Introduction to Reproducible Computational Environments

./sdc

Haroon Chughtai | 2019-07-19

In this session, we'll discuss...

- Why is reproducibility useful to you?
- What is a computational environment?
- How to capture environments?
  - Package Management
  - Virtual Machines
  - Containers

Why is reproducibility useful to you?

What does reproducible mean?

		Data	
		Same	Different
Analysis	Same	Reproducible	Replicable
	Different	Robust	Generalisable

https://dx.doi.org/10.6084/m9.figshare.7140050

As a researcher....

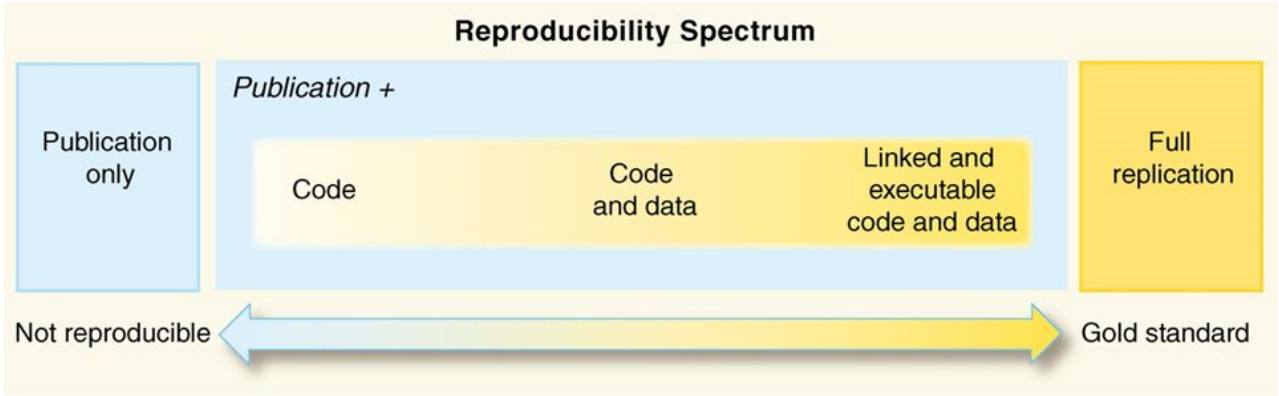
The science is the code, and reproducibility of your analysis is the aim.

Code and data need the computational environment to be executed.

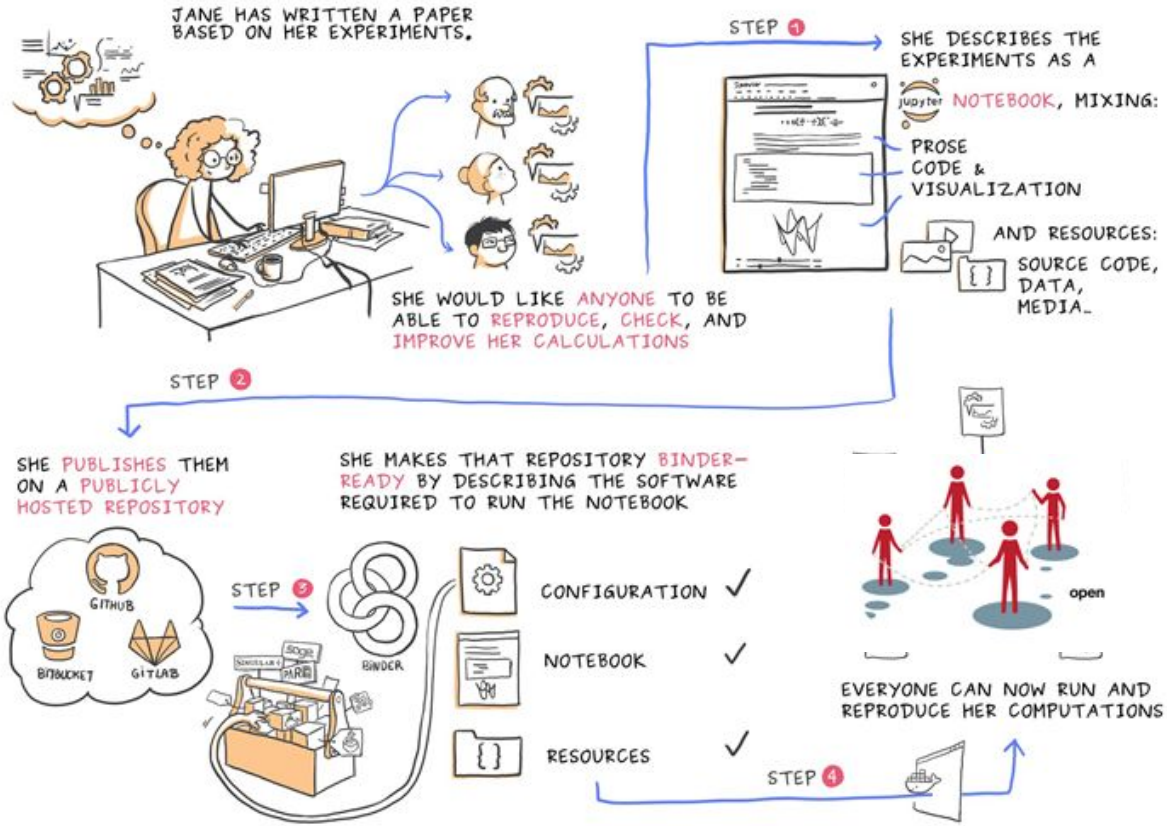
As a developer....

