Simplify your Jenkins Projects with Docker Multi-Stage Builds

Eric Smalling - Solution Architect, Docker Inc.

Jenkins World

A global DevOps event

2017

Simplify your Jenkins Projects with Docker Multi-Stage Builds

Eric Smalling
Solution Architect
Docker Inc.

Agenda

- Docker images 101
- Building images via Jenkins
- Image size challenges
- Old way: Using Docker image builder pattern
- New way: Docker multi-stage builds
- Demos
- Resources for more info

Introduction

- Eric Smalling
  - Solution Architect
    Docker Customer Success Team
- ~25 years in software development, architecture, version control admin, etc…
  - Java / J2EE, C, Python, etc
  - Puppet, Ansible
  - Git, SVN, CVS, ClearCase, VSS, PVCS
- ~10 years in build & test automation
  - Primarily Jenkins for CI, including some very large scale implementations
  - Testing with Selenium, Fitnesse, RESTAssured, SOAPUI, Puppet-RSpec, etc
  - Docker user since pre-1.0 days
- Dallas/Fort Worth Jenkins Area Meetup (JAM) coordinator.

Docker images 101

- Images are built up from a series of layers
- Each layer represents changes from the prior
- Layers are immutable (read-only) and referenced by hashes and optional tags
- Containers run with a file system based on an image with a thin read/write layer added.
  (plus any volume mounts specified)

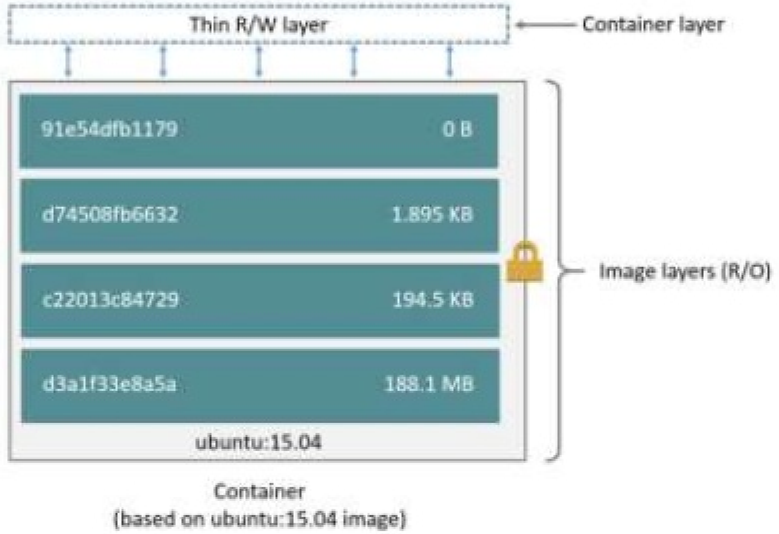


Image credit: https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers

Docker images 101

- Multiple containers running on the same image share the underlying layers.
- Each container overlays it's own container read/write layer.

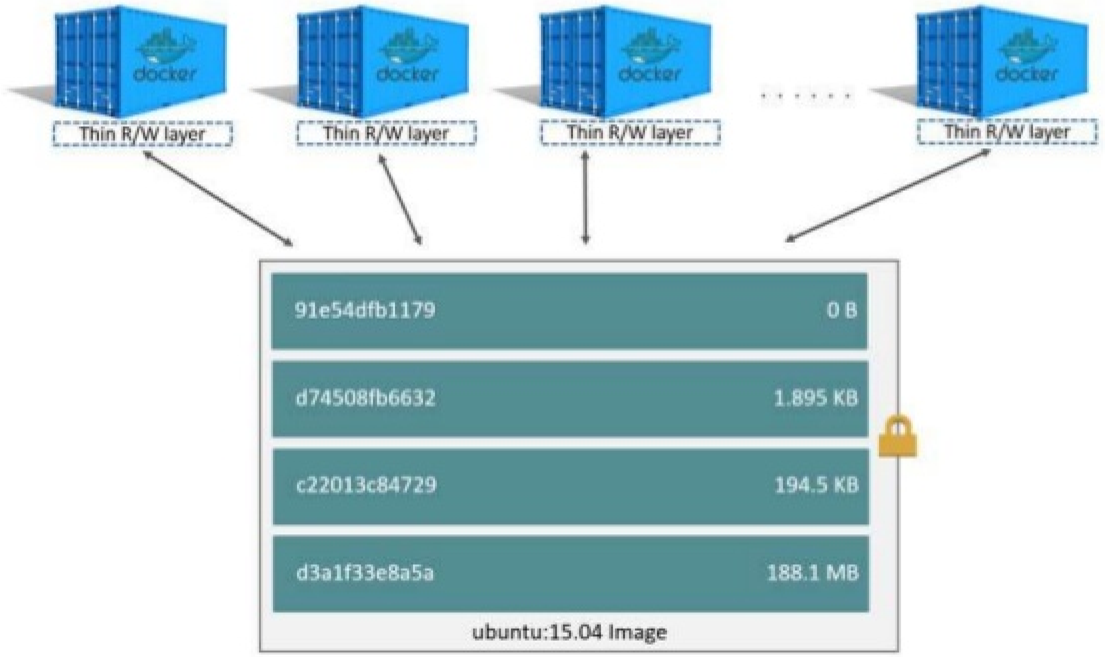
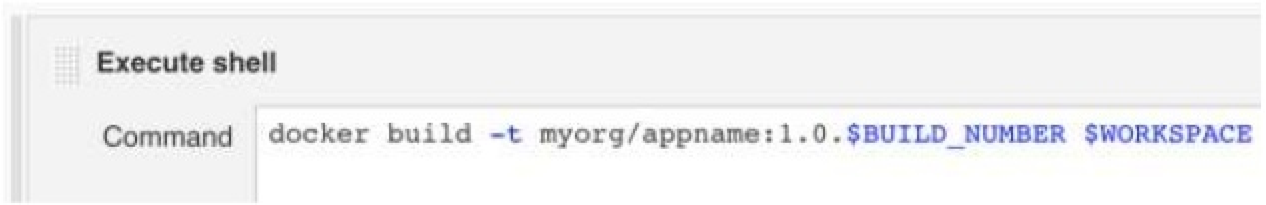


