Multi-Stage Docker Robot Deployments

Levon Avagyan and Xu Han

Multi-Stage Docker Robot Deployments

Levon Avagyan and Xu Han

“Developers these days don’t know how to deploy code, so they just ship their development environment instead”

- commenter on *stack**overflow*** on using docker

- ❏ Keep update sizes as small as possible
- ❏ Minimize the time and effort it takes to make a release
- ❏ Make upgrades/downgrades hermetic, no accidental halfway upgrades

- ❏ Make it easy to have multiple versions and developers on one robot
- ❏ Make it so that developers can build a release locally
- ❏ Make developer, testing, and production environments as similar as possible
- ❏ Make setting up a robot development and testing environment seamless and easy
- ❏ Minimize time spent compiling

Before multi-stage docker deploys

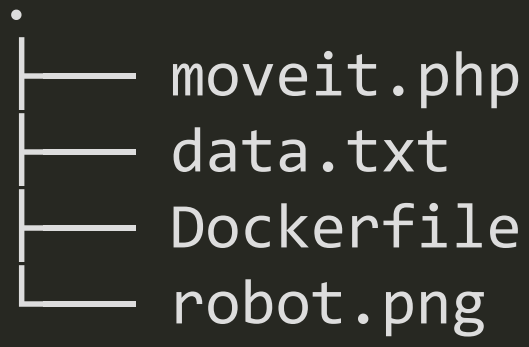
- Form of containerization
- Encapsulates system dependencies, environment, executables, file systems, etc…
- Not a VM, uses underlying kernel and has minimal virtualization overhead (in most cases)
- Dockerfiles - description files for configuring Docker Images
- Docker Compose - description files for starting sets of Docker Containers
- Relies on concept of image layers that can be cached, reused, and transferred

```
# Base Image
FROM ubuntu:18.04

RUN echo “hello world” > hello_world.txt

RUN date > date.txt

RUN cp date.txt date2.txt

RUN rm date.txt

RUN echo “I am done”
```

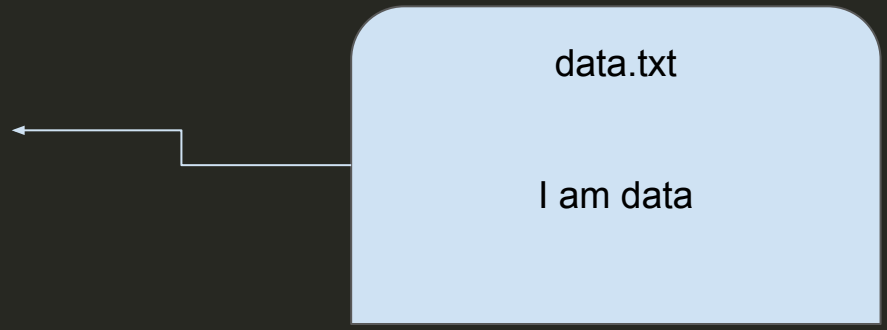
```
# Base Image
FROM ubuntu:18.04 CACHED

RUN echo “hello world” > hello_world.txt CACHED

RUN date > date.txt CACHED

RUN cp date.txt date2.txt CACHED

RUN rm date.txt CACHED

RUN echo “I am done” CACHED
```

```
# Base Image
FROM ubuntu:18.04 CACHED

RUN echo “bye world” > hello_world.txt    CHANGED

RUN date > date.txt BROKEN

RUN cp date.txt date2.txt BROKEN

RUN rm date.txt BROKEN

RUN echo “I am done” BROKEN
```

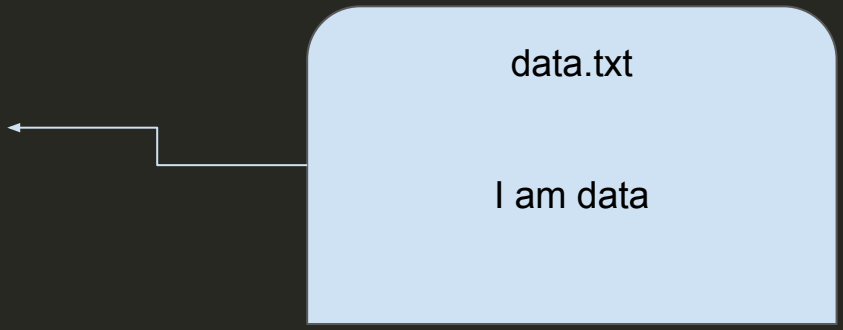
```
# Base Image
FROM ubuntu:18.04 CACHED

RUN echo “bye world” > hello_world.txt    CACHED

RUN date > date.txt CACHED

RUN cp date.txt date3.txt CHANGED

RUN rm date.txt BROKEN

RUN echo “I am done” BROKEN
```


