Carnegie Mellon University
Software Engineering Institute

Advancing Algorithms For File Deduplication Across Containers

NOVEMBER 14–16, 2022

Kevin Pitstick
Senior Software Engineer

Document Markings

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM22-0864

Introduction

Problem: The self-contained nature of container images leads to storage waste, which conflicts with resource limitations at the edge.

Solution: Develop an automated technology that minimizes the storage size of a set of container images for developers to use before deployment to the edge.

Impact: The technology allows DoD organizations to field more capability per SWaP at faster deployment speeds.

Storage Waste in Container Images

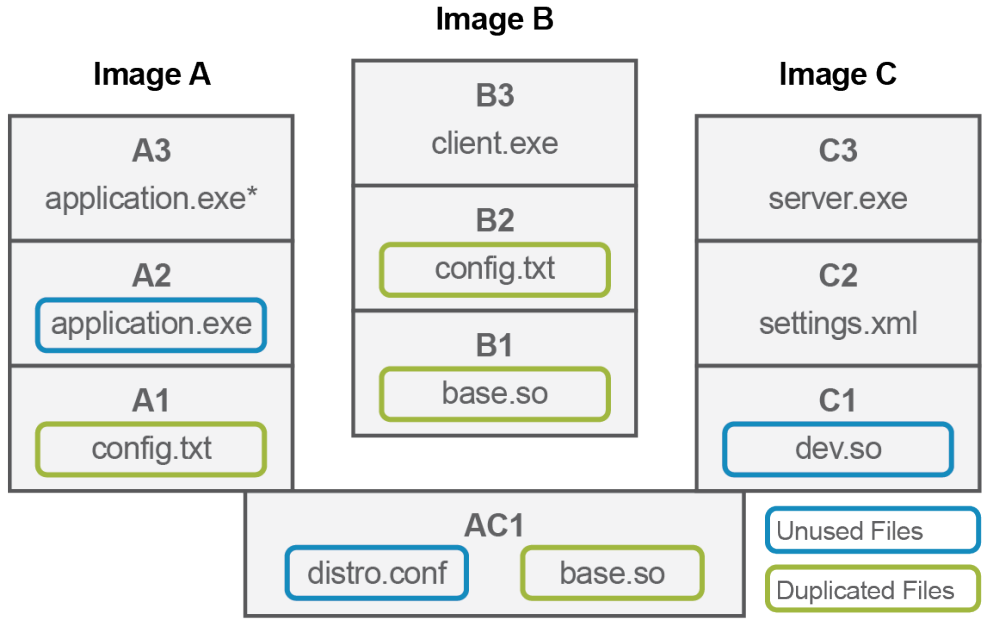
Sources of Storage Waste

unused files

- development files (dev.so)
  - unused distro files (distro.conf)
  - overwritten files (application.exe)

