Welcome Back To Dependency Hell!

OSS Compliance in the Age of Software Reuse

Nisha Kumar
Open Source Engineer @ VMware
Open Source Summit North America 2019
Who Am I and What Am I selling

- Maintainer for Tern (github.com/vmware/tern) - a container image inspection tool for Open Source Software compliance
- 4 Years Experience in DevOps for embedded systems
- 3 Years in Manufacturing RF modules for cellphones (different kind of pipeline)
- I advocate for Software Distribution Best Practices
Quick Poll
Why should you care about Open Source Software Compliance?
Why should you care about Open Source Software Compliance?

Because you are probably shipping software you did not write.
How do I know?
How do I know?

Because it is the norm
How do I know?

Let me explain...
What is a dependency?
"hello world" in C

#include <stdio.h> // this program depends on this code existing

int main()
{
    printf("Hello, World!\n");
    return 0;
}

$ gcc -o hello_world.o hello_world.c

$ ./hello_world.o

Hello, World!
What is a dependency?
"hello world" in C

```bash
$ ldd hello_world.o

linux-vdso.so.1 (0x00007ffce6ca4000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fc341acc000)
/lib64/ld-linux-x86-64.so.2 (0x00007fc3420bf000)
```

gcc has dynamically linked your dependencies and it’s dependencies

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Project</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>linux-vdso</td>
<td>Linux Kernel</td>
<td>GPL-2.0-only</td>
</tr>
<tr>
<td>libc</td>
<td>GNU C Library</td>
<td>LGPL (various)</td>
</tr>
<tr>
<td>ld-linux-x86-64</td>
<td>GNU C Dynamic Linker/Loader</td>
<td>LGPL (various)</td>
</tr>
</tbody>
</table>
What is a dependency?

"hello world" in C

```
$ gcc -static -o hello_world.o hello_world.c
$ ldd hello_world_static.o
    not a dynamic executable
```

gcc has combined your dependencies' bytecode with yours

```
hello_world_dynamic.o 8.2K
hello_world_static.o 825K
```
Why would anyone statically compile?

Portability!

Only targets that have all the dependencies are supported

```
$ docker run -it debian:buster <- similar target

root@d2e126a1d43d:/# ./hello_world_dynamic.o
Hello, World!
```
Why would anyone statically compile?

Portability!

Combining all the dependencies together allows you to “build once run anywhere”

$ docker run -it alpine:3.7 <- unsupported target

/ # ./hello_world_dynamic.o
/bin/sh: ./hello_world_dynamic.o: not found

/ # ./hello_world_static.o
Hello, World!
The trouble with binary blobs

- You cannot tell which bits belong to what project
- You cannot tell where the blobs came from
- You cannot tell whether you can trust them or not
- You can compare them with something else that is known and guess
- You can look at everything else that came with it and guess

License Compliance is only one area where this becomes a problem. Security and Build Repeatability are also affected.
What do dependencies look like nowadays?

“hello world” in Golang

package main

import "fmt" // a dependency from the standard library

func main() {
    fmt.Println("hello world")
}

$ go build hello_world.go

$ ./hello_world

hello world

go build will produce a binary blob containing all the dependencies. This blob is 2M in size.
What do dependencies look like nowadays?

“hello world” in Golang

```
$ ldd `which go`
```

```
linux-vdso.so.1 (0x00007ffc10bf9000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fc182328000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fc181f37000)
/lib64/ld-linux-x86-64.so.2 (0x00007fc182547000)
```

Part of these dependencies are from the go toolchain itself and others are from somewhere else. It’s hard to tell what bits came from where.

Go is heavily used in developing components for cloud native environments, on top of which folks develop cloud native applications.
What do dependencies look like nowadays?

“hello world” in Node.js

```javascript
console.log("hello world");
```

You need the node binary to run any node program. This depends on 8 shared libraries (as best as we can tell).

You need the standard library. Node has 55 built-in modules compiled into bytecode that also need to be included with the app.

npm is part of the binary installation. It has 808 modules it is dependent on.
What do dependencies look like nowadays?

“hello world” in Node.js

Luckily, the core belongs to 2 projects which have their licenses reasonably well documented. Node and npm provide the source code to all their releases.

Unluckily, creating an npm module is not hard, resulting in an explosion of modules dependent on other modules.

You may not even know you are using these modules.

You’ll often hear this referred to as ”transitive dependencies”.
For decades, discussion of software reuse was far more common than actual software reuse. Today, the situation is reversed: developers reuse software written by others every day, in the form of software dependencies, and the situation goes mostly unexamined.

- Russ Cox

https://research.swtch.com/deps
Now let’s talk about containers
Unit of Deployed Software

- C
- Python
- Golang
- Node
- Containerized app

Number of Dependencies
A container image is, in effect, a package with an app and all of its dependencies.

