Accelerating the CI/CD-to-Robot Cycle by 10x for 1/10th the cost

Ruffin White
Marcus Scheunemann

ROSCon 2024 | Odense, Denmark

DEXORY

About Me 👨‍💻 https://about.me/ruffin

Ruffin White Ph.D.

@ruffsl

Motivation 🔭

CI/CD-to-Robot Cycle

Speedup software deployment

Time is money

⏱️ = 💸

Scaleup integration validation

More features, more variants

🛳️⛴️🚢

Streamline robot testing

Simulation-only doomed to succeed

🖥️ ≠ 🤖

Simplify Developer Experience

Poor DevEx impacts morale

🔥 🧯 😓

Optimize CI/CD costs

Budgets are finite

🏦💸⏳

Motivation 🔭

CI/CD-to-Robot Cycle

Speedup software deployment
- Reduce time from pushed PR to ROS run
- Deduplicate work via distributed caches

Scaleup integration validation
- Expanding platform functionality
- Without compromising test coverage

Streamline robot testing
- Automate real and sim tests via PRs
- Enable hybrid tests w/ hardware + sim

Simplify Developer Experience
- Discrepancies between Dev and Prod adds friction to reviewing & debugging

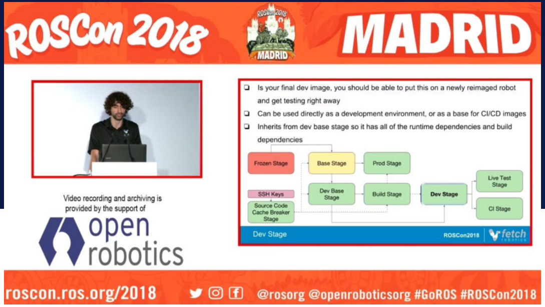
Optimize CI/CD costs
- GitHub hosted GPU runners - $$$
- AWS egress bandwidth - also $$$

Related Work 📚

Prior art utilizes multi-staging to prevent churn when re-deploying, or forgoes debian entirely for more rigorous build environments & OS.

Can cache granularity be improved to extend its distributed lifecycle?

Can cache determinism be better without retraining and retooling?

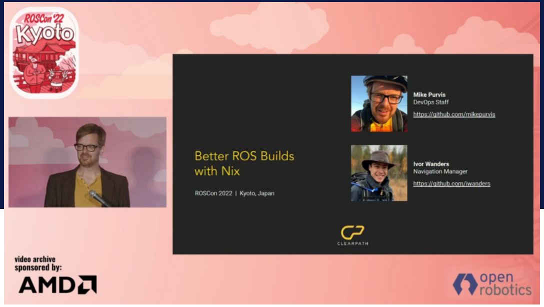

ROSCon 2018 MADRID

Video recording and archiving is provided by the support of open robotics

- Is your final dev image, you should be able to put this on a newly reimaged robot and get testing right away
- Can be used directly as a development environment, or as a base for CI/CD images
- Inherits from dev base stage so it has all of the runtime dependencies and build dependencies

Dev Stage ROSCon2018 fetch robotics

roscon.ros.org/2018 @rosorg @openroboticsorg #GoROS #ROSCon2018


ROSCon '22 Kyoto

Better ROS Builds with Nix

ROSCon 2022 | Kyoto, Japan

Mike Purvis
DevOps Staff
https://github.com/mikepurvis

Ivor Wanders
Navigation Manager
https://github.com/iwanders

CLEARPATH

video archive sponsored by: AMD

open robotics

docker

🎓 Great introduction to intermediate patterns

📟 Predates modern BuildKit advancements

```
Hermetic Robot Deployment
Using Multi-Stage Dockers
ROSCon '18 | Madrid, Spain
```

NixOS

♻️ Purely deterministic and cacheable builds

🚧 Radical departure from Tier-1 support

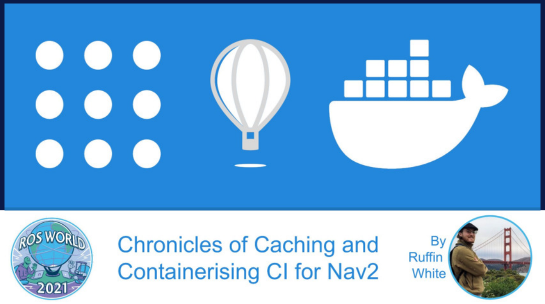
```
Better ROS Builds with Nix
ROSCon '22 | Kyoto, Japan
```

Related Work 📚

Previous approaches demonstrate ways to optimize workspace builds using incremental compilation while unifying CI with local development.