You want to be able to operationalise and scale up your software easily.

Code should run reproducibly on dev, test and live machines' environments.



Peng, Science, 2011

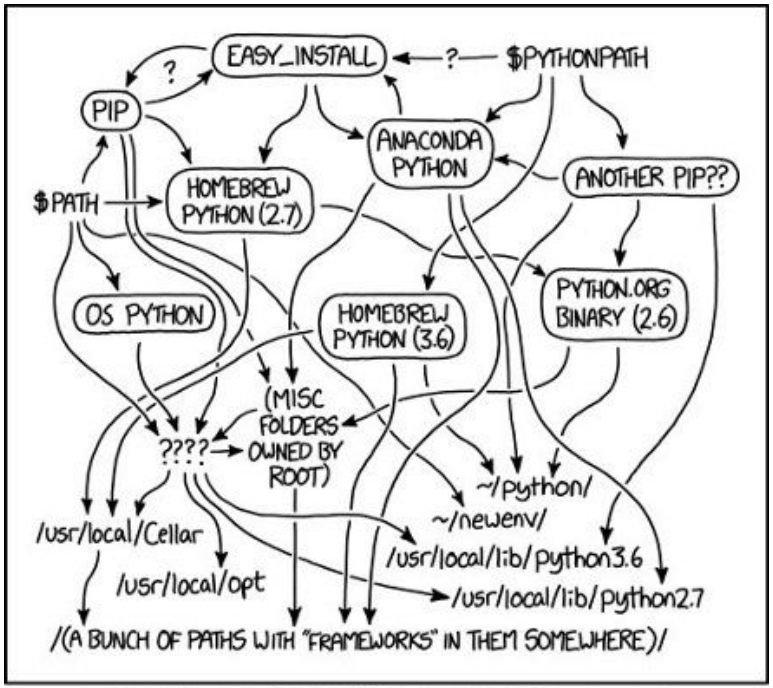


Courtesy of Juliette Taka:
https://twitter.com/mybinderteam/status/1082556317842264064

What is a computational environment?

What is a computational environment?

- Hardware
  - CPU, GPU
- Operating Systems
- Software
  - Language version
  - Package versions
  - Dependencies



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

How to capture environments?

Package Management

Package managers install and keep track of the different software packages (and their versions) that you use within an environment.

- Tools such as: pip, conda, virtual environments
- Without management, you must choose:
    - Using the older version of package C forever and not benefiting from updates and bugfixes in later versions.
    - Installing the updated version of the package and hoping that it doesn't impact Project One.
    - Installing the updated version of the package for use in Project Two then uninstalling it and reinstalling the old one whenever they need to do work on Project One.

Project One

Package name	Version
Package A	1.5.2
Package B	2.1.10
Package C	0.7.9

Project Two

Package name	Version
Package B	2.1.10
Package C	1.2.4
Package D	1.5.2
Package E	3.7.1

