
AWS
re:Invent
NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

CON325-R

Building containers for AWS

Phil Estes
Principal Engineer
AWS

Jesse Butler
Principal Technologist
AWS

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

Containers and images
Open-source container development tooling
Building images and planning for distribution and deployments

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Containers and images

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Containers on AWS



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Containers as a compute paradigm

From orchestration of large multicluster estates to one-off web applications

Integrations with other AWS services, across a number of use cases

Reproducible and consistent through the entire software lifecycle



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Containers for packaging and distribution

Every instance based on a versioned and immutable artifact

All components and dependencies are defined and provable at build time

Registries are the foundation, both build destination and deployment origin



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

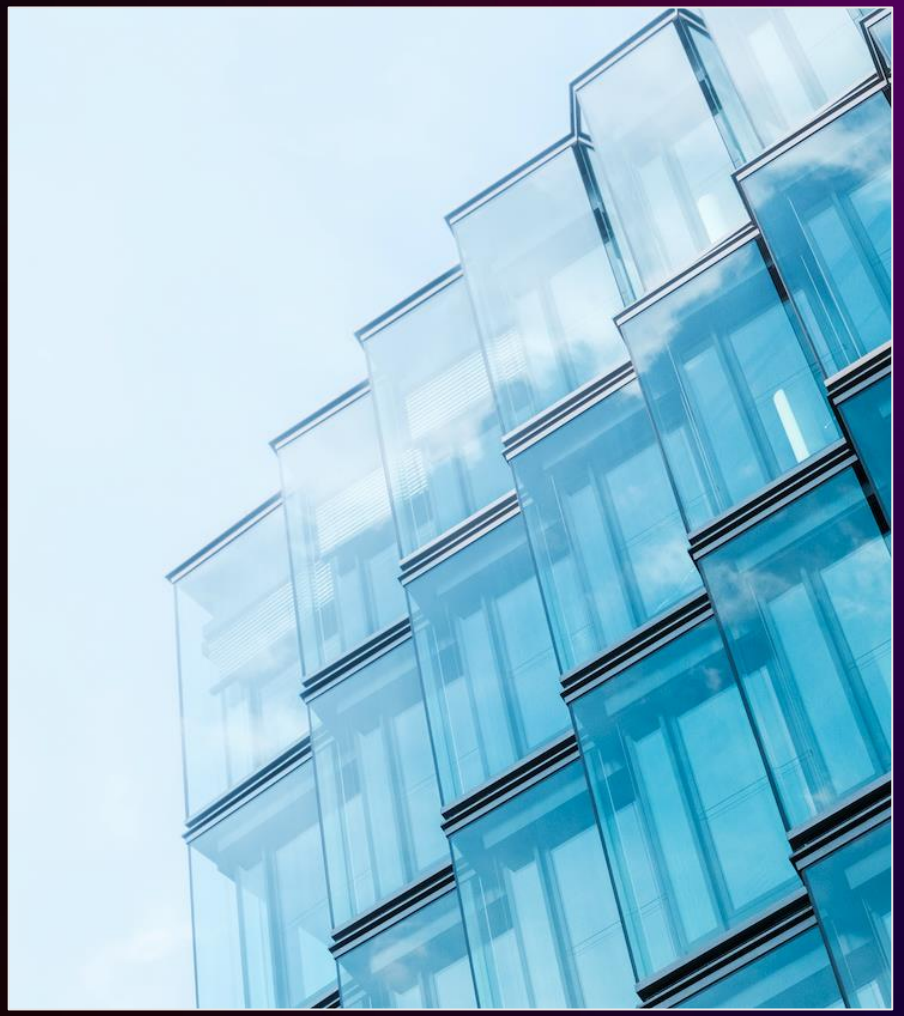
Containers are simple, scale is complex

It’s very easy to get started with containers, but complexity can creep in

Long-term adoption is more than deploying containerized workloads

Image build standards, software dependencies, and distribution practices all matter at scale

How long to deploy? How often? How large are your images, and how long to build?



aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Understanding Linux containers

No matter what AWS services you use with containers, to run workloads or simplify packaging, you start with images

If you build these images as part of your development work, you need good tools and some context for what problems they are solving

To set that context, we'll start with a quick tour of how containers work, and the software that handles the various moving parts

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Linux containers, for context

There is no such object known as a “container” in Linux

Containers are a combination of kernel features and process isolation constructs

The goal of various container tools is to start a process in a seemingly private compute environment given a configuration and bundle of software

Whenever a container is started from this bundle, it should be consistent each and every time as there is no impact from the surrounding host system

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Namespaces

Enables us to create seemingly isolated instances of otherwise global resources

In containers, we configure namespaces for a number of resources

Mount, PID, Network, UTS, IPS and User are examples

Namespaces are what makes a container appear to be an isolated system

NAMESPACES(7) **Linux Programmer's Manual** **NAMESPACES(7)**

NAME top

namespaces - overview of Linux namespaces

DESCRIPTION top

A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. Changes to the global resource are visible to other processes that are members of the namespace, but are invisible to other processes. One use of namespaces is to implement containers.

Linux provides the following namespaces:

Namespace	Constant	Isolates
Cgroup	**CLONE_NEWCGROUP**	Cgroup root directory
IPC	**CLONE_NEWIPC**	System V IPC, POSIX message queues
Network	**CLONE_NEWNET**	Network devices, stacks, ports, etc.
Mount	**CLONE_NEWNS**	Mount points
PID	**CLONE_NEWPID**	Process IDs
User	**CLONE_NEWUSER**	User and group IDs
UTS	**CLONE_NEWUTS**	Hostname and NIS domain name

This page describes the various namespaces and the associated */proc* files, and summarizes the APIs for working with namespaces.

The namespaces API

As well as various */proc* files described below, the namespaces API includes the following system calls:

clone(2)

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Control groups (cgroups)

Provides limits and monitors for a process, or set of processes

CPU, memory, network IO, and disk IO are all resources that can be controlled by a cgroup

The second major component for building Linux contains

CGROUPS(7) Linux Programmer's Manual CGROUPS(7)

NAME top

cgroups - Linux control groups

DESCRIPTION top

Control cgroups, usually referred to as cgroups, are a Linux kernel feature which allow processes to be organized into hierarchical groups whose usage of various types of resources can then be limited and monitored. The kernel's cgroup interface is provided through a pseudo-filesystem called cgroupfs. Grouping is implemented in the core cgroup kernel code, while resource tracking and limits are implemented in a set of per-resource-type subsystems (memory, CPU, and so on).

Terminology

A *cgroup* is a collection of processes that are bound to a set of limits or parameters defined via the cgroup filesystem.

A *subsystem* is a kernel component that modifies the behavior of the processes in a cgroup. Various subsystems have been implemented, making it possible to do things such as limiting the amount of CPU time and memory available to a cgroup, accounting for the CPU time used by a cgroup, and freezing and resuming execution of the processes in a cgroup. Subsystems are sometimes also known as *resource controllers* (or simply, controllers).

The cgroups for a controller are arranged in a *hierarchy*. This hierarchy is defined by creating, removing, and renaming subdirectories within the cgroup filesystem. At each level of the hierarchy, attributes (e.g., limits) can be defined. The limits, control, and accounting provided by cgroups generally have effect throughout the subhierarchy underneath the cgroup where the attributes are defined. Thus, for example, the limits placed on a cgroup at a higher level in the hierarchy cannot be exceeded by descendant cgroups.

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Tooling to the rescue

You might be thinking at this point, we're a few slides in and this is already more than you want to worry about

We agree! This is why reliable container tools are so important

Containers are a great solution for a number of problems:

- Consistent and repeatable runtime environments
- Dependency management, packaging, and distribution
- Immutable deployment artifacts, tied to distribution by unique identifier

Containers are very *useful*, our tools are what make them *usable*

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Where to start

As we describe components of development tooling for containers, and later provide some guidance for building your images, we'll need some context:

Starting from the bottom up, we need a runtime

A container runtime abstracts the kernel features and other component software and puts a smooth handle on the creation and execution of a container instance

The runtime needs an environment to execute processes within, which leads to the need for portable file systems or images

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

A brief history, Open Containers Initiative

The Open Containers Initiative was created to set standards for container image formats and runtimes

OCI Runtime and Image specifications v1.0 were released in July 2017



aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Creating containers

The OCI Runtime Specification specifies the configuration, execution environment, and lifecycle of containers

Configuration is specified in a config file, the environment specification ensures that containers are consistent

These come together in a standard file system bundle, from which a container can be created



aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The reference implementation, runc

Implemented by an engineer at Docker

Donated as the OCI runtime specification reference implementation

Given a compliant bundle, a container is born

It requires a root file system to be already in place

```
$ pwd
/acontainer
$ head -5 config.json
{
        "ociVersion": "1.0.2-dev",
        "process": {
                "terminal": true,
                "user": {
$ ls rootfs/
bin  dev  etc  home  proc  root  sys  tmp  usr  var
$ runc run acontainer
/ # uname -sm
Linux x86_64
/ # ps -ef
PID   USER     TIME  COMMAND
    1 root      0:00 sh
    7 root      0:00 ps -ef
/ # ls
bin   dev   etc   home  proc  root  sys   tmp   usr   var
/ #
```

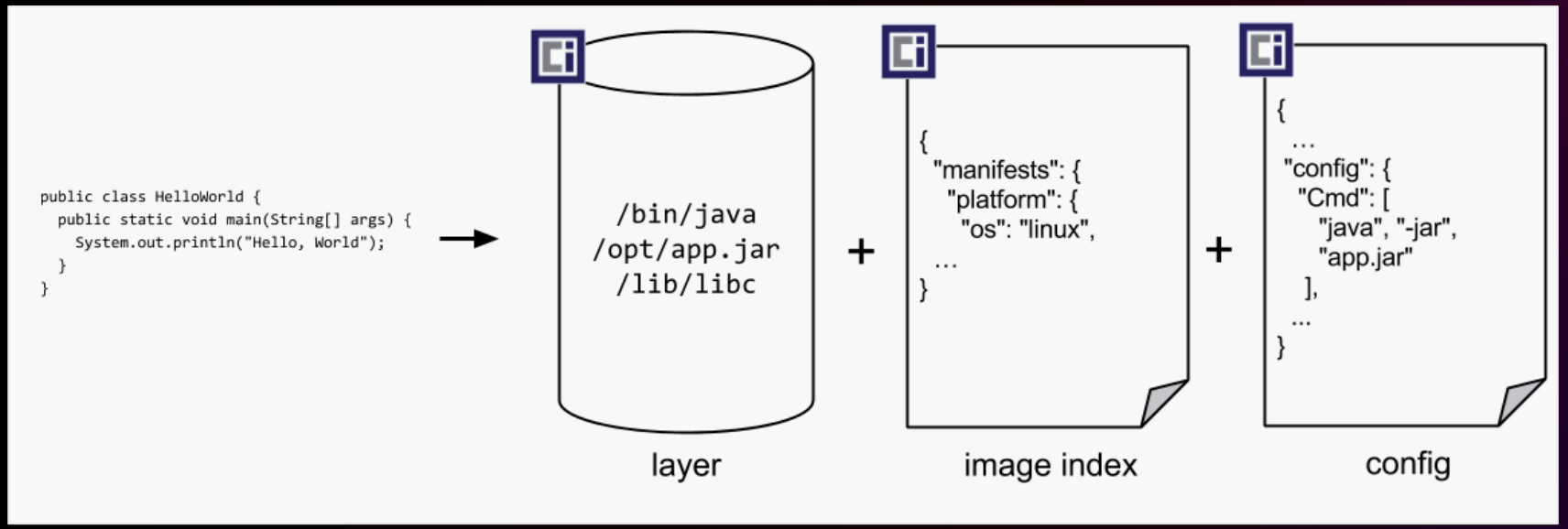
aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Container images