Can CI/CD speed be scaled further while minimizing overhead costs?

Can dev workflows be simplified despite infrastructure complexities?



circleci

⏱️ Optimize CI pipeline for speed and caching

💰 Non-optimal cost for large scale deployment

```
Chronicles of Caching and
Containerising CI for Nav2
ROS World ‘21 | Earth, Sol
```

By: Ruffin White, Gianluca Caiazza
Repeatable Reproducible Accessible ROS Development via Dev Containers

VSCode

⚒️ Demos containers as dev environments

🚢 Omits infrastructure to build/ship containers

```
Repeatable Reproducible
Accessible ROS Development
via Dev Containers
ROSCon ‘23 | New Orleans, USA
```

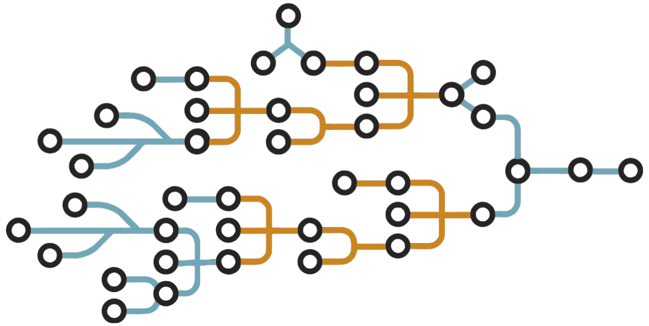
Approach | BuildKit 🐳

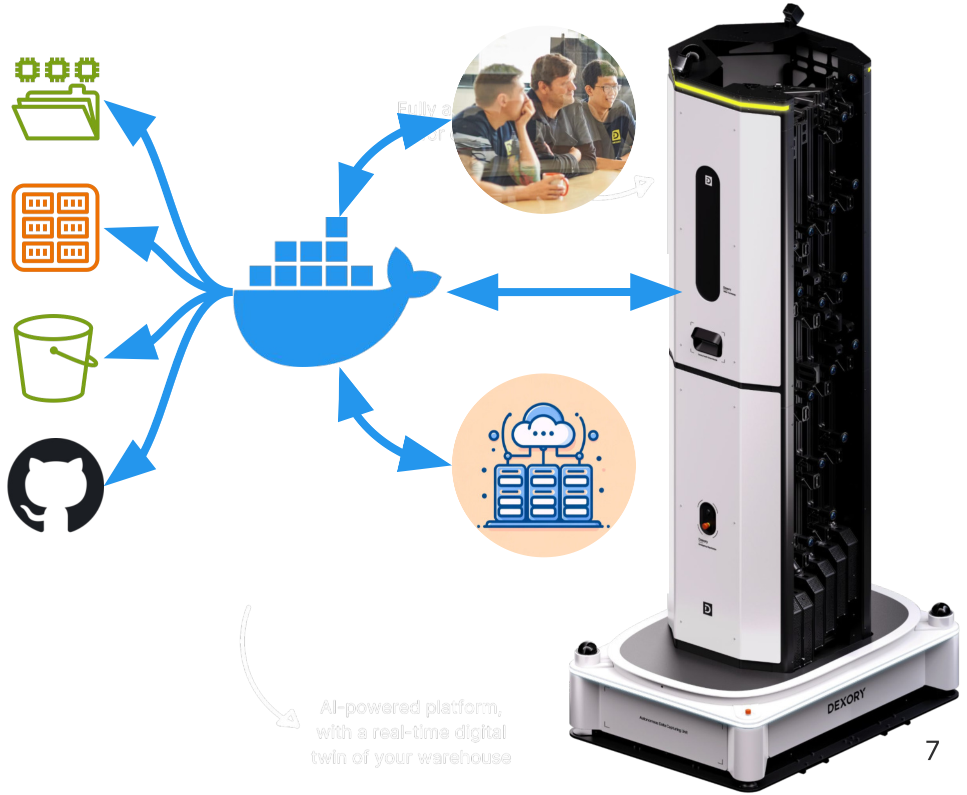
Granular. More caching options with Dockerfile directives such as `COPY --link` and `RUN --mount` features