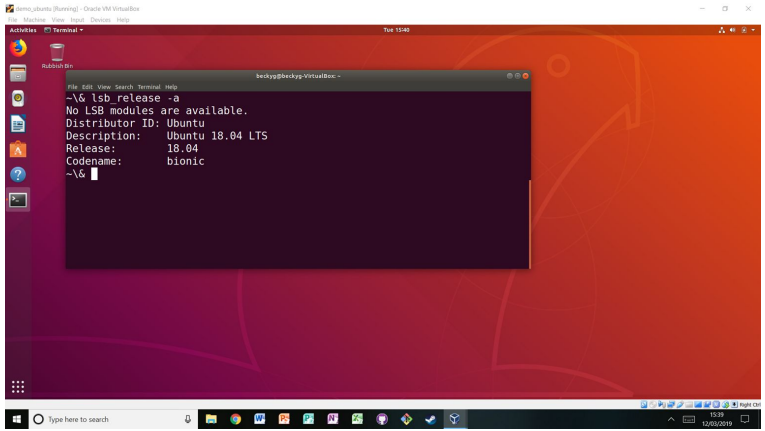
Virtual Machines

Virtual machines (VMs) essentially package a whole computer as an app that can be run

Example: Even moderately complex projects, the size of the software dependency stack can be huge.

To run a simple pipeline to build a pdf report for an analysis scripted in R using Rmarkdown:

1. The respective R packages need to be installed
2. The R version needs to be the same, but also
3. The versions of pandoc and LaTeX need to be exactly the same as during runtime.



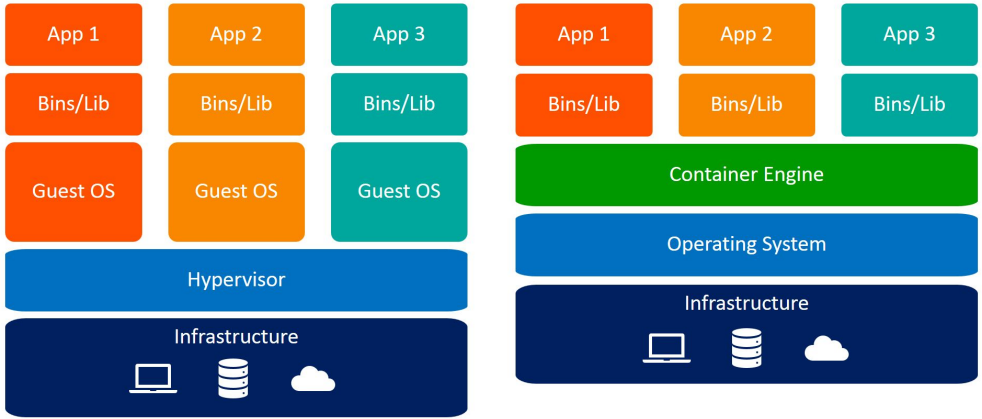
Containers

Containers allow developer to package up a project with all of the parts it needs, such as libraries, dependencies, and system settings and ship it all out as one package.

Anyone can then open up a container and work within it, viewing and interacting with the project as if the machine they are accessing it from is identical to the machine specified in the container - regardless of what their computational environment actually is.