The OCI Image Specification enables tooling for building, transporting, and preparing container images

It specifies an OCI image consisting of an image manifest, an image index, a set of file system layers, and a configuration



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Building images

Linux container images are built as a stack of layers

Each layer depends on the prior one in the stack

Layers are built in order, from the bottom

For the purposes of our session, we'll focus on BuildKit and Dockerfiles as to tool and input construct for images

```
$ ls
Dockerfile      message
$ cat message
hi!
$ cat Dockerfile
FROM public.ecr.aws/amazonlinux/amazonlinux:2022
COPY message /app-message
ENTRYPOINT echo `cat /app-message`
$ _
```

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Building images

BuildKit is a mature and feature-rich image building system

It handles the actual work of building image layers and configuration

The client will pass a *build context* to it, usually a path in your build environment

The build context is loaded, and BuildKit iterates through the Dockerfile

```
$ docker build . -t hi
[+] Building 0.1s (7/7) FINISHED
 => [internal] load build definition from Dockerfile                    0.0s
 => => transferring dockerfile: 36B                                     0.0s
 => [internal] load .dockerignore                                       0.0s
 => => transferring context: 2B                                         0.0s
 => [internal] load metadata for public.ecr.aws/amazonlinux/amazonlinux:2  0.0s
 => [internal] load build context                                       0.0s
 => => transferring context: 28B                                        0.0s
 => [1/2] FROM public.ecr.aws/amazonlinux/amazonlinux:2022              0.0s
 => CACHED [2/2] COPY message /app-message                              0.0s
 => exporting to image                                                  0.0s
 => => exporting layers                                                 0.0s
 => => writing image sha256:aa5a90e424921163f9c3d262bbf5cea00105ea200879e  0.0s
 => => naming to docker.io/library/hi                                   0.0s
$ docker run --rm hi
hi!
$
```

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Builds from Dockerfiles

Each line in the Dockerfile is processed, top to bottom

Mutating instructions result in image layer builds, which we think of as bottom-up in the final image

Instructions that change the rootfs, like COPY, or a RUN command, which installs a package, will result in a new layer

```
FROM public.ecr.aws/docker/library/ubuntu:rolling

RUN apt-get update && apt-get install -y python3 python3-pip
RUN pip install flask==2.1.*

COPY app.py /

ENV FLASK_APP=app
EXPOSE 8000
CMD flask run --host 0.0.0.0 --port 8000
```

aws

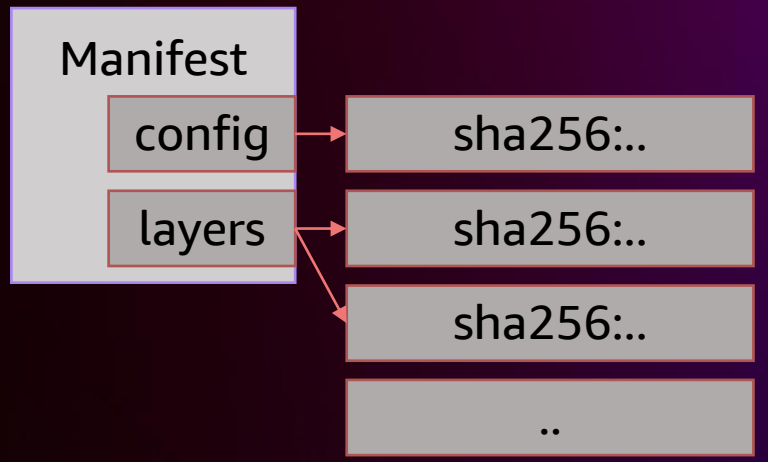
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What is an image?

OCI images are described by a manifest, which specifies the image layers by digest along with other metadata

The layers built from a Dockerfile (read and built top down) represent the image layers, bottom up

Images are effectively stacks of tarballs, which are dependent on the layer below



aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

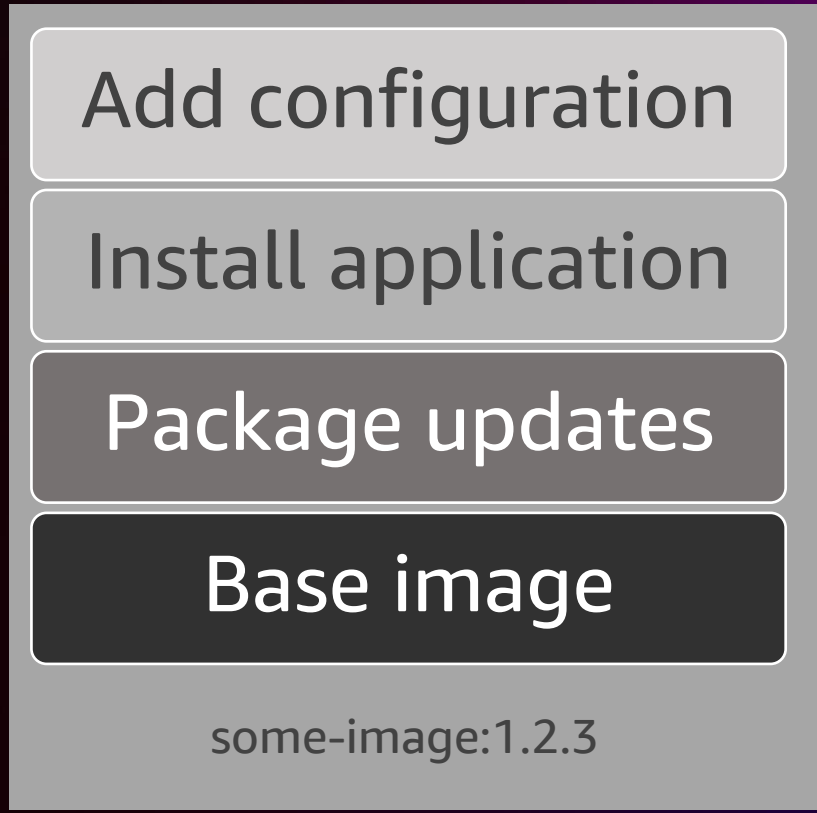
Image layers

Only the changes for a given layer are stored; the layers are then reassembled when a container is created

When a container is created locally, these layers are reassembled

The layers are immutable and identified by a digest of their content

The image is immutable, but typically you want your processes in a container to be able to write to the file system



aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Immutable images

New containers start with a read-write layer at the top of the stack

The underlying layers are not modified, and identified by their digest

Image manifests are also identified by a digest

We can always describe image content by its specific digest

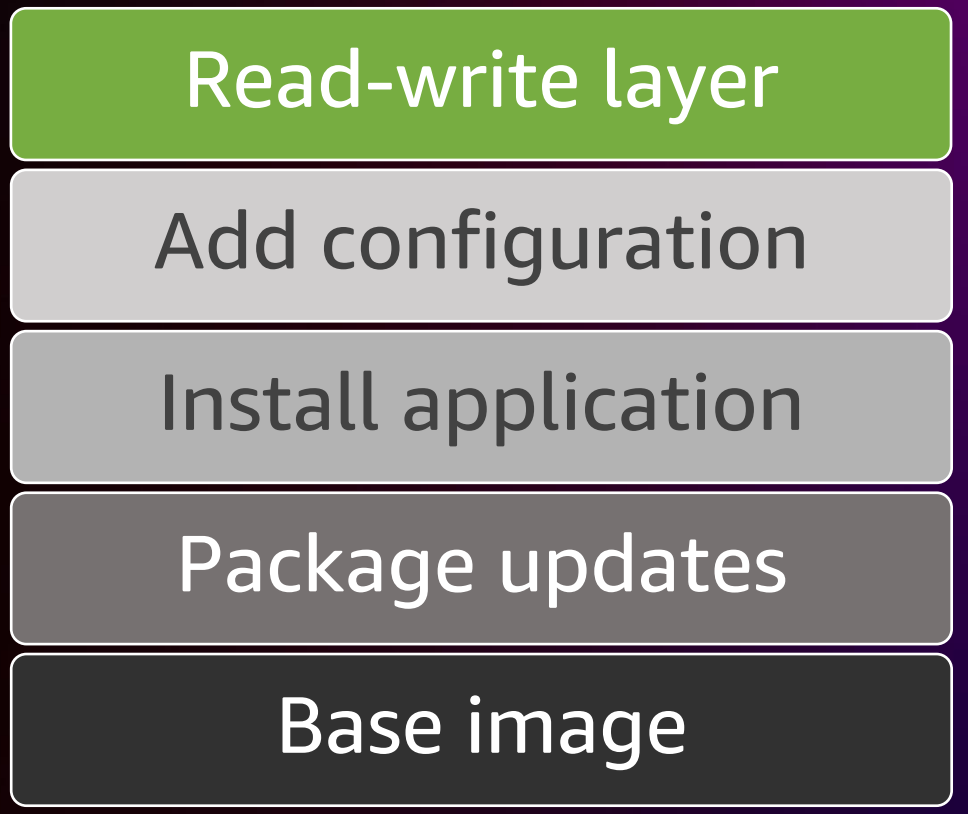


Image deployed as container rootfs

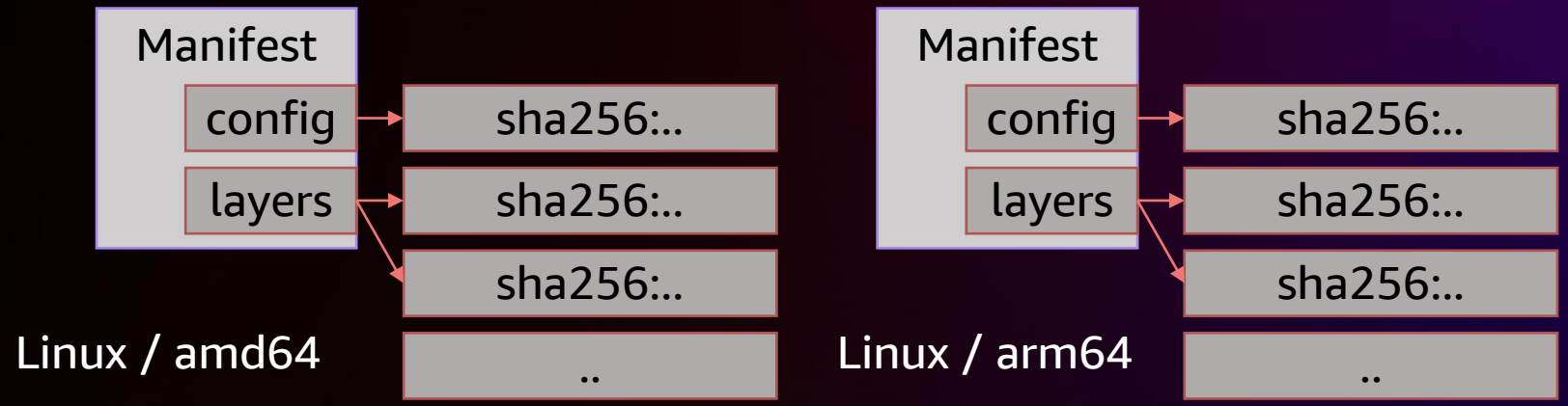
aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Multiplatform images