Image credit: https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers

Building images via Jenkins

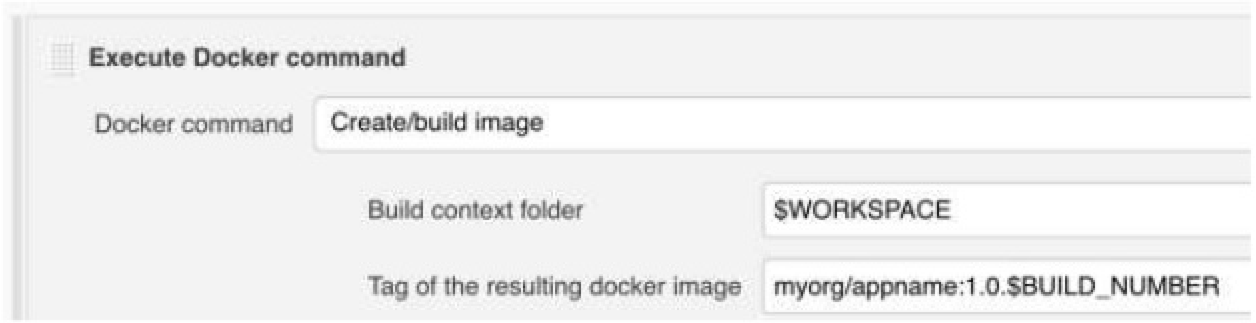
- 3 usual methods:
  - Shell step
  - Docker build step *(plugin)*
  - Pipeline DSL



Execute shell

Command

```
docker build -t myorg/appname:1.0.$BUILD_NUMBER $WORKSPACE
```

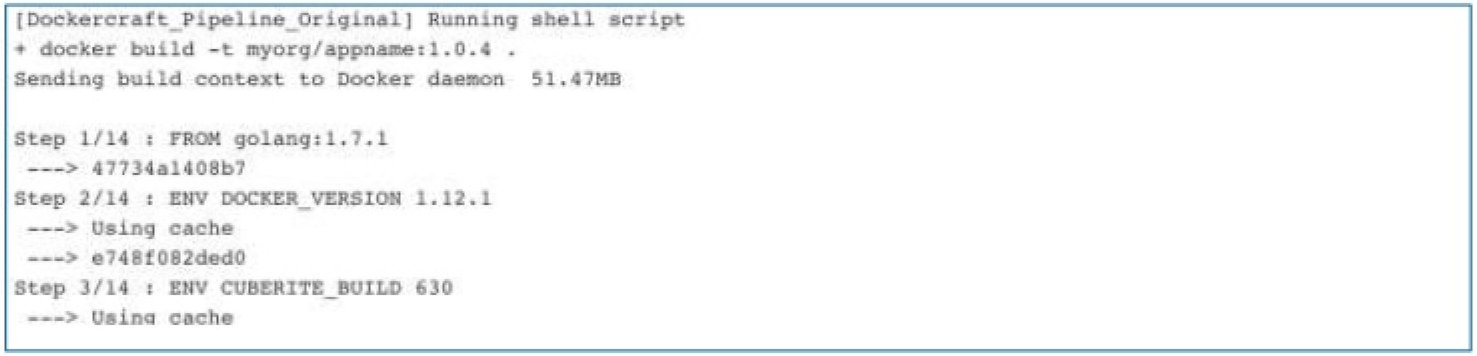


Execute Docker command

Docker command	Create/build image
Build context folder	$WORKSPACE
Tag of the resulting docker image	myorg/appname:1.0.$BUILD_NUMBER

