GRAMMATECH

Minimizing Software Containers: Forethought or Hindsight

Jonathan Dorn, Denis Gopan, Deby Katz, Lucja Kot, Junghee Lim,
Adam Seitz, Thomas Wahl

October 18, 2024

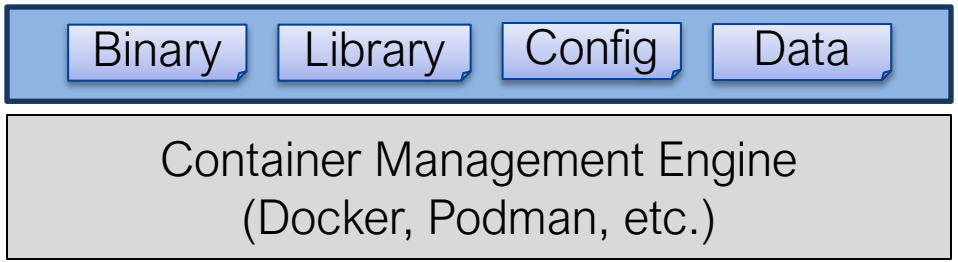
Overview

- Bloat in software containers.
- Hindsight and its limitations.
- Potential for forethought.
- Open questions and directions.
- Discussion.

Container Bloat

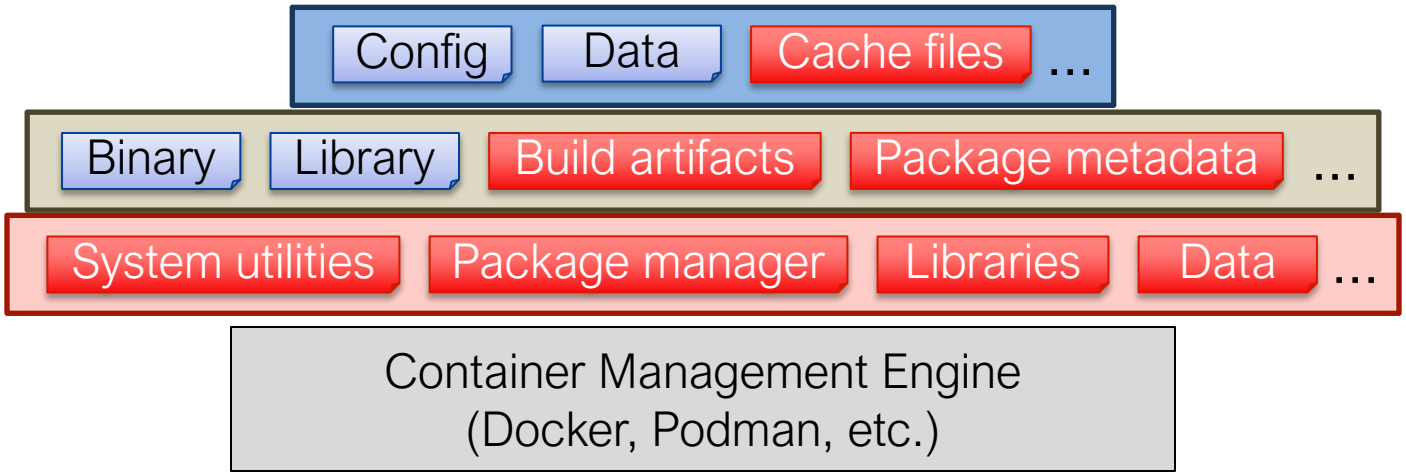
Containers in Theory

- Wrap application in a lightweight, portable environment.

