Containerd Internals: Building a Core Container Runtime

Stephen Day, Docker
Phil Estes, IBM

September 11, 2017
#OSSummit
A Brief History

**APRIL 2016**
Containerd “0.2” announced, Docker 1.11

*Management/Supervisor for the OCI runc executor*

**DECEMBER 2016**
Announce expansion of containerd OSS project

**MARCH 2017**
Containerd 1.0: A core container runtime project for the industry

Containerd project contributed to CNCF
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
Architecture

- GRPC
  - Content
  - Snapshot
  - Diff
  - Images
  - Containers
  - Tasks
  - Events
- Metrics
- Storage
- Metadata
- OS
- Runtimes
Architecture

API Client
(moby, containerd-cri, etc.)

containerd

OS (Storage, FS, Networking)

Runtimes
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
Events

containerd

Publish  Events  Subscribe
Pulling an Image

What do runtimes need?
Image Formats

Docker and OCI

Index (Manifest List)

Manifests:
- linux amd64
- linux arm64
- linux ppc64le
- windows amd64

Manifest
Config:
- C

Layers:
- L₀
- L₁
- ... 
- Lₙ

OCI Spec
process
- args
- env
- cwd
- ...
root
mounts

Root Filesystem
- /usr
- /bin
- /dev
- /etc
- /home
- /lib
digest.FromString("foo") -> "sha256:2c26b46b68ffe68ff99b453c1d30413413422d706483bfa0f98a5e886266e7ae"
digest.FromString("foo tampered") -> "sha256:51f7f1d1f6bebed72b936c8ea257896cb221b91d303c5b5c44073f6e33ab8dd8"
digest.FromString("bar sha256:2c...") -> "sha256:2e94890c66fbc0cca9ad680e1b1c933cc323a5b4bcb14cc8a4bc78bb88d41055"
Image Formats

Docker and OCI

Index (Manifest List)

Manifests:
- linux amd64
- linux arm64
- linux ppc64le
- windows amd64

Digest

Layer File 0

Layer File 1

Layer File N

Manifest Config:

Layers:

Digest
Resolution

Getting a digest from a name:

```
sha256:71cd81252a3563a03ad8daee81047b62ab5d892ebbf71cf53415f29c130950
```
Pulling an Image

Data Flow

Remote

Events

Fetch

Content

Unpack

Images

Snapshots

Pull

Mounts
Remotes  
(hint: think git remotes)

Locators and Resolution

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

Endlessly Configurable!

```go
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)
}
```
Example: Pull an Image

Via **ctr** client:

```
$ export CONTAINERD_NAMESPACE=example

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

$ ctr image ls
...
```

```go
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)
```
Snapshotters

How do you build a container root filesystem?
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 Snapshottter 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
)```
Active

a

a'

a''

Committed

Prepare(a, P_0)

Commit(P_1, a')

Prepare(a'', P_1)

Commit(P_2, a'')

Remove(c)

P_0

P_1

P_2

Snapshot Model
Example: Investigating Root Filesystem

$ ctr snapshot ls
...

$ ctr snapshot tree
...

$ ctr snapshot mounts <target> <id>
Pulling an Image

1. **Resolve** manifest or index (manifest list)
2. **Download** all the resources referenced by the manifest
3. **Unpack** layers into snapshots
4. **Register** the mappings between manifests and constituent resources
Pulling an Image

Data Flow

Remote → Fetch → Pull → Unpack → Events → Content, Images, Snapshots → Mounts
Starting a Container

1. **Initialize** a root filesystem (RootFS) from snapshot
2. **Setup** OCI configuration (config.json)
3. **Use** metadata from container and snapshotter to specify config and mounts
4. **Start** process via the task service
Starting a Container
Example: Run a Container

Via `ctr` client:

```bash
$ export \
    CONTAINERD_NAMESPACE=example

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

$ ctr c
...
Example: Kill a Task

Via `ctr` client:

```bash
$ export \n    CONTAINERD_NAMESPACE=example

$ ctr t kill redis-server

$ ctr t ls

...
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
Customization

- Linux Namespaces -> WithLinuxNamespace
- Networking -> WithNetwork
- Volumes -> WithVolume
Use cases

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

- **FUTURE/POTENTIAL**
  - IBM Cloud/Bluemix
  - OpenFaaS
  - \{your project here\}
Evolution

containerd
Going further with **containerd**

- **Contributing**: [https://github.com/containerd/containerd](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
Moby Summit at OSS NA

Thursday, September 14, 2017

“An open framework to assemble specialized container systems without reinventing the wheel.”

Tickets:
https://www.eventbrite.com/e/moby-summit-los-angeles-tickets-35930560273
Thank You! Questions?

- **Stephen Day**
  - [https://github.com/stevvooe](https://github.com/stevvooe)
  - stephen@docker.com
  - Twitter: @stevvooe

- **Phil Estes**
  - [https://github.com/estesp](https://github.com/estesp)
  - estesp@gmail.com
  - Twitter: @estesp