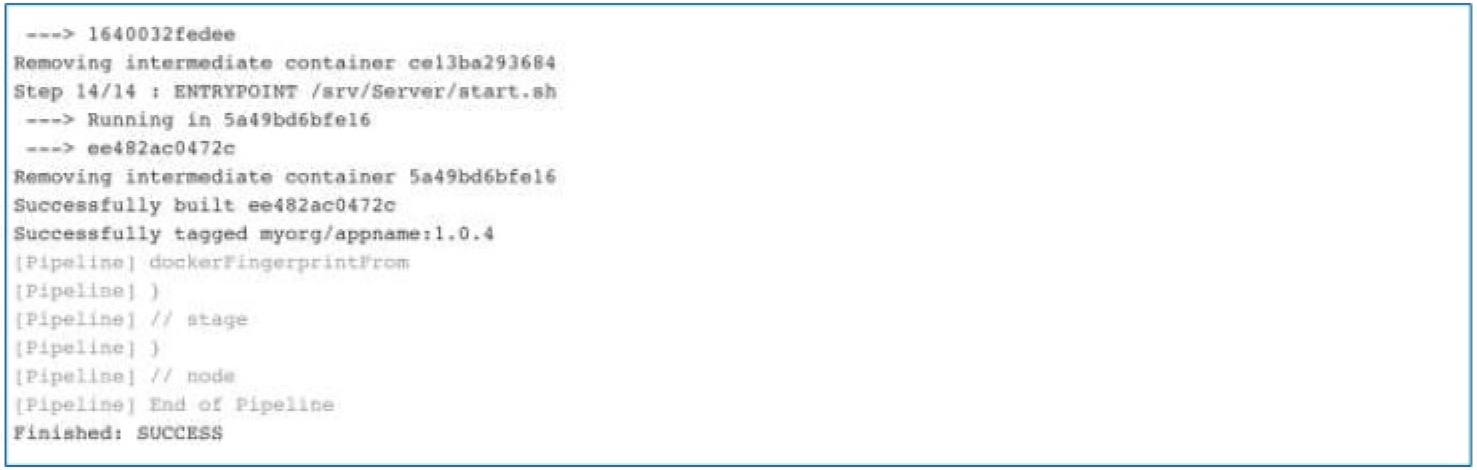
```
stage ('Build Image') {
      app = docker.build("myorg/appname:1.0.$BUILD_NUMBER")
}
```

Building images via Jenkins



```
[Dockercraft_Pipeline_Original] Running shell script
+ docker build -t myorg/appname:1.0.4 .
Sending build context to Docker daemon  51.47MB

Step 1/14 : FROM golang:1.7.1
 ---> 47734a1408b7
Step 2/14 : ENV DOCKER_VERSION 1.12.1
 ---> Using cache
 ---> e748f082ded0
Step 3/14 : ENV CUBERITE_BUILD 630
 ---> Using cache
```

...



```
 ---> 1640032fedee
Removing intermediate container ce13ba293684
Step 14/14 : ENTRYPOINT /srv/Server/start.sh
 ---> Running in 5a49bd6bfe16
 ---> ee482ac0472c
Removing intermediate container 5a49bd6bfe16
Successfully built ee482ac0472c
Successfully tagged myorg/appname:1.0.4
[Pipeline] dockerFingerprintFrom
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Image size challenges

- Image layers have size and they add up!
- CI can compound this
- Example: dockercraft
  - Base golang:1.7.1 = 672MB
  - Docker 17.06 CE = 98MB
  - Cuberite = 12MB
  - Misc = 56 MB
  - Total: 838MB!

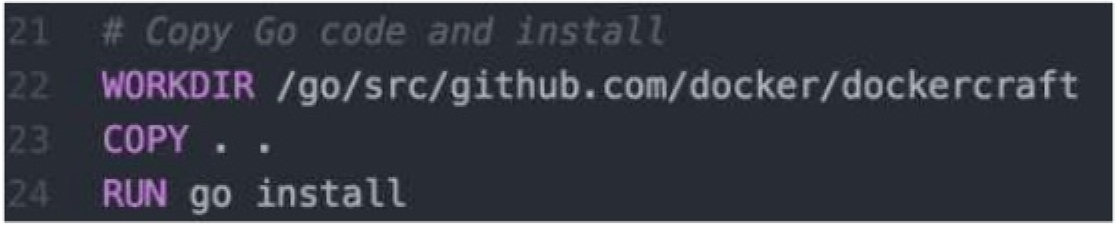
Image size challenges

- Image layers have size and they add up!
- CI can compound this
- Example: dockercraft
  - Base ~~golang:1.7.1 = 672MB~~ debian:jessie = 123MB
  - Docker 17.06 CE = 98MB
  - Cuberite = 12MB
  - Misc = 56 MB
  - Total: ~~838MB!~~ 289MB

Changing the base all by itself saves us 549MB!

Image size challenges

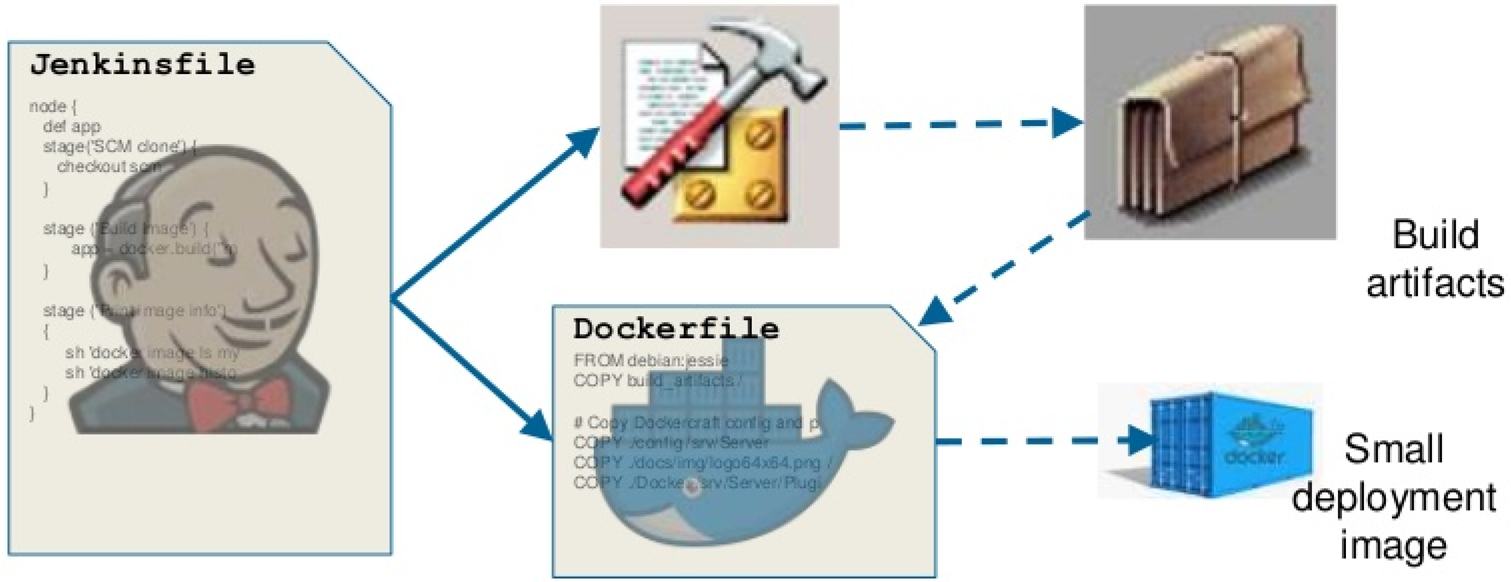
How can we get rid of GoLang if it's needed?