Parallelized. Independent stages build simultaneously, using Low-Level Build (LLB) to optimize shared layers

Distributed. Multiple caches backend supported: local directory, remote registry, s3 bucket, GitHub cache, ...

Procedural. Code up bake files to string together Docker contexts, build args, tag settings, etc, using HCL syntax




AI-powered platform, with a real-time digital twin of your warehouse
DEXORY

Approach | RunsOn 🚀

Join companies that get faster CI, at lower cost

RunsOn is used by companies of all sizes, from startups to large enterprises. Millions of runners have been run with RunsOn since its inception in January 2024.

⏱️ **Faster**. Raw CPU performance is up 30% compared to GitHub hosted runners.

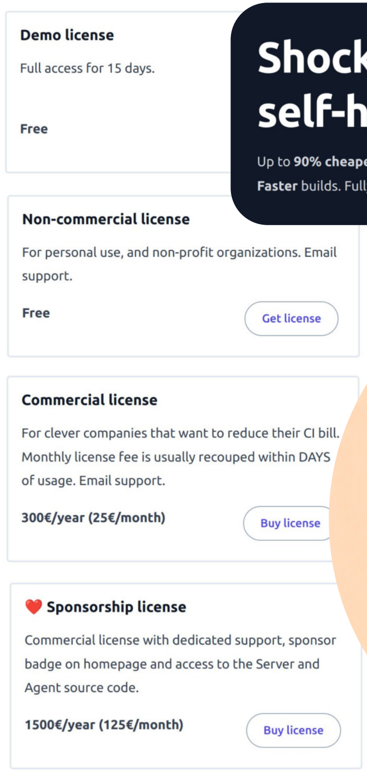
💰 **Cheaper**. Between 7x to 15x cheaper than GitHub hosted runners via spot instance pricing.

🏗️ **Scalable**. Handles bursts of multiple hundred jobs at once without issue. No concurrency limit.

✅ **Compatible**. With native GitHub workflows via public default AMIs, or customize your own.

🧘‍♀️**Low maintenance**. A single CloudFormation template with all the resources, 1-click install, 1-click upgrades. Costs $1.5/month.

runs-on.com



Demo license

Full access for 15 days.

Free

Non-commercial license

For personal use, and non-profit organizations. Email support.

Free

Commercial license

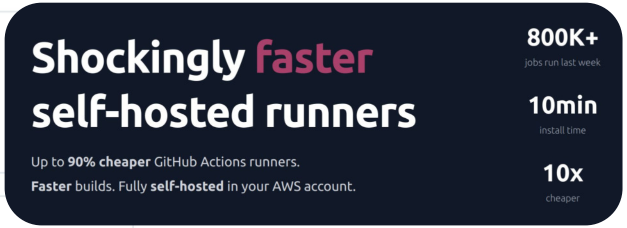
For clever companies that want to reduce their CI bill. Monthly license fee is usually recouped within DAYS of usage. Email support.

300€/year (25€/month)

❤️ **Sponsorship license**

Commercial license with dedicated support, sponsor badge on homepage and access to the Server and Agent source code.

1500€/year (125€/month)



Shockingly faster self-hosted runners

Up to **90% cheaper** GitHub Actions runners.
Faster builds. Fully **self-hosted** in your AWS account.

