



RESEARCH REVIEW 2024

**Carnegie  
Mellon  
University**  
Software  
Engineering  
Institute

# Vessel: Reproducible Container Builds

**NOVEMBER 13, 2024**

Kevin Pitstick

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

©2024 Carnegie Mellon University



# Document Markings

Copyright 2024 Carnegie Mellon University.

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.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

References herein to any specific entity, product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute nor of Carnegie Mellon University - Software Engineering Institute by any such named or represented entity.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at [permission@sei.cmu.edu](mailto:permission@sei.cmu.edu).

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM24-1486

# Introduction

**Problem:** Lack of reproducibility in state-of-the-practice container build processes inhibits trust in build outputs, conflicting with best supply chain security practices:

- NSA/ODNI/CISA's developer guide: *Securing the Software Supply Chain*
- Level 4 of the Supply Chain Levels for Software Artifacts security framework

**Solution:** Analyze open-source and DoD container build environments to identify common reproducibility issues and prototype tools for automatically detecting and repairing these issues.

**Impact:** This reproducibility technology allows DoD organizations to detect malicious tampering during their build processes, adding trust to their build products.

# 2020 SolarWinds Hack

**Summary:** The Sunspot malware ran on a build server waiting for a build command, injected the backdoor into the source code prior to compilation, and then restored the original source after build completion.

**Damages:** The malware was downloaded by 18,000 customers. A 2021 report estimated monetary damages at \$90 million.

**Prevention:** Multiple analyses have noted that reproducible builds with hash validation of the build output would have detected the tampering. This attack led SolarWinds's R&D SVP to announce in 2021 efforts to adopt reproducible build.

*“...it appears that the **source code** wasn't compromised, and the **distribution system** wasn't compromised. Instead, the **build system** was compromised. This is exactly the kind of attack that is countered by reproducible builds. Thus, the recent **SolarWinds** subversion is a very good argument for why it's important to have **reproducible builds** (and to verify builds using reproducible builds).”*






—David Wheeler, Director of Open Source Supply Chain Security at the Linux Foundation

# Background

**Reproducible build:** When given the same build environment, any party can recreate bit-by-bit identical copies of all specified artifacts.<sup>1</sup>

**Reproducibility failures:** Instances when two builds with the same build environment produce different artifacts.

Failures occur when a **reproducibility issue** aligns with a specific **change in external factors**.

|  |   |   |  |   |
|--|---|---|--|---|
|  |  |  |  |  |
| Reproducibility issue  |   | External factor   |  | Reproducibility failure   |
| Build script downloads a package without version specified                       |   | New package release   |  | Container images have different versions of the package                             |
| Build script generates a file with a timestamp                                   |   | Time changes  |  | Container files have different timestamps   |
| Build script creates an archive of a directory                                   |   | Filesystem lists files in a different order                                       |  | Container archive file lists files in a different order                             |
| Build script generates a UUID  |   | Pseudo-random number generator  |  | Container files contain different UUIDs   |

1. Paraphrased from <https://reproducible-builds.org/docs/definition/>

# State of the Art

| Dockerfile linting/repair tools  | Reproducible Builds project <sup>1</sup>  |
|--|---|
| <b>Focus:</b> Detecting best practice violations in Dockerfiles  | <b>Focus:</b> Guidance and support tools for building reproducible packages for package managers  |
| <b>Tool examples:</b> hadolint, <sup>2</sup> shipwright <sup>3</sup>   | <b>Tool examples:</b> disorderfs, strip-nondeterminism, reprotest   |
| <b>Gaps:</b> <ul style="list-style-type: none"><li>• Only handles identify/repair for 1 issue (package versioning)</li><li>• Does not address non-Dockerfile build files</li></ul> | <b>Gaps:</b> <ul style="list-style-type: none"><li>• Only handles identify for 2 issues (unstable input order, archive file non-determinism) and repair for 1 issue (archive)</li><li>• Guidance does not address container reproducibility</li></ul> |