```
# Copy Go code and install
WORKDIR /go/src/github.com/docker/dockercraft
COPY . .
RUN go install
```

Build in Jenkins-land, then Dockerize

- Use Jenkins build steps to do the construction in the workspace followed by a Docker image build with just the needed artifacts

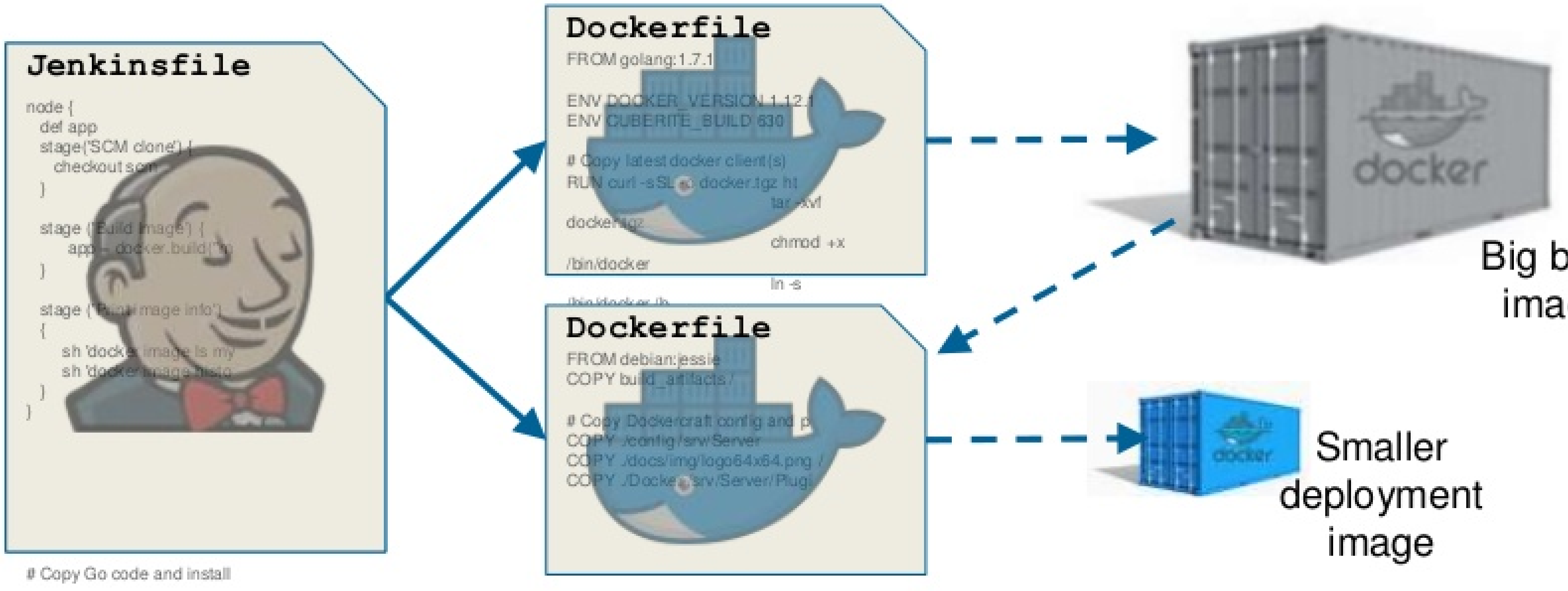