*ALL* the dependencies.

All the parts of the operating system the app needs.

Everything except the kernel.
These dependencies are not well defined.

They are arbitrarily installed and are frozen in time.

It is standard practice to run random install scripts in what is basically a chroot jail.

It is standard practice to reuse existing container images.
Example of container dependencies: golang:1.11

```
[nisha@localhost tern]$ docker history docker.io/golang

<table>
<thead>
<tr>
<th>IMAGE</th>
<th>CREATED</th>
<th>CREATED BY</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>fb7a47d8605b</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) WORKDIR /go</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c mkdir -p &quot;$GOPATH/src&quot; &quot;$GOPATH&quot;...</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) ENV PATH=/go/bin:/usr/...</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) ENV GOPATH=/go</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c set -eux; dpkgArch=&quot;$dpkg --...</td>
<td>341 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) ENV GOLANG_VERSION=1.11</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c apt-get update &amp;&amp; apt-get insta...</td>
<td>162 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c apt-get update &amp;&amp; apt-get insta...</td>
<td>142 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c set -ex; if ! command -v gpg &gt;...</td>
<td>7.8 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c apt-get update &amp;&amp; apt-get insta...</td>
<td>23.2 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c #(nop) CMD [&quot;bash&quot;]</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c #(nop) ADD file:58d5c21fcabcf...</td>
<td>101 MB</td>
</tr>
</tbody>
</table>
```

Example of container dependencies: golang:1.11

<table>
<thead>
<tr>
<th>IMAGE</th>
<th>CREATED</th>
<th>CREATED BY</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>fb7a47d8605b</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) WORKDIR /go</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c mkdir -p &quot;$GOPATH/src&quot; &quot;$GOPATH...</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) ENV PATH=/go/bin:/usr/...</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) ENV GOPATH=/go</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c set -eux; dpkgArch=&quot;$(dpkg --...</td>
<td>341 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) ENV GOLANG_VERSION=1.11</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c apt-get update &amp; apt-get i...</td>
<td>162 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c apt-get update &amp; apt-get i...</td>
<td>142 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c apt-get update &amp; apt-get i...</td>
<td>7.8 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c apt-get update &amp; apt-get i...</td>
<td>23.2 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c #(nop) CMD [&quot;bash&quot;]</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c #(nop) ADD file:58d5c21fcabcf1e...</td>
<td>101 MB</td>
</tr>
</tbody>
</table>

Example of container dependencies: golang:1.11

[nisha@localhost tern]$ docker history docker.io/golang

<table>
<thead>
<tr>
<th>IMAGE</th>
<th>CREATED</th>
<th>CREATED BY</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>fb7a47d8605b</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) WORKDIR /go</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c mkdir -p &quot;$GOPATH/src&quot; &quot;$GOPATH...</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) ENV PATH=/go/bin:/usr/l...</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) ENV GOPATH=/go</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c set -eux; dpkgArch=&quot;$(dpkg --...</td>
<td>341 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) ENV GOLANG_VERSION=1.11</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c apt-get update &amp;&amp; apt-get insta...</td>
<td>162 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c apt-get update &amp;&amp; apt-get insta...</td>
<td>142 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c set -ex; if ! command -v gpg &gt;...</td>
<td>7.8 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c apt-get update &amp;&amp; apt-get insta...</td>
<td>23.2 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c #(nop) CMD [&quot;bash&quot;]</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c #(nop) ADD file:58d5c21fcabcf1e...</td>
<td>101 MB</td>
</tr>
</tbody>
</table>

https://github.com/docker-library/buildpack-deps/blob/1845b3f918f69b4c97912b0d4d68a5658458f/stretch/scm/Dockerfile
Example of container dependencies: golang:1.11

```
[nisha@localhost tern]$ docker history docker.io/golang

<table>
<thead>
<tr>
<th>IMAGE</th>
<th>CREATED</th>
<th>CREATED BY</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>fb7a47d8605b</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) WORKDIR /go</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c mkdir -p &quot;$GOPATH/src&quot; &quot;$GOPATH...</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) ENV PATH=/go/bin:/usr/l...</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) ENV GOPATH=/go</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c set -eu; dpkgArch=&quot;$(dpkg --...</td>
<td>341 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) ENV GOLANG_VERSION=1.11</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>4 weeks ago</td>
<td>/bin/sh -c apt-get update &amp; &amp; apt-get insta...</td>
<td>162 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c apt-get update &amp; &amp; apt-get insta...</td>
<td>142 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c set -ex; if ! command -v gpg &gt;...</td>
<td>7.8 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c apt-get update &amp; &amp; apt-get insta...</td>
<td>23.2 MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c #(nop) CMD [&quot;bash&quot;]</td>
<td>0 B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>5 weeks ago</td>
<td>/bin/sh -c #(nop) ADD file:58d5c21fcabcf1e...</td>
<td>101 MB</td>
</tr>
</tbody>
</table>
```

https://github.com/docker-library/buildpack-deps/blob/1845b3f918f69b4c97912b0d4d68a5658458e84f/stretch/scm/Dockerfile
https://github.com/docker-library/buildpack-deps/blob/1845b3f918f69b4c97912b0d4d68a5658458e84f/stretch/curl/Dockerfile
### Example of container dependencies: golang:1.11