duplicated files

- same files stored in different layers

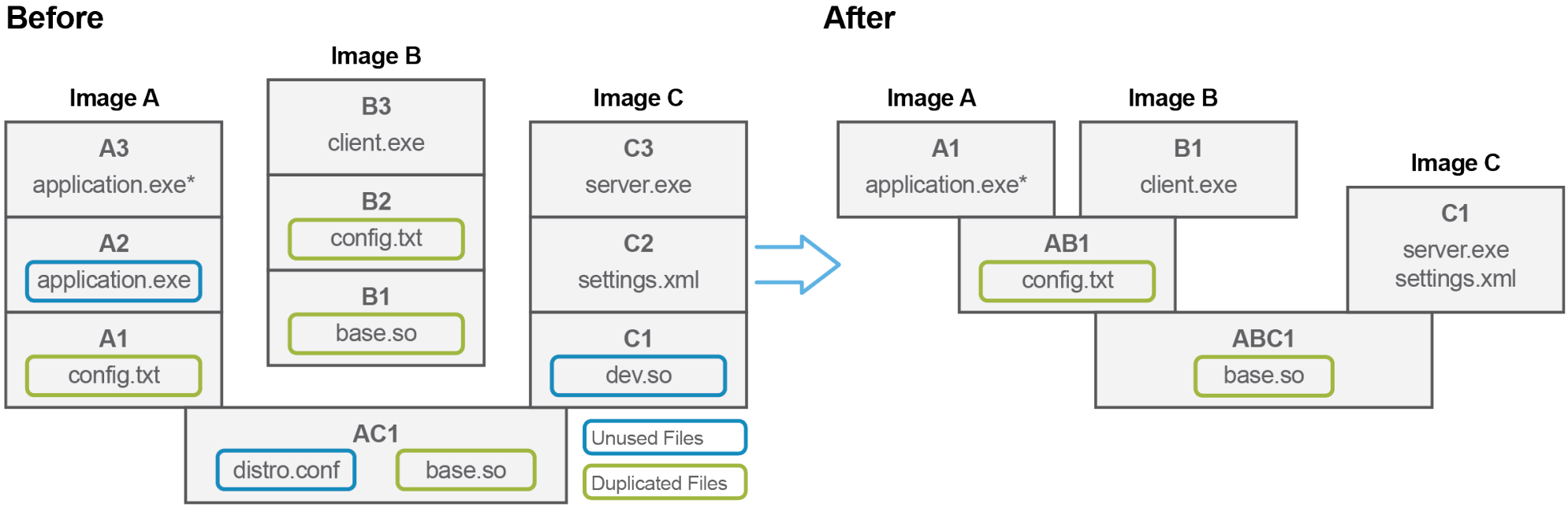


Minimization Terminology

- *pruning* – the removal of unnecessary and unused files from containers
- *deduplication* - combining shared files from multiple images into common container layers

Container Minimization Tool (CMT)

Develop an automated technology that minimizes the storage size of a set of container images that developers can use before deployment to the edge.



Deduplication Algorithm

Overview

Objective: Develop a file deduplication algorithm for reorganizing the layers for a set of images to reduce storage and network costs

- method proposed by Skourtis[1]
- structured as an optimization problem to minimize value of cost function