800K+	jobs run last week
10min	install time
10x	cheaper

Implementation | BuildKit 🐳

- Stagger dependency installation
  - Optimizes image size and layer reuse
  - Run-time < Test-time < Build-time
- Lazy evaluation of build contexts
  - Why do now, what you can do later?
  - Delay volatile branch inputs for last

- Cache disk and network IO
  - Using typed cache mounts
  - E.g apt downloads for later rebuilds
- Export cache mode=max
  - To preserve intermediate stages/work
  - No need to inline them in final image

```
FROM baser AS cacher

# copy overlay source
COPY ./src ./src

# generate typed dependency lists
SHELL ["/bin/bash", "-o", "pipefail", "-c"]
RUN dep_types=(\
     "exec:--dependency-types=exec" \
     "test:--dependency-types=test" \
     "build:"\
   ) && \
   for dep_type in "${dep_types[@]}"; do \
     IFS=":"; set -- $dep_type; \
     rosdep install -y \
       --from-paths src \
       --ignore-src \
       --reinstall \
       --simulate \
       ${2} \
       | grep 'apt-get install' \
       | awk -F' ' '{print $4}' | sed "s/'//g" \
       | sort > /tmp/${1}_debs.txt; \
   done
```

```
COPY --link --from=cacher /tmp/exec_debs.txt /tmp/exec_debs.txt
RUN --mount=type=cache,sharing=locked,target=/var/cache/apt \
   < /tmp/exec_debs.txt xargs apt-get install -y --no-install-recommends
```

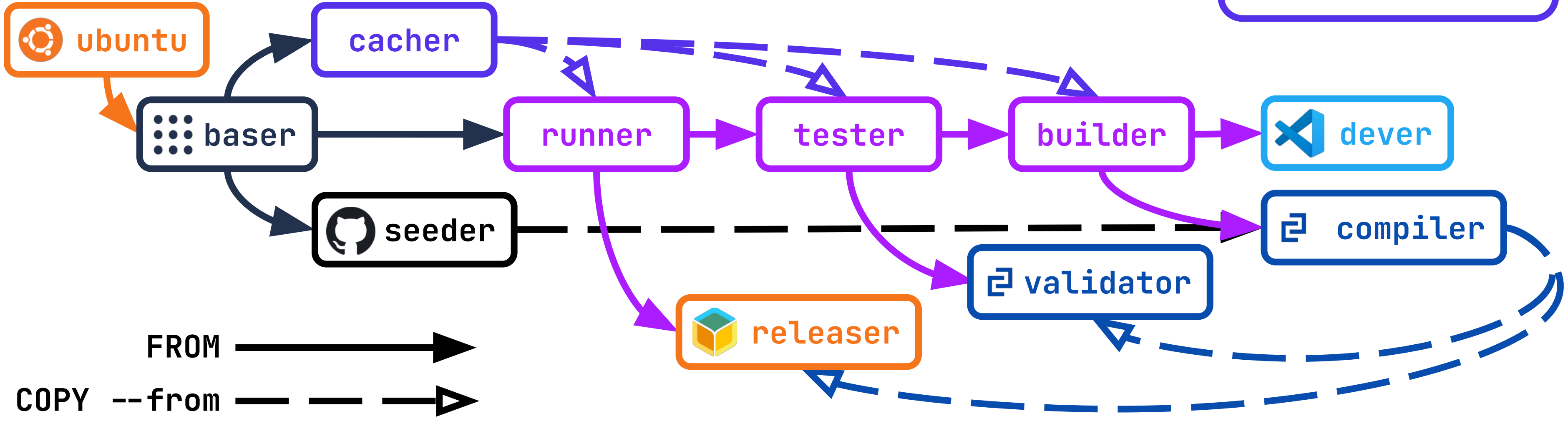
Implementation | BuildKit 🐳 Dockerfile Graph

Each stage builds FROM only what it needs,
yet will COPY --from what it needn't rebuild.

