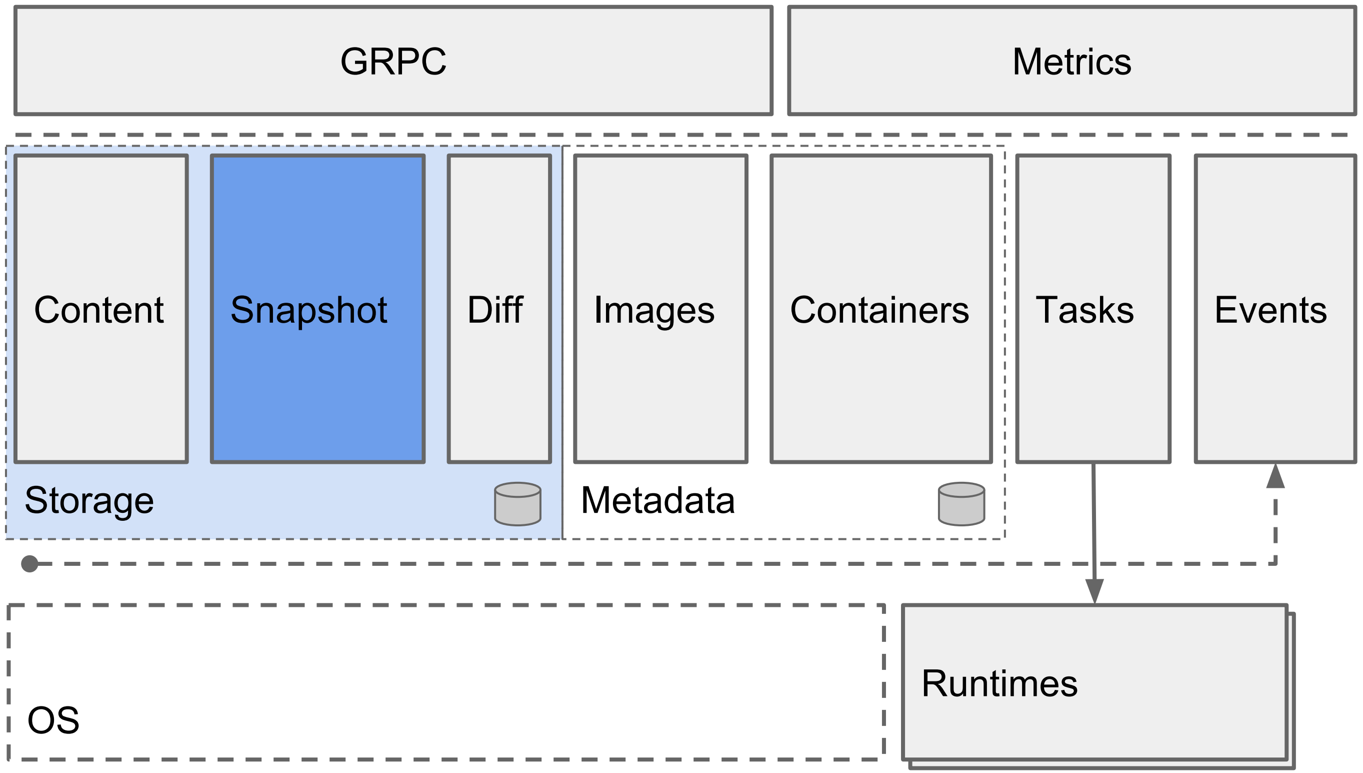
containerd · oci · containers · docker · cncf

