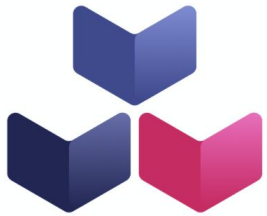
Intro to Cloud Native Buildpacks



Buildpacks.io

CLOUD NATIVE
COMPUTING FOUNDATION

$ whoami

Shaun McLernon

shaun@codesome.tech

Independent software engineer with 20+ years in development roles, and 5 years in ”devops” space. My current focus is working as a platform engineer.

Github: shaunmclernon

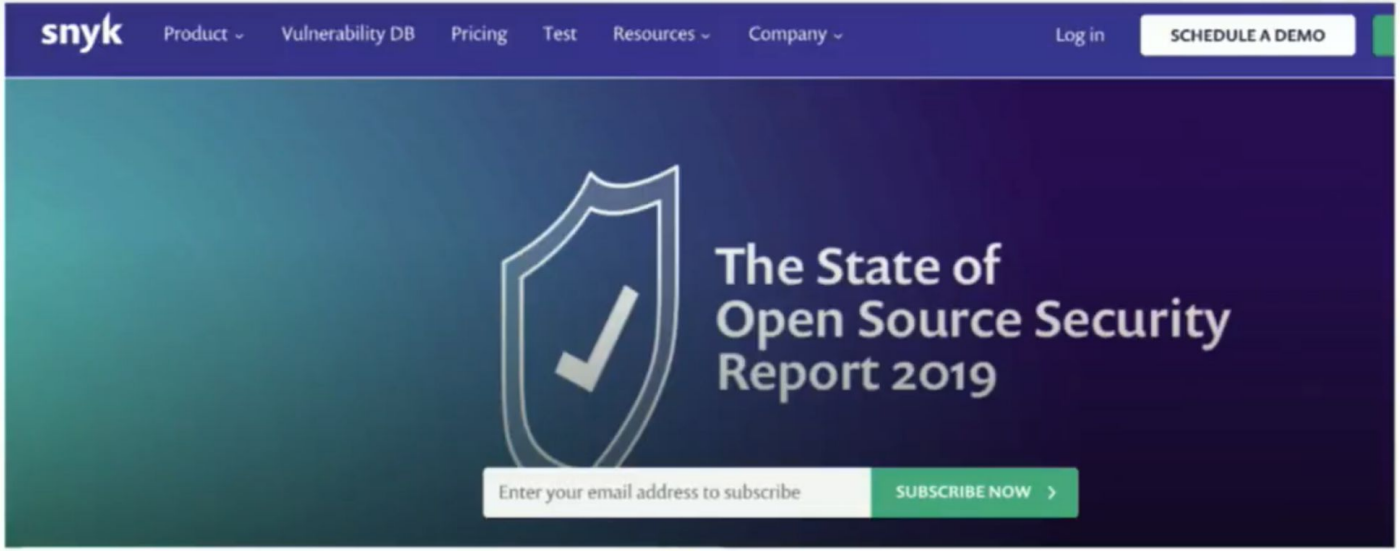
Twitter: @mclernon

Cloud Native Birmingham Slack: @shaun


CONTAINERIZE
ALL THE THINGS

Dockerfile bad practise - can you spot them?

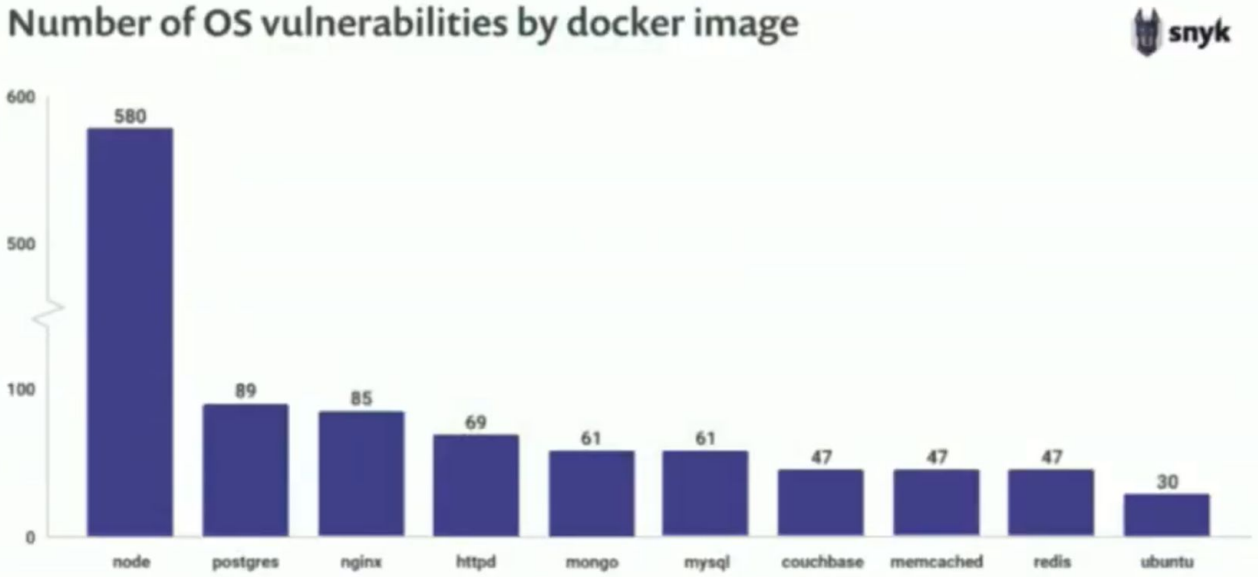
```
# DO NOT USE THIS DOCKERFILE AS AN EXAMPLE, IT IS BROKEN
FROM python:3

COPY yourscript.py /

RUN pip install flask

CMD [ "python", "./yourscript.py" ]
```



Top ten most popular docker images each contain at least 30 vulnerabilities

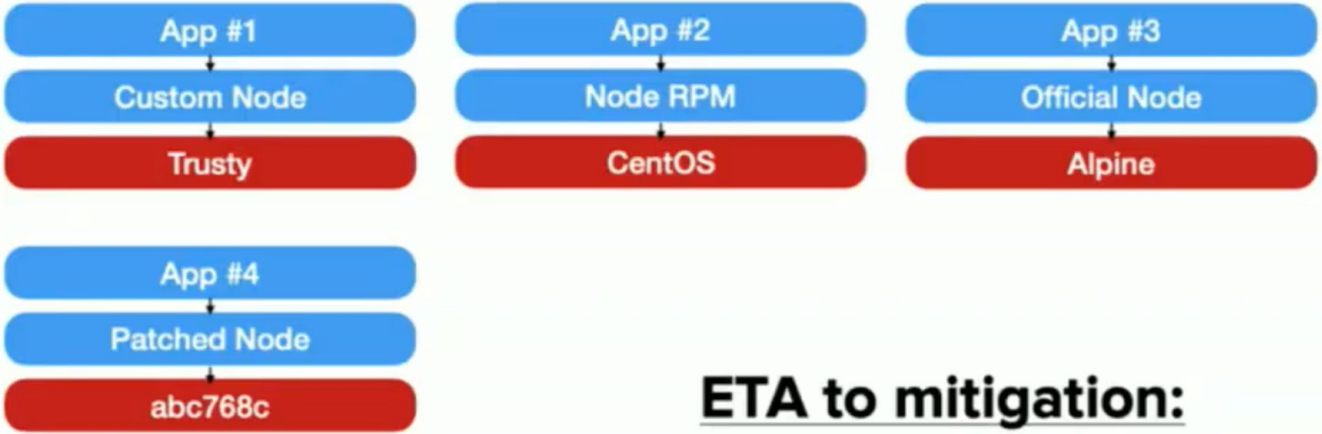
FEBRUARY 26, 2019 | IN ECOSYSTEMS, OPEN SOURCE | BY LIRAN TAL

https://snyk.io/blog/top-ten-most-popular-docker-images-each-contain-at-least-30-vulnerabilities/



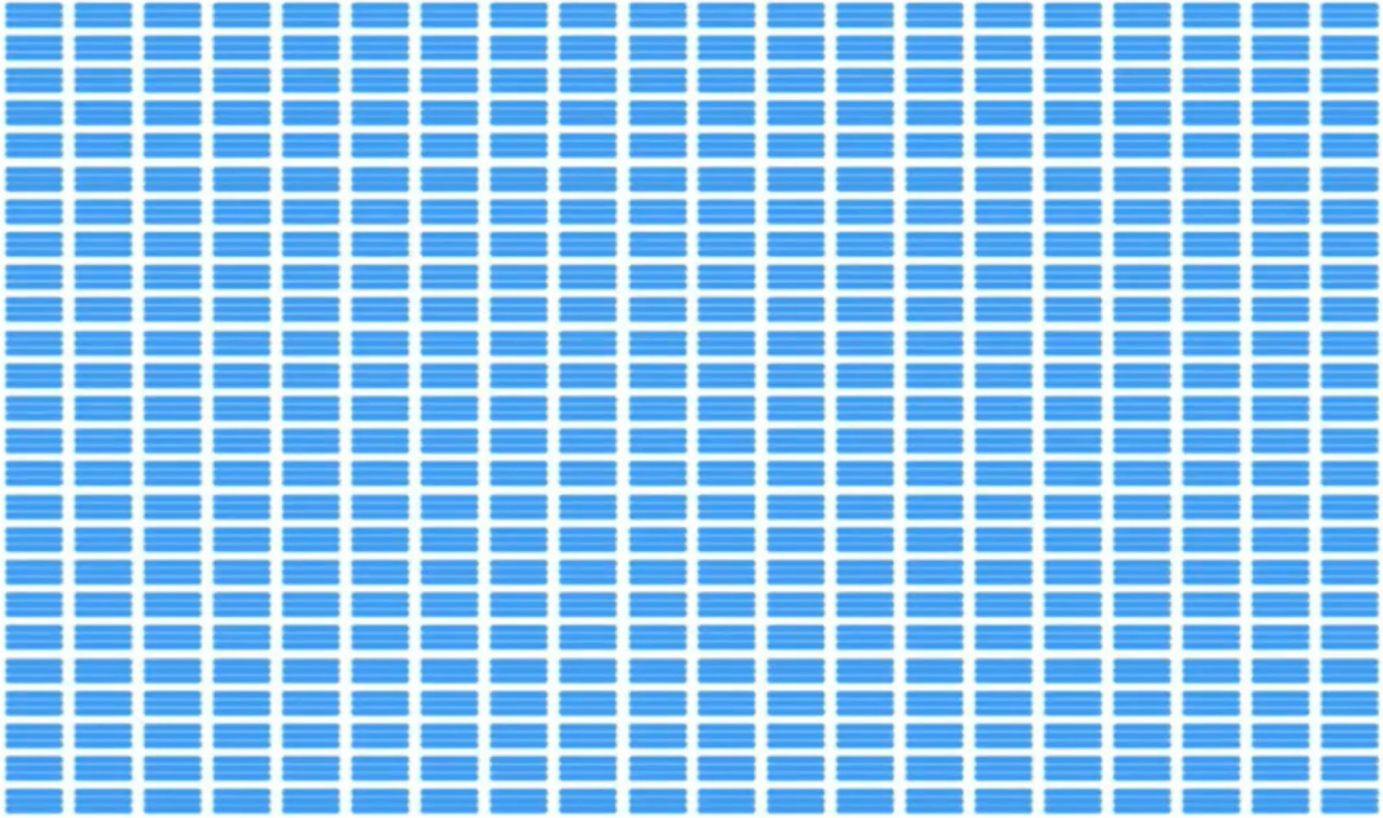
Fix can be easy if you're aware. 20% of images can fix vulnerabilities simply by rebuilding a docker image, 44% by swapping base image

