CnTR

Lightweight OS Containers

Jörg Thalheim, Pramod Bhatotia, Pedro Fonseca, Baris Kasikci

THE UNIVERSITY OF EDINBURGH

UNIVERSITY OF WASHINGTON

UNIVERSITY OF MICHIGAN

USENIX ATC 2018
Container-based virtualization

- Process-level virtualization
  - Namespaces
  - Cgroups
- Extensively used in production
- Lightweight isolation
Why lightweight containers are important?

- Fast deployment
- Low resource usage
- Low build times
Containers are NOT lightweight anymore!

Case study: Top 50 Docker Hub container images

Limitations: Inefficient development and deployment of containers
Why containers are becoming heavyweight?

Build description: e.g. Dockerfile

Container image

Application (MySQL)

Additional tools (Coreutils, ...)

Container

Host

Container images are large due to additional tools!
Additional tools

- **What are these additional tools?**
  - Debuggers, editors, coreutils, shell, etc.

- **Why are they important?**
  - Debugging, inspection, monitoring, management, etc.

Additional tools are NOT used in the common use case
Cntr: Split container images

Original image

- Slim image
  - Slim container
    - Runs the application
    - Common use case

- Fat image
  - Fat container
    - Serves tools to the user
    - Deployed on demand

- CNTR
  - Provides access
Design
Design goals

- Generality
  - Support a wide range of workflows (debugging, inspection, etc.)

- Transparency
  - No modifications to the OS, container engine, and application

- Efficiency
  - No performance overhead on the application
Overview

User

Slim container

Nested namespace

App (MySQL)

Access the application

Access tools via FUSE

Fat container

CntrFS server

Tools (Gdb, coreutils...)

Access tools
Nested namespace

- Nested namespace filesystem view

- Implemented on top of existing OS features
  - Namespaces
  - FUSE

“Fat” image

“Slim” image
Process and CntrFS server can run in different namespaces (container)
Implementation

● Lightweight deployment
  ○ Single 1.2 MB static binary

● Easy to use
  ○ `root@fat-container $ cntr attach slim-container`
    `root@slim-container $`

● Supports all popular containers
  ○ Docker, LXC, LXD, Systemd-nspawn, rkt, etc.
Evaluation
Evaluation

● Questions:
  1. Is the implementation complete?
  2. What are the performance overheads?
  3. How effective is the approach in reducing container image sizes?

● Experimental testbed:
  ○ M4.xlarge VM on EC2
  ○ 100 GB device of type GP2 (SSD-backed network storage)
  ○ Base filesystem: Ext4
#1: Completeness

- Benchmark: Xfstests regression test suite

<table>
<thead>
<tr>
<th>Tests</th>
<th>Supported tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
<td>90 (95.74%)</td>
</tr>
</tbody>
</table>

- Unsupported tests are minor Linux-specific implementation details
- 3 of 4 unsupported tests also don’t work on overlayfs (default on Docker)

**Cntr can already be used in production**
#2 (a): Overheads for the “slim” container

0%

For the common use case of accessing the slim container
#2 (b): Overheads for the “fat” container

Phoronix test suite

Cntr incurs reasonable overhead for management tasks
#3: Effectiveness

Average reduction is 66% of the container size

Containers with static Go binaries

Top 50 containers on Docker Hub

Majority of containers contains unnecessary data

Average reduction is 66% of the container size
Demo!
Demo setup

Host: NixOS

Access
Via CNTR

“Slim”
container: Busybox

$ sudo cntr attach mycontainer

$ sudo docker run --name mycontainer busybox
Summary

● Containers are NOT lightweight in practice
  ○ Limitation: Inefficient development and deployment of containers

● CNTR: Lightweight OS Containers
  ○ Splits the container image into fat and slim parts
  ○ Leverages FUSE to expose additional tools in a nested namespace

Generic + Transparent + Efficient

Try it out!

https://github.com/Mic92/cntr