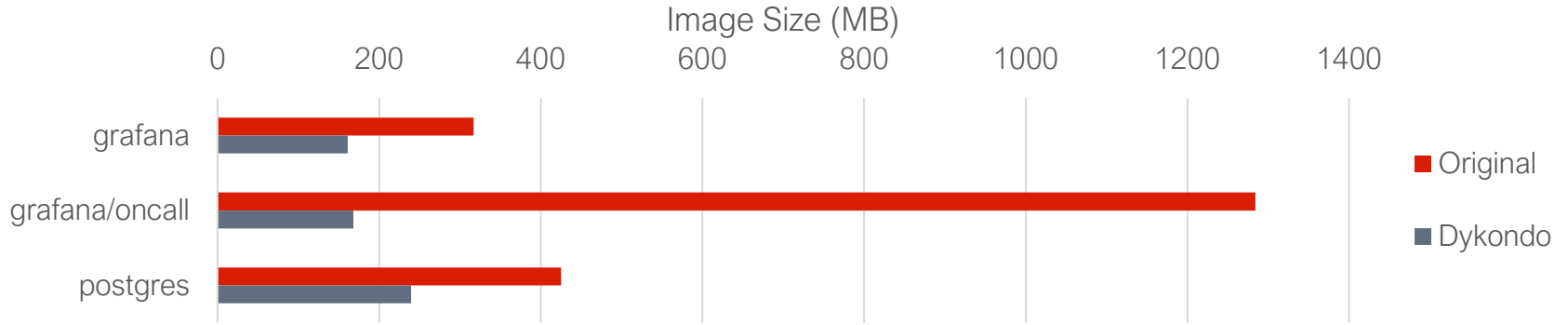


Containers in Reality

- Wrap application in a ~~lightweight~~, portable? environment.

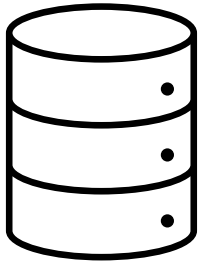


Bloat in Open-Source Containers

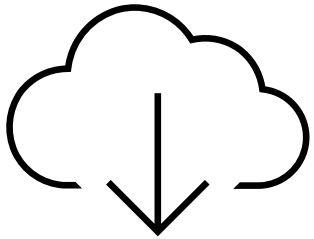


Application	Version	Type	Language	Bloat
Grafana	9.5.3	Web Dashboard	Golang, Javascript	49%
Grafana OnCall	1.3.80	On-call Management	Python	87%
PostgreSQL	16.1	Database	C	44%

Bloat Impacts



Wasted Storage



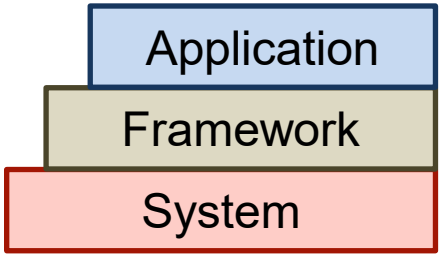
Large Downloads

Expanded
Attack Surface

Container Debloating

Typical Scenario

- Simplified REST application.
  1. System layer provides common Linux utilities.
  2. Framework layer provides REST infrastructure.
  3. Application layer implements endpoints.
- Four distinct actors:
  - Three development teams.
  - One end-user.