Dockerfiles at Scale

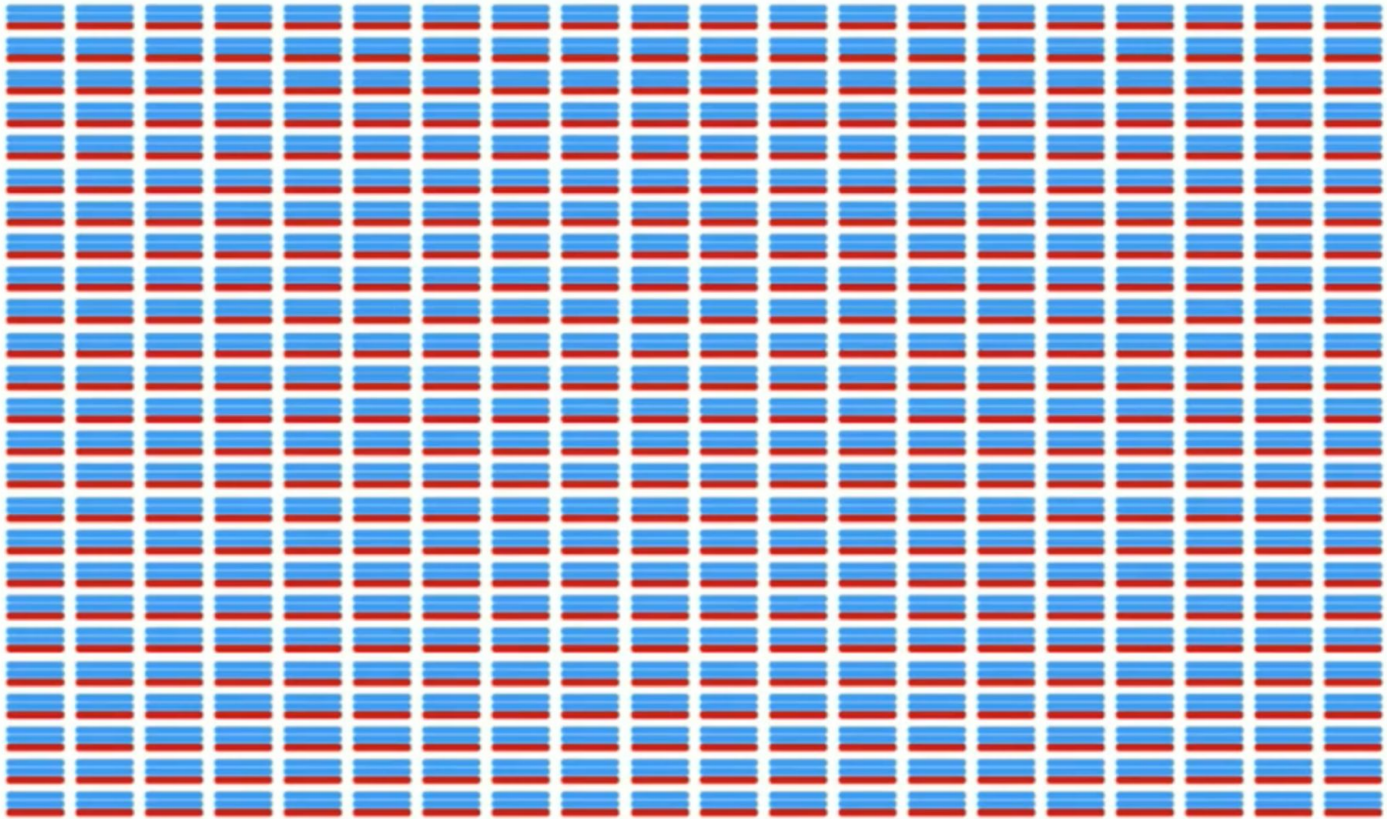


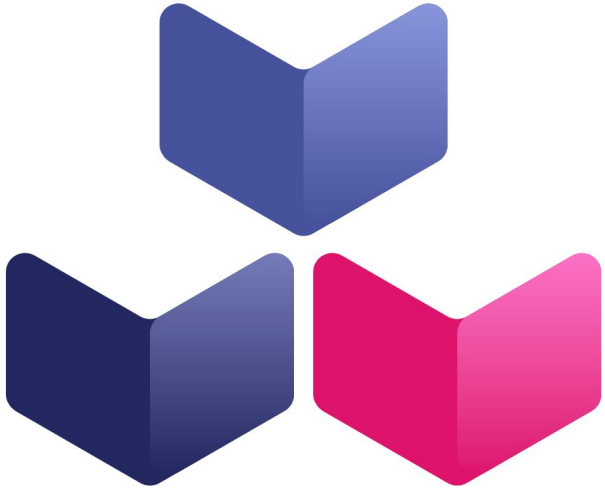
ETA to mitigation:
months? years?