.
├── moveit.php
├── data.txt
├── Dockerfile
└── robot.png

```
# Base Image
FROM ubuntu:18.04

COPY data.txt data.txt

RUN cat data.txt
```



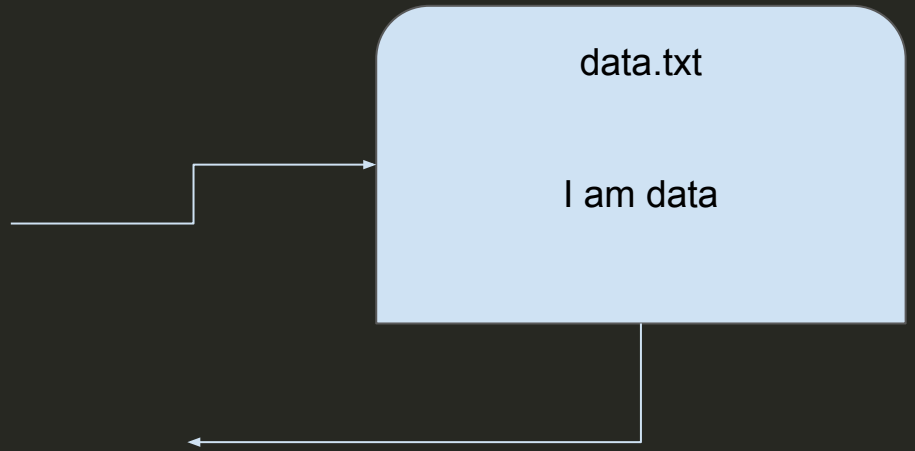
```
# Base Image
FROM ubuntu:18.04 CACHED

COPY data.txt data.txt CACHED

RUN cat data.txt CACHED
```



```
# Base Image
FROM ubuntu:18.04 CACHED

COPY data.txt data.txt CHANGED

RUN cat data.txt BROKEN
```

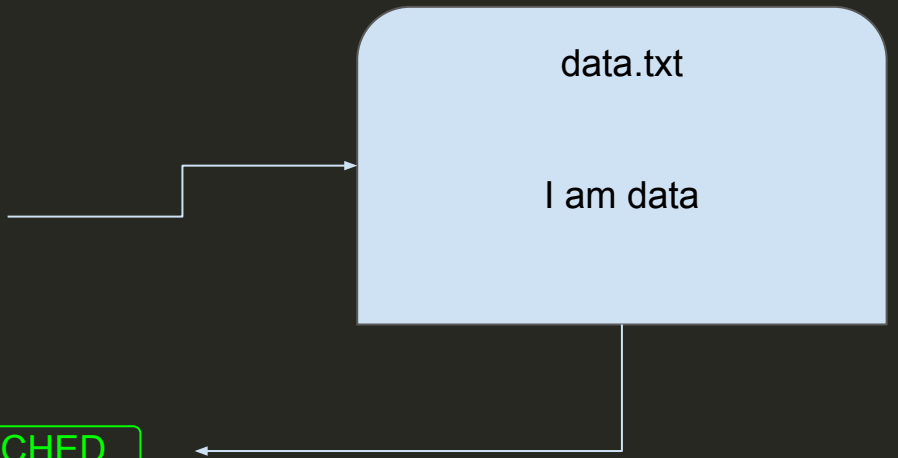
- Allows copying from intermediate stages without having those layers show up in the final image
- Cached separately from mainline docker image
- Replaces bash scripts clobbering together different build steps

```
# Copy Stage
FROM ubuntu:18.04 as copy_stage

RUN echo “copy from me!”

RUN echo “I am data” > data.txt

# Base Image
FROM ubuntu:18.04

COPY --from=copy_stage data.txt data.txt

RUN cp date.txt date2.txt

RUN echo “I am done”
```