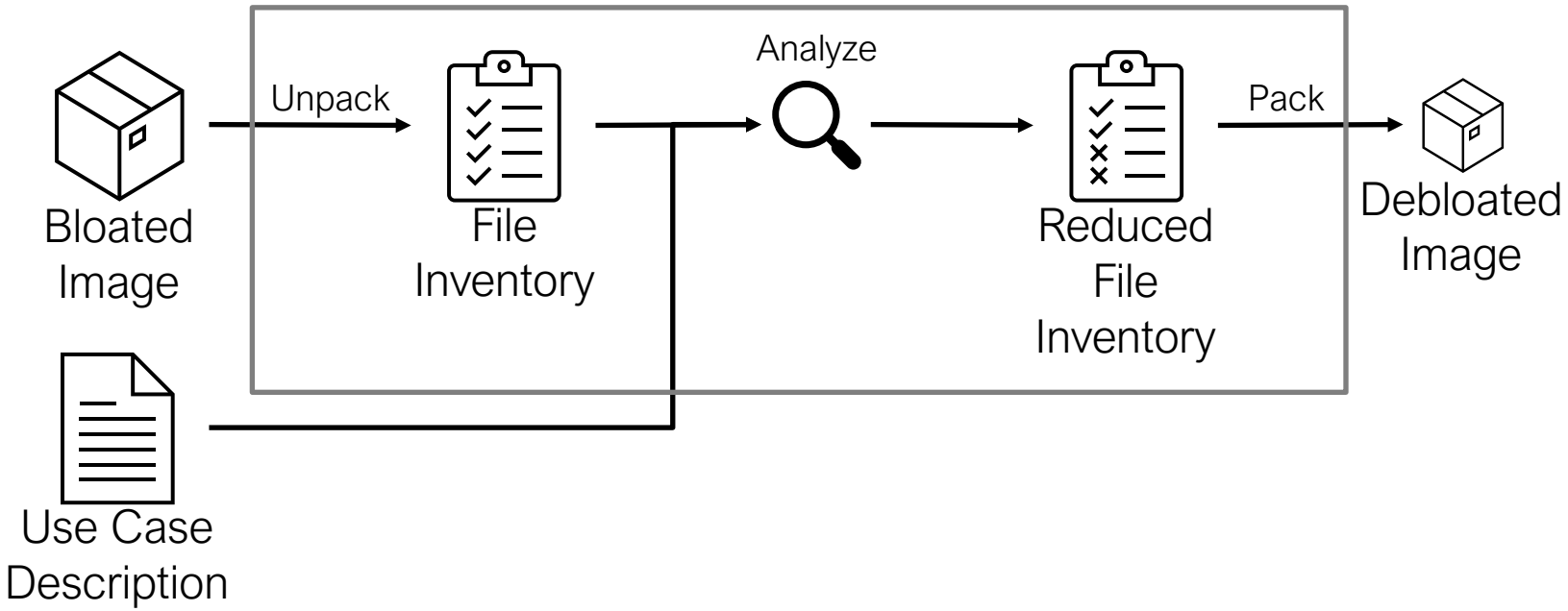


Traditional Bloat Mitigation

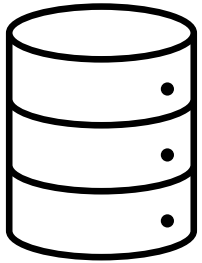
- Dominated by (manual) best practices.[1]
- Mostly containerfile design:
  - Multi-stage builds—exclude bloat in final stage.
  - Explicitly remove caches and build artifacts.
  - Use fine-grained dependencies when possible.
    - Depends on upstream developer effort.

[1] E.g., https://docs.docker.com/build/building/best-practices

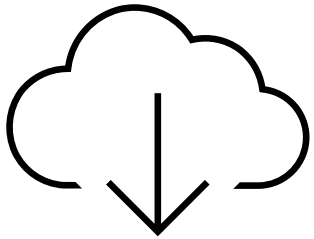
Hindsight: Automatic Bloat Repair



Problem Solved?