500 Apps



500 Apps

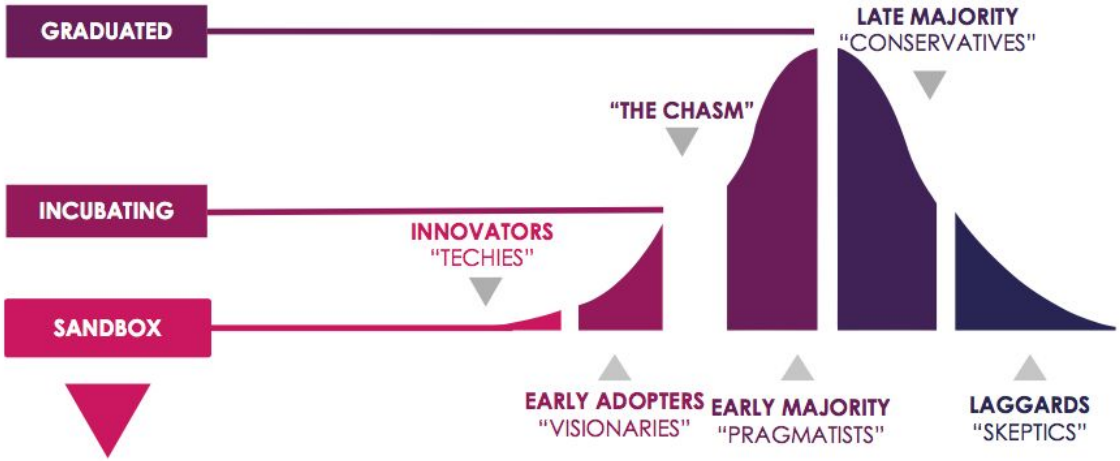


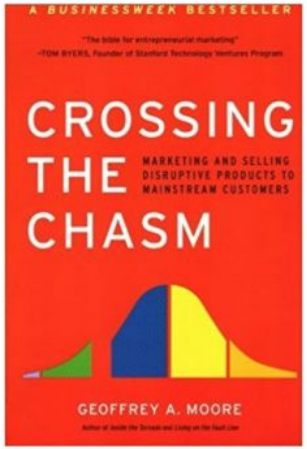


Cloud Native Buildpacks

A CNCF Sandbox project

CNCF Project Maturities

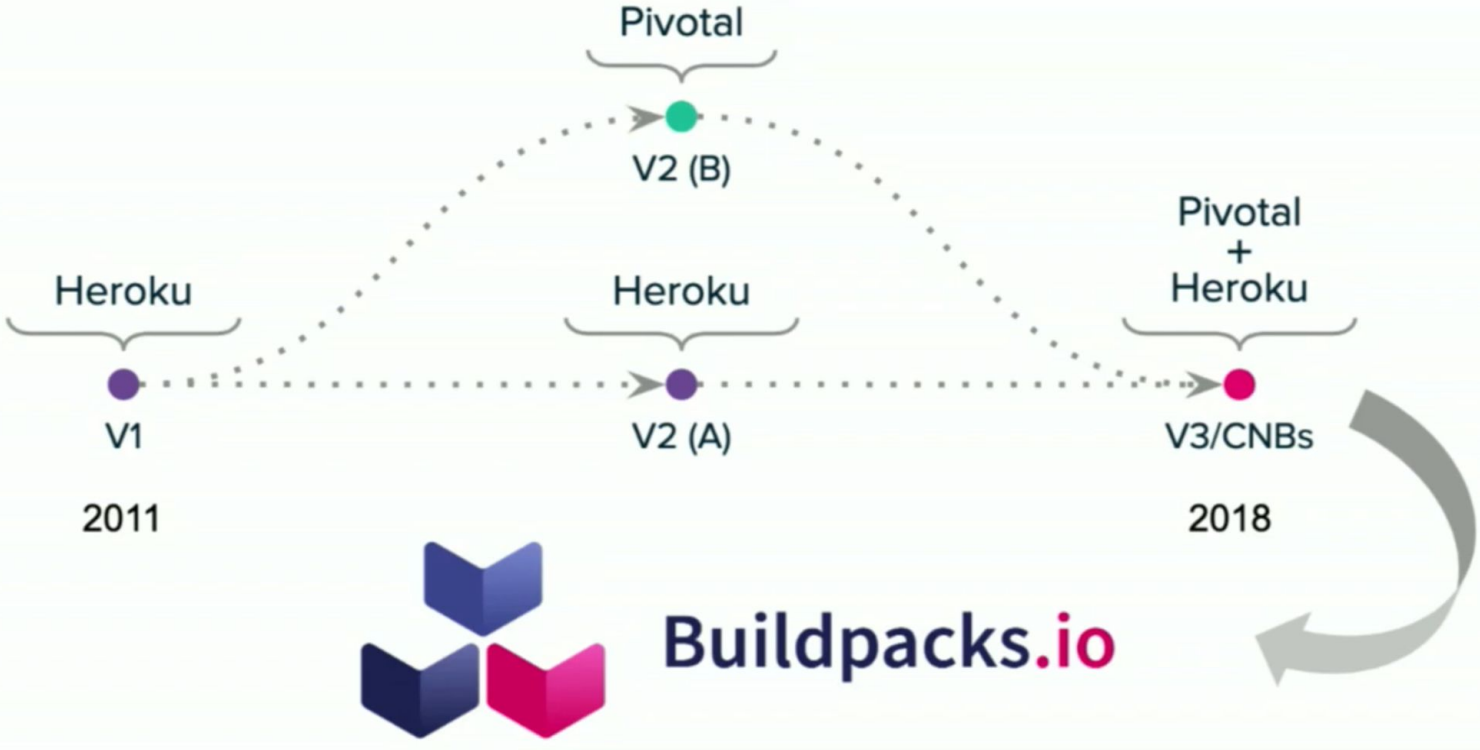


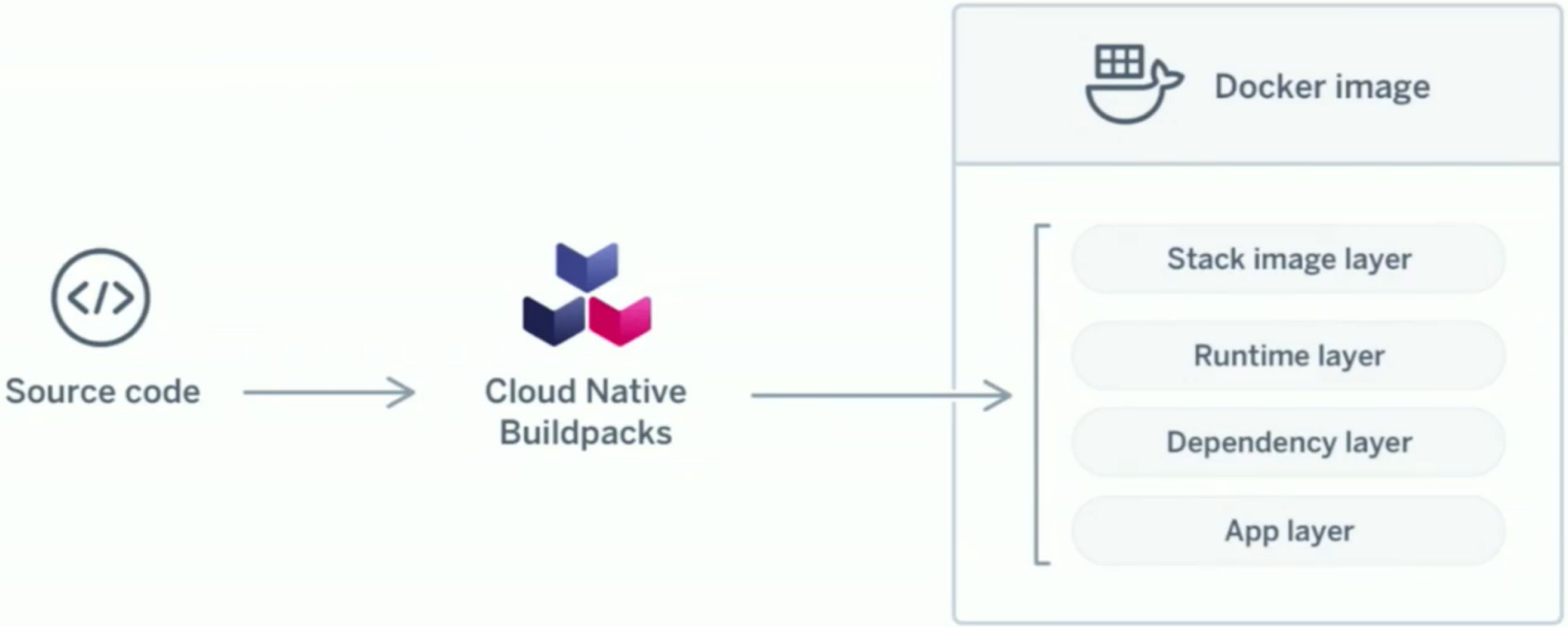


SANDBOX

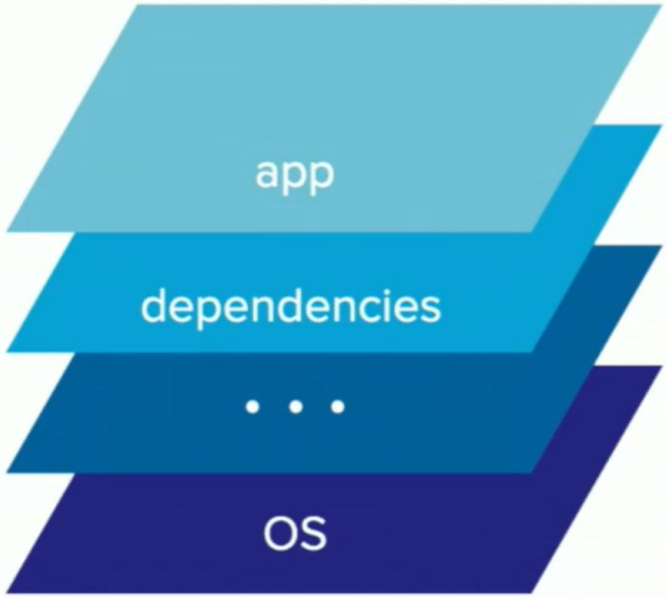
spiffe	SPIRE		OPENMETRICS	cortex	Buildpacks.io	Dragonfly	Virtual Kubelet	KubeEdge	BRIGADE
Identity Spec	Identity	Tooling	Metrics Spec	Monitoring	Packaging Spec	Image Distribution	Nodeless	Edge	Scripting
Network Service Mesh	OpenTelemetry	OpenEBS	Thanos	flux	STRIMZI	in-toto	KubeVirt	LONGHORN	ChubaoFS
Networking	Telemetry Spec	Storage	Monitoring	GitOps	Kafka Operator	Security	VM Operator	Storage	Storage

History of Buildpacks




Source code
Cloud Native Buildpacks
Docker image
Stack image layer
Runtime layer
Dependency layer
App layer

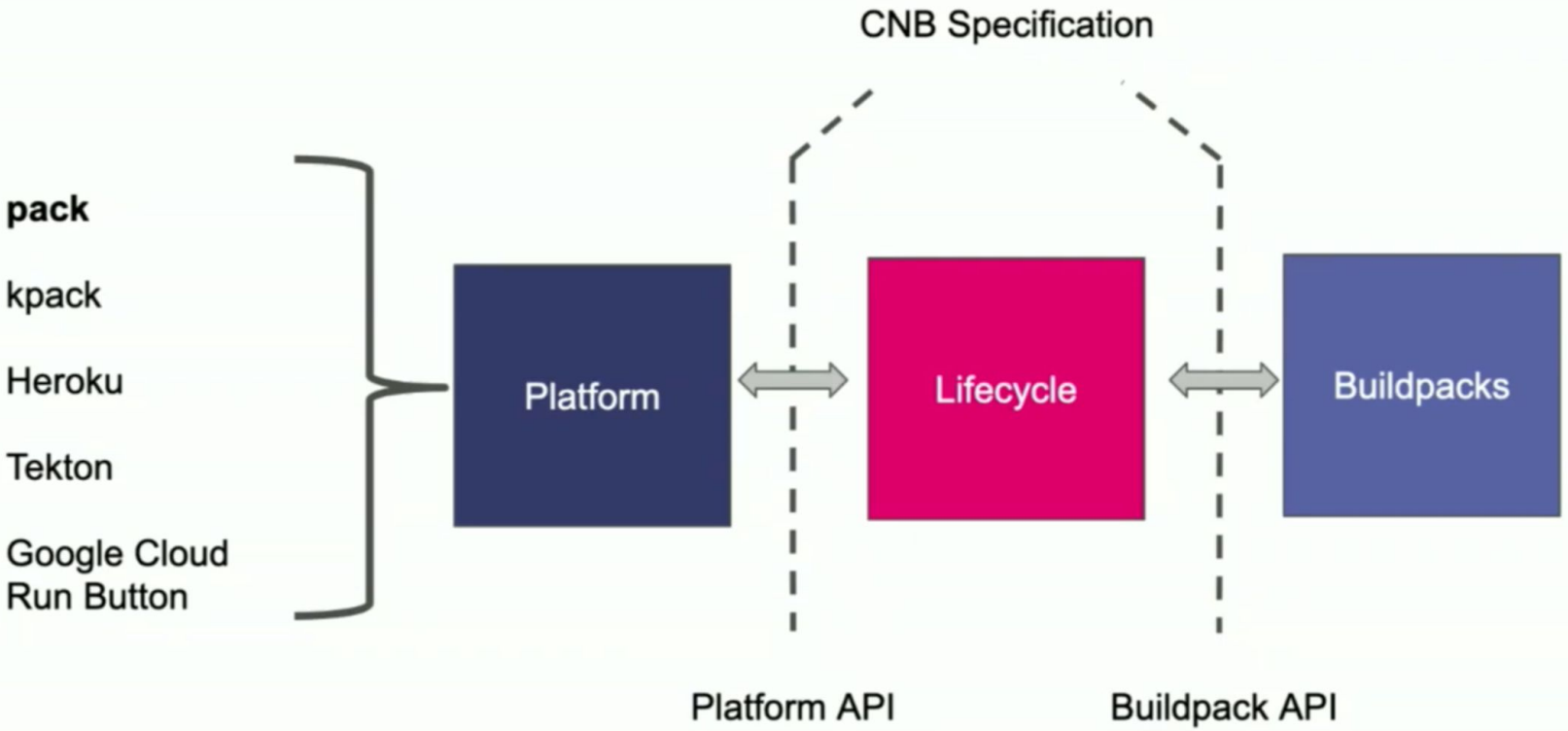
The Result



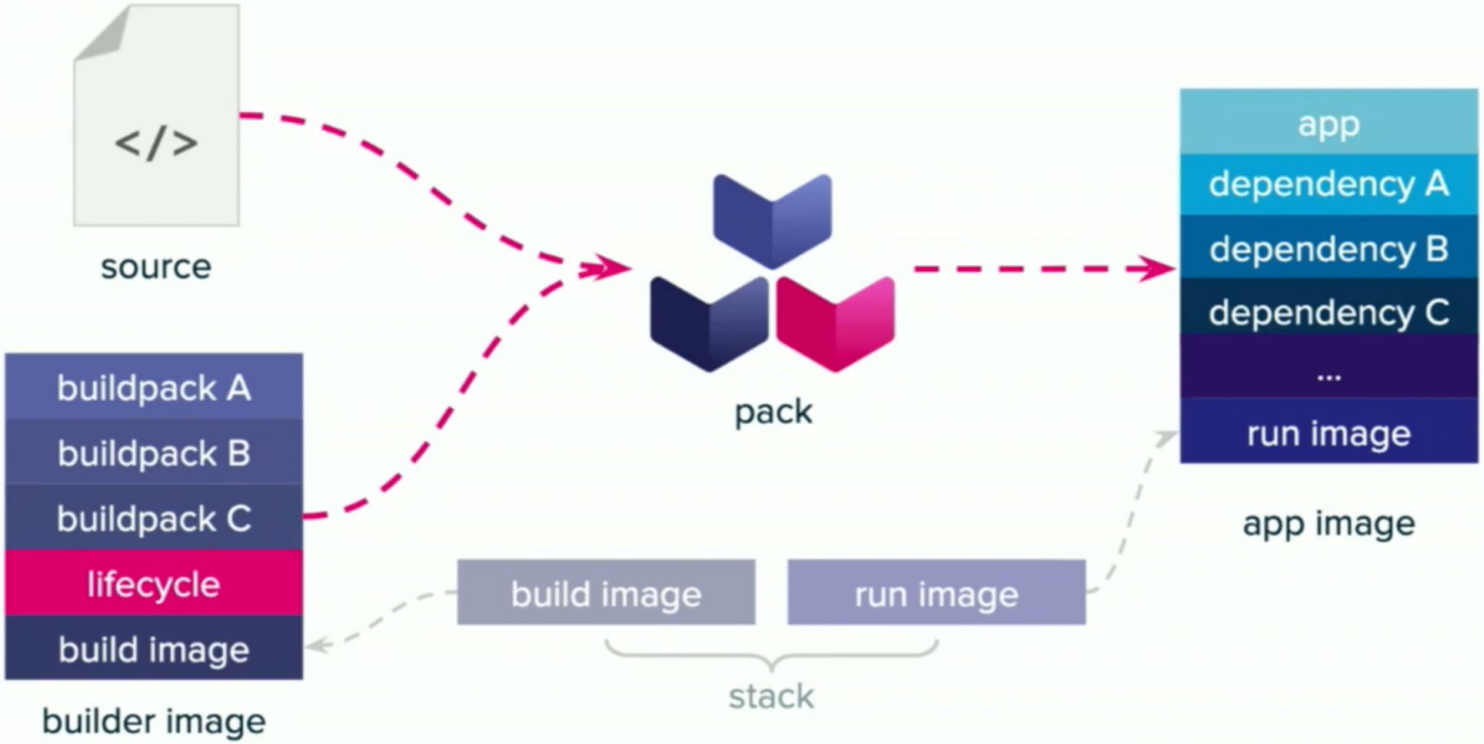
App image that has

- a reproducible build
- metadata that can be inspected
- logical mapping of layers to components

CNB: An Open Standard



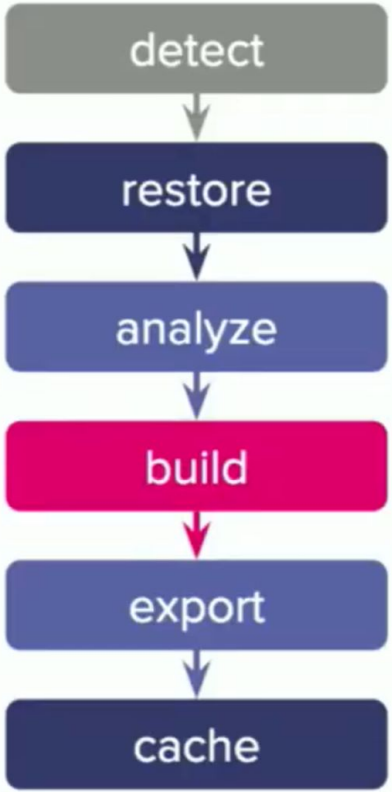
Build


</>
source
buildpack A
buildpack B
buildpack C
lifecycle
build image
builder image
pack
app
dependency A
dependency B
dependency C
...
run image
app image
build image
run image
stack

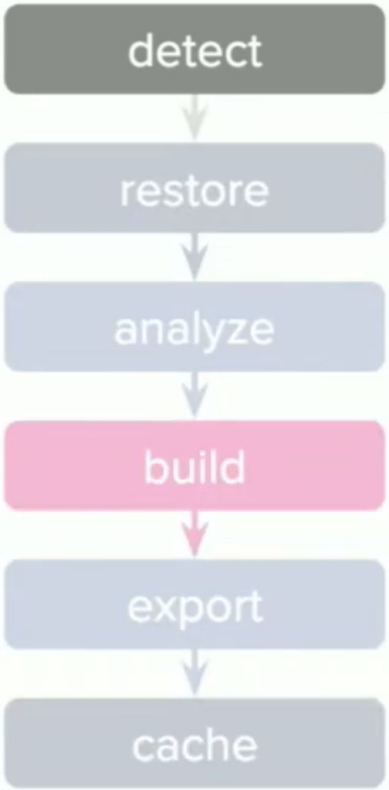
So… lets build an app...

So what did we just do?