<table>
<thead>
<tr>
<th>IMAGE</th>
<th>CREATED</th>
<th>docker.io/golang Created By</th>
<th>SIZE</th>
<th>docker history</th>
</tr>
</thead>
<tbody>
<tr>
<td>fb7a47d8605b</td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) WORKDIR /go</td>
<td>0 B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 weeks ago</td>
<td>/bin/sh -c mkdir -p &quot;$GOPATH/src&quot; &quot;$GOPATH...</td>
<td>0 B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) ENV PATH=/go/bin:/usr/l...</td>
<td>0 B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) ENV GOPATH=/go</td>
<td>0 B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 weeks ago</td>
<td>/bin/sh -c set -eux; dpkgArch=&quot;$(dpkg --...</td>
<td>341 MB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 weeks ago</td>
<td>/bin/sh -c #(nop) ENV GOLANG_VERSION=1.11</td>
<td>0 B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 weeks ago</td>
<td>/bin/sh -c apt-get update &amp; &amp; apt-get insta...</td>
<td>162 MB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 weeks ago</td>
<td>/bin/sh -c apt-get update &amp; &amp; apt-get insta...</td>
<td>142 MB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 weeks ago</td>
<td>/bin/sh -c set -e: if ! command -v gpg &gt;...</td>
<td>7.8 MB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 weeks ago</td>
<td>/bin/sh -c apt-get update &amp; &amp; apt-get insta...</td>
<td>23.2 MB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 weeks ago</td>
<td>/bin/sh -c #(nop) CMD [&quot;bash&quot;]</td>
<td>0 B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 weeks ago</td>
<td>/bin/sh -c #(nop) ADD file:58d5c21fcabcfle...</td>
<td>101 MB</td>
<td></td>
</tr>
</tbody>
</table>

- **golang:1.11**
- **buildpack-deps:stretch-scm**
- **buildpack-deps:stretch-curl**
- **debian:stretch**

 GitHub URLs:
- [golang:1.11 Dockerfile](https://github.com/docker-library/golang/blob/ed78459fac108dab72556146b759516cc65ee109/1.11/stretch/Dockerfile)
- [buildpack-deps:stretch-scm Dockerfile](https://github.com/docker-library/buildpack-deps/blob/1845b3f918f69b4c97912b0d4d68a5658458e84f/stretch/scm/Dockerfile)
- [buildpack-deps:stretch-curl Dockerfile](https://github.com/docker-library/buildpack-deps/blob/1845b3f918f69b4c97912b0d4d68a5658458e84f/stretch/curl/Dockerfile)
- [debian:stretch Dockerfile](https://github.com/debuerreotype/docker-debian-artifacts/blob/ed15c6a0b511d2985ca252f59f4318b1fe2a7a59/stretch/Dockerfile)
Example of container dependencies: golang:1.11

[nisha@localhost ~]$ mkdir image
[nisha@localhost ~]$ time docker save docker.io/golang | tar xC image/.
real  0m19.899s
user  0m0.326s
sys   0m1.788s