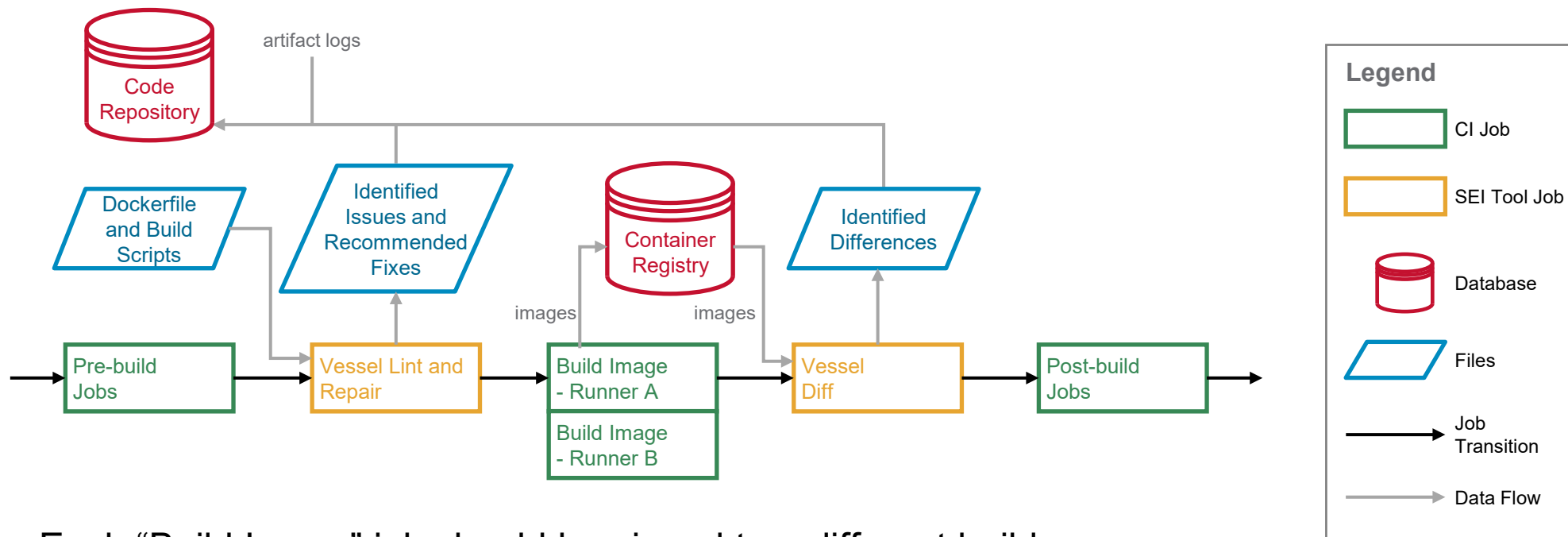
### Addressing the Gaps

Develop tooling to

- detect benign and potentially malicious reproducibility issues in container images
- repair reproducibility issues in container build files

[1] Lamb, C. Reproducible Builds. n.d. <https://reproducible-builds.org/>  
[2] Lorenzo, J. Haskell Dockerfile Linter. GitHub. Nov. 2022. <https://github.com/hadolint/hadolint>  
[3] Henkel, J. et al. Shipwright: A Human-in-the-Loop System for Dockerfile Repair. In *Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. May 2021.

# Vessel Overview



- Each “Build Image” job should be pinned to a different build runner.
- Artifact logs from Vessel jobs can provide reporting for Certificate to Field as well as feedback to developers for improving Dockerfiles and build scripts.

# Reproducibility Issues

## High-Level Classes of Reproducibility Issues

- Lack of explicit dependency version pinning
- Use of current date/time
- Installation of recommended dependencies
- Using host environment variables
- Not clearing package manager caches
- Reliance on filesystem ordering
- Use of randomly generated data
- Reliance on platform-specific data