Lifecycle



Lifecycle: Detect



- Tests groups of buildpacks against source, in order (via each buildpack's `detect` binary)
- First group that passes is selected

```
src/
  package.json
  yarn.lock
  ...
```

Node CNB Yarn CNB

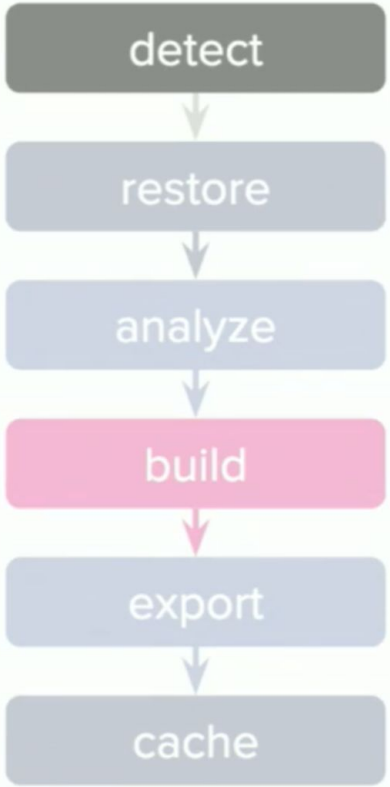
Node CNB NPM CNB

```
src/
  package.json
  ...
```

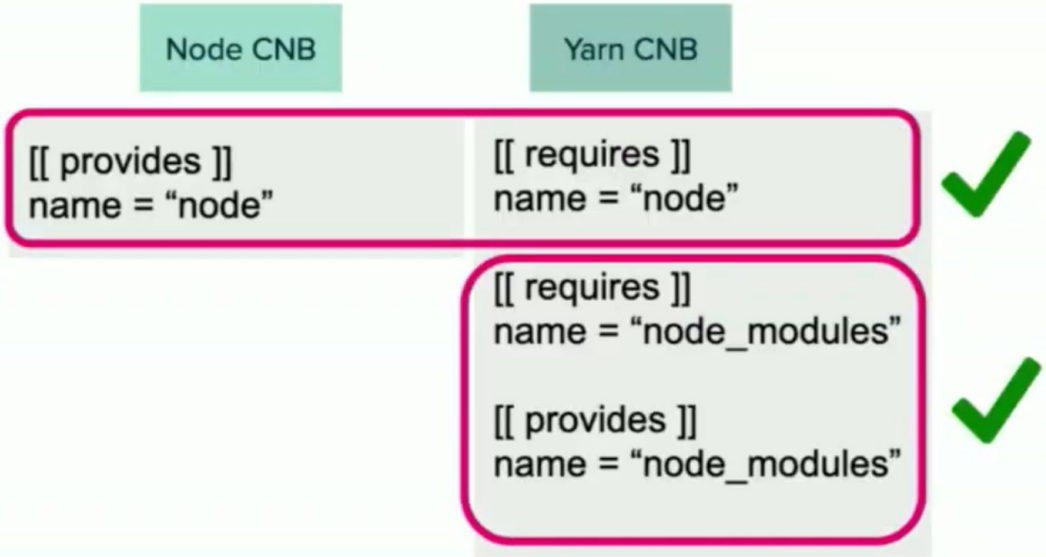
Node CNB Yarn CNB

Node CNB NPM CNB

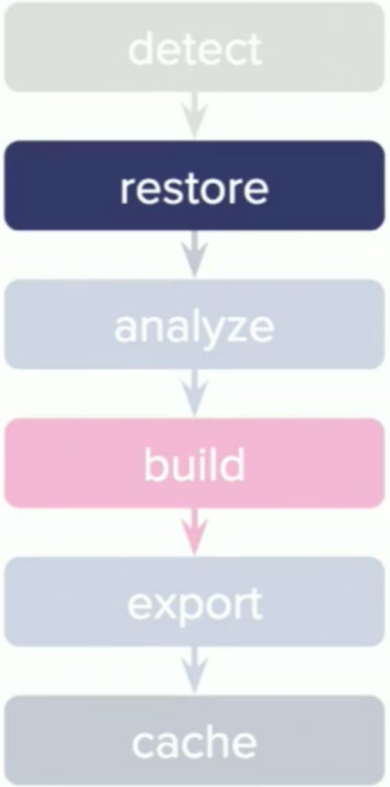
Lifecycle: Detect - Build Plan



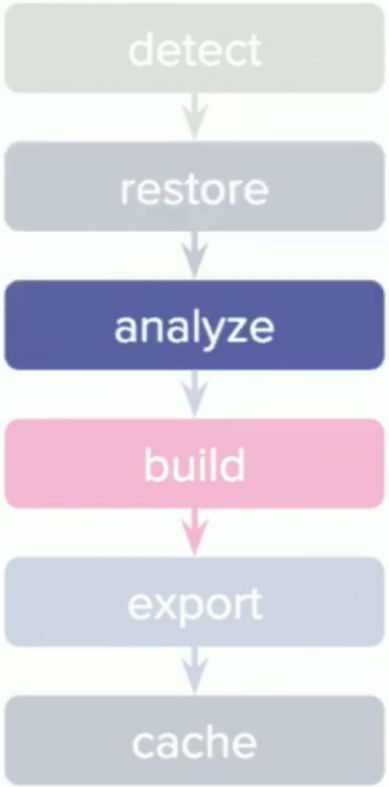
- Buildpacks detect in parallel
- The build plan allow buildpacks to coordinate during detection
- Composability allows for easy customization or extension



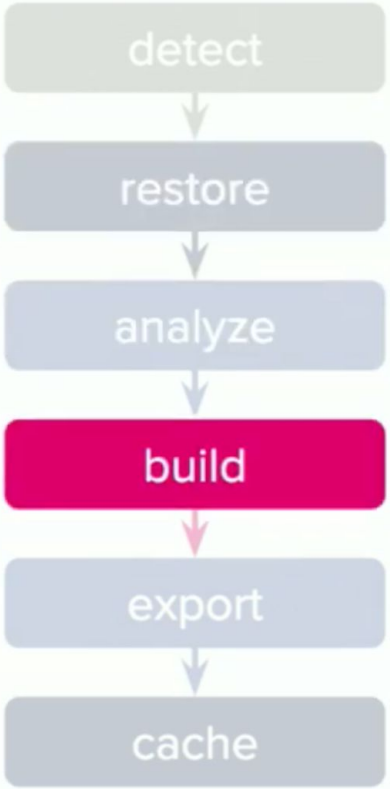
Lifecycle: Restore



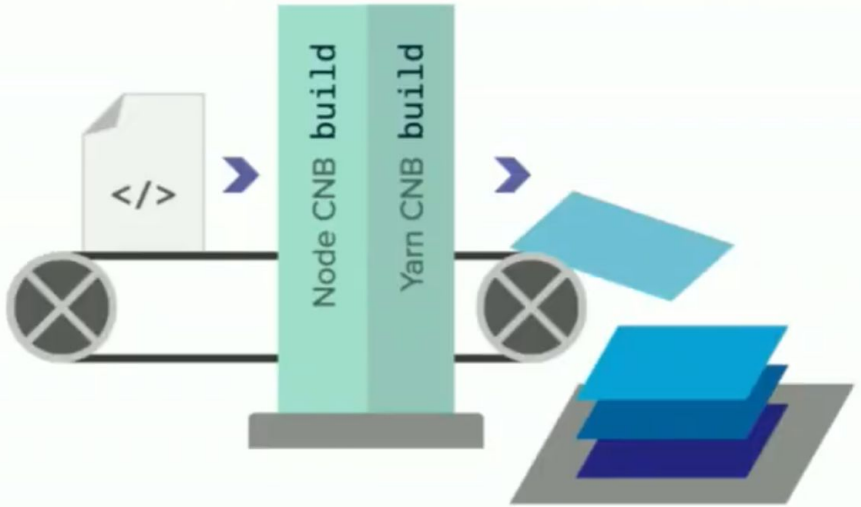
Lifecycle: Analyze



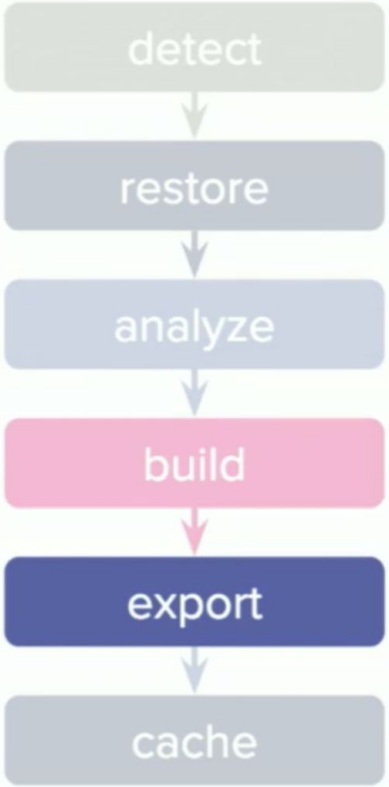
Lifecycle: Build



- For previously-selected group, executes each buildpack's `build` binary, in order
- *Recall:* `build` gathers dependencies, compiles app (if needed), and sets launch command

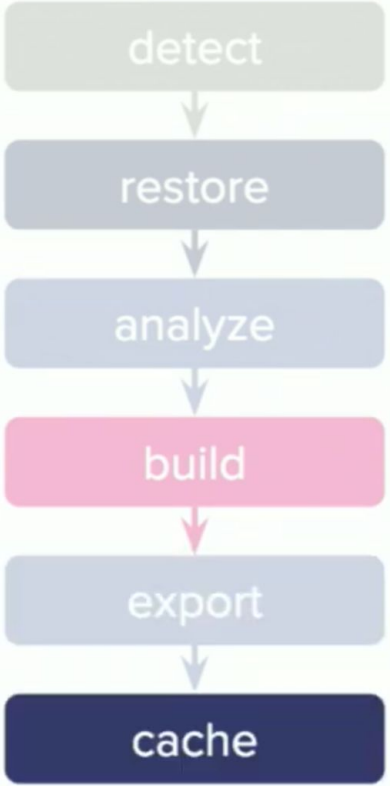


Lifecycle: Export

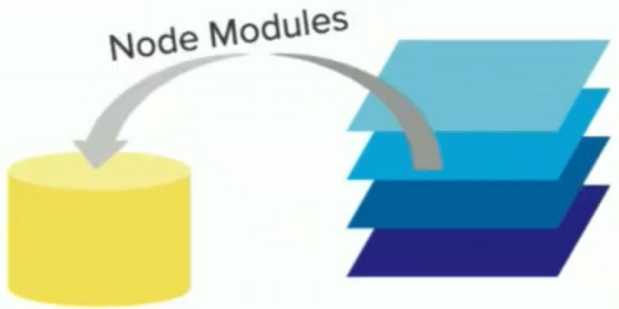


- Assembles final layers into image
- Combines information from `analyze` phase to ensure only changed layers are updated

Lifecycle: Cache



- Caches any necessary dependencies
- Retrieved on next build's `restore` phase



First Build

Detect: Node.js + Yarn

Restore

Analysis

Build
install node.js runtime
yarn install

Export
create app layer
create node engine layer
create npm modules layer
create OS layer

Caching
create cache layers

OCI Image

- node_modules
- node-engine
- app
- ubuntu:18.04
- configuration

Cache Layers

- node-engine
- node_modules
- npm cache

First Build

Detect: Node.js + Yarn

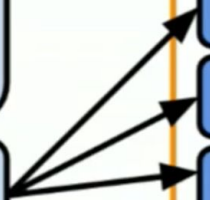
Restore

Analysis

Build
install node.js runtime
yarn install

Export
create app layer
create node engine layer
create npm modules layer
create OS layer

Caching
create cache layers



OCI Image

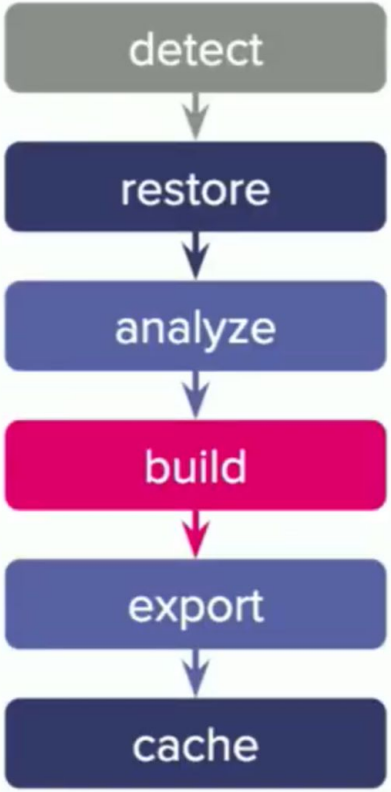
- node_modules
- node-engine
- app
- ubuntu:18.04
- configuration