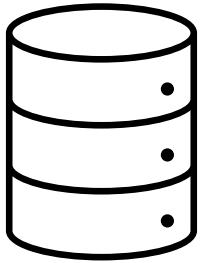
Wasted Storage



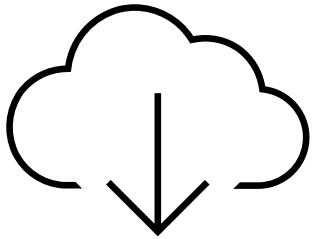
Large Downloads

Expanded
Attack Surface

Problem Solved? Yes



Wasted Storage



Large Downloads

Expanded
Attack Surface

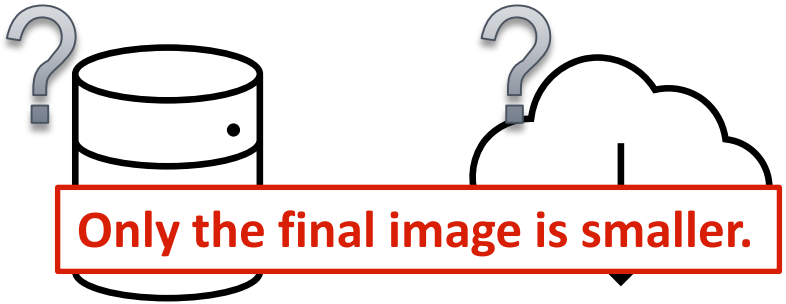
Problem Solved? Sort Of

Wasted Storage

Large Downloads

Expanded Attack Surface

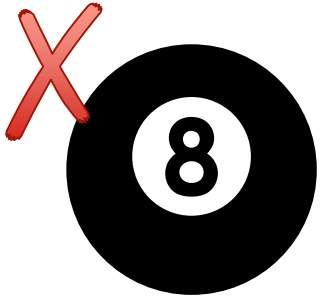
A New Challenge Arises



Wasted Storage

Large Downloads

Reduced
Attack Surface



Oracle Problem

The Oracle Problem

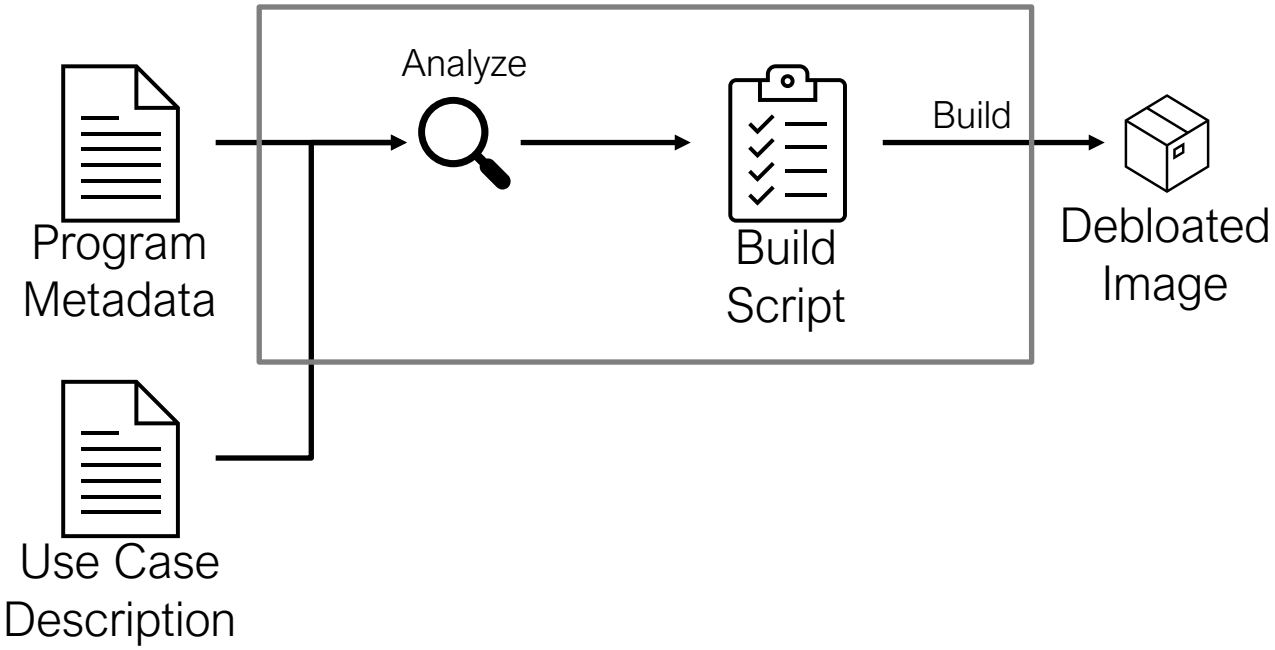
- Software testing:
  - Determine whether program behavior on an arbitrary input is correct.
- Container debloating:
  - Determine whether reduced container is correct.
  - Responsibility for checking is on the debloater: **end-user** or **Application developer**.