CI may restore partial builds for incremental compilation
by seeding caches that are key'd to builder's image hash

```
COPY --link --from=cacher /tmp/<exec/test/build>.txt ...
RUN --mount=type=cache,sharing=locked,target=/var/cache/apt \
   < /tmp/<exec/test/build>.txt xargs apt-get install ...
```

```
RUN dep_types=(\
  ...
  rosdep install -y \
    ...
    | sort > /tmp/${1}_debs.txt
```



Implementation | RunsOn 🚀

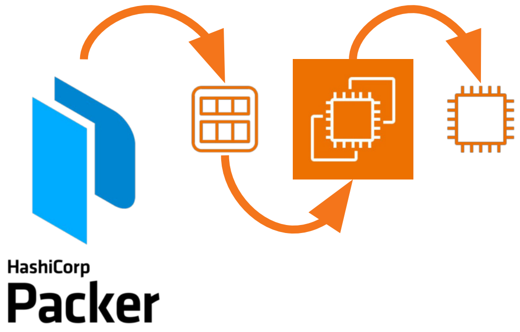
Config file/redirect is committed into repo
Defines available images and runners
Images (or AMIs) could be public or private
Example Packer template for Nvidia enabled AMI

- github.com/runs-on/runner-images-for-aws/pull/5

Runners (or instances) can be specified by:

- CPU architecture
- vCPU count
- RAM/HDD size
- Spot pricing pref
- SSH debug enable

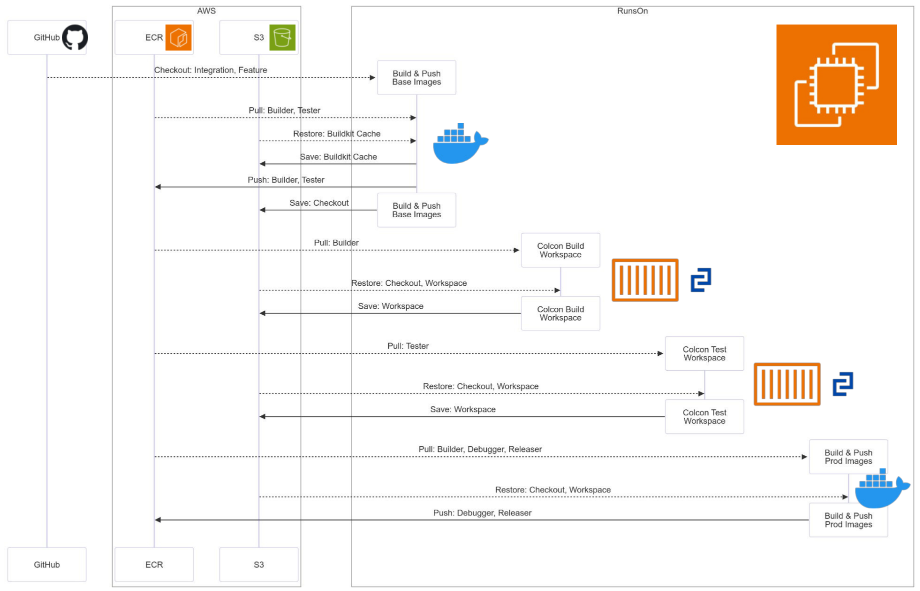
Preinstall enables ECR login, prior to container job startup, enabling use of private images, equivalent to GCR via GitHub