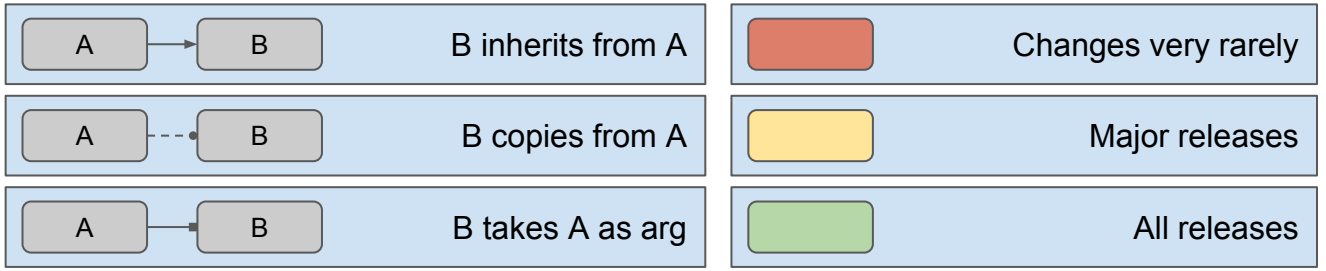


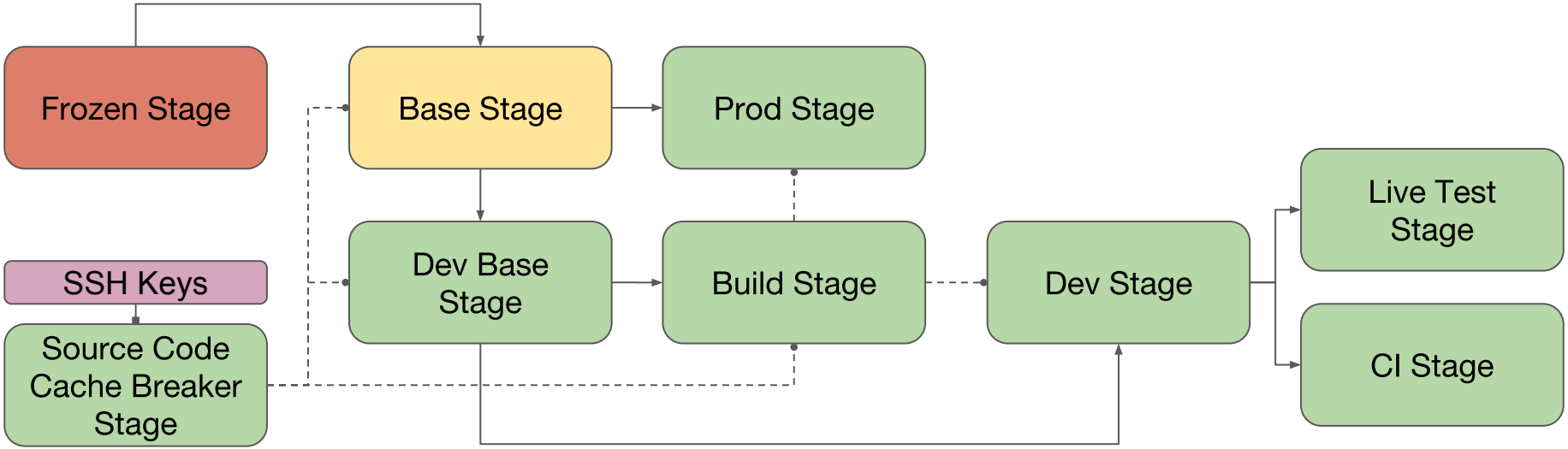
```
# Copy Stage
FROM ubuntu:18.04 as copy_stage CACHED

RUN echo “copy from me!” CACHED

RUN echo “I am data” > data.txt CACHED

# Base Image
FROM ubuntu:18.04 CACHED

COPY --from=copy_stage data.txt data.txt CACHED

RUN cp date.txt date2.txt CACHED

RUN echo “I am done” CACHED
```

```
# Copy Stage
FROM ubuntu:18.04 as copy_stage        CACHED

RUN echo “I am changed!”               CHANGED

RUN echo “I am data” > data.txt        BROKEN

# Base Image
FROM ubuntu:18.04                      CACHED

COPY --from=copy_stage data.txt data.txt   CACHED

RUN cp date.txt date2.txt              CACHED

RUN echo “I am done”                   CACHED
```