Images are a discrete artifact, with content for a given OS and architecture

Containers are supported on multiple systems and architectures

Multiplatform images include a manifest list, or OCI index

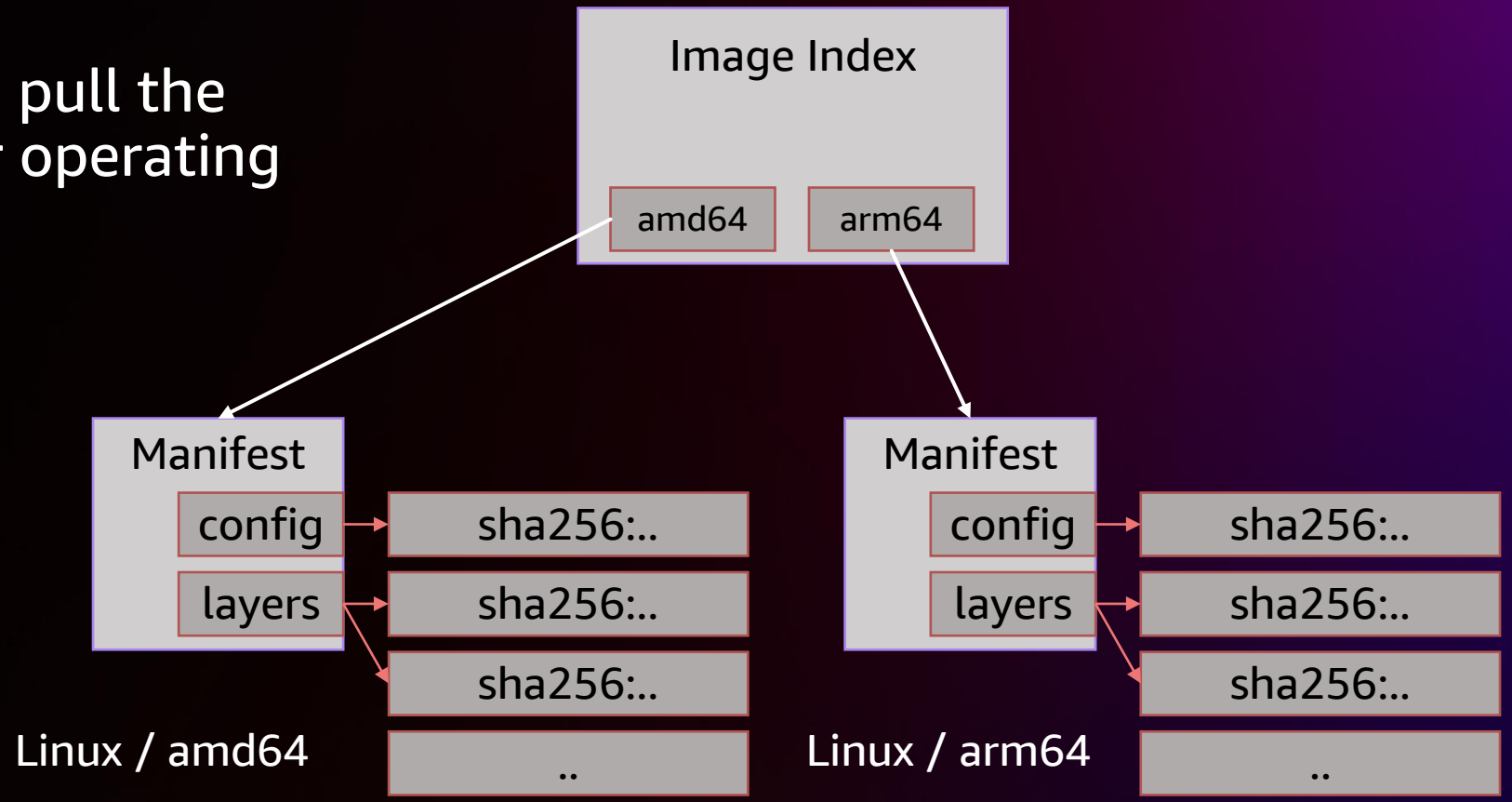


© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Multiplatform images

This is an index of other manifests, annotated for platform

Clients are implemented to pull the appropriate image for their operating system and architecture



aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Organizing it all with containerd

Now we have a way to create a container, and images to create them from, and a way to distribute those images

We need something to manage multiple container instances, their images, and make a reasonable system



aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Organizing it all with containerd

containerd is the standard container supervisor for Linux, also started by an engineer at Docker and donated (this time to the CNCF)

It runs as a system service where you run your containers, and is responsible for creating and managing container instances, pulling images, and more

containerd is used in a number of development tools, and is the default runtime for Kubernetes, as well as AWS Fargate



aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Tooling to the rescue

This was a whirlwind cursory tour, because it’s good to be familiar with what’s being abstracted

Customers want to focus on building their software, not their tools

Good tools abstract complexity and enable simple workflows



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Open-source container development tooling

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

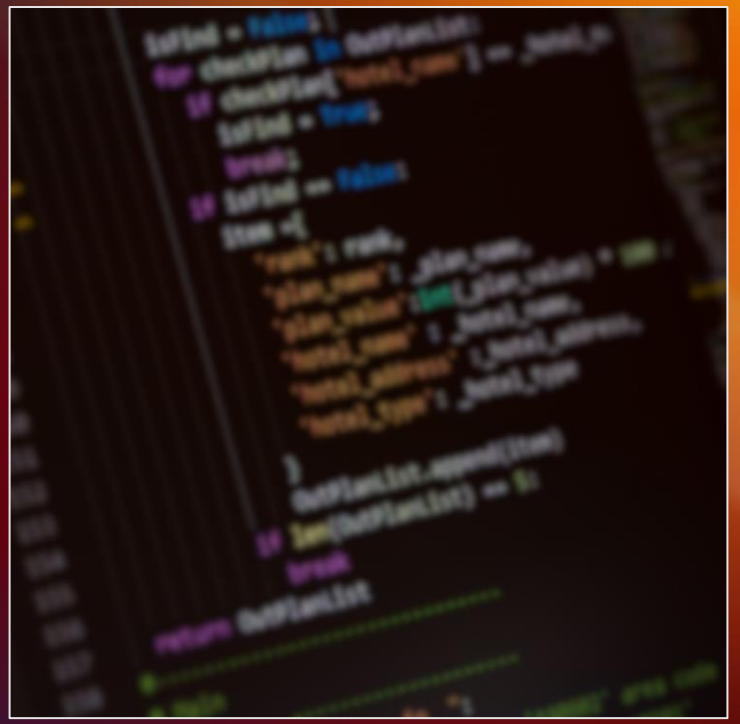
Container development

In this section of our session, we'll look at container development tools

Specifically, we want to talk about Linux container tools for non-Linux environments like macOS

We will focus on open-source components that meet the requirements we have

And we'll dive into a new open-source project that can help you use them easily



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Virtual machines

Most container development targets Linux

Most development environments commonly use macOS or Windows

As we've seen, we need Linux to run Linux containers and build Linux images

Virtualization to the rescue; initially ad hoc, but over time, several end-to-end solutions have been created to hide this complexity from the user

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

End-to-end solutions

Docker Desktop was the first to meet this need, and is still Docker's flagship container development product, using HyperKit

Red Hat introduced Podman, a command line utility provided as part of the libpod library, as well as Podman Desktop earlier this year, leveraging QEMU

Rancher Desktop expanded its desktop Kubernetes solution with nerdctl- and Docker-based container management, leveraging Lima (which manages QEMU)

Colima is a very popular open-source project that makes it simple to use nerdctl or Docker in a Linux VM, using Lima

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Introducing Finch

Finch is a new open source client for container development

It offers a macOS-native client that integrates directly with nerdctl, a client for containerd

It is installed with an opinionated virtual machine configuration managed by Lima

Finch focuses on integrating with and extending nerdctl and containerd and related software

There is a lot of interest already, with over 1k stars on GitHub in the first few days



aws

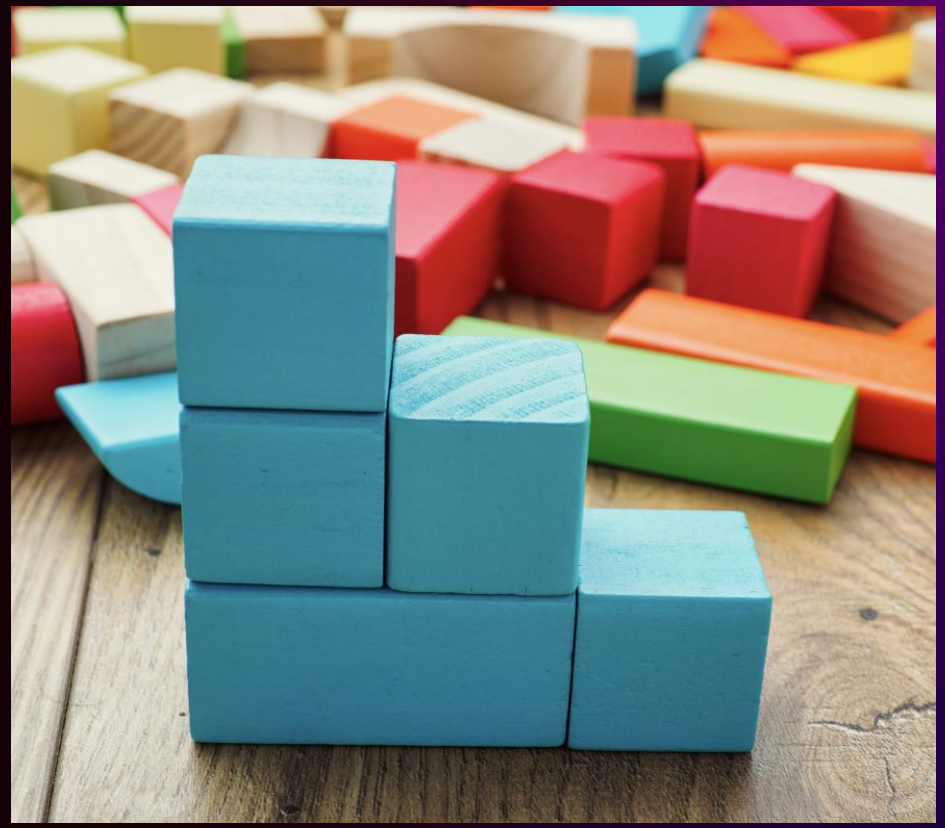
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Open source building blocks

Finch depends on many existing and innovative open source components

containerd, nerdctl, Lima, and BuildKit are the primary components

Finch makes it easy to adopt these with a curated, integrated solution



aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Why Finch?