cost = α*operation

- Cost of too many layers

+ β*storage

- Cost of duplicate files in layers

+ γ*network

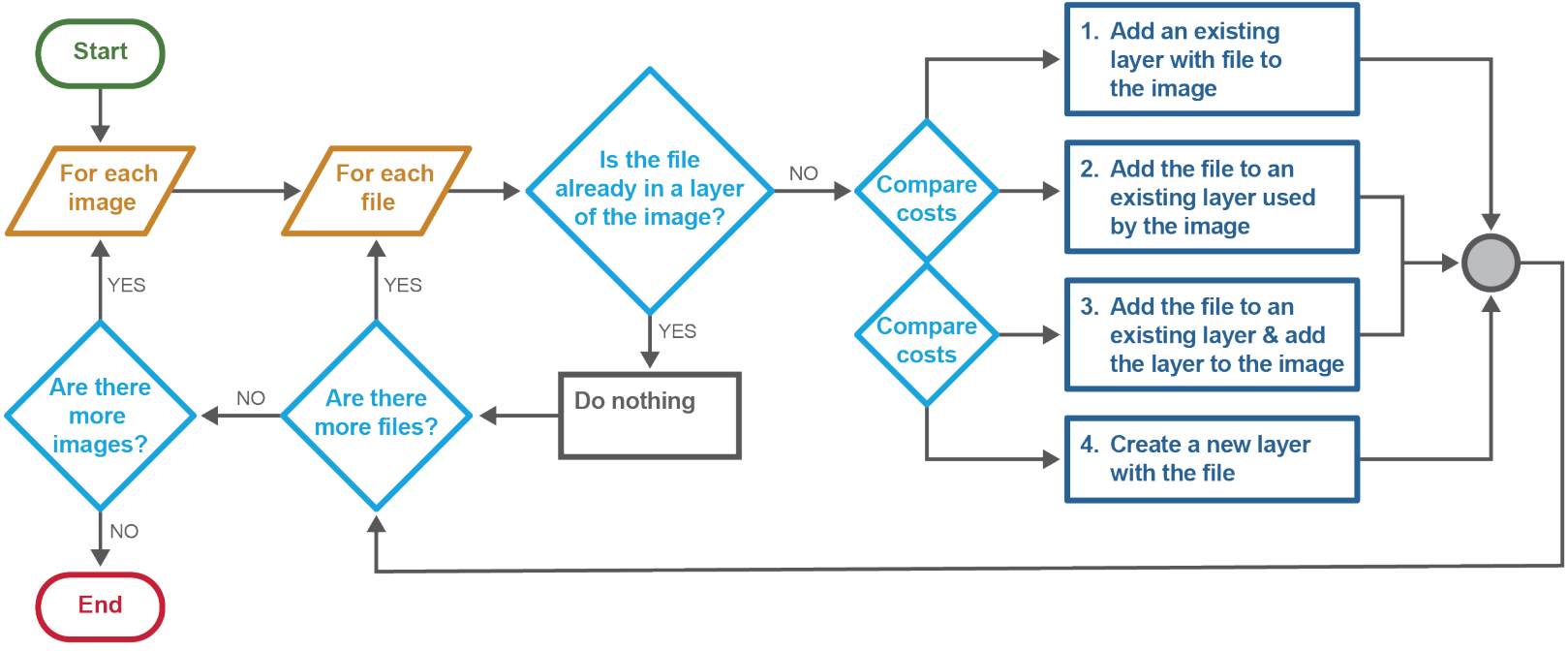
- Cost of too few layers

[1] Skourtis et al. Carving Perfect Layers out of Docker Images. 2019. https://www.usenix.org/system/files/hotcloud19-paper-skourtis_0.pdf

Deduplication Algorithm

Steps

The solution is a greedy algorithm that creates a new set of layers for every image with minimal duplicates.



Deduplication Algorithm

Challenges

- The authors did not release the code.

- The authors implemented an abstract, simplified algorithm with no output of real images.
  - ignores the complexity of creating working final images

- The authors did not release benchmark information on how long it takes to run the algorithm.

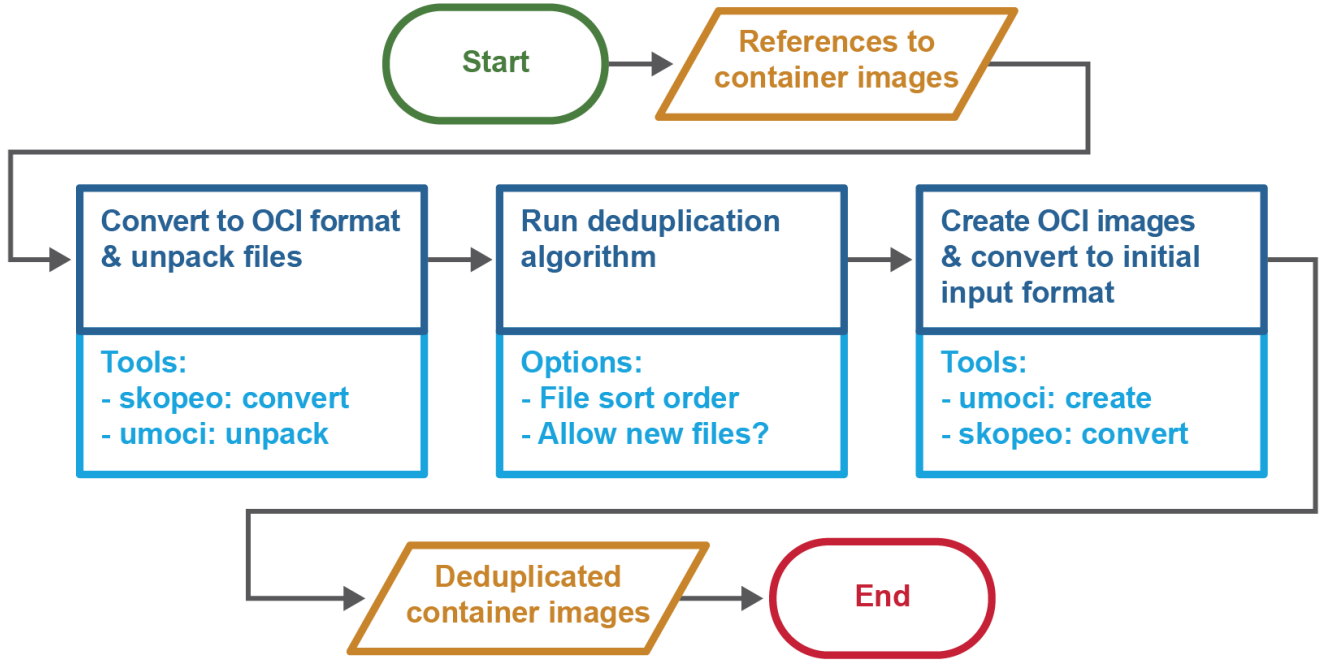
Implementation

Create end-to-end tool that deduplicates images.