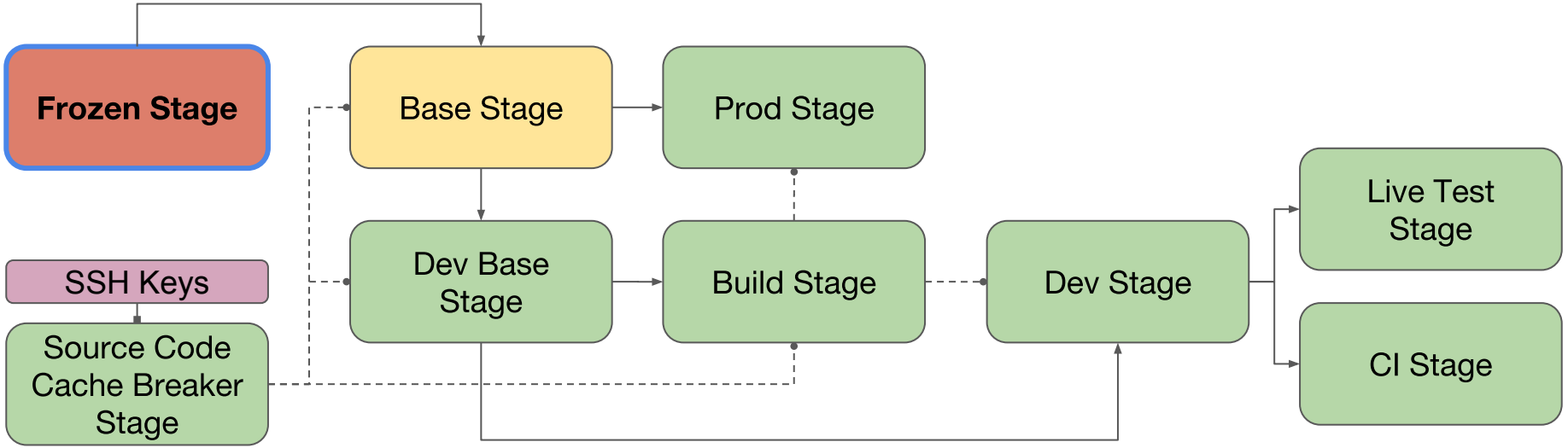


```
# Copy Stage
FROM ubuntu:18.04 as copy_stage          CACHED

RUN echo “I am changed!”                 CACHED

RUN echo “I am groot” > data.txt         CHANGED

# Base Image
FROM ubuntu:18.04                        CACHED

COPY --from=copy_stage data.txt data.txt CHANGED

RUN cp date.txt date2.txt                BROKEN

RUN echo “I am done”                     BROKEN
```


A
B
B inherits from A
A
B
B copies from A
A
B
B takes A as arg
Changes very rarely
Major releases
All releases