Containers

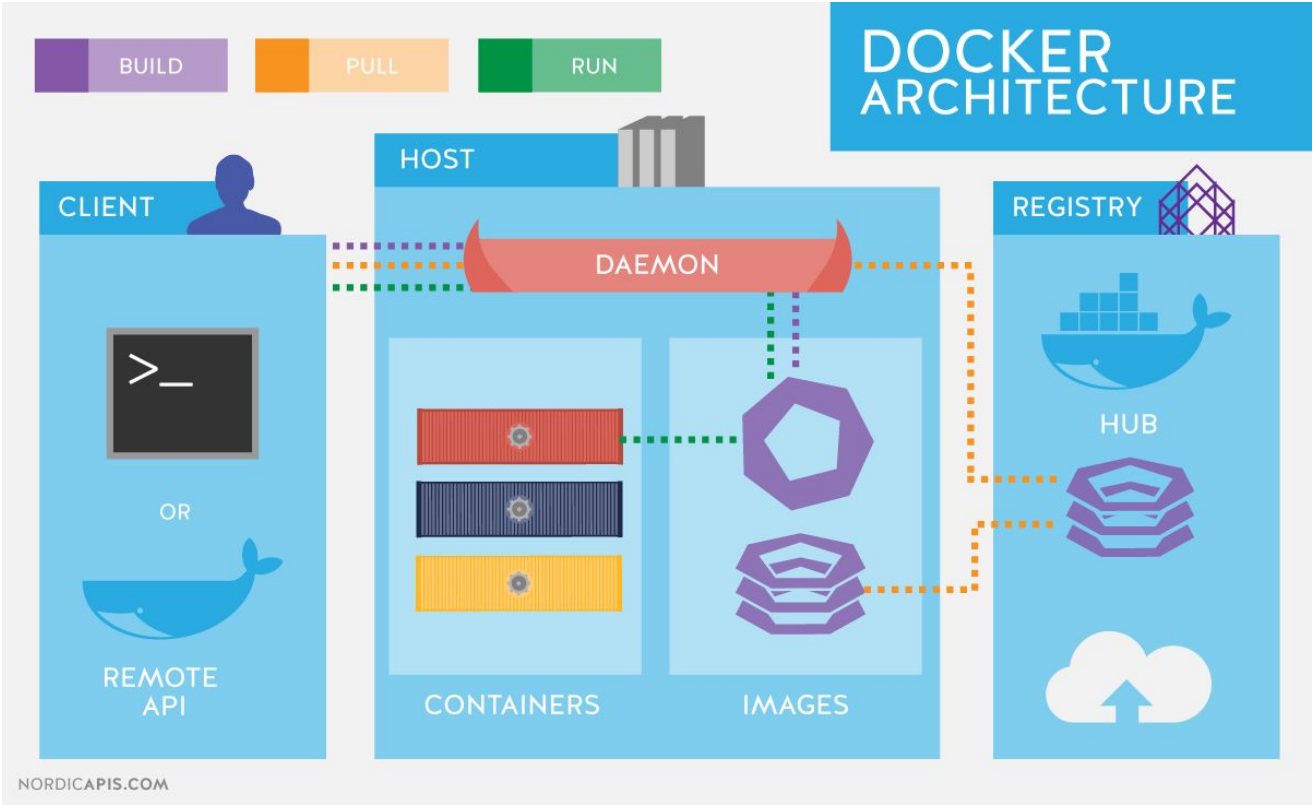
Unlike a virtual machine, rather than creating a whole virtual operating system plus all the software and tools typically packaged with one, containers only contain the individual components they need in order to operate the project they contain.



Virtual Machines

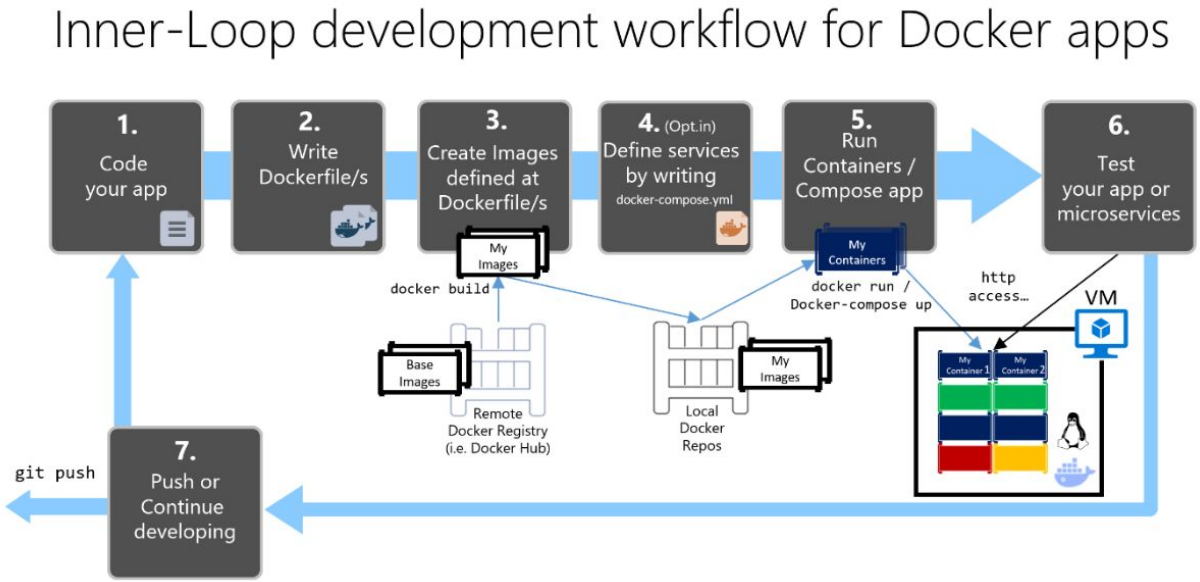
Containers

Docker 101


BUILD
PULL
RUN
DOCKER ARCHITECTURE
HOST
CLIENT
DAEMON
REGISTRY
>_
OR
REMOTE API
HUB
CONTAINERS
IMAGES
NORDICAPIS.COM

Docker 101

Images are the files used to generate containers. Humans don't make images, they write the recipes to generate images. Containers are then identical copies instantiated from images.