- *input:* references to container images
- *output:* deduplicated container images

Utilize OCI image format to increase applicability.

- Use open source tools skopeo & umoci



Experiments

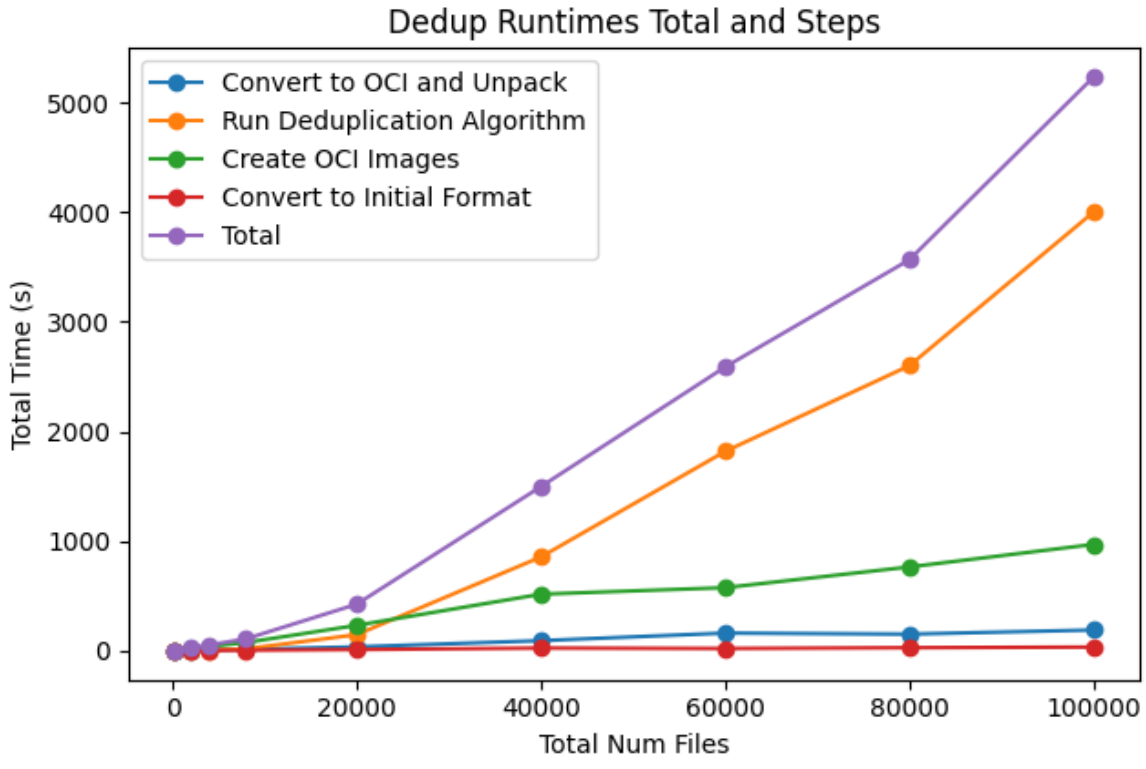
Time Profiling

Goal: Measure effect of number of files on time needed to deduplicate.

- 2 images
- equal number of files per image
- 10 files duplicated between images

Results:

- The time is $O(n^2)$ on number of files.
- ”Run Deduplication Algorithm” takes the largest percentage of time.
- After multiple optimizations, we reduced algorithm runtime by 30x from our initial implementation.



Experiments

Deduplication Evaluation

Goal: To measure the ability to deduplicate depending on amount of duplicated files

- We generated images with different percentages of duplicated files (size 100kb) for two images: 0% to 100%.
- We kept the total number of files (1000 files) constant between experiments.

Results:

- For all cases, duplication is 0% for all deduplicated images.

Experiments
Real Images

We ran both the time profiling and deduplication evaluation experiments on real sets of images.