Cache Layers

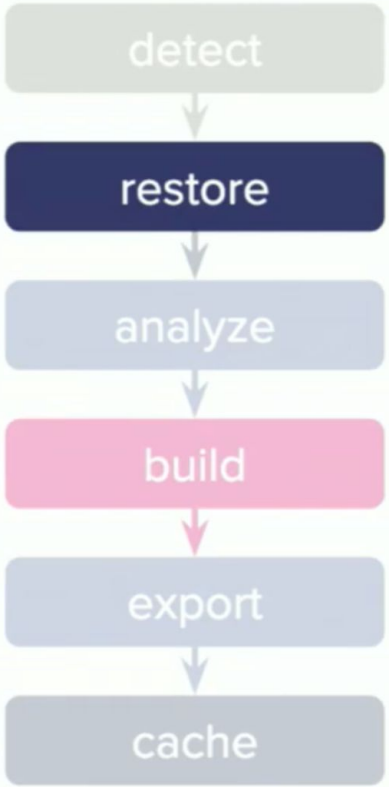
- node-engine
- node_modules
- npm cache

Update some dependencies

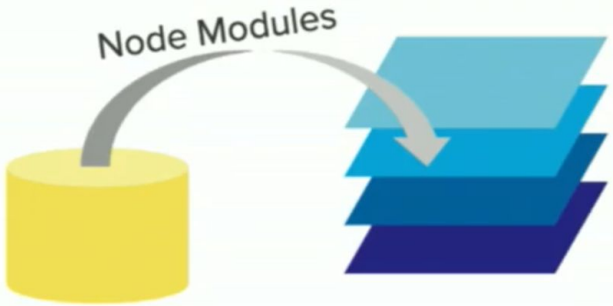
Lifecycle



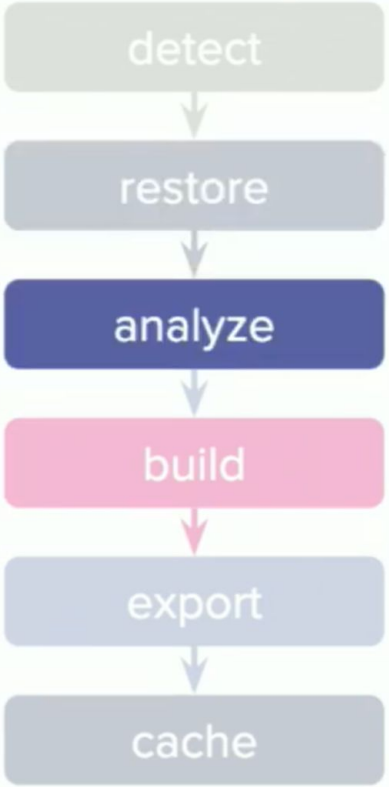
Lifecycle: Restore



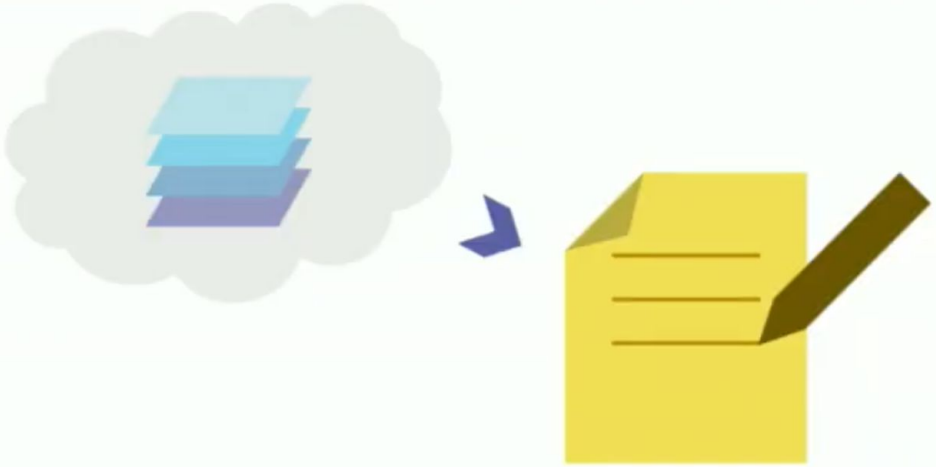
- Restores any previously-cached dependencies
- Cache is local to platform
  (local machine, k8s cluster PVC, etc.)



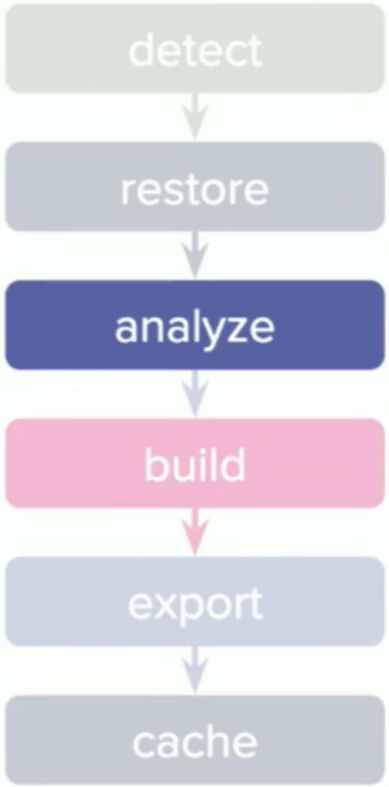
Lifecycle: Analyze



- Gathers metadata about previously-built image
- Used during `export` to avoid re-uploading unchanged layers



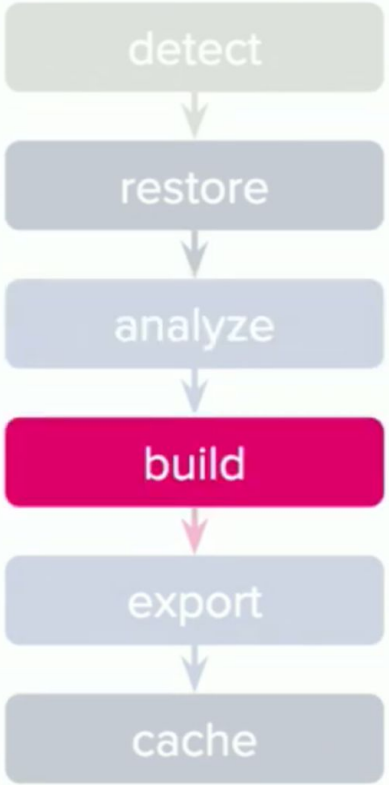
Lifecycle: Analyze - example metadata file



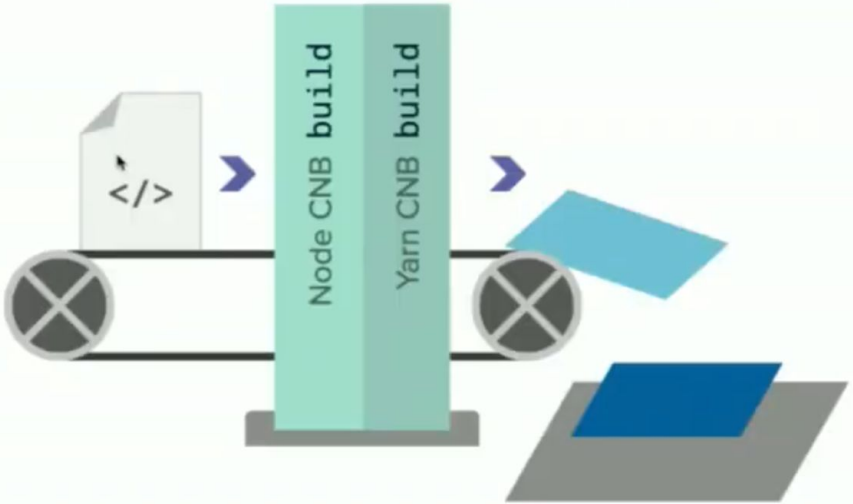
Node CNB → node.toml

```
build = true
cache = true
launch = true

[metadata]
  id = "node"
  name = "Node Engine"
  version = "10.17.0"
  uri = "https://nodejs.org/dist/v10.17.0/node-v10.17.0-linux-x64.tar.gz"
  sha256 = ...
```

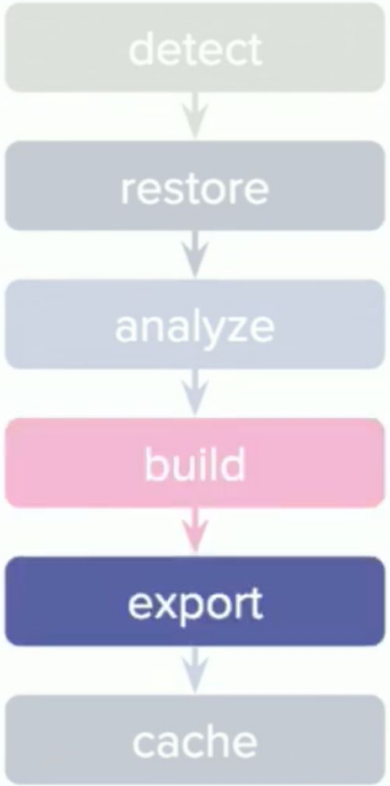
Lifecycle: Build



- Only rebuilds layers that should change

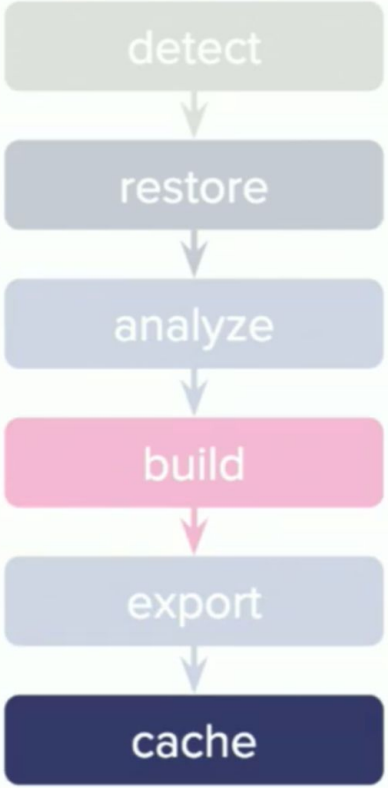


Lifecycle: Export

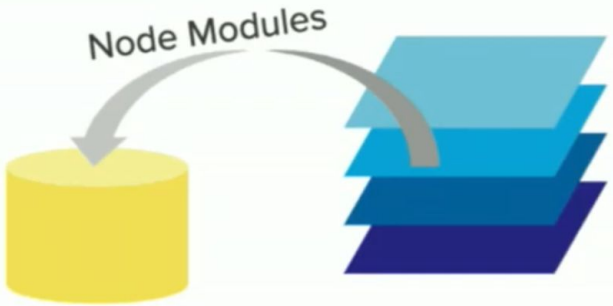


- Assembles final layers into image
- Combines information from `analyze` phase to ensure only changed layers are updated

Lifecycle: Cache



- Caches any necessary dependencies
- Retrieved on next build's `restore` phase



Second Build

Detect: Node.js + Yarn

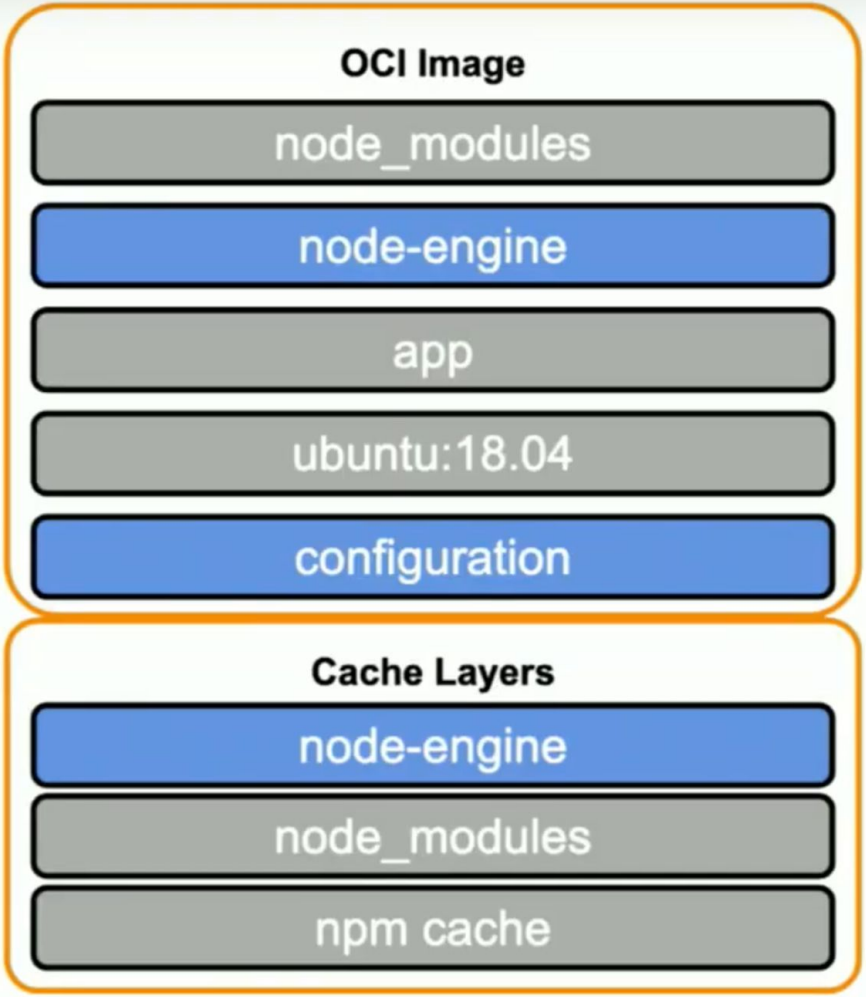
Restore
read cached layers
write cached layers to disk