We are customer obsessed at AWS, and customer conversations guided this project

Some customers wanted our support with a simple, open source solution

Other customers were asking for a starting point for their own customizations, for us to take on the integration and release work so they could focus on their needs

This led us to thinking in terms of a distribution of open source components, coupled with a lightweight client and installer

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Where does it fit?

Open source is a place where we can work across project boundaries, build on existing foundations, and create something new

It’s also about recognizing what already works, and helping improve and maintain these projects

One of the first things the team did was to survey the existing set of software for our requirements

Insert “reinvent” pun here 🙂

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Finch as a distribution

AWS invests in containerd, with several maintainers on staff

Leaning into this investment, our components became clear

nerdctl is a containerd client, which is now stable having achieved a 1.0 release

Lima is a newer project, started by the same engineer who started nerdctl



aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Finch ❤️ Lima

Lima is a main component of the Finch distribution

Lima is a newer project, but very popular and recently joined the CNCF as a Sandbox project

Rather than build something new, the team is contributing to Lima actively

Finch deploys an opinionated Lima configuration as part of its installation

This includes nerdctl, containerd, BuildKit, and all of the dependencies to get started



aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

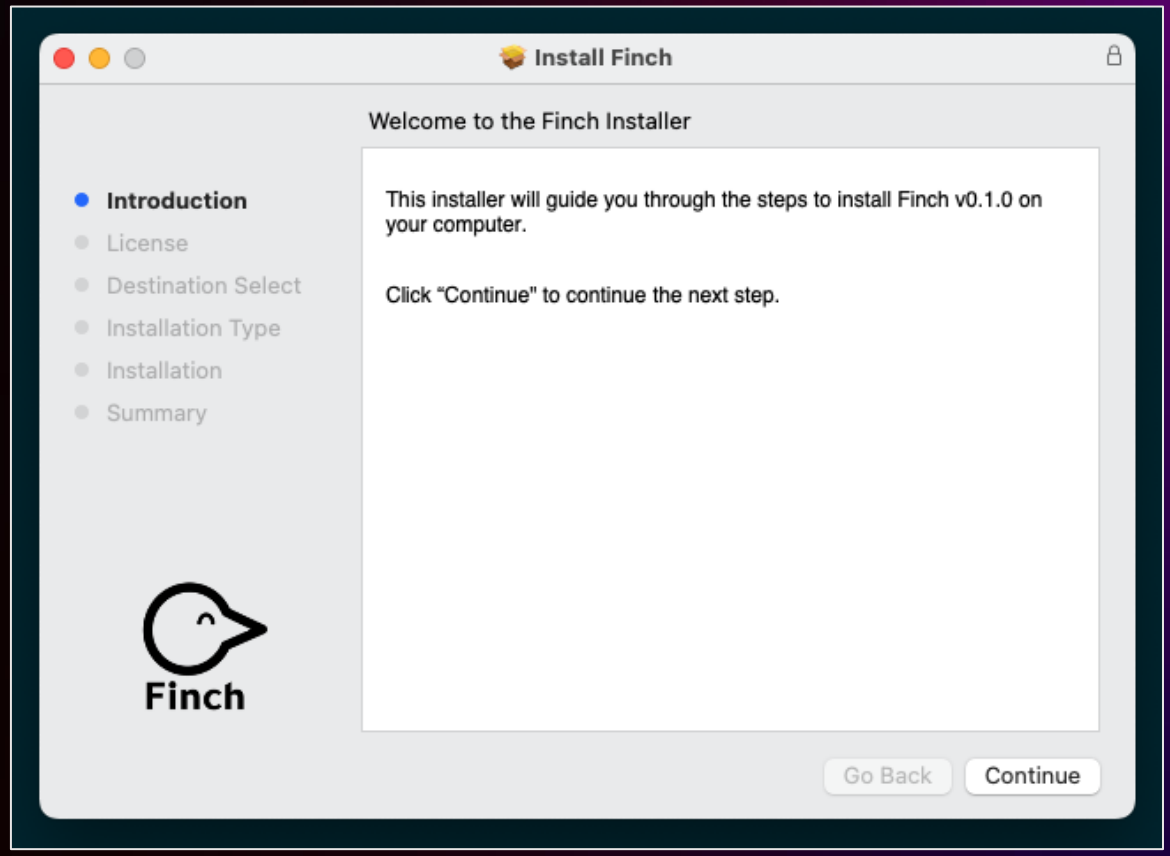
Installing Finch

Finch 0.1 is now available via a pkg installer downloadable from GitHub

We plan to add Homebrew support

At first launch, macOS is supported

- Minimum macOS version 10.15
- Intel (x86_64) and Apple Silicon (M1)



aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Getting started with Finch

The Finch client itself is macOS native

Windows support is on the list of features we plan to work on

Finch runs natively on your host OS, and integrates directly with nerdctl in the Linux environment

You don’t need to interact with the underlying VM other than to initialize it and start it

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

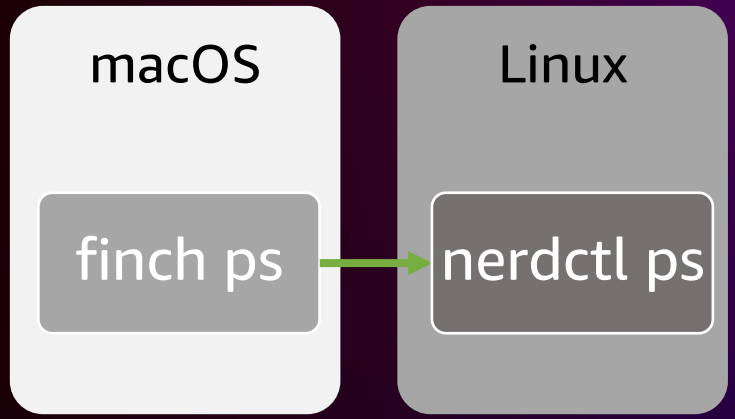
Getting started with Finch

Finch passes your commands from macOS through to nerdctl

nerdctl adopted the de facto standard commands and options established by the Docker client

Experienced developers can get started right away with familiar commands

For new developers, numerous existing blogs and tutorials remain useful



aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Running and testing containers

```
$ finch run --rm public.ecr.aws/amazonlinux/amazonlinux:2022 uname -sm
public.ecr.aws/amazonlinux/amazonlinux:2022:                              resolved       |++++++++++++++++++++++++++++++++++++++|
index-sha256:549759288ea88d4f0069efe695401fb62c56d8aac3e5495d079bd098cd58fb84:    done           |++++++++++++++++++++++++++++++++++++++|
manifest-sha256:2d51a2f176099da9870cfec8e8ccb44abf62b0752cafeeed81d072e2e0a40987: done           |++++++++++++++++++++++++++++++++++++++|
config-sha256:aeb97c8a250a690152cc87faf046796353c0f819ef270ca8d89f1e24810f980f:   done           |++++++++++++++++++++++++++++++++++++++|
layer-sha256:809bf5ee98e0790bf442d3910c9262a97ae40dbaadb2411ab6521f9a98a099e8:    done           |++++++++++++++++++++++++++++++++++++++|
elapsed: 1.6 s                                                            total:  53.9 M (33.7 MiB/s)
Linux aarch64
$
$ for i in 1 2 3 ; do
for> finch run public.ecr.aws/amazonlinux/amazonlinux:2022 sleep 1
for> done
$
$ finch rm $(finch ps -qa)
1d49f9a74ff7
360e19ad82ba
a3cdfe69ef62
$
$ finch images
REPOSITORY                                TAG       IMAGE ID       CREATED           PLATFORM          SIZE         BLOB SIZE
public.ecr.aws/amazonlinux/amazonlinux    2022      549759288ea8   47 seconds ago    linux/arm64/v8    173.3 MiB    53.9 MiB
public.ecr.aws/finch/hello-finch          latest    a71e474da9ff   6 minutes ago     linux/arm64       1.7 MiB      1007.7 KiB
$
```

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Running and testing containers

```
$ cat > ${HOME}/file
hi there
$ finch run --rm -v ${HOME}/file:/data/file al:2022 cat /data/file
hi there
$
```

```
$ finch run -d -p 8080:8080 hello
6a052a176c880d7c0ff425caba8914a4398c0426f87699a09d5adc05dd570ece
$
$ curl localhost:8080/hello
{"arch":"aarch64","message":"Hello, there!","os":"Linux 6.0.7-301.fc37.aarch64"}
$
```

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Running and testing containers

```
$ finch compose up
INFO[0000] Creating network localtest_default
INFO[0000] Ensuring image mreferre/yelb-db:0.6
INFO[0000] Ensuring image redis:4.0.2
INFO[0000] Ensuring image mreferre/yelb-appserver:0.6
INFO[0000] Ensuring image mreferre/yelb-ui:0.10
INFO[0000] Creating container localtest_yelb-db_1
INFO[0000] Creating container localtest_redis-server_1
INFO[0000] Creating container localtest_yelb-ui_1
INFO[0000] Creating container localtest_yelb-appserver_1
INFO[0000] Attaching to logs
redis-server_1   |1:C 27 Nov 21:42:04.904 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
redis-server_1   |1:C 27 Nov 21:42:04.904 # Redis version=4.0.2, bits=64, commit=00000000, modified=0, pid=1, just started
redis-server_1   |1:C 27 Nov 21:42:04.904 # Warning: no config file specified, using the default config. In order to specify a config file u
se redis-server /path/to/redis.conf
redis-server_1   |1:M 27 Nov 21:42:04.905 # You requested maxclients of 10000 requiring at least 10032 max file descriptors.
redis-server_1   |1:M 27 Nov 21:42:04.905 # Server can't set maximum open files to 10032 because of OS error: Operation not permitted.
redis-server_1   |1:M 27 Nov 21:42:04.905 # Current maximum open files is 1024. maxclients has been reduced to 992 to compensate for low uli
mit. If you need higher maxclients increase 'ulimit -n'.
redis-server_1   |1:M 27 Nov 21:42:04.905 * Running mode=standalone, port=6379.
redis-server_1   |1:M 27 Nov 21:42:04.905 # Server initialized
```

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Support for cross architectures