```
node {
  def app
  stage('SCM clone') {
    checkout scm
  }

  stage ('Build image') {
      app = docker.build("
  }

  stage ('Print image info')
  {
     sh 'docker image ls my
     sh 'docker image histo
  }
}
```

```
FROM debian:jessie
COPY build_artifacts /

# Copy Dockercraft config and p
COPY ./config /srv/Server
COPY ./docs/img/logo64x64.png /
COPY ./Docker /srv/Server/Plugi
```

Docker build pattern

- Using Docker image builder pattern, we break the Dockerfile up into multiple files, the final one being the one that produces the image to ship.



Jenkinsfile

```
node {
  def app
  stage('SCM clone') {
    checkout scm
  }

  stage ('Build image') {
      app = docker.build("
  }

  stage ('Print image info')
  {
     sh 'docker image ls my
     sh 'docker image histo
  }
}
```

```
# Copy Go code and install
```

Dockerfile

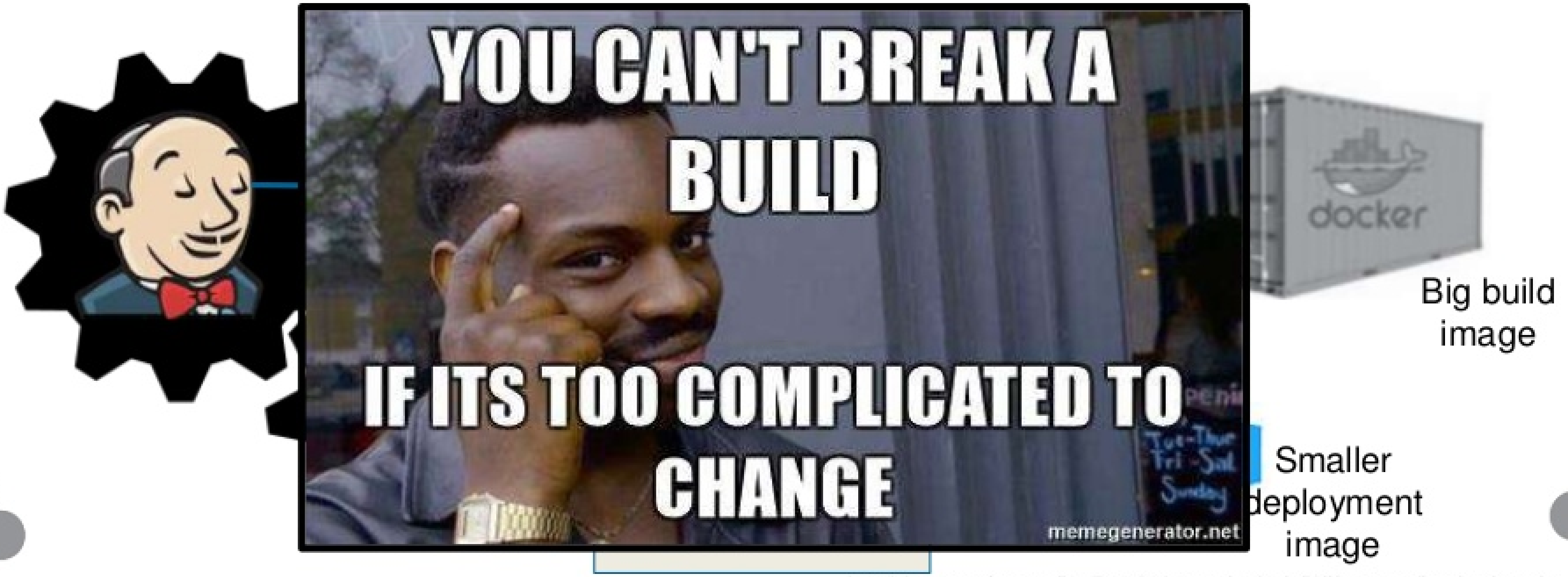
```
FROM golang:1.7.1

ENV DOCKER_VERSION 1.12.1
ENV CUBERITE_BUILD 630

# Copy latest docker client(s)
RUN curl -sSL -o docker.tgz ht
                                  tar -xvf
docker.tgz
                                  chmod +x
/bin/docker
                                  ln -s
/bin/docker /b
```

Dockerfile