Analysis
read metadata about layers
write metadata to disk for build

Build
install new node engine

Export
update node-engine layer
update configuration layer

Caching
update node-engine



Second Build

Detect: Node.js + Yarn

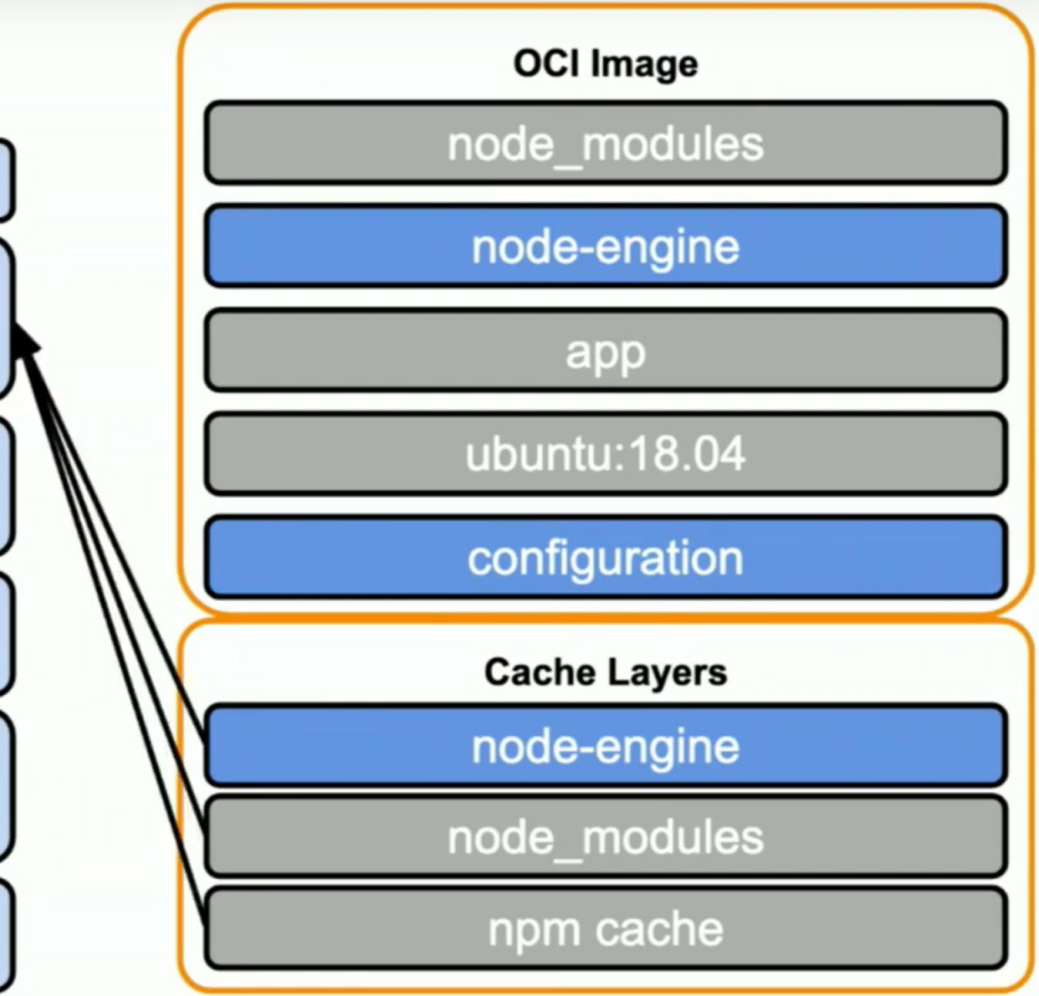
Restore
read cached layers
write cached layers to disk

Analysis
read metadata about layers
write metadata to disk for build

Build
install new node engine

Export
update node-engine layer
update configuration layer

Caching
update node-engine



Second Build

Detect: Node.js + Yarn

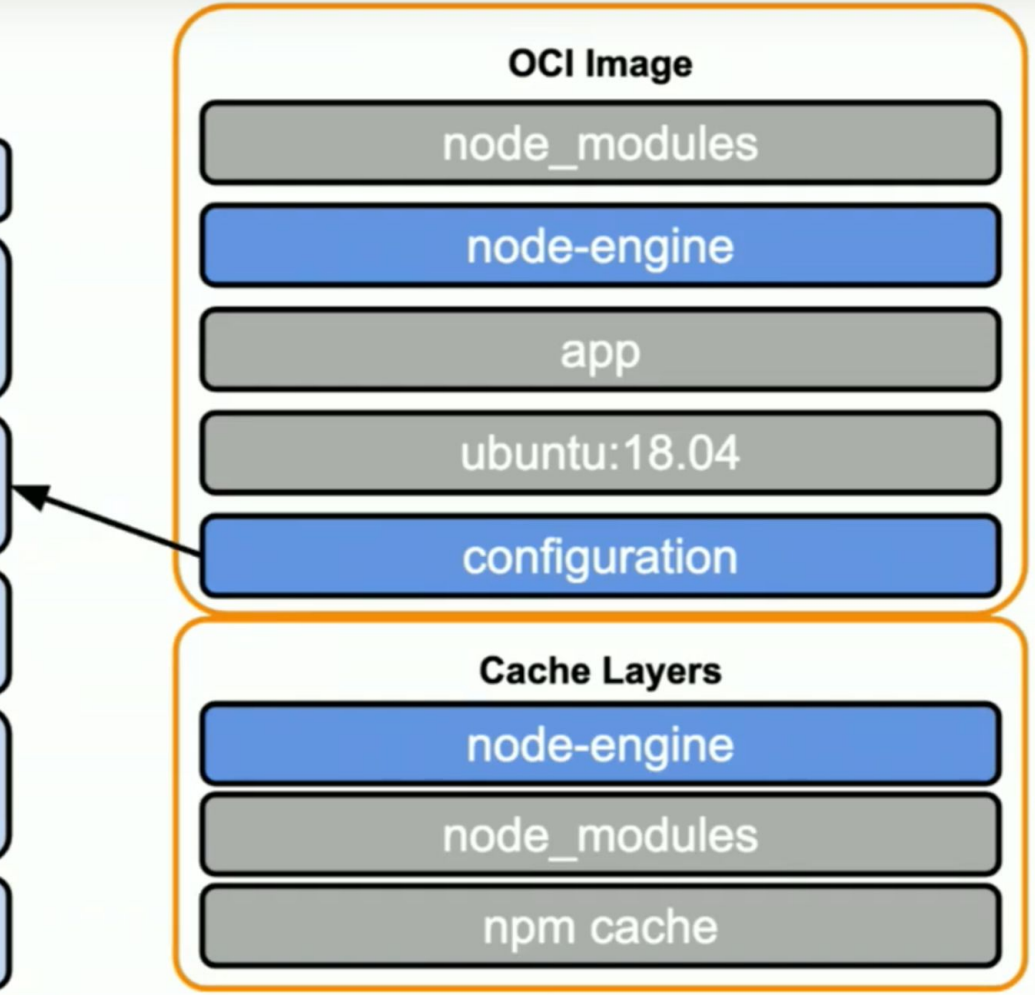
Restore
read cached layers
write cached layers to disk

Analysis
read metadata about layers
write metadata to disk for build

Build
install new node engine

Export
update node-engine layer
update configuration

Caching
update node-engine



Second Build

Detect: Node.js + Yarn

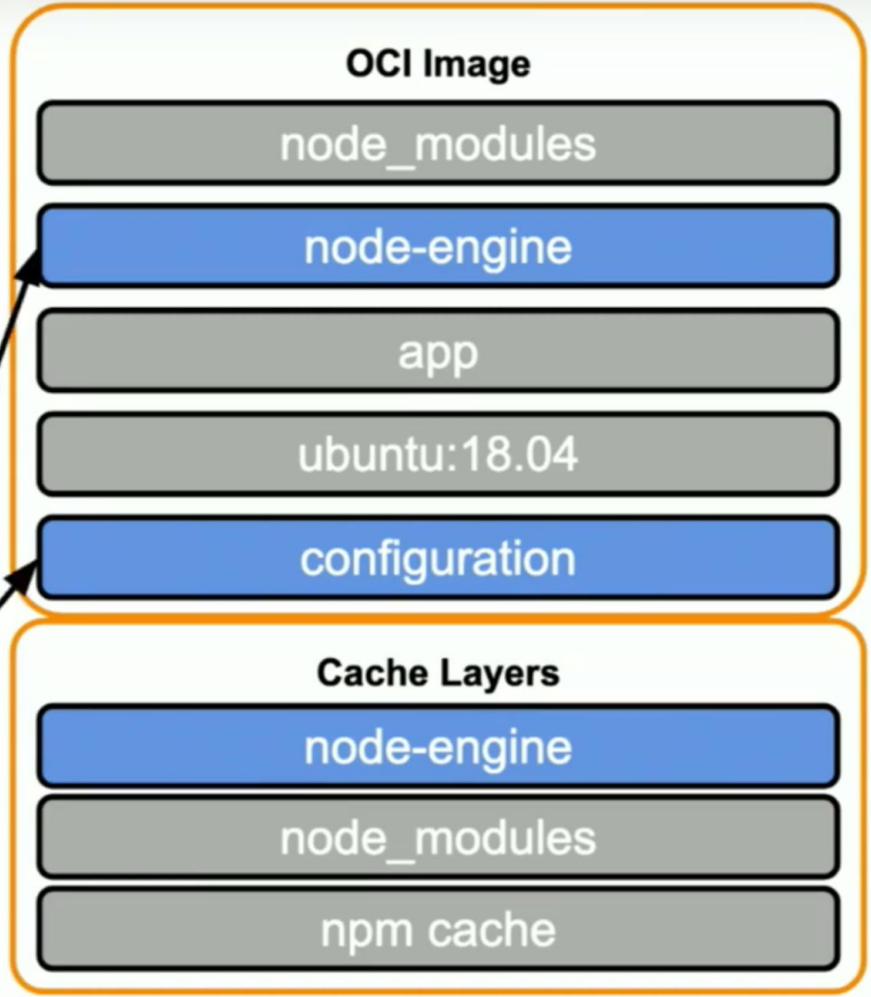
Restore
read cached layers
write cached layers to disk