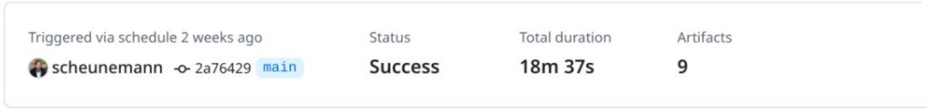


```
# .github/runs-on.yml

images:
 cpu_image:
   platform: "linux"
   arch: "x64"
   owner: "012345678901"
   # Default AMI with GitHub Action runner installed
   name: "runs-on-dev-cpu-full-x64-*"
   # Login with AWS ECR for container on startup
   preinstall: &preinstall_script |
     #!/bin/bash
     set -e
     su - runner -c "\
       aws ecr get-login-password --region eu-west-2 \
       | docker login --username AWS --password-stdin \
         012345678901.dkr.ecr.eu-west-2.amazonaws.com"
 gpu_image:
   platform: "linux"
   arch: "x64"
   owner: "012345678901"
   # Custom AMI with Nvidia drivers and container
runtime
   name: "runs-on-dev-gpu-full-x64-*"
   preinstall: *preinstall_script

runners:
 gpu_runner:
   # Specify GPU instance for simulations
   family: ["g4dn.4xlarge" ]
   image: gpu_image

 cheap:
   # For workflows that don’t require much of CPU /
RAM.
   ram: [2, 4, 8]
   # Burstable instances, valid for both x64 and arm64
   family: ["t3", "t4"]
   image: other_image
   ssh: false
 fast:
   cpu: 32
   # Increase disk size to 8
   hdd: 80
   family: ["c7a", "m7a"]
   spot: false
   image: cpu_image

admins:
 - ruffsl
```

Implementation 🛠️

- Sequence Diagram
  - Resource interaction for each job
- Flow Diagram
  - DAG of jobs for integration workflow

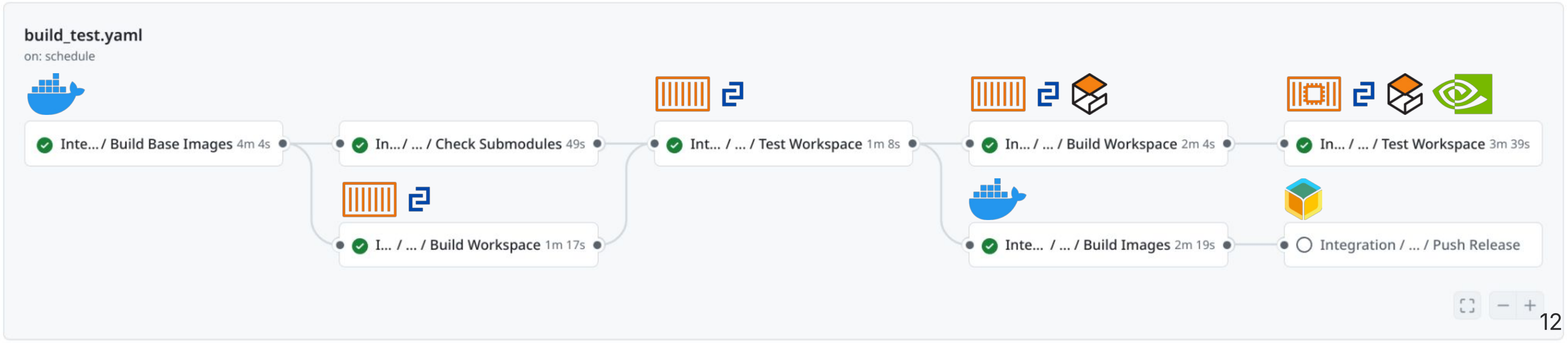



Triggered via schedule 2 weeks ago
scheunemann 2a76429 main

Status
Success

Total duration
18m 37s

Artifacts
9


build_test.yaml
on: schedule

- Inte... / Build Base Images 4m 4s
- In... / ... / Check Submodules 49s
- I... / ... / Build Workspace 1m 17s
- Int... / ... / Test Workspace 1m 8s
- In... / ... / Build Workspace 2m 4s
- Inte... / ... / Build Images 2m 19s
- In... / ... / Test Workspace 3m 39s
- Integration / ... / Push Release