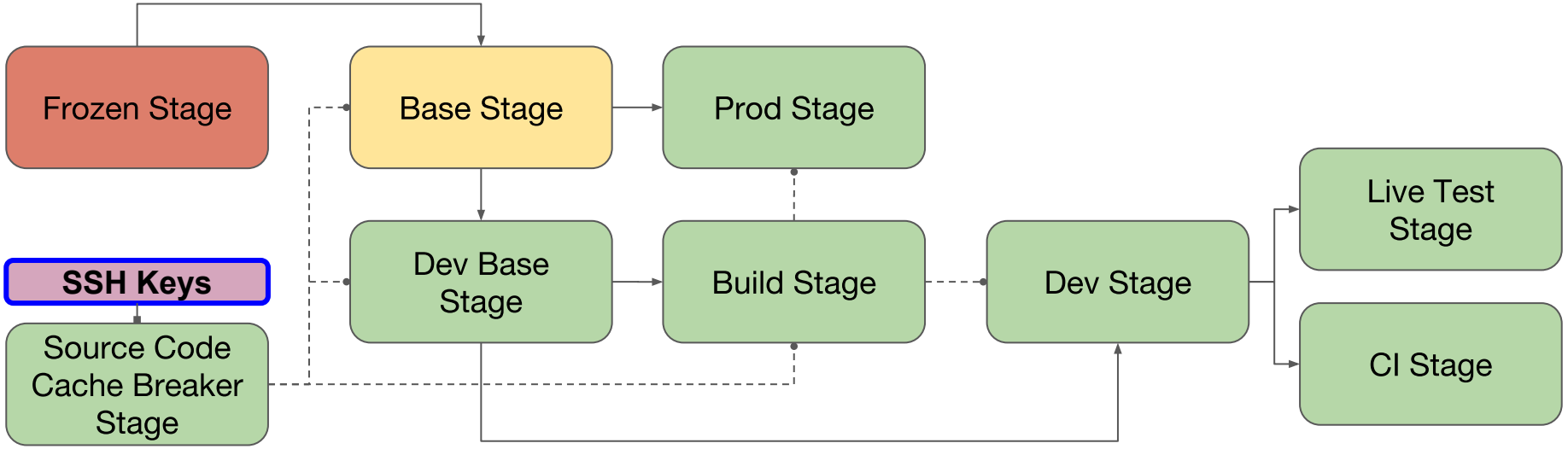

Frozen Stage
Base Stage
Prod Stage
SSH Keys
Source Code Cache Breaker Stage
Dev Base Stage
Build Stage
Dev Stage
Live Test Stage
CI Stage

- Needs to be updated extremely rarely
- Only needs to be built once and can be stored in a registry
- Can either be in same, or different Dockerfile



```
FROM ubuntu:18.04

# Install basic packages
RUN DEBIAN_FRONTEND=noninteractive apt-get update -y && \
                    apt-get install software-properties-common curl -y
```

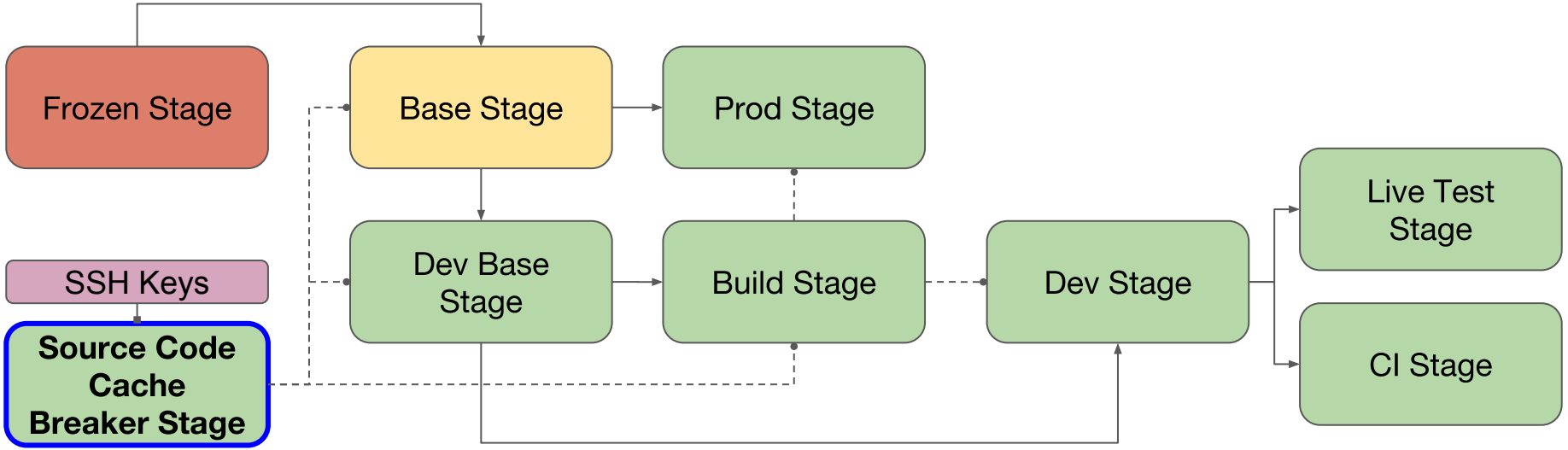
- Allows the use of secrets without leaking them into output images
- Can be combined with cache breaker stage to git clone private repos
- You don't have to share build keys, everyone can use their own (though the cache will break)



```
# Cache Breaking Stage
FROM ubuntu:18.04 as secret_stage

# Inject the secret we want to use
ARG SUPER_SECRET_PASSWORD
# Download remote file that may change between builds
RUN wget --user roscon --password $SUPER_SECRET_PASSWORD
https://legitexample.com/requirements.txt

# Work Stage
FROM python:3.7.0

# We can copy the requirements file but the password ARG never interacts with this stage's
filesystem
COPY --from=secret_stage /requirements.txt /requirements.txt
RUN pip install -r requirements.txt
```

```
docker build --build-arg SUPER_SECRET_PASSWORD="hunter2" -t robot:roscon
```