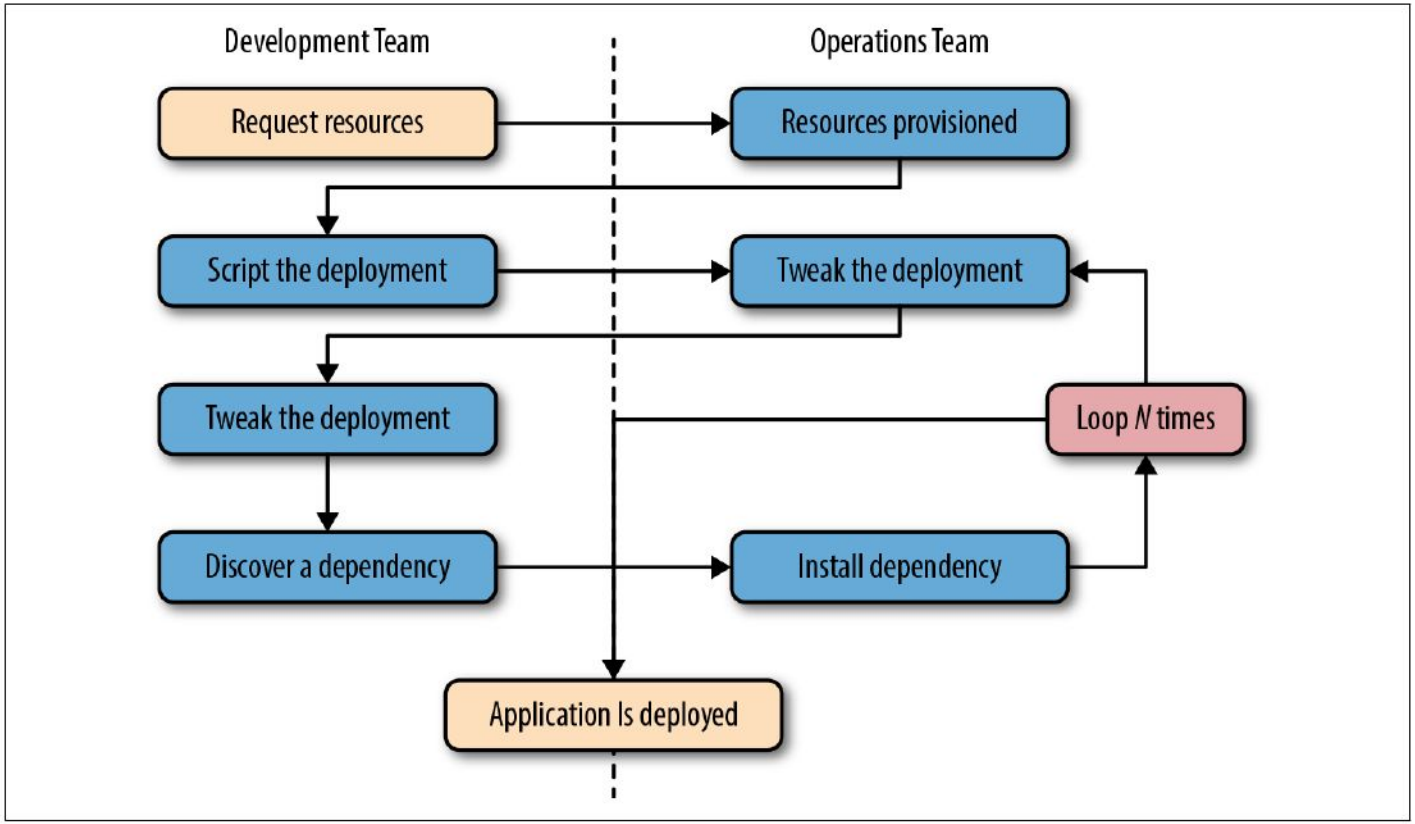


Figure 2-1. A traditional deployment workflow (without Docker)