Foresight: Bloat-Free Synthesis

Goal

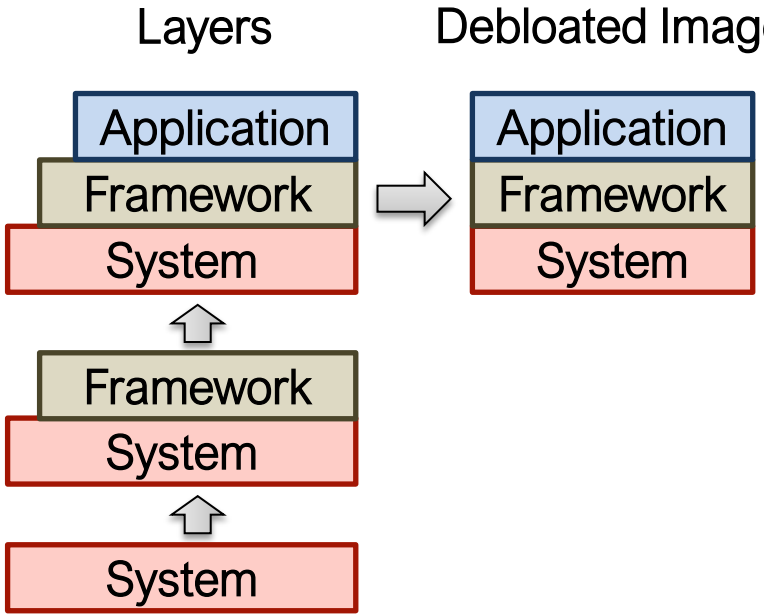
Construct minimal containers in the first place.

Automatic Bloat-Free Synthesis

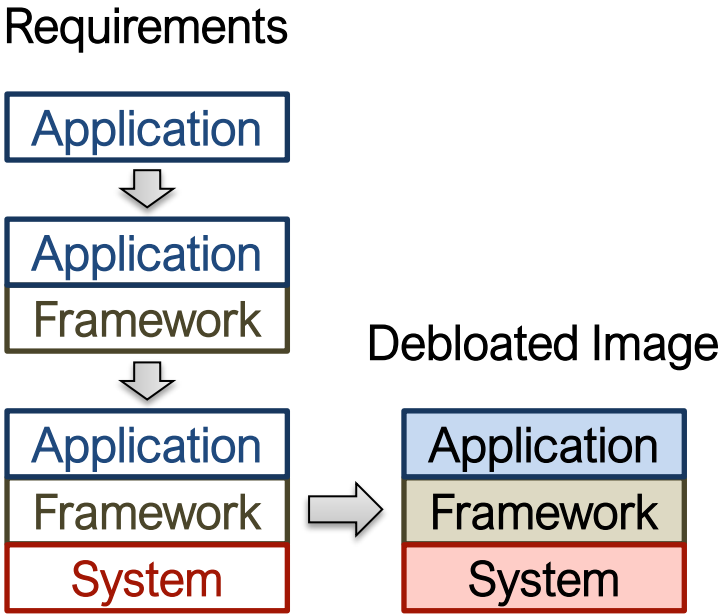


Approach Comparison

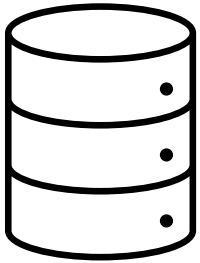
Repair



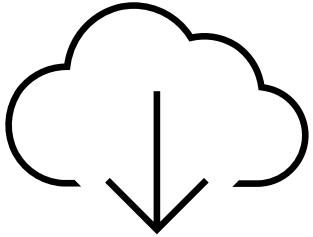
Synthesis



Problem Solved?



Wasted Storage

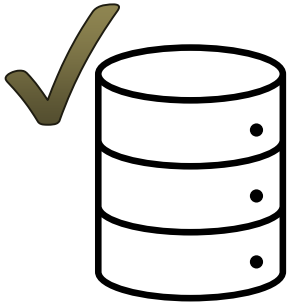


Large Downloads

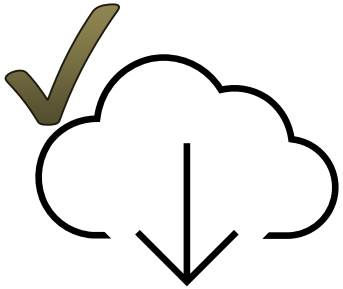
Reduced
Attack Surface

Oracle Problem

Problem Solved?