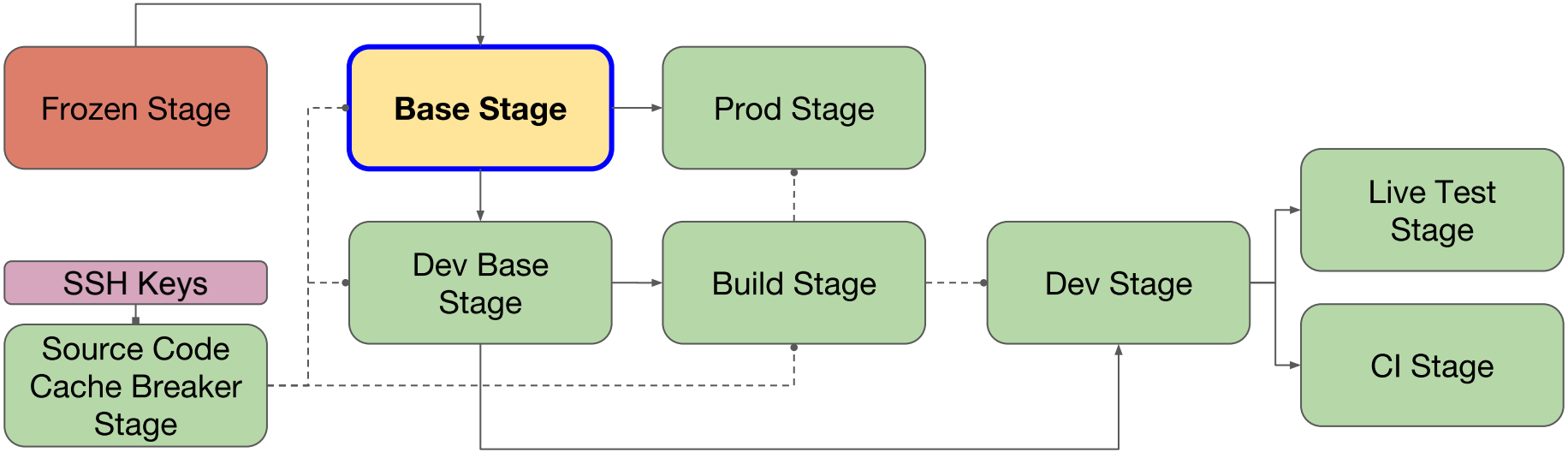
- Used to force a particular section to never be cached, without breaking the cache for the entire Dockerfile
- Can be used to force updates, download source code, get time/date related data
- If the data copied out of this stage has not changed, the stage that copies from it will remain cached



```
# Cache Breaking Stage
FROM ubuntu:18.04 as cache_breaker_stage

# Breaks the cache every time this arg changes
ARG RANDOMIZER
# Download remote file that may change between builds
RUN wget https://legitexample.com/requirements.txt

# Work Stage
FROM python:3.7.0

COPY --from=cache_breaker_stage /requirements.txt /requirements.txt
# This will be cached so long as requirements.txt has not changed
RUN pip install -r requirements.txt
```

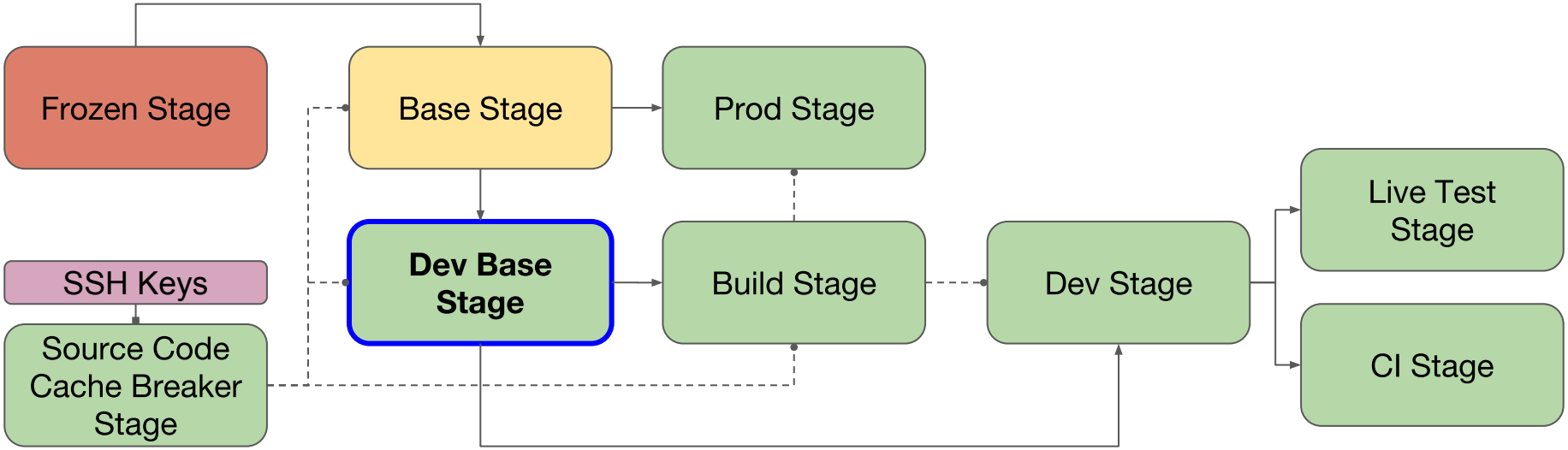
```
docker build --build-arg RANDOMIZER="$(date|md5sum)" -t robot:roscon
```