Finch takes advantage of userspace emulation in the underlying VM

The `--platform` option supports running either amd64 or arm64 containers

```
$ finch run --rm public.ecr.aws/amazonlinux/amazonlinux:2 uname -sm
Linux aarch64
$
$ finch run --rm --platform amd64 public.ecr.aws/amazonlinux/amazonlinux:2 uname -sm
Linux x86_64
```

Planned: integrated multiarchitecture image builds

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Building images

Dockerfile builds are supported through the BuildKit integration

`--platform` can also be used to build across architectures (amd64, arm64)

```
$ finch build . --platform amd64 -t myimage:amd64
[+] Building 2.2s (8/10)
 => [internal] load build definition from Dockerfile                                0.1s
 => => transferring dockerfile: 270B                                                0.0s
 => [internal] load .dockerignore                                                   0.1s
 => => transferring context: 2B                                                     0.0s
 => [internal] load metadata for public.ecr.aws/docker/library/golang:1.19          1.6s
 => [builder 1/4] FROM public.ecr.aws/docker/library/golang:1.19@sha256:dc76ef03e54c34a00dcdca81e55c242d  0.0s
 => => resolve public.ecr.aws/docker/library/golang:1.19@sha256:dc76ef03e54c34a00dcdca81e55c242d24b34d23  0.0s
 => [internal] load build context                                                   0.0s
 => => transferring context: 270B                                                   0.0s
```

Planned: support for custom BuildKit syntax

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Publishing images

Authentication to your registries follows a familiar workflow, too

Simply login, tag, and push

```
$ finch tag hello-finch:amd64 public.ecr.aws/finch/hello-finch:amd64
$ aws ecr-public get-login-password --region us-east-1 | \
        finch login --username AWS --password-stdin public.ecr.aws

Login Succeeded
$ finch push public.ecr.aws/finch/hello-finch:arm64
..
```

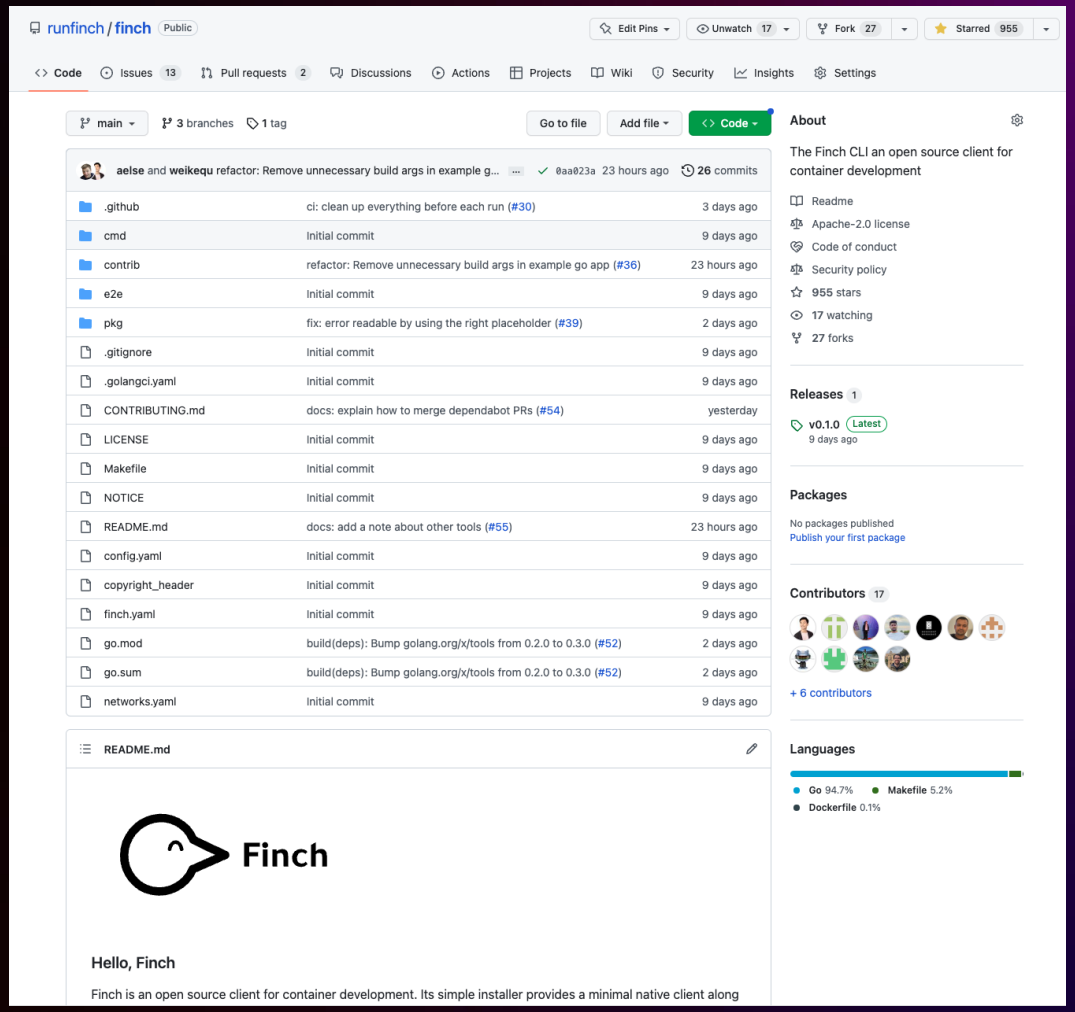
Planned: Simpler installation of credential helpers

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Join us!

We’d love to have you give Finch a try

The team is ready for issues, and pull requests are very welcome!



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Building images, and planning for distribution and deployments

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

High-level goals

General guidance for image builds

- Smaller images
- Faster builds
- Limit scope

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

High-level goals

General guidance for image builds

- Smaller images for quicker pulls, less attack surface, and ideally less frequent updates
- Faster builds to get your changes out into production fast
- Limit scope to embrace the value of containers as a means of enabling loosely coupled services as large-scale systems

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Let’s talk about base images

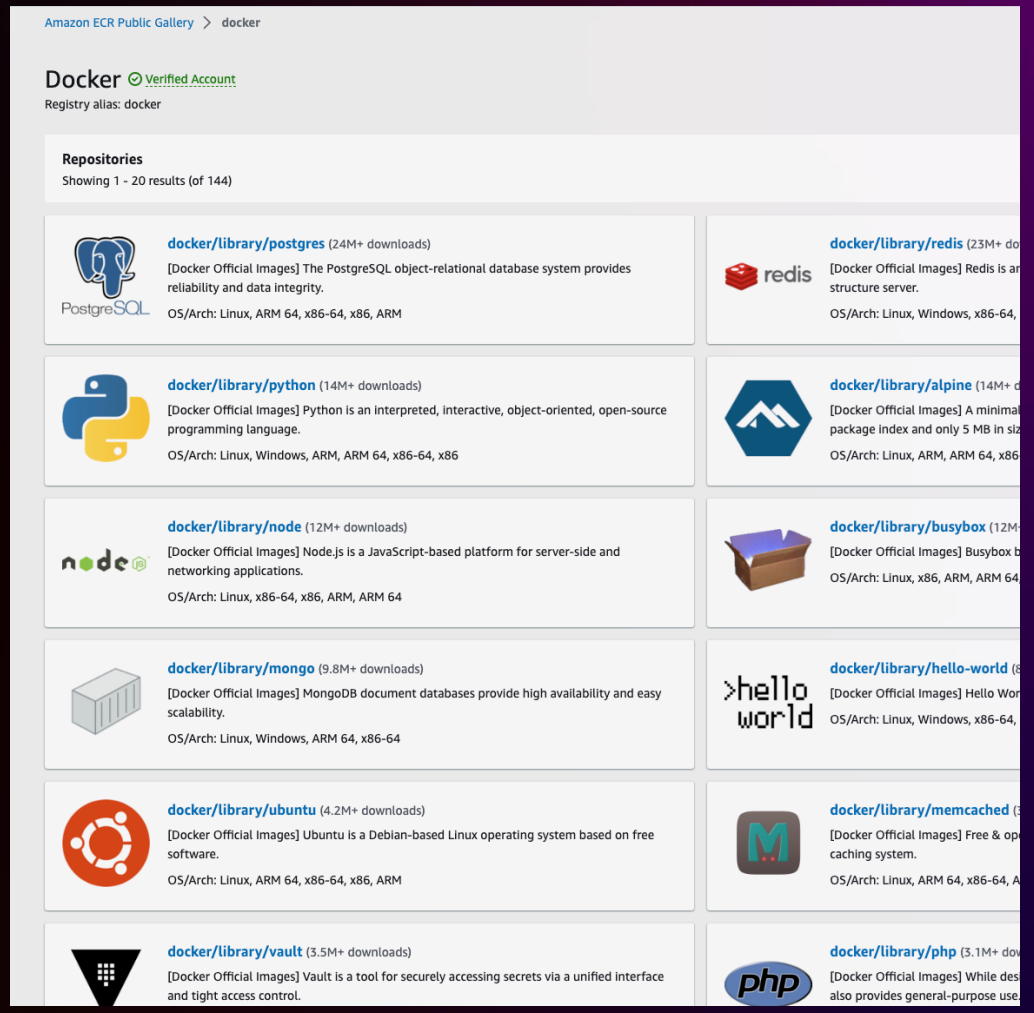
These are the images pointed at in FROM lines in a Dockerfile