Analysis
read metadata about layers
write metadata to disk for build

Build
install new node engine

Export
update node-engine layer
update configuration

Caching
update node-engine



Second Build

Detect: Node.js + Yarn

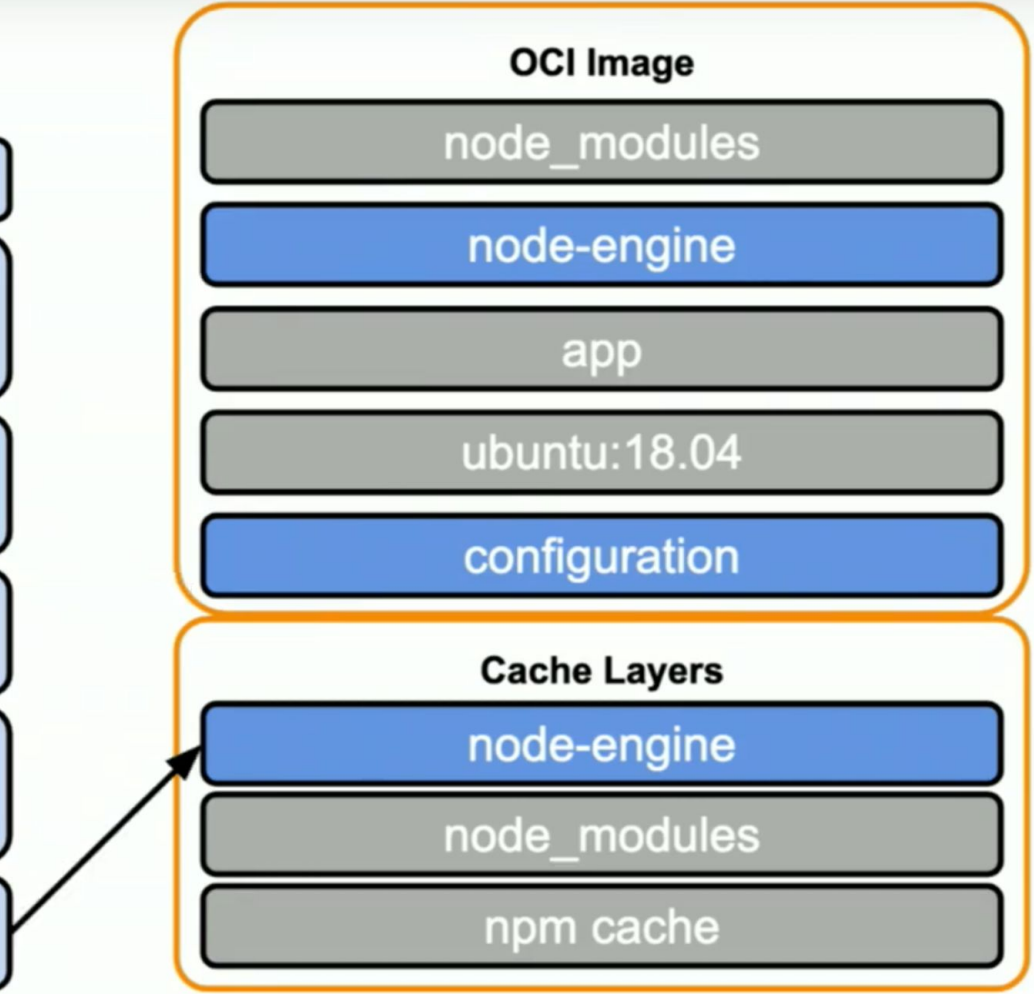
Restore
read cached layers
write cached layers to disk

Analysis
read metadata about layers
write metadata to disk for build

Build
install new node engine

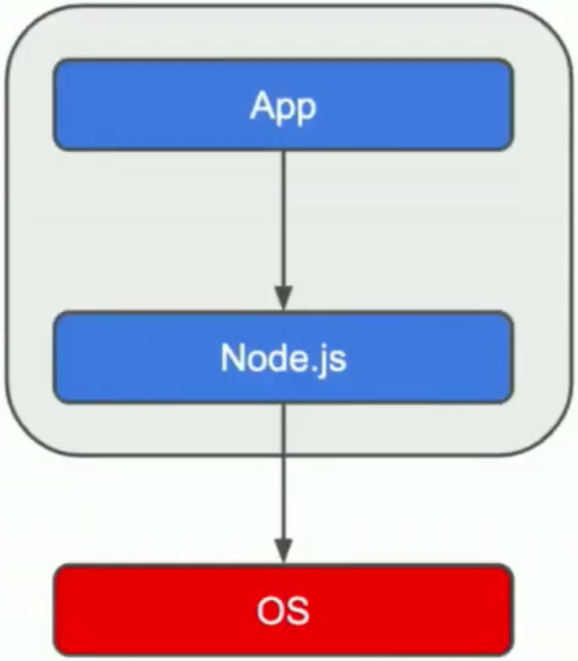
Export
update node-engine layer
update configuration