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

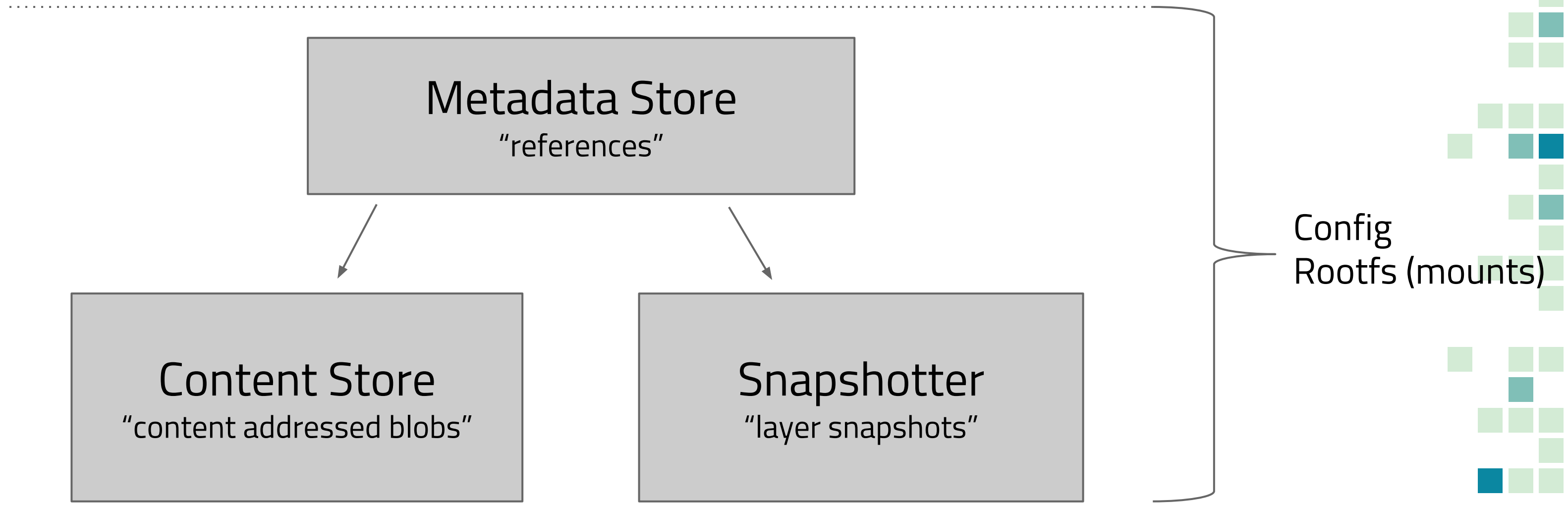
mlaventure Merge pull request #1665 from crosbymichael/bump-runc ... Latest commit 3679a55 3 days ago