This is the bottom layer of file system content, and you can create your own

Docker Official Images have become the go-to standard for base images

These typically have ‘slim’ versions, which are minified equivalent images

Available on ECR Public from the Docker publisher page



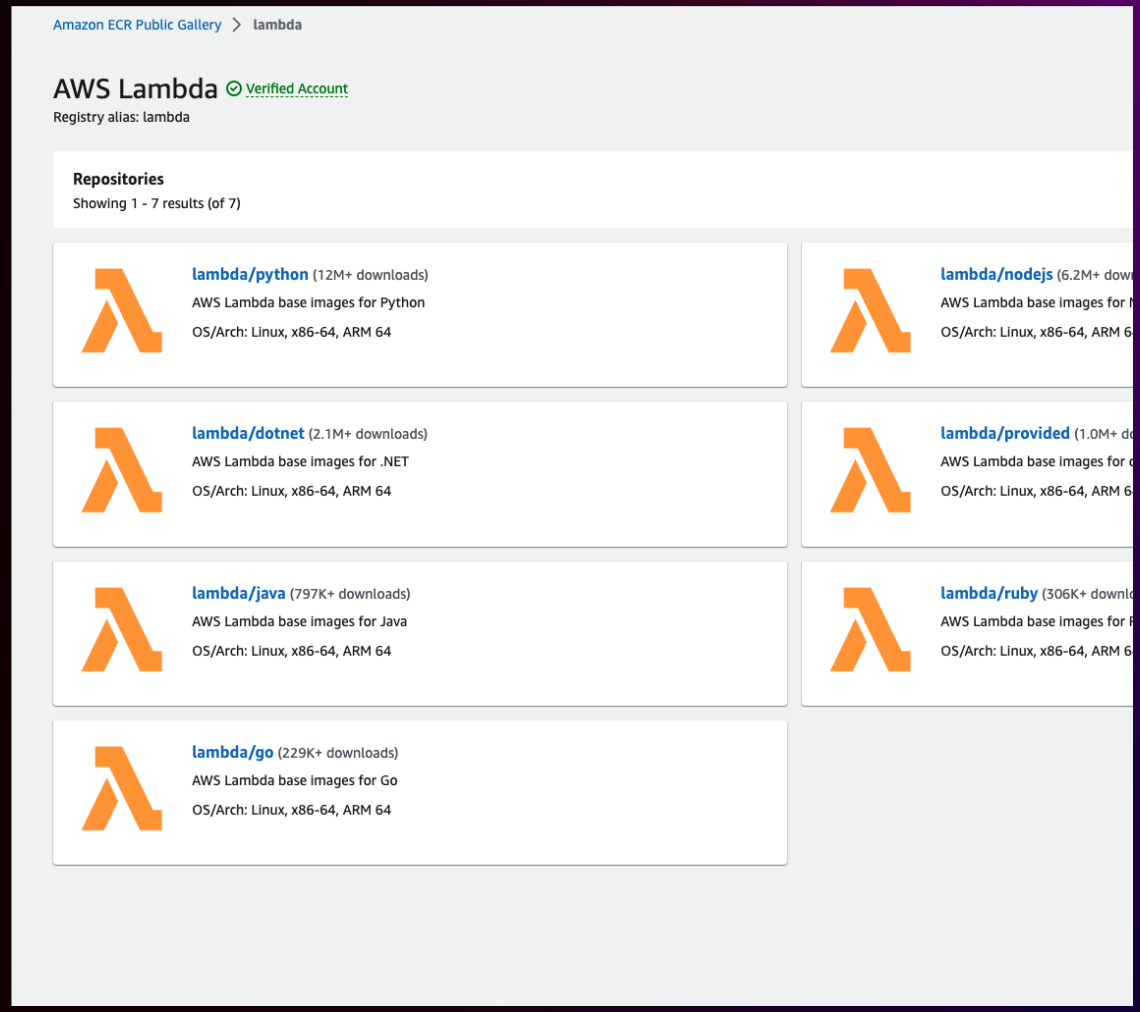
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Let’s talk about base images, Lambda edition

AWS curates a set of images for use with Lambda

While it’s possible to build your own base images, you should have a good reason to do so

These are ready-to-go images, perfect to use as base images in your builds



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Make base images your own

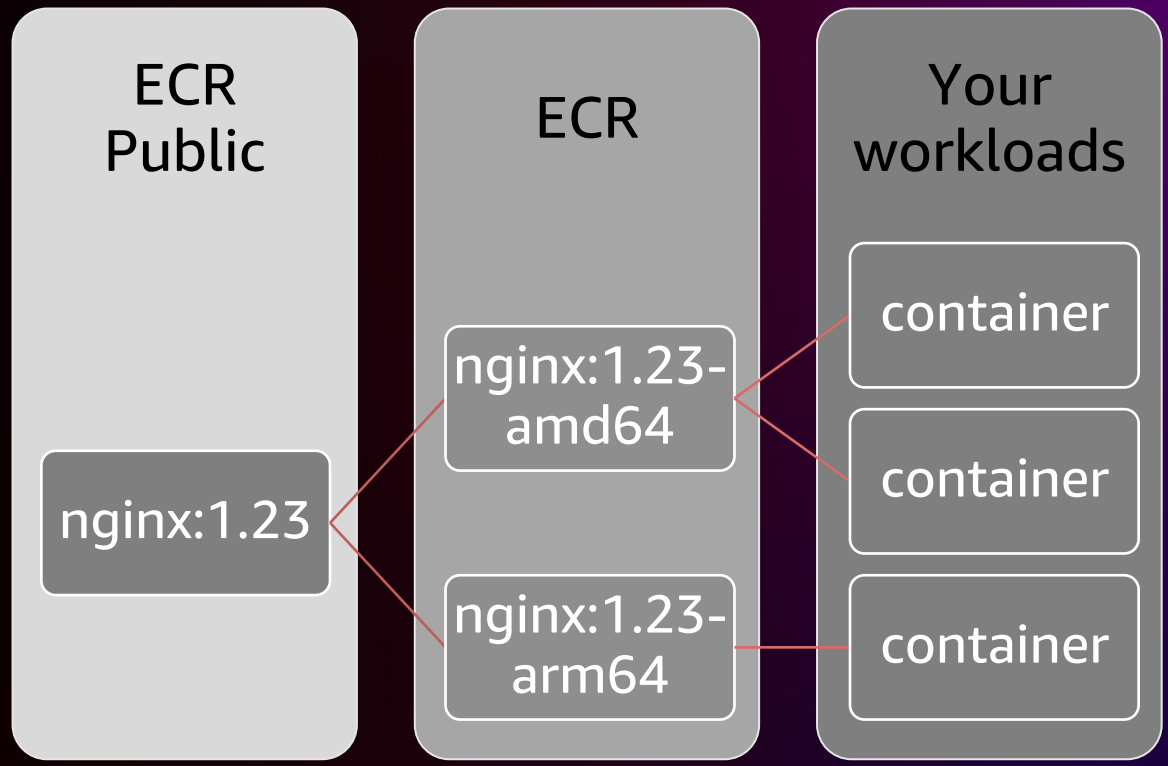
Base images are the root of your own image builds

Or, you may use these images directly and unmodified in your systems

Consider curating your own in your ECR registry instances

Control availability, potentially harden, scan, and anything else of value

ECR Pull Through Cache can help



aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Use explicit tags, as often as you can (always)

‘latest’ is handy for one-off tests

For production, it’s best to be specific

Know your dependencies

This is advice for your builds as well

You would not build a large compiled software project against “whatever is latest in the repo”

```
FROM public.ecr.aws/greatstuff/whatyouneed:latest
```

What’s this?

```
FROM public.ecr.aws/greatstuff/whatyouneed:1.23.2
```

We know exactly what this is.

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Multistage builds

An easy win for building smaller images is to use multistage builds

First available in Docker and now all BuildKit integrations (including Finch)

Simple modification of your Dockerfiles and reduce layers and overall image size

Basically, declare more than one FROM, and copy artifacts forward

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Multistage builds

Install compilers, linters, other build tools, and dependencies

Once the build instructions are complete, declare a new stage

Copy the artifacts and dependencies from the build stage into the final

```
FROM public.ecr.aws/docker/library/python:3.9.15-slim AS build

RUN python3 -m venv /opt/venv
RUN apt-get update && apt-get install -y build-essential
COPY requirements.txt .
RUN pip install wheel && pip install -r requirements.txt

COPY setup.py .
RUN pip install .

FROM public.ecr.aws/docker/library/python:3.9.15-slim AS final

COPY --from=build /opt/venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

ENTRYPOINT ["app"]
```

This saves about 140 MB

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Static binaries and scratch images

This is especially cool if you can compile to static binaries

Using the `scratch` image for your final stage gets us very small images

```
FROM public.ecr.aws/docker/library/golang:1.19 AS builder
WORKDIR /build
COPY . .

RUN CGO_ENABLED=0 go build -a -o hello-finch .

FROM scratch
COPY --from=builder /build/hello-finch /app/
WORKDIR /app
ENTRYPOINT ["./hello-finch"]
```

This image is 1.7 MB

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Free speed-up: Multistage parallel builds

BuildKit will analyze your Dockerfile ahead of starting a build

If it determines that a stage has no dependencies, it will start building it

If there are more than one, they will build in parallel, resulting in a faster build overall

Take advantage of this by avoiding unnecessary dependencies across stages

```
FROM public.ecr.aws/amazonlinux/amazonlinux:2022 AS base

FROM base AS foobuilder
RUN touch /foo
RUN sleep 8

FROM base AS barbuilder
RUN touch /bar
RUN sleep 4

FROM base
COPY --from=foobuilder /foo /app/
COPY --from=barbuilder /bar /app/
```

A little over 8 seconds to build

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Reduce layers by combining related operations

Often there are instructions that are related, for example, package operations

When possible, combine these into a single instruction

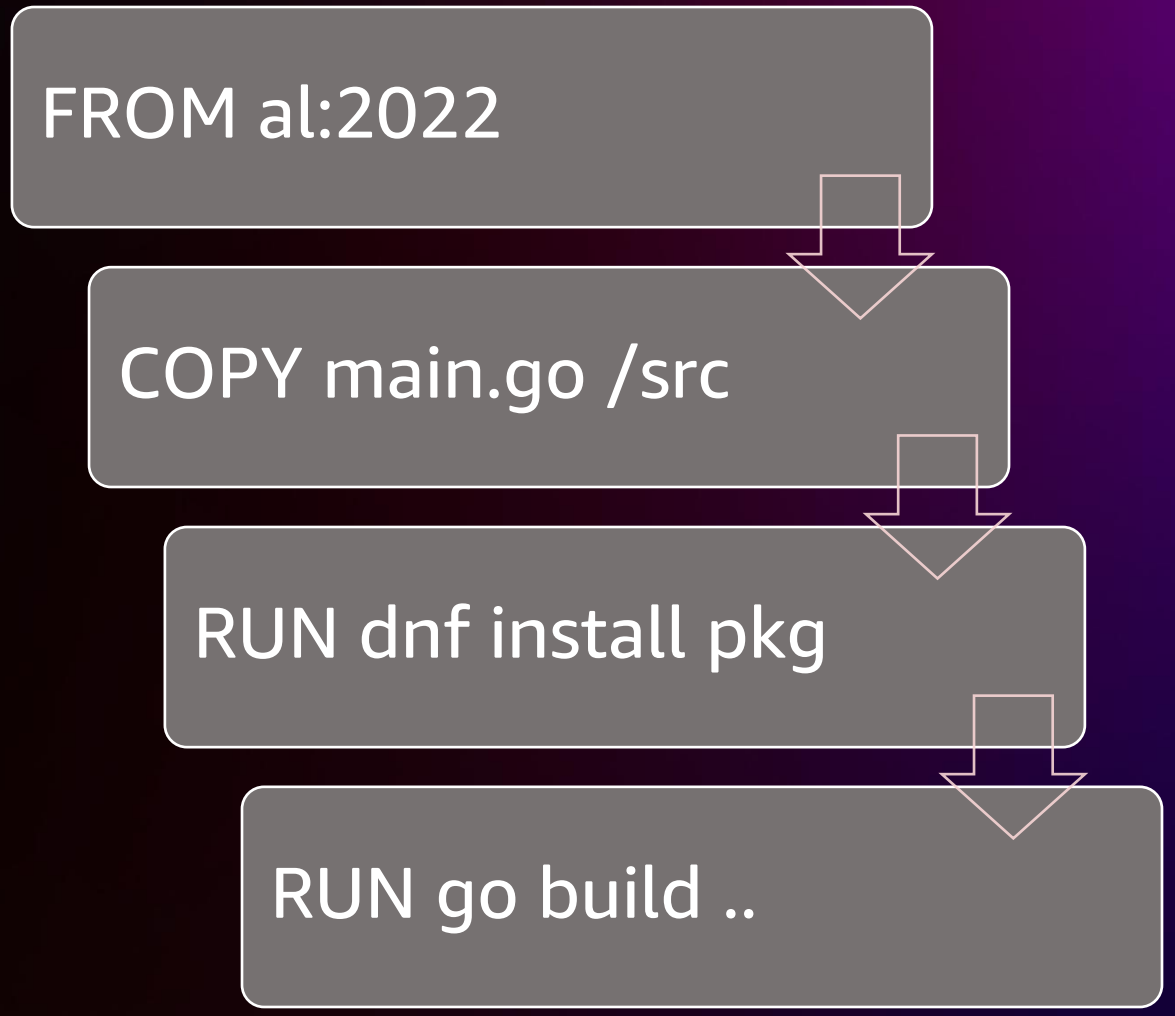
Same commands, resulting in a single layer

```
RUN dnf install -y gcc automake build-essential \
        && dnf clean all \
        && rm -rf /var/cache/yum
```