Caching
update node-engine



Let’s Swap the Base Layers

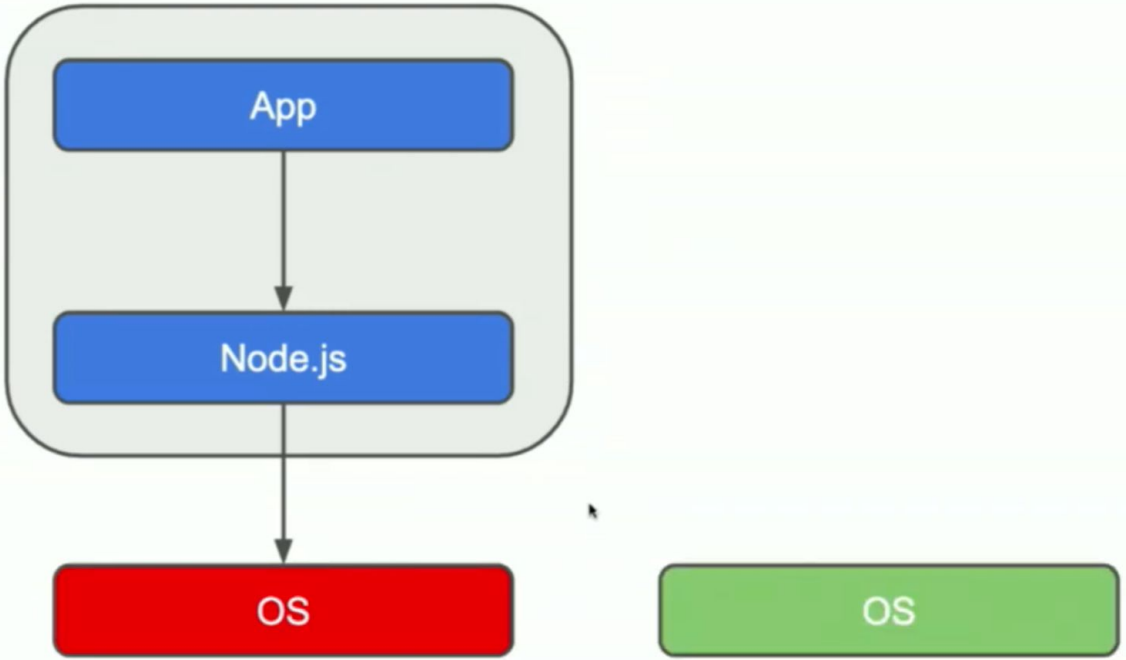
Node.js App



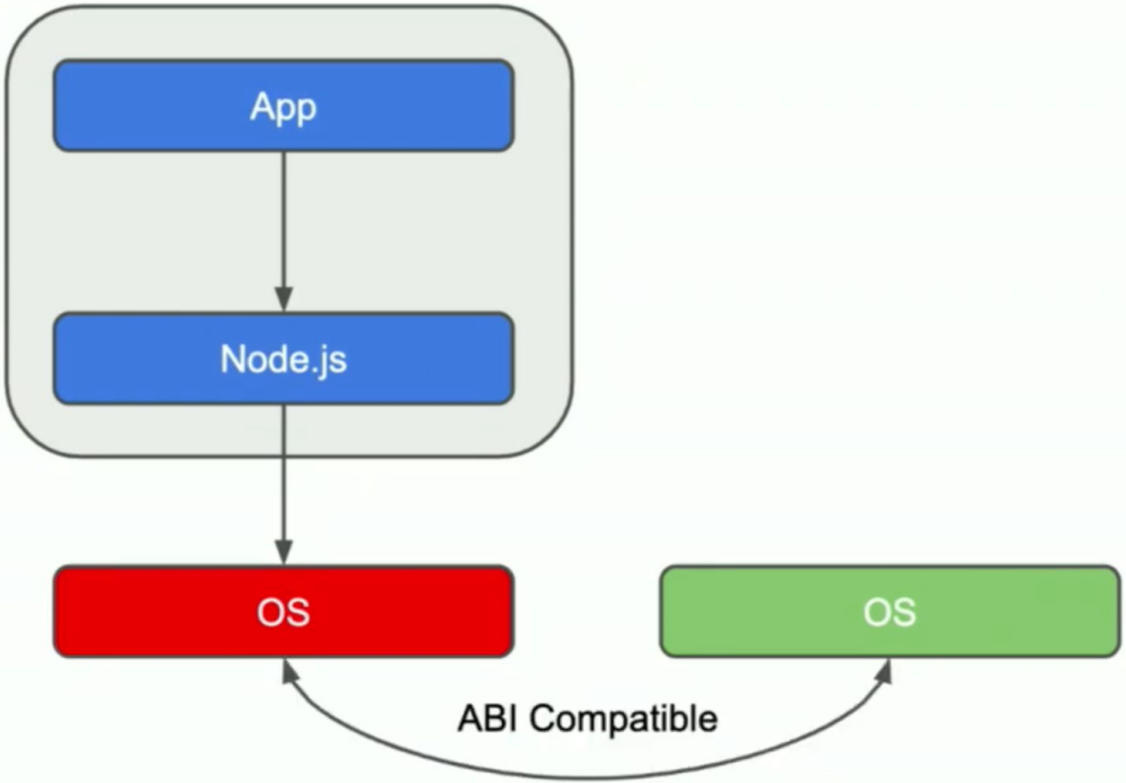
Buildpack: cloudfoundry/node

FROM ubuntu:trusty

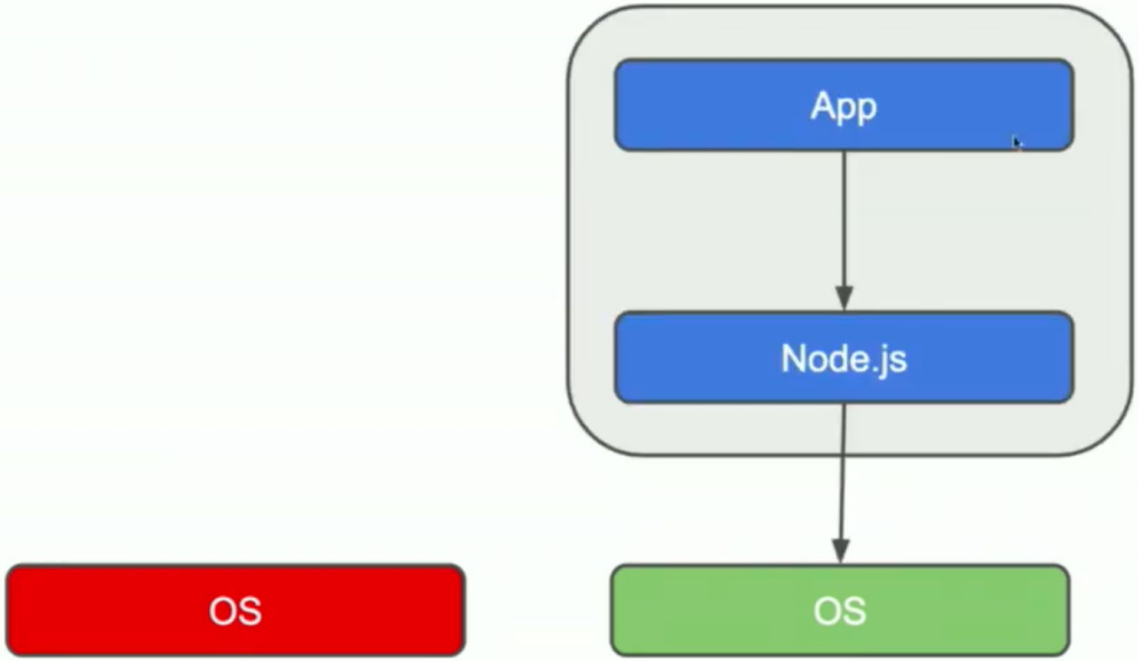
Rebase



Rebase

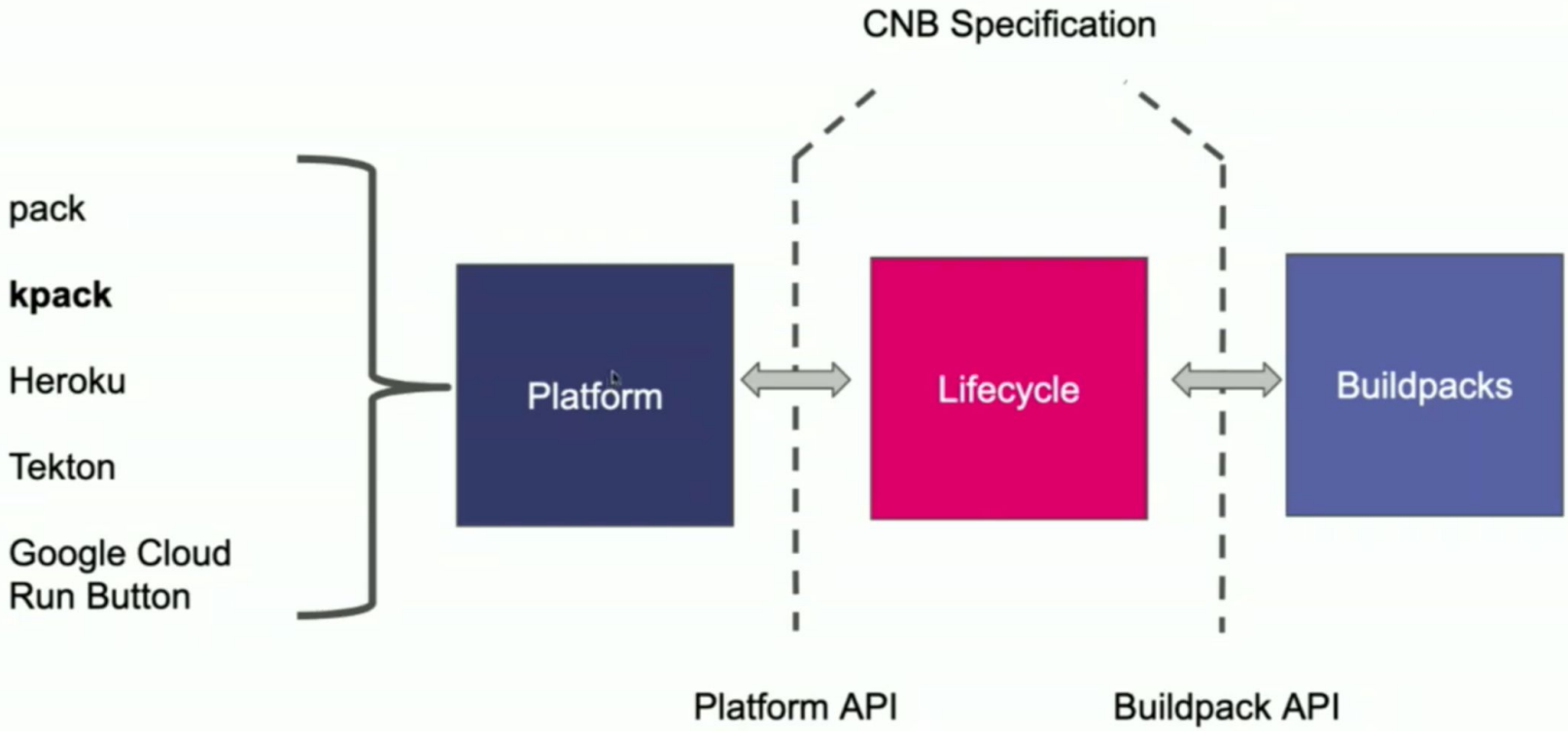


Rebase

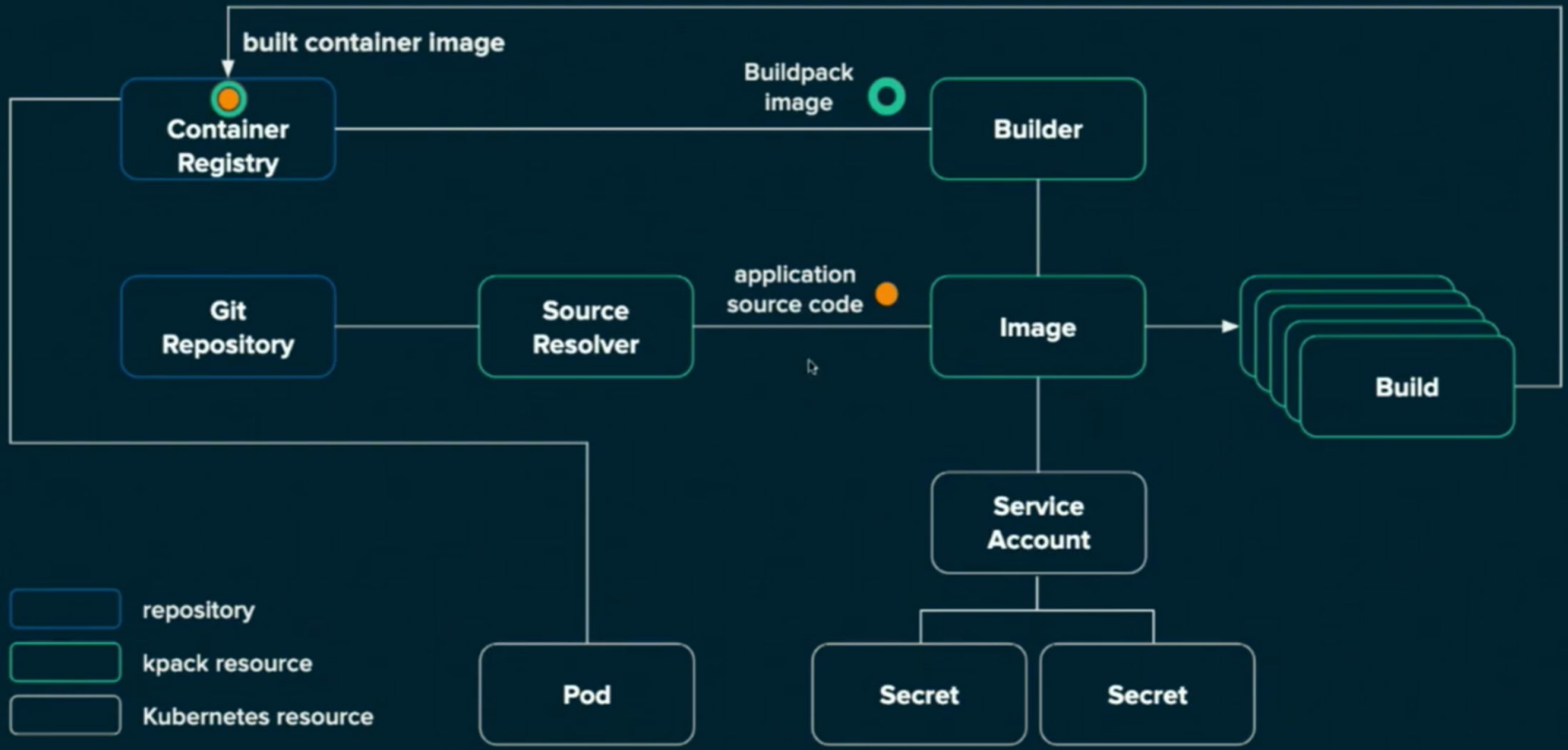


Buildpacks at Scale

CNB at Scale

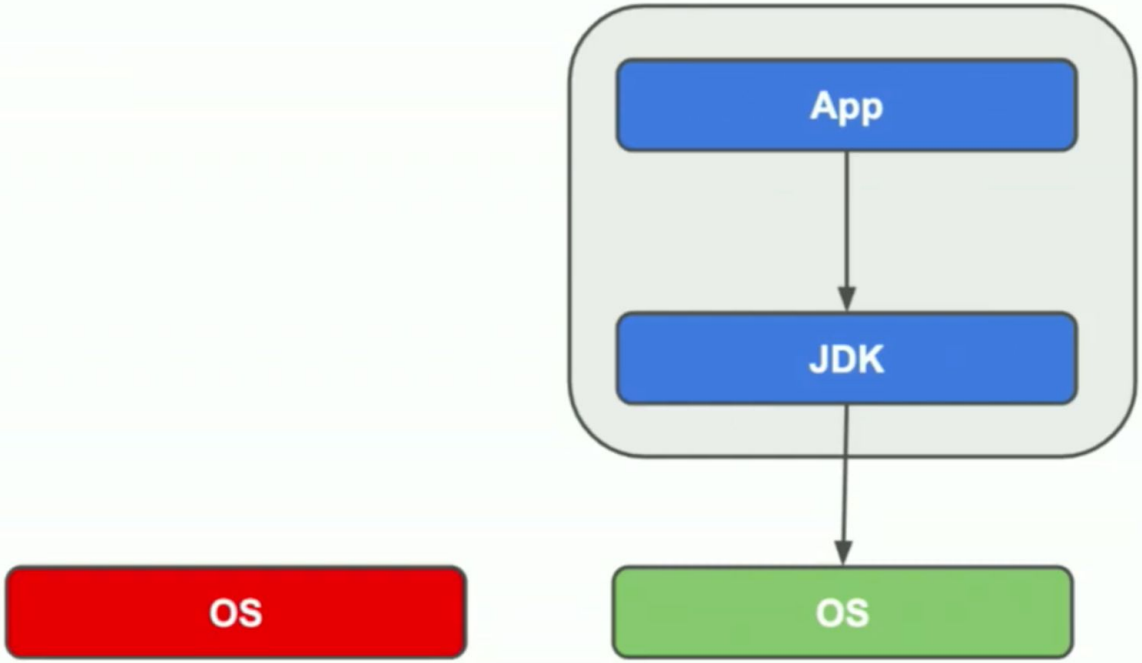

CNB Specification
pack
kpack
Heroku
Tekton
Google Cloud Run Button
Platform
Lifecycle
Buildpacks
Platform API
Buildpack API

kpack


built container image
Container Registry
Buildpack image
Builder
Git Repository
Source Resolver
application source code
Image
Build
Service Account
repository
kpack resource
Kubernetes resource
Pod
Secret
Secret

Let’s rebase multiple images

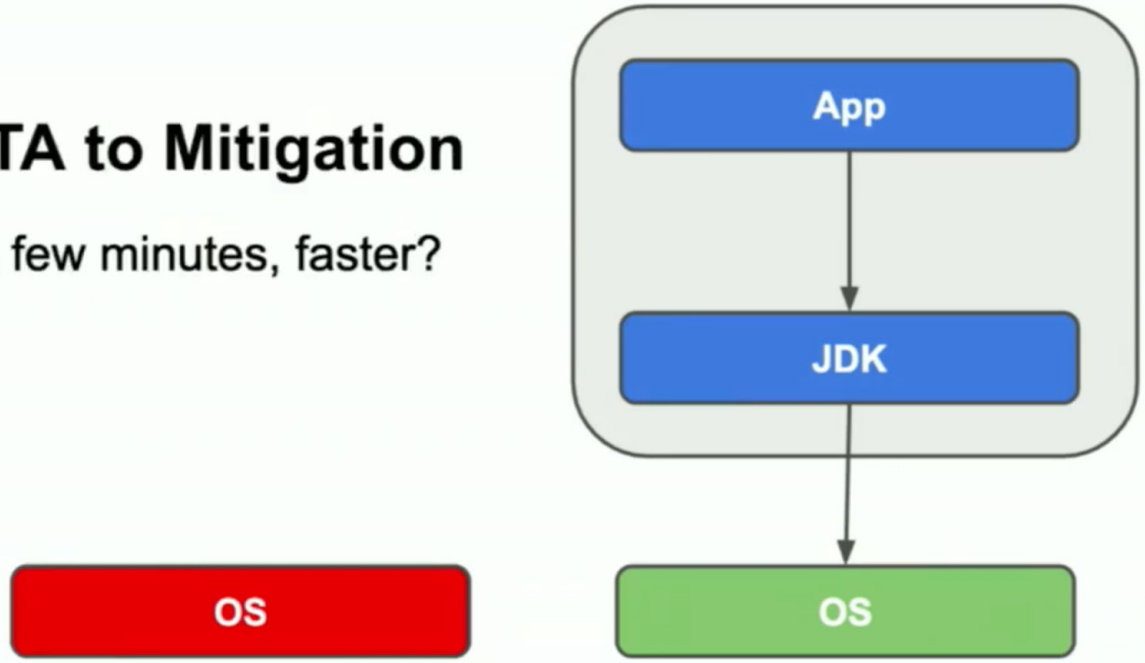
Rebase with kpack



Rebase with kpack

ETA to Mitigation

A few minutes, faster?



Cloud Native Buildpacks are...

Reusable	Fast	Modular	Safe
use the same buildpack on many apps	only re-builds and uploads layers when necessary	combine buildpacks to create composite images	apps meet security requirements w/o developer intervention

Try it out!

Buildpacks.io