Wasted Storage

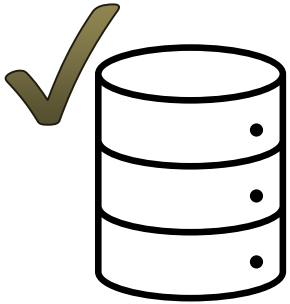


Large Downloads

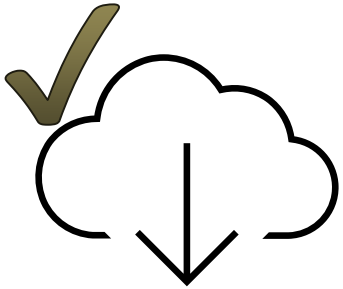
Reduced
Attack Surface

Oracle Problem

Problem Solved?



Wasted Storage



Large Downloads

Reduced
Attack Surface

Oracle Problem

The Specification Problem

- Oracle problem → specification problem.
  - Container will be correct if dependencies and build are correctly specified for each layer.
- Developers, not end-users, create specs.
  - Responsibility distributed across layers.
- Specs used when building debloated image.
  - Compatibility is critical.

Open Questions and Research Directions

- How do we represent specifications?
- Can dev tools help create specifications?
- How can layer developers protect IP?
- Hybrid workflows: synthesize some layers, repair the rest.
- What are the implications for software development and deployment practices?

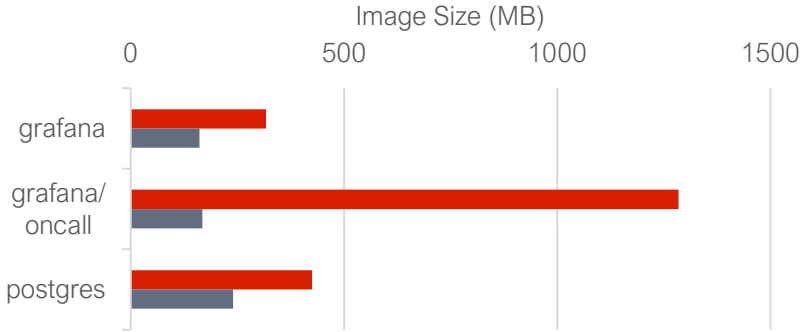
Challenge: Disclosure and Build Costs

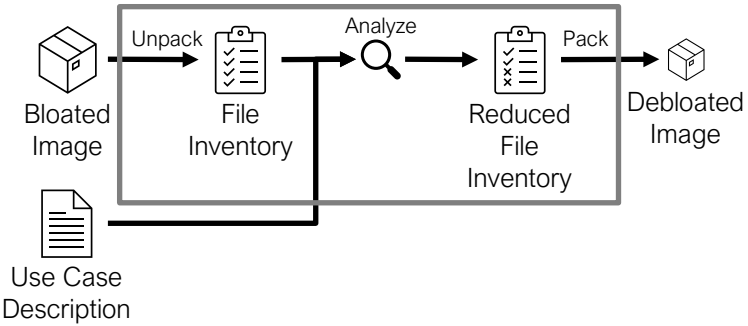
- Layers cannot be built until end-user requirements are known.

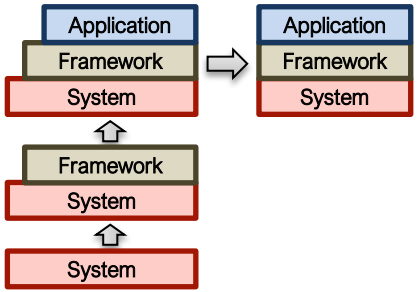
- Option 1: End-user builds all layers.
  - How can layer developers protect their IP?
- Option 2: Layer developer builds their own layer.
  - Significant ongoing resource costs.
  - Significant long-term commitment.

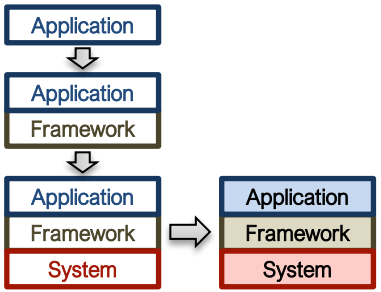
Discussion

Discussion









GRAMMATECH