```
FROM debian:jessie
COPY build_artifacts /

# Copy Dockercraft config and p
COPY ./config /srv/Server
COPY ./docs/img/logo64x64.png /
COPY ./Docker /srv/Server/Plugi
```

Old School Way

- Implementing some fashion of the builder pattern via build steps or multiple projects triggering each other.



https://cdn.meme.am/instances/500x/76014521/you-cant-break-a-build-if-its-too-complicated-to-change.jpg

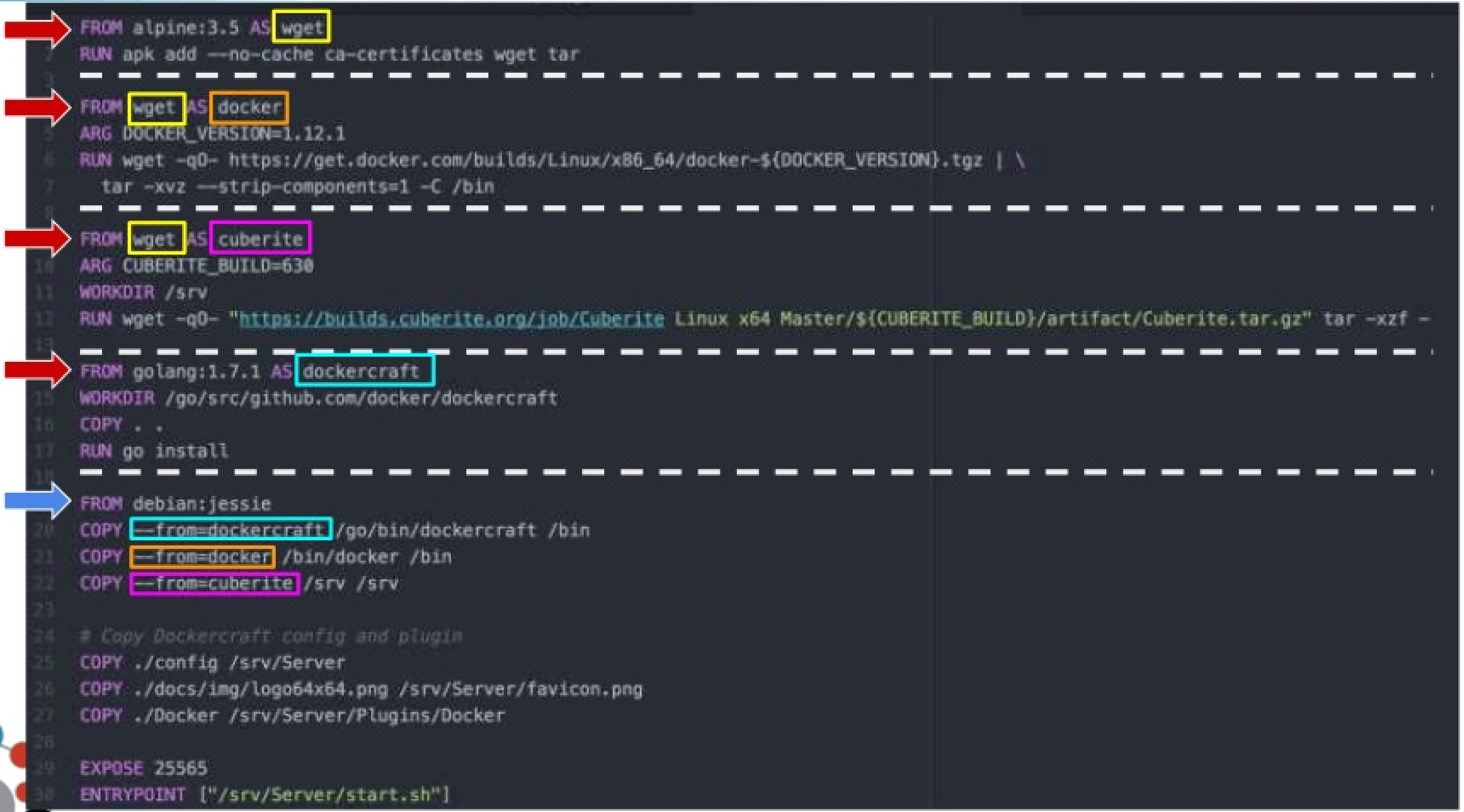