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Leverage the build cache

BuildKit manages a local build cache to make subsequent builds faster

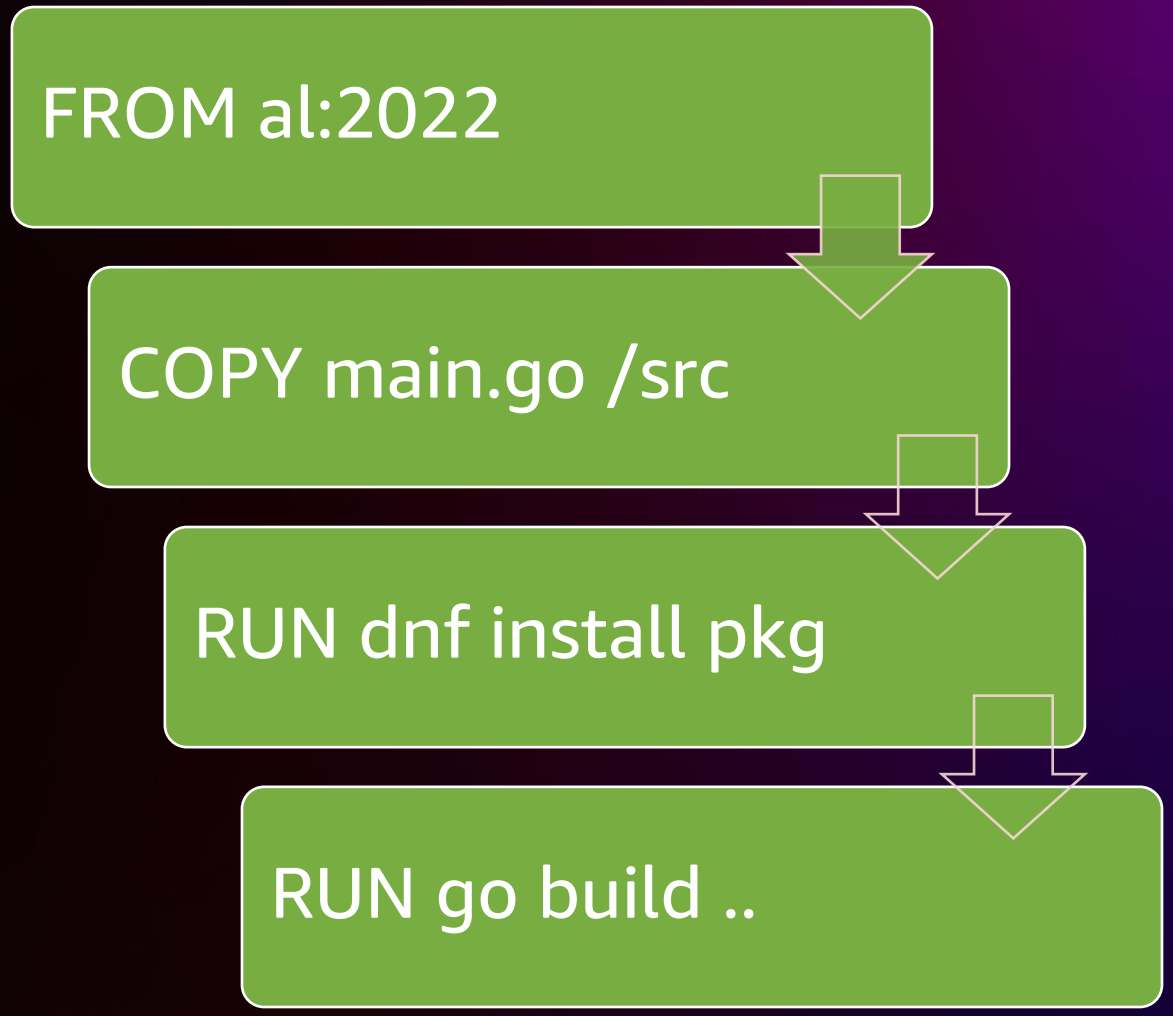


© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Leverage the build cache

BuildKit manages a local build cache to make subsequent builds faster

Once a cache is warmed with layers, unmodified layers are used



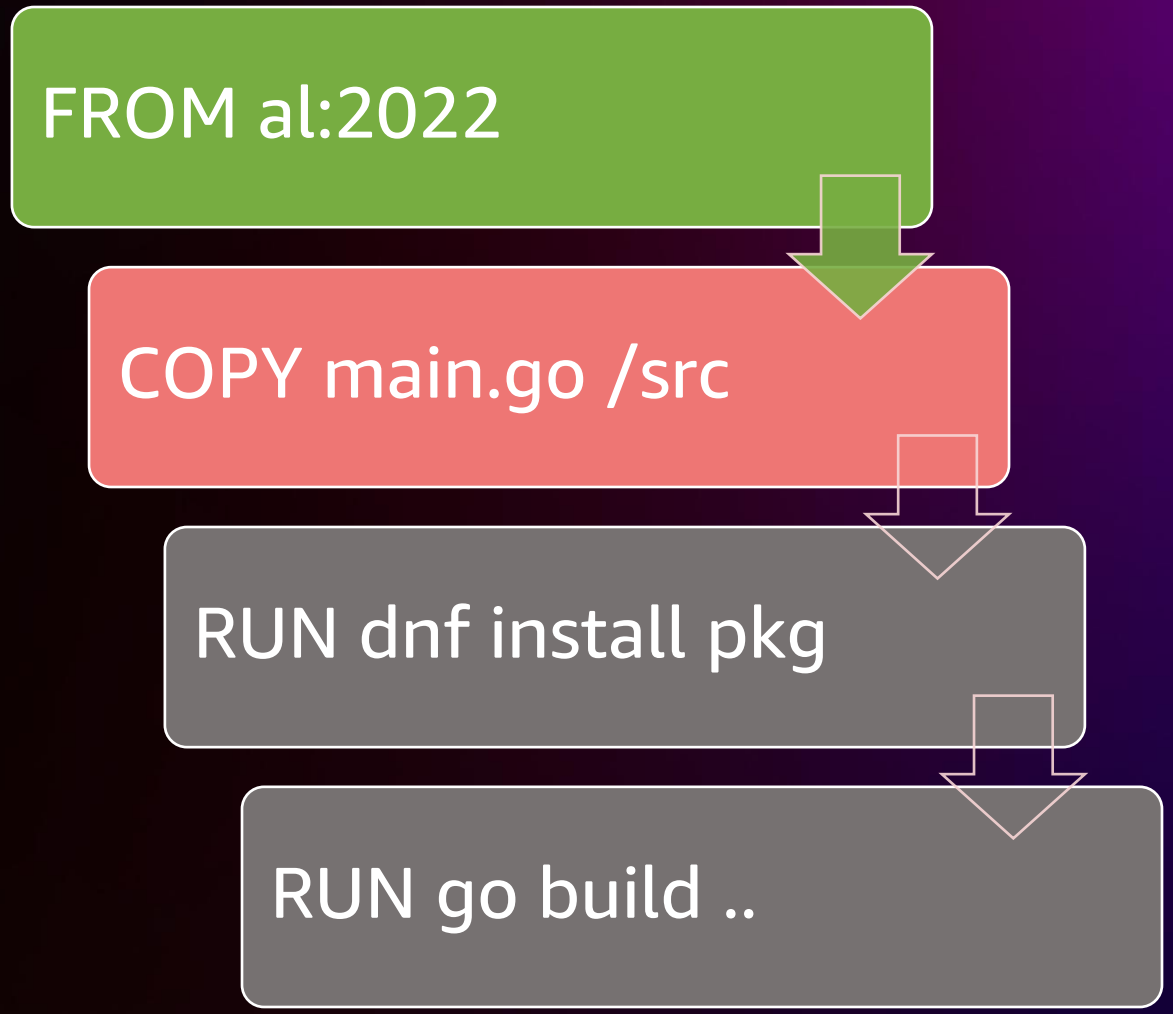
aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Leverage the build cache