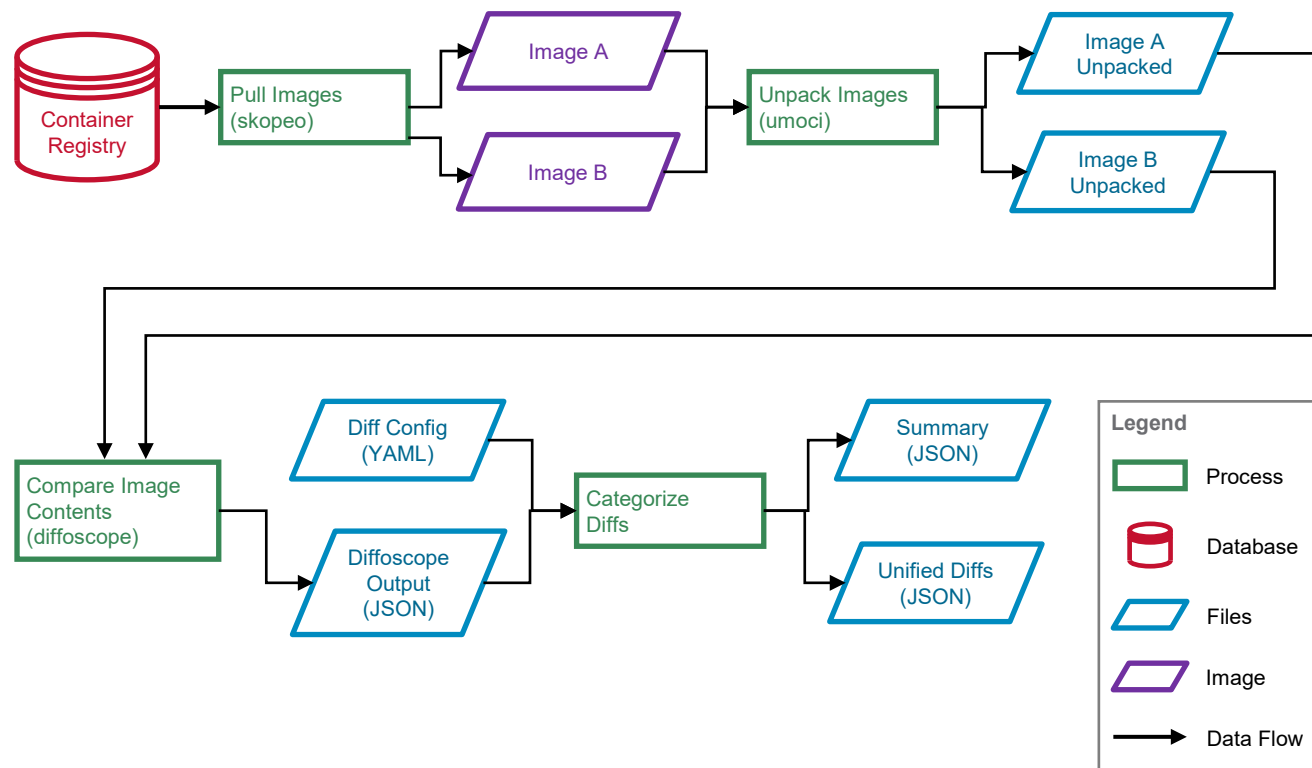
## External Factors

- External dependency changes
- Build platform changes
- Randomness
- Current time changes
- Network availability



# Diff Tool

## Overview



**Goal:** Identify reproducibility issues by comparing differences in two container images. Categorization helps separate benign vs. malicious differences.

### Diff Tool Stages

1. *Pull Images*: Run skopeo to pull in Open Container Initiative (OCI) images from a variety of sources (e.g., Docker registries, OCI tars).
2. *Unpack Images*: Run umoci to unpack OCI images into a folder structure for comparison.
3. *Compare Image Contents*: Run diffoscope to generate human-readable comparisons of many binary formats, such as archive files (tar, zip), database files (sqlite, bdb), PDFs, and Git metadata.
4. *Categorize Diffs*: Categorize diffs into reproducibility issues based on regex mapping across file path, file type, comparison command, diffoscope comment, and unified diff.

# Diff Tool

## Configuration

### Diff flag format:

- *id*: unique identifier associated with reproducibility issue
- *description*: more details on the issue
- *filepath*: regex to match path of files in both images
- *filetype*: regex to match type (libmagic) of files in both images
- *command*: regex to match comparison command
- *comment*: regex to match comparison comment(s)
- *indiff*: regex to match lines in both files

### Example Flags:

```

1 - id: 'pyc-format'
2   description: 'Pyc format differences'
3   filepath: '\.pyc$'
4   filetype: '.'
5   command: '.'
6   comment: 'but no file-specific differences were detected'
7   indiff: '.'

```

```

1 - id: 'git-index-modified'
2   description: 'Git index modified date/times'
3   filepath: '.*\/\.git\/index'
4   filetype: 'Git index'
5   command: '.'
6   comment: '.'
7   indiff: 'Modified:\s*\d+[.]\d+'

```

```

1 - id: 'file-time'
2   description: 'A timestamp diff caused when files are modified at different times.'
3   filepath: '.'
4   filetype: '.'
5   command: 'stat {}'
6   comment: '.'
7   indiff: 'Modify: [0-9]{1,4}-[0-9]{1,2}-[0-9]{1,2}[ ]+[0-9]{1,2}\: [0-9]{1,2}\: [0-9]{1,2}(\. [0-9]{1,2})?'