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

Architecture



containerd Storage Architecture



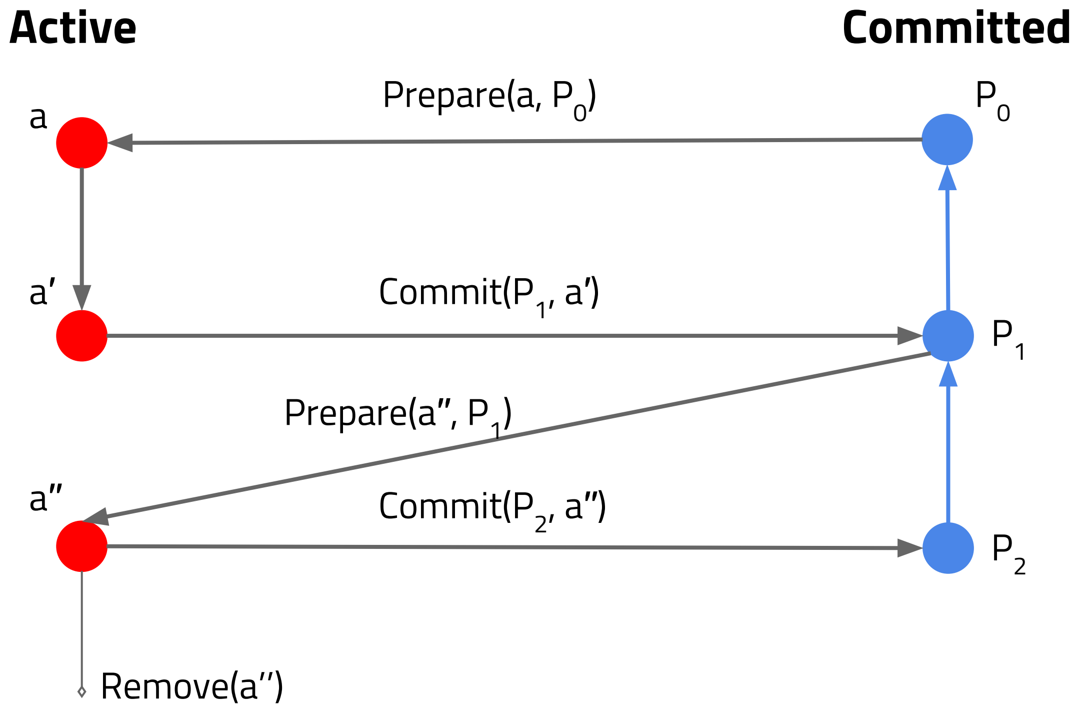
Evolved from Graph Drivers

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

Snapshotter Properties

- 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

Snapshot Lifecycle



Example: Handcrafting Snapshots

```
$ ctr snapshot prepare active0
$ ctr snapshot mounts <target> active0
$ touch <target> /hello
$ umount <target>
$ ctr snapshot commit foo active0
```

Example: Investigating Root Filesystem

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

Demo

```go
type Snapshotter interface {
        Stat(ctx context.Context, key string) (Info, error)
        Update(ctx context.Context, info Info, fieldpaths ...string) (Info, error)
        Usage(ctx context.Context, key string) (Usage, error)
        Mounts(ctx context.Context, key string) ([]mount.Mount, error)
        Prepare(ctx context.Context, key, parent string, opts ...Opt) ([]mount.Mount, error)
        View(ctx context.Context, key, parent string, opts ...Opt) ([]mount.Mount, error)
        Commit(ctx context.Context, name, key string, opts ...Opt) error
        Remove(ctx context.Context, key string) error
        Walk(ctx context.Context, fn func(context.Context, Info) error) error
}
```

```go
type Kind uint8

// definitions of snapshot kinds
const (
        KindUnknown Kind = iota
        KindView
        KindActive
        KindCommitted
)
```

```go
type Info struct {
        Kind    Kind                         // active or committed snapshot
        Name    string                       // name or key of snapshot
        Parent  string            `json:",omitempty"` // name of parent snapshot
        Labels  map[string]string `json:",omitempty"` // Labels for snapshot
        Created time.Time         `json:",omitempty"` // Created time
        Updated time.Time         `json:",omitempty"` // Last update time
}
```

Applying a Layer

```go
// ApplyLayer applies a single layer on top of the given provided layer chain,
// using the provided snapshotter and applier. If the layer was unpacked true
// is returned, if the layer already exists false is returned.
func ApplyLayer(ctx context.Context, sn snapshots.Snapshotter, a diff.Applier, ...) (bool, error) {
        // Prepare snapshot with from parent, label as root
        mounts, err := sn.Prepare(ctx, key, parent.String(), opts...)
        diff, err = a.Apply(ctx, layer.Blob, mounts)
        sn.Commit(ctx, chainID.String(), key, opts...)
}
```

Considerations

Rootless

- Mounts and uid mapping present problems
- Snapshot model doesn’t need to modified

Daemonless

- Snapshot packages can be used without daemon

Status

- **Implementations**: btrfs, overlay, zfs, aufs and native
- **Testsuite**: Full behavioral testing of snapshotters

Going Further

- https://github.com/containerd/containerd
  - Experiment and file bugs
- Documentation: https://godoc.org/github.com/containerd/containerd/snapshots#Snapshotter
-

KubeCon Talks

- **containerd Deep Dive**
  - Friday May 4, 2018 15:40 - 16:15
  - B5-M1+3

Thank You! Questions?

- **Stephen Day**
  - https://github.com/stevvooe
  - @stevvooe
  - Docker Community Slack
  - Kubernetes Community Slack