Docker image builds demo (old way)

Demo time!

Docker Multi-Stage builds

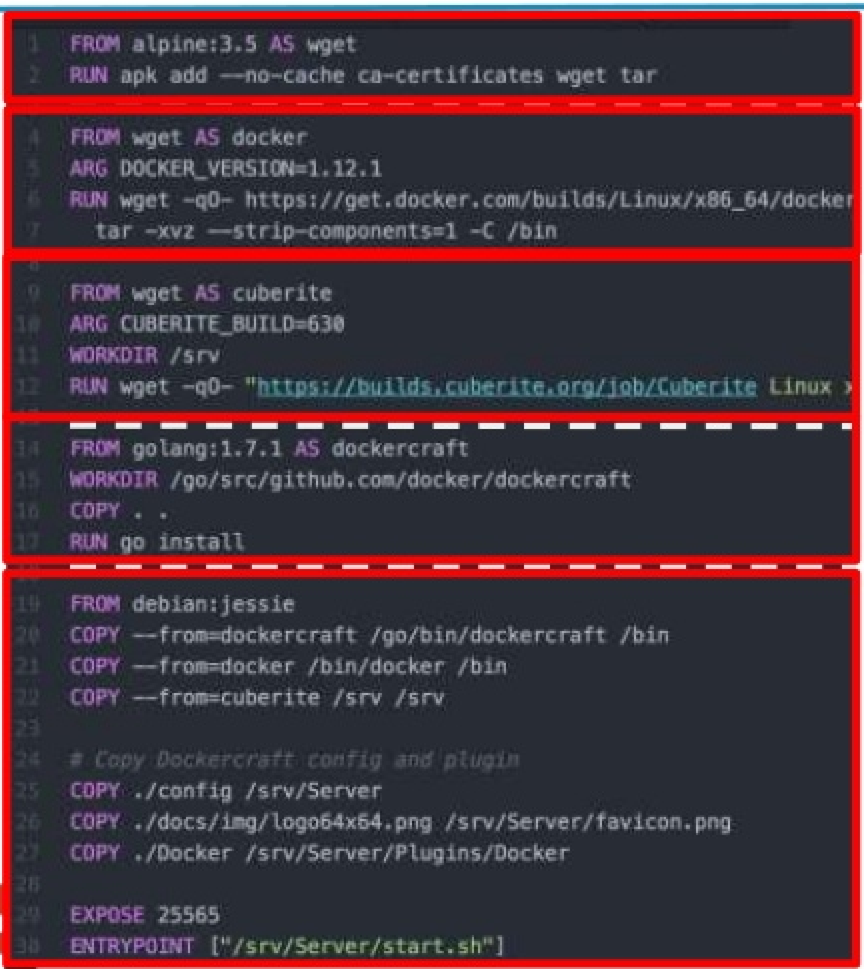
- Multiple stages in one Dockerfile
- Each stage starts with a new “FROM” line
- Layers from final stage are the only ones in your image
- Stages can refer back to and copy files from earlier stages
- Requires Docker CE 17.05+ / EE 17.06+

Example Dockerfile with multi-stage:

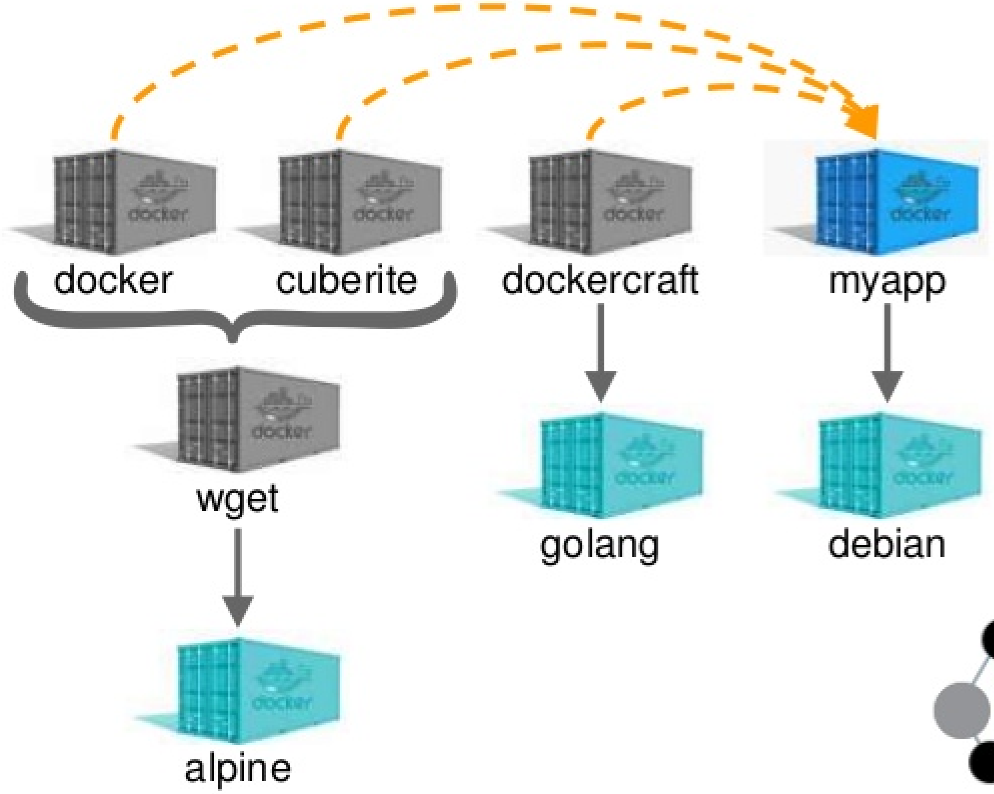