```

# Diff Tool

## Output

### summary.json

```

1 {
2   "summary": {
3     "unknown_issues": 1626,
4     "flagged_issues": 70014
5   },
6   "Diffs": [
7     {
8       "source1": "/data/vessel/kevin/dockerfile-dataset/experiments/apprtc/2024-10-01_09-18-29_vs_2024-10-01_21-45-04/2024-10-07_10-32-51/data/umoci-unpack-apprtc.2024-10-01_09-18-29/rootfs/cert"
9     },
10    {
11      "source2": "/data/vessel/kevin/dockerfile-dataset/experiments/apprtc/2024-10-01_09-18-29_vs_2024-10-01_21-45-04/2024-10-07_10-32-51/data/umoci-unpack-apprtc.2024-10-01_21-45-04/rootfs/cert"
12    },
13    {
14      "unified_diff_id": 1,
15      "command": "stat {}",
16      "flagged_issues": [
17        {
18          "id": "RI-2-8",
19          "description": "Logging time difference.",
20          "minus_file_line_number": 6,
21          "plus_file_line_number": 6,
22          "minus_diff_line_number": 6,
23          "plus_diff_line_number": 7,
24          "minus_matched_str": "2024-10-01 19:31:10.",
25          "plus_matched_str": "2024-10-02 01:50:33."
26        }
27      ]
28    }
29  ]
30 }

```

### unified\_diffs.json

```

1 {
2   "1": [
3     "@@ -1,8 +1,8 @@",
4     " ",
5     "   Size: 4096          \tBlocks: 8           IO Block: 4096   directory",
6     " Links: 2",
7     " Access: (0755/drwxr-xr-x) Uid: (   0/   root) Gid: (   0/   root)",
8     " ",
9     "-Modify: 2024-10-01 19:31:10.000000000 +0000",
10    "+Modify: 2024-10-02 01:50:33.000000000 +0000",
11    " ",
12    " ",
13  ],

```

# Dataset Creation

```

1  {
2    "success": [
3      {
4        "name": "kafka-docker",
5        "url": "https://github.com/wurstmeister/kafka-docker",
6        "commit_hash": "901c084811fa9395f00af3c51e0ac6c32c697034",
7        "programming_language": "Shell",
8        "last_modified": "2024-05-08T12:11:08Z"
9      },
10     {
11       "name": "buildkit",
12       "url": "https://github.com/moby/buildkit",
13       "commit_hash": "6c7ea85dc9f8c4aecc9372cf5a9462acc736e02c",
14       "programming_language": "Go",
15       "last_modified": "2024-08-02T08:28:23Z"
16     }
17   ]
18 }

```

```

1  {
2    "failure": [
3      {
4        "name": "kafka-manager-docker",
5        "url": "https://github.com/sheepkiller/kafka-manager-docker",
6        "commit_hash": "N/A",
7        "programming_language": "Shell",
8        "last_modified": "2019-03-01T14:21:42Z"
9      },
10     {
11       "name": "dockerfile-from-image",
12       "url": "https://github.com/CenturyLinkLabs/dockerfile-from-image",
13       "commit_hash": "N/A",
14       "programming_language": "Ruby",
15       "last_modified": "2020-01-24T17:21:18Z"
16     }
17   ]
18 }

```

The team used an automated script to search GitHub for repositories matching the following criteria:

- “Dockerfile” in base directory of repository
- at least 50 stars

The script then clones the repositories, attempts to build images, and tracks success/failure in a JSON file. This JSON file provides a record (with commit ids) of all repositories in the sample dataset. Only successfully built repositories are included in the sample dataset.

# Dataset Statistics

Total repositories in dataset matching search filters: 727

Total repositories pulled for sample dataset: 110 (~15% of population)

| Language Distribution | Full Dataset | Sample Dataset |
|-----------------------|--------------|----------------|
| Shell                 | 32.2%        | 30.1%          |
| Dockerfile            | 23.7%        | 24.3%          |
| Python                | 11.3%        | 9.7%           |
| Go                    | 7.0%         | 12.6%          |
| JavaScript            | 5.4%         | 9.7%           |
| TypeScript            | 2.2%         | 1.0%           |
| Java                  | 1.9%         | 0.0%           |
| Makefile              | 1.8%         | 1.0%           |
| HTML                  | 1.3%         | 1.0%           |
| Ruby                  | 1.2%         | 1.0%           |
| Jupyter Notebook      | 1.0%         | 0.0%           |
| C#                    | 0.9%         | 1.0%           |
| ...                   | ...          | ...            |

| Last Modified Year | Full Dataset | Sample Dataset |
|--------------------|--------------|----------------|
| 2024               | 43.5%        | 43.3%          |
| 2023               | 14.4%        | 21.2%          |
| 2022               | 9.6%         | 8.7%           |
| 2021               | 7.8%         | 5.8%           |
| 2020               | 6.9%         | 7.7%           |
| 2019               | 6.3%         | 5.8%           |
| 2018               | 5.0%         | 4.8%           |
| 2017               | 4.3%         | 1.0%           |
| 2016               | 1.9%         | 1.0%           |
| 2015               | 1.3%         | 1.0%           |
| 2014               | 0.8%         | –              |
| 2013               | 0.1%         | –              |