BuildKit manages a local build cache to make subsequent builds faster

Once a cache is warmed with layers, unmodified layers are used

Cached layers are invalidated if the instruction results in a change



aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

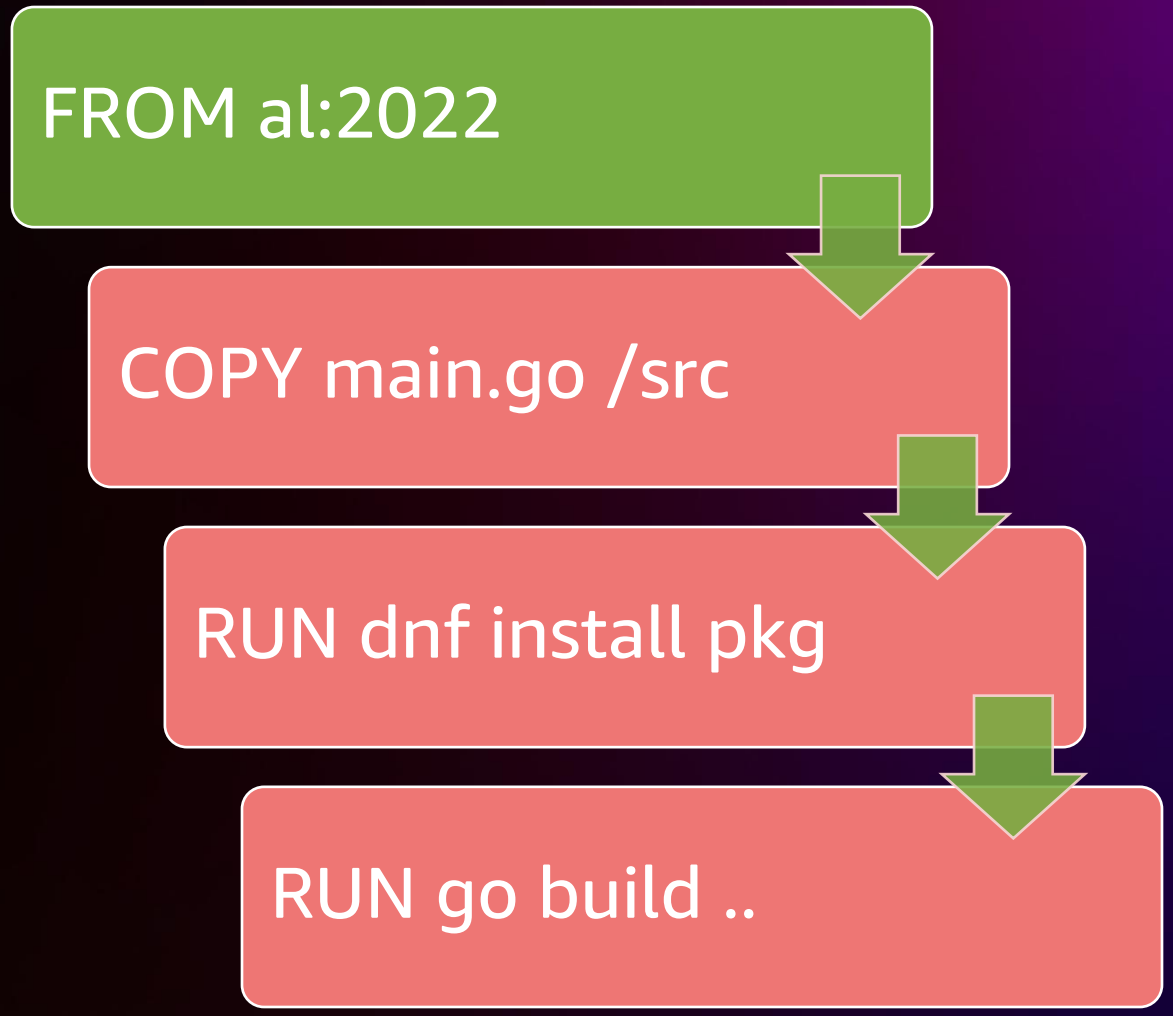
Leverage the build cache

BuildKit manages a local build cache to make subsequent builds faster

Once a cache is warmed with layers, unmodified layers are used

Cached layers are invalidated if the instruction results in a change

Since layers depend on each other, all subsequent layers are also invalidated



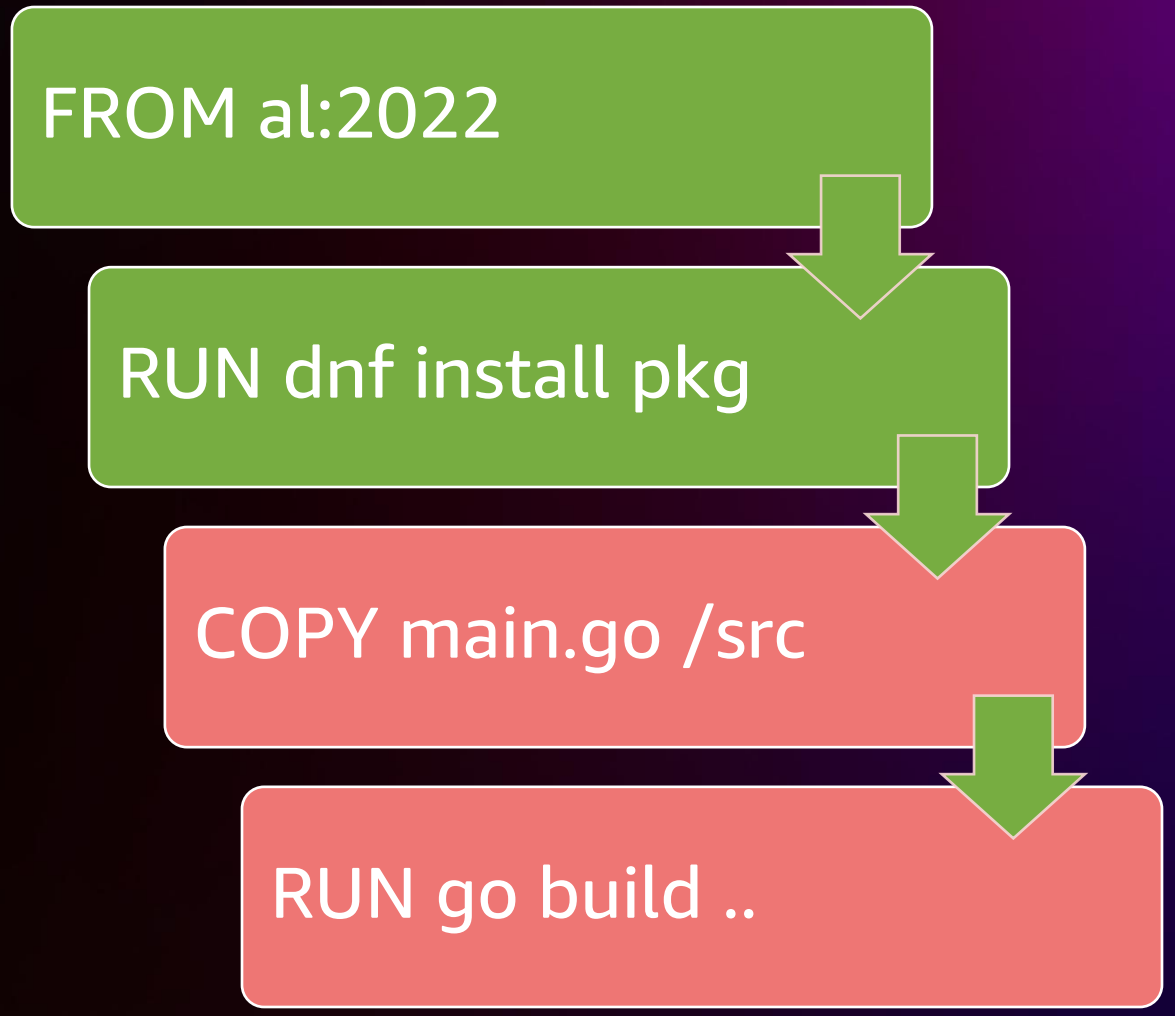
aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Leverage the build cache

Order your instructions top-down from least frequently changed

For example, setting up package installs for dependencies come early

Building your code comes later



aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

ENTRYPOINT or CMD

This one isn’t so hard and fast

Both provide default startup processes

- CMD can be overridden from CLI
- ENTRYPOINT takes args from the CLI

Typically, if you don’t have a reason to use CMD, use ENTRYPOINT

If you want the user to define their own starting process, use CMD

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Related, shell vs. exec formats

Again, both have their use cases (less so)

This is really just something to be aware of more than guidance

- Shell format will invoke all commands through a shell, interpreting environment each time
- Exec format simply runs the commands specified

Typically, if you don’t have a reason to use Shell format, use Exec

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Use secrets, not build arguments

Do not pass secrets to container builds with --build-arg, these are embedded

Definitely do not add a secret file then remove it – it remains

Instead, use BuildKit secrets

These use a special mount, and do not leave behind any trace of your secret

```
FROM public.ecr.aws/myvendor/coolsoftware:1.2.3

COPY . .
RUN --mount=type=secret,id=myvendor-api.key ./init-sdk
```



```
$ finch build --secret id=myvendor-api.key,src=host/path/to/vendor-api.key .
```

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Limit your scope

The UNIX philosophy of “do one thing and do it well” holds especially true with containers

It can be tempting to simply containerize a monolith, but this does not unlock the value of containers

Arguably, it’s easier to run a monolith on an instance

Embrace, refactor, and minimize

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Wrapping up

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

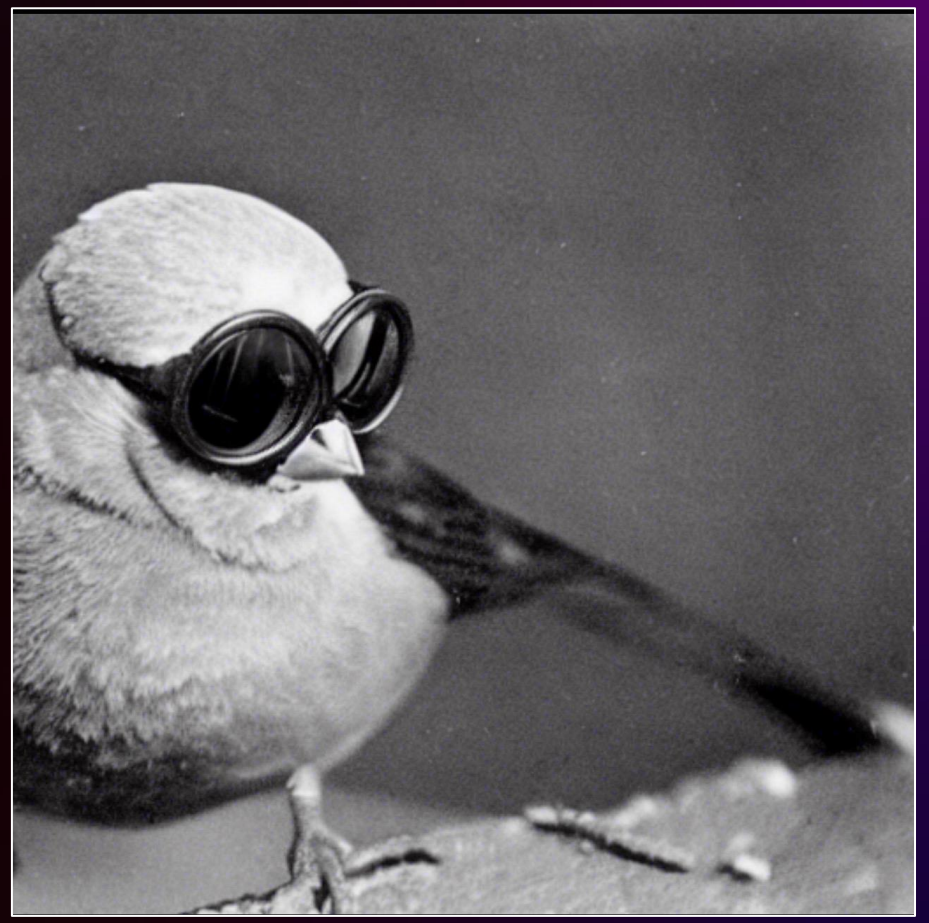
Connect with us, check out Finch!

Please come check out our new project

We'd love to hear what we can do better, and talk about what we can build together

Discussions, Issues, and Pull Requests all welcome!

A more refined public roadmap is in the works, as well as a dedicated site



aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

Phil Estes
Principal Engineer, AWS
@estesp

Jesse Butler
Principal Technologist, AWS
@jlb13



Please complete the session survey in the **mobile app**

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.