```dockerfile
FROM alpine:3.5 AS wget
RUN apk add --no-cache ca-certificates wget tar

FROM wget AS docker
ARG DOCKER_VERSION=1.12.1
RUN wget -qO- https://get.docker.com/builds/Linux/x86_64/docker-${DOCKER_VERSION}.tgz | \
  tar -xvz --strip-components=1 -C /bin

FROM wget AS cuberite
ARG CUBERITE_BUILD=630
WORKDIR /srv
RUN wget -qO- "https://builds.cuberite.org/job/Cuberite Linux x64 Master/${CUBERITE_BUILD}/artifact/Cuberite.tar.gz" tar -xzf -

FROM golang:1.7.1 AS dockercraft
WORKDIR /go/src/github.com/docker/dockercraft
COPY . .
RUN go install

FROM debian:jessie
COPY --from=dockercraft /go/bin/dockercraft /bin
COPY --from=docker /bin/docker /bin
COPY --from=cuberite /srv /srv

# Copy Dockercraft config and plugin
COPY ./config /srv/Server
COPY ./docs/img/logo64x64.png /srv/Server/favicon.png
COPY ./Docker /srv/Server/Plugins/Docker

EXPOSE 25565
ENTRYPOINT ["/srv/Server/start.sh"]
```

Example Dockerfile with multi-stage:



```
FROM alpine:3.5 AS wget
RUN apk add --no-cache ca-certificates wget tar

FROM wget AS docker
ARG DOCKER_VERSION=1.12.1
RUN wget -qO- https://get.docker.com/builds/Linux/x86_64/docker
  tar -xvz --strip-components=1 -C /bin

FROM wget AS cuberite
ARG CUBERITE_BUILD=630
WORKDIR /srv
RUN wget -qO- "https://builds.cuberite.org/job/Cuberite Linux x

FROM golang:1.7.1 AS dockercraft
WORKDIR /go/src/github.com/docker/dockercraft
COPY . .
RUN go install

FROM debian:jessie
COPY --from=dockercraft /go/bin/dockercraft /bin
COPY --from=docker /bin/docker /bin
COPY --from=cuberite /srv /srv

# Copy Dockercraft config and plugin
COPY ./config /srv/Server
COPY ./docs/img/logo64x64.png /srv/Server/favicon.png
COPY ./Docker /srv/Server/Plugins/Docker

EXPOSE 25565
ENTRYPOINT ["/srv/Server/start.sh"]
```



Minecraft, really?

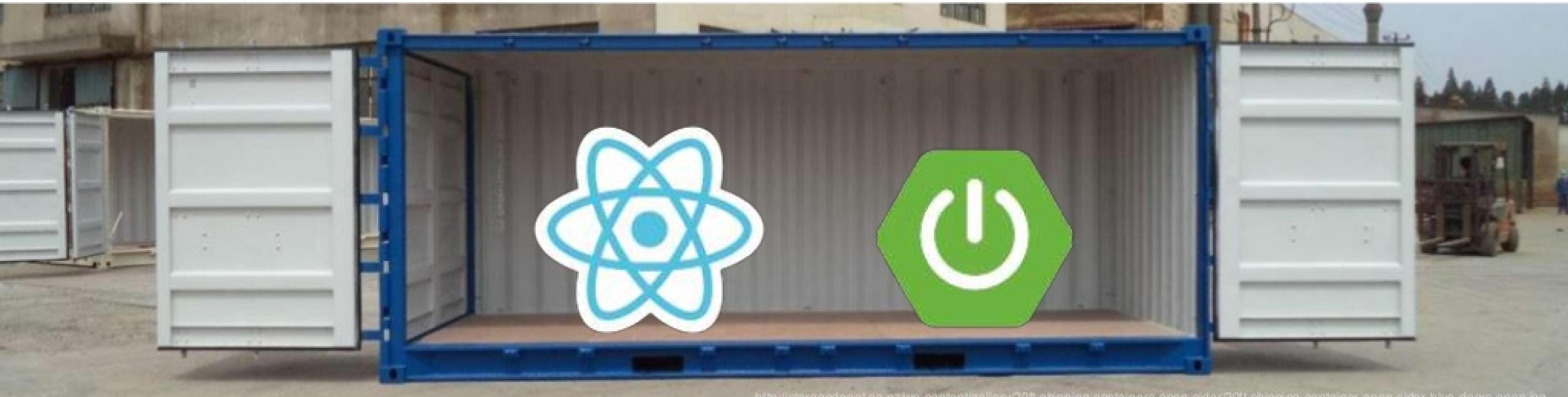
Not everyone works at Unicorn shops that code in GoLang all day and visualize their Docker containers in Minecraft

http://gph.is/1UK1ZFa

AtSeaApp Demo

Simple web app:

- ReactJS front-end
- Spring Boot, Java back-end
- Deployed in single container



AtSeaApp Demo

Demo time!

Resources

- Dockercon 2017 keynote introducing Multi-Stage Builds: https://youtu.be/hwkqju_BXEo?t=24m26s
- Abby Fuller, *Creating Effective Images*: https://youtu.be/pPsREQbf3PA
- Nicolas Frankel, *A Dockerfile for Maven-Based GitHub Projects*: https://goo.gl/hWv3NM

Resources

Jenkins World
A global DevOps event
2017

docker
con 17 EU

16-19 OCTOBER

COPENHAGEN, DENMARK

Q & A

Contact me

Eric Smalling
Solution Architect at Docker
eric.smalling@docker.com

#irc } ericsmalling

16-19 OCTOBER
COPENHAGEN, DENMARK

Jenkins World

A global DevOps event

2017