# Experiments

## Steps

- For each repository in the open-source dataset,
  - build OCI image 1 using BuildKit
  - build OCI image 2 using BuildKit
  - run Vessel Diff tool on both images

## Results

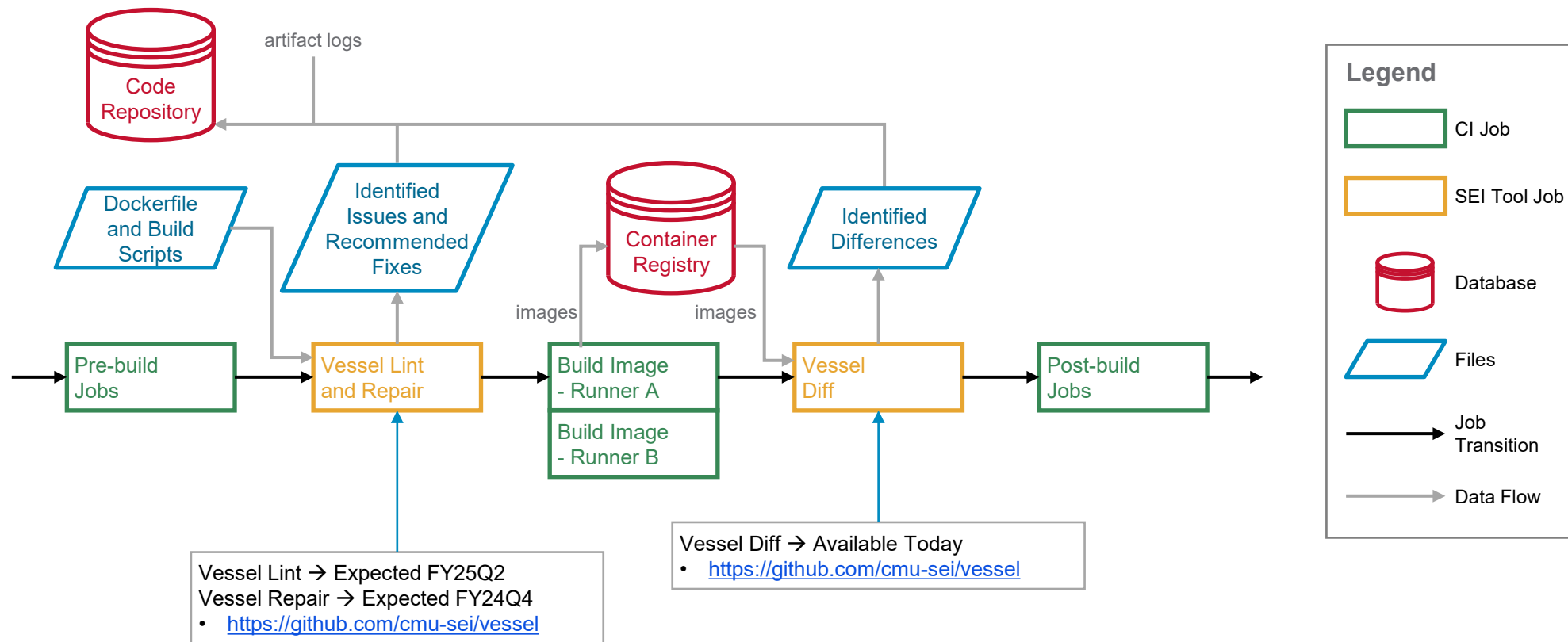
On average, ~91% of differences in each repository were flagged as known reproducibility issues.

- 84% of flagged differences were Golang binary debugging metadata.
- 7% of flagged differences were modified file or logging timestamps.
- 5% of flagged differences were Golang build cache files.
- 3% of flagged differences were time and inode differences in Git index files.

Three repositories were outliers, each with ~1% of differences flagged. The differences were traced back to different versions of packages installed because dependencies were unpinned.

- We expect these issues to be easier to detect through pre-build linting (Dockerfile analysis) than comparing final images post-build.

# Looking Forward



# Team



**Kevin Pitstick**

Principal Investigator  
Senior Software Engineer



**Alex Derr**

Associate Software Engineer



**Lihan Zhan**

Associate Software Engineer



**Sebastián Echeverría**

Senior Software Engineer



# Contact Us