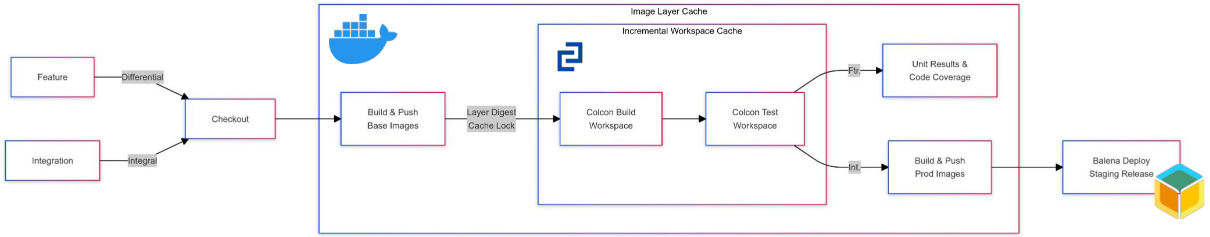
Conclusion 🏁

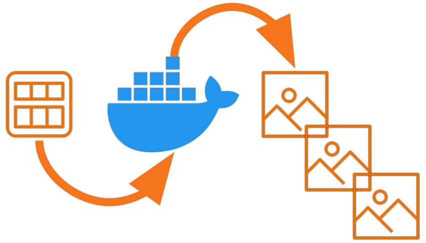
Container Images

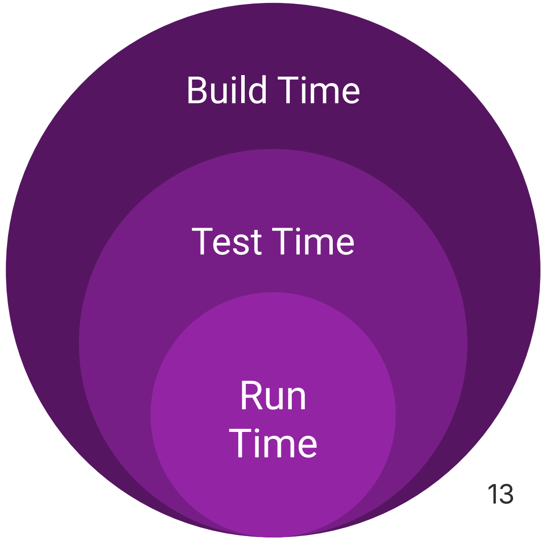
- Cache via BuildKit
  - Building w/ cache mounts
  - Deterministic multistages
  - Cache storage backends
- Unify via BakeFiles
  - Robot Development
  - CI for Testing
  - CD for Production

Container Runtimes

- Save via self hosting
  - Co-locate Cache & Compute
- Simplify via RunsOn
  - Maintain GitHub Integration
- Optimize Resources
  - Provision per requirements





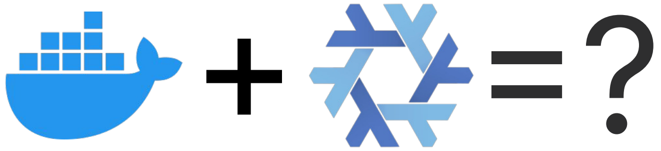


Future work 🔮

- Codespaces, but $elf Hosted with GPUs
  - Simplify onboarding and remote collaborations
  - Simulate scenarios too demanding for dev laptops
  - Improve cache locality/bandwidth via AWS region
  - Forward graphics via NICE DCV or Moon/Sunlight

- Combining containers with Nix packaging
  - Improve build determinism and cacheabilty
  - Minimize diffs via per package COPY --link
  - Lower learning curve for dev adoption
  - Reusing existing deployment infrastructure
  - [Docker and Nix (DockerCon 2023)]()

- Optimizing ephemeral RunsOn runners
  - Recycling EC2 instances for similar workflows
  - [github.com/runs-on/runs-on/discussions/72]()
  - Reduce pull times via AMI/EBS docker tricks
  - [github.com/awslabs/amazon-eks-ami/issues/1273]()



References 🧭

RunsOn

- runs-on.com
- github.com/runs-on/runs-on
- github.com/runs-on/cache

BuildKit

- github.com/moby/buildkit
- docs.docker.com/build/buildkit
- docs.docker.com/build/bake
- 🚧 Concurrent requests may skip loading some cache paths 🚧
  - github.com/moby/buildkit/issues/4674

Open Source example

- 🚧 Refactor Docker and Dev Container setup using Buildkit 🚧
  - github.com/ros-navigation/navigation2/pull/4392

DEXORY

Questions?