Test Case 1: ClearML[1]

- 5 unique images (8 services)
- 64,138 files (2872 MB)
  - 3,948 shared files (25.2 MB)
  - All of them are duplicated

Results:

- The method deduplicated images in 24 minutes (on server with 3.1 Ghz Xeon processor).
- The method achieved full deduplication when files were processed in order of most used.
  - 2872 MB => 2847 MB
- When files were processed alphabetically, results were worse (only 1.9 MB of 25.2 MB deduplicated).

[1] https://clear.ml/

Experiments
Real Images

We ran both the time profiling and deduplication evaluation experiments on real sets of images.

Test Case 2: Stan's Robot Shop[1]

- 10 unique images (10 services)
- 225,915 files (4.62 GB)
  - 112,687 shared files (3.3 GB)
  - 27,033 of those are duplicated (724 MB)

Results:

- The method deduplicated images in 81 minutes (on server with 3.1Ghz Xeon processor).
- The method achieved full deduplication when files were processed in order of most used.
  - 4.62 GB => 3.89 GB
- When files were processed alphabetically, results were worse (only 77 MB of 724 MB deduplicated).

[1] https://github.com/instana/robot-shop

Use Cases for DoD Systems

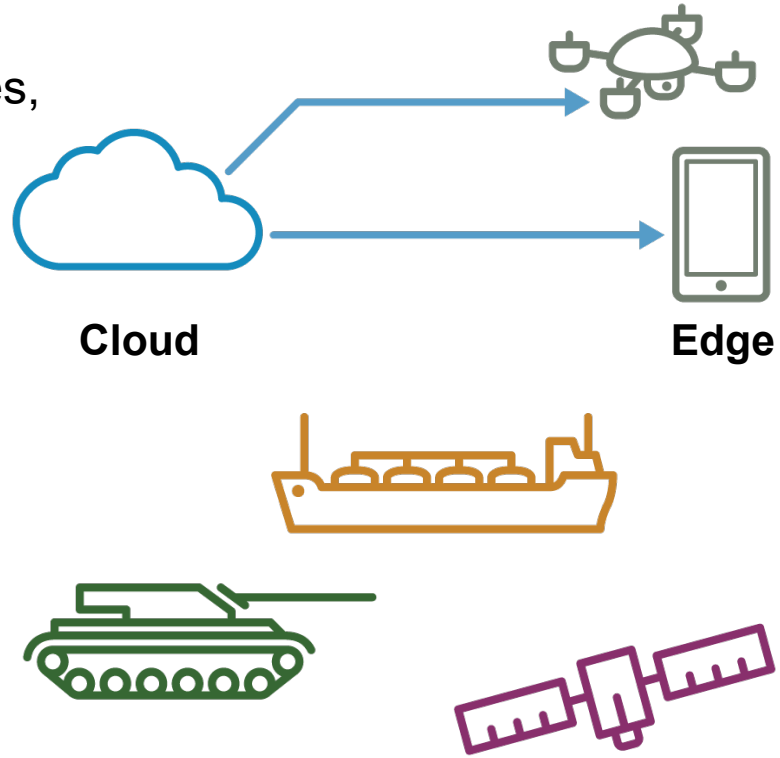
Who: organizations deploying containers to very SWAP-constrained edge systems (e.g., legacy systems, satellites, drones, etc.)

When: Organizations can use image deduplication

- when existing build practices utilize different kinds of base images.
- when container images from multiple organizations are deployed together.
- after pruning container images to rebuild shared base layers.

Why: Our results indicate that we can reduce container image storage usage and update bandwidth.

- by up to 5-15% for multi-container deployments
- by up to 10-30% for pruned container deployments



Next Steps

Algorithm

- Increase algorithm speed to accommodate larger sets of images (e.g., parallelize).
- Add an optimized special case when developers update only a subset of images.

Testing

- Test and evaluate the algorithm with other sets of real-world images.
  - More images/files, higher duplication percentage

Release

- Put code through release review and release it on GitHub.
- Publish testing results.

Carnegie Mellon University Software Engineering Institute

Team

Kevin Pitstick
Principal Investigator
Senior Software Engineer

Sebastián Echeverría
Senior Software Engineer

Brandon Born
Associate Software Engineer

Brent Clausner
DevOps Engineer

Carl Gruhn
Assistant Software Engineer

Gary Zhang
Software Developer Intern

Lihan Zhan
Assistant Software Engineer

Carnegie Mellon University
Software Engineering Institute

Questions?

Please reach out at info@sei.cmu.edu