[nisha@localhost ~]$ cd image/
[nisha@localhost image]$ ls -al

```
total 52
drwxrwxr-x.  9 nisha nisha 4096 0ct 12 17:49 .
drwx--------. 25 nisha nisha 4096 0ct 12 17:48 ..
drwxr-xr-x.  2 nisha nisha 4096 Sep 5 04:06 1eff74c9194e6e74bea2cfffdec00eaf66367daff22f97b4cf4d89012aada4cfb2
...```
Example of container dependencies: golang:1.11

[nisha@localhost image]$ ls 2fd07b43bcbcbcf7f629b6ea299040133321899b8c0b9fb8eb02621bd4c6f35/contents/etc/
All of it needed to build a hello_world app
What does an “app” and it’s dependencies look like in the cloud?

Microservices

https://blog.twitter.com/engineering/en_us/a/2013/observability-at-twitter.html
https://www.slideshare.net/adriancockcroft/dockercon-state-of-the-art-in-microservices
https://medium.com/@LachlanEvenson/practicing-microservices-ac43c8b3b712

Twitter

Netflix

Amazon
We’re standing on the shoulders of giants
So how do we keep track of our dependencies?
So how do we keep track of our dependencies?

Package Managers!
So how do we keep track of our dependencies?

Package Managers!

Except not all package managers do this well...
# Things that package managers should do

<table>
<thead>
<tr>
<th>Toolchain</th>
<th>Package Manifest</th>
<th>Dependency Resolution</th>
<th>Snapshots/Lockfiles</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpkg</td>
<td>descriptive</td>
<td>exists</td>
<td>not really</td>
<td>yes</td>
</tr>
<tr>
<td>bundler</td>
<td>missing license</td>
<td>exists</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>npm</td>
<td>descriptive</td>
<td>exists</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>go</td>
<td>nope</td>
<td>not needed</td>
<td>maybe</td>
<td>use git</td>
</tr>
<tr>
<td>docker</td>
<td>nope</td>
<td>not needed</td>
<td>nope</td>
<td>not needed</td>
</tr>
</tbody>
</table>
Containers don’t manage anything

FROM <image>

RUN <script>

COPY ..

Diff files:
install app dependencies and extra modifications

Diff Files:
Copy app into container

Diff Files: etc/ca-certificates/* usu/share/ca-certificates/*
OS & language package manager dependencies

BaseOS: bin boot etc home lib opt root tmp usr var
Debian, Photon, Alpine

Completely arbitrary build machinery

Intended image with unknown content
Container images are basically random executables downloaded off the internet.
But it gets worse
There are a lot of questionable standards and practices advocated by thought leaders
Some standard practices for building container images

Pull from the latest and greatest.
No, we won’t tell you what changed between the last latest and greatest you pulled and this one.

Just trust us!
Some standard practices for building container images

You can start off with other container images.

Go ahead and use this one!

What’s in it? Everything you’ll ever need, baby!
Some standard practices for building container images

Run what you like in this container. Do what ever it takes to get your thing to work.

No, we won’t guarantee that it’ll work again.

But you’re a 10x engineer! You’ll figure it out!
Some standard practices for building container images

You can absolutely use a container as a build stage for your app.

Just pull from latest...
Some standard practices for building container images

Go ahead and use Docker’s multistage builds. It’s the best way to reduce the size of that hideous container image.

What’s that? Your build image gets deleted? That’s totally OK! Containers are supposed to be thrown away!
What are the implications of not having a package manager for microservices?
You’re the package manager
Build Repeatability, License Compliance and Security Updates are your problem.
What do we do?
Look at OS Suppliers
Look at OS Suppliers

Their build system usually have defined and descriptive inputs

Their build system is usually repeatable

You can usually get corresponding sources

They always closely follow upstream projects
Use language package managers with caution
Use language package managers with caution

Ruby, Python and Rust are not bad. Node.js is hard. Golang is harder.

Use as much of the machinery as you can and roll your own glue code for the rest. Some examples for glue code: mirror upstream, build from source, label artifacts and byproducts, document document document document.

If you’re using a container as a build stage, don’t throw it away. Record the image digest with the rest of the build documentation. Make sure you use a reliable container OS supplier.
Use minimized container images like you would a pot of boiling water
Stripped down containers

Container images with just the app’s runtime dependencies are a good idea. Just make sure they also come from a well documented, reproducible build and release pipeline and are traceable to source.

Docker’s multistage build is non-reproducible. Avoid it if you can.

Keep and document all of your intermediate containers. You will need to go back to them later.
Get involved in the Open Source Community
You’re part of the community

It’s a global community

You’re heavily dependent on them

Not your OSPO
YOU!!!
You’re part of the community

Don’t wait for the community to come up with something. Get involved.

Help them implement best practices. Tell them why they are needed. Submit PRs. Help them with CVEs.

They won’t mind.
You’re part of the community now

Be nice!!!

Give creators credit for their work.

Follow their wishes on how they would like their code to be used and distributed.
Think about what “best practices” looks like in this new era of Software Reuse.

Develop software against these best practices.
Resources that can help

Linux Foundation ACT projects: FOSSology, QMSTR, SPDX Tools, Tern

Scancode is a tool that scans files for licenses

Tern finds licenses for container images

reproducible-builds.org does good work on defining build and release best practices

SPDX is standardized language to communicate license information

OCI accepts proposals to make container images and distribution better

https://github.com/nexB/scancode-toolkit
https://github.com/vmware/tern
https://reproducible-builds.org/docs/
https://spdx.org/specifications
https://github.com/opencontainers
Thank You!

Tern: http://github.com/vmware/tern
github: @nishakm
twitter: @nishakmr, @vmwopensource
website: https://nishakm.github.io

Come see me at VMware’s Expo booth #31