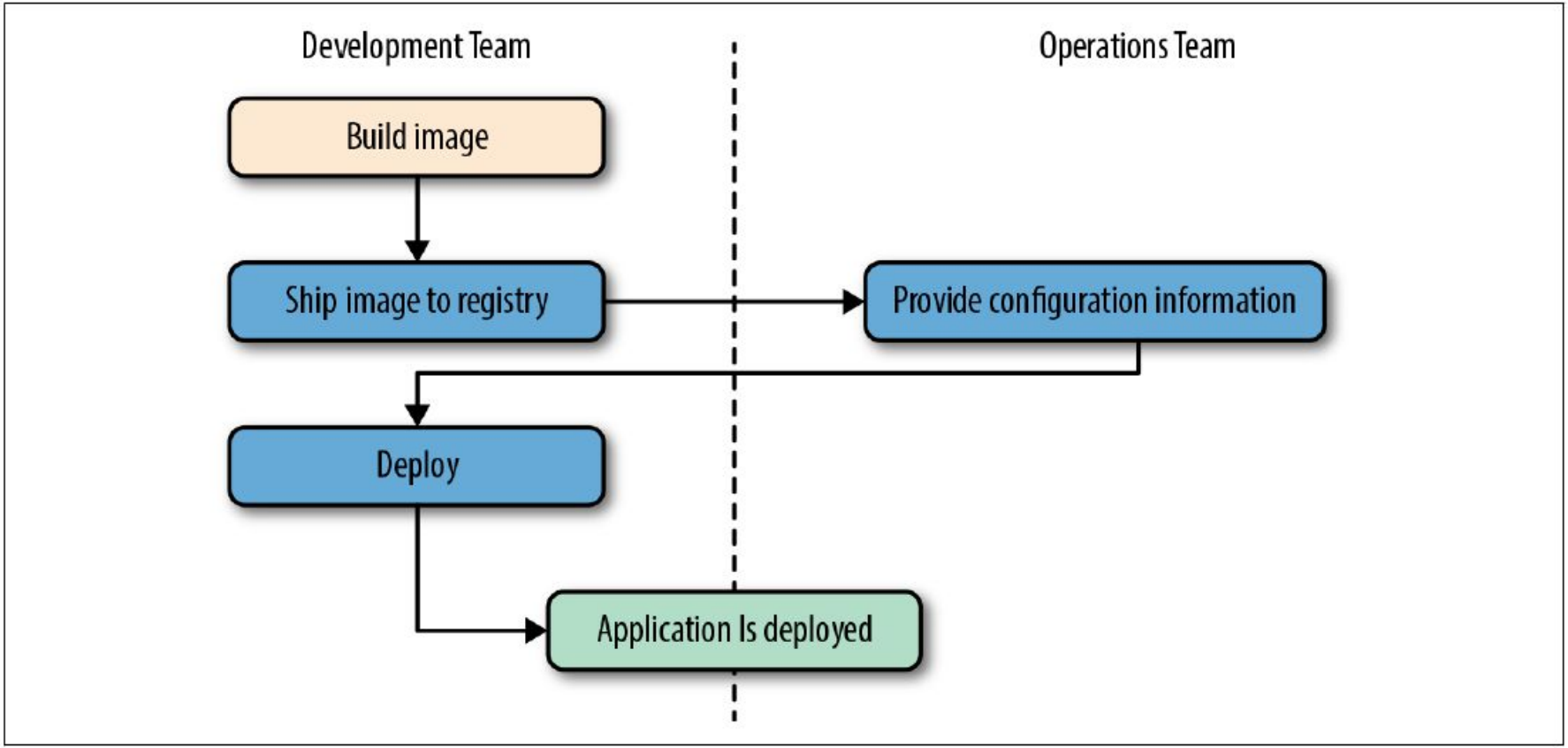


Figure 2-2. A Docker deployment workflow

DEMO

$ docker pull alpine

$ docker run alpine echo "hello from alpine"

$ docker run -it alpine /bin/sh

$ docker ps -a

$ docker run --name static-site -e AUTHOR="Your Name" -d -P dockersamples/static-site

$ docker stop static-site

$ docker rm static-site

Sample dockerfile

```
# our base image
FROM alpine:3.5

# Install python and pip
RUN apk add --update py2-pip

# install Python modules needed by the Python app
COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt

# copy files required for the app to run
COPY app.py /usr/src/app/
COPY templates/index.html /usr/src/app/templates/

# tell the port number the container should expose
EXPOSE 5000

# run the application
CMD ["python", "/usr/src/app/app.py"]
```

Any Questions?

Resource List

What this talk was based on
https://the-turing-way.netlify.com/reproducible_environments/reproducible_environments.html

Greg Wilson's blog - Co-Founder of SW Carpentry
http://third-bit.com/

Docker for beginners (for distributed apps)
https://docker-curriculum.com/

Docker for reproducible research
https://arxiv.org/pdf/1410.0846.pdf