- Contains all runtime (production) requirements for the robot
- If you were to copy the built binaries into this stage, they should be able to run
- Common base for dev, production, and testing images for a given release, and inherits from frozen stage



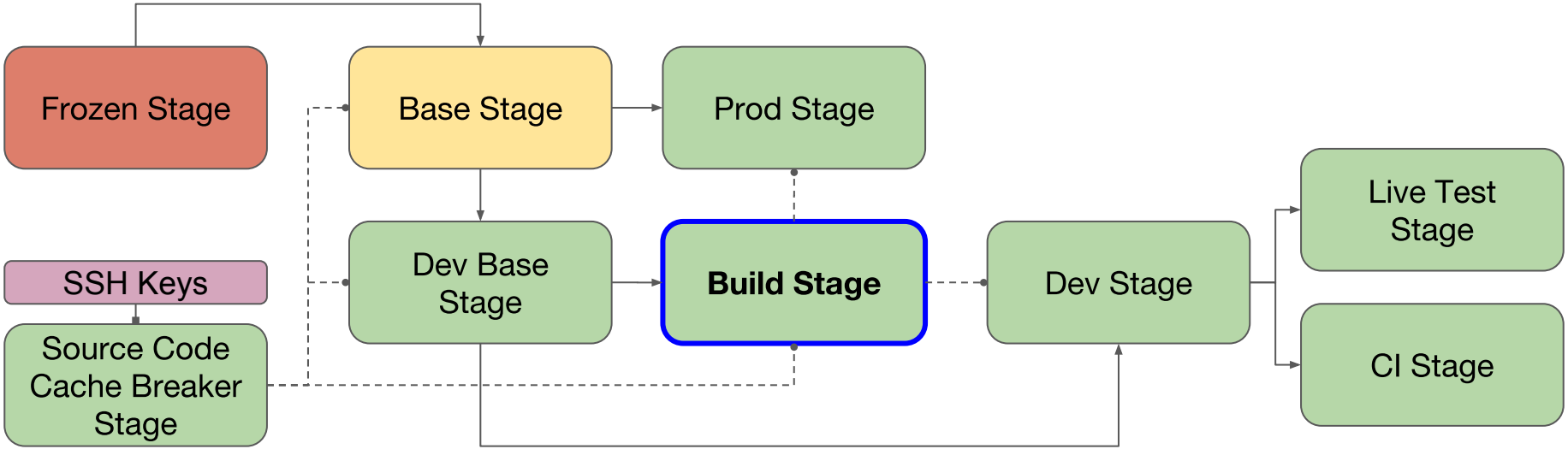
```
# Cache Breaking Stage
FROM ubuntu:18.04 as cache_breaker_stage

ARG SUPER_SECRET_PASSWORD
ARG RANDOMIZER

# Download remote file that may change between builds
RUN wget --user roscon --password $SUPER_SECRET_PASSWORD
https://legitexample.com/requirements.txt

# Base Stage
FROM frozen_stage as base_stage

# Install all of our runtime requirements using pip, apt, curl, etc...
COPY --from=cache_breaker_stage /requirements.txt /requirements.txt
RUN pip install -r requirements.txt
```

- Contains all build time requirements for the robot, as well as source code
- At the end of this stage, if you were to call *build* on any of your projects, they should be successful
- Can be used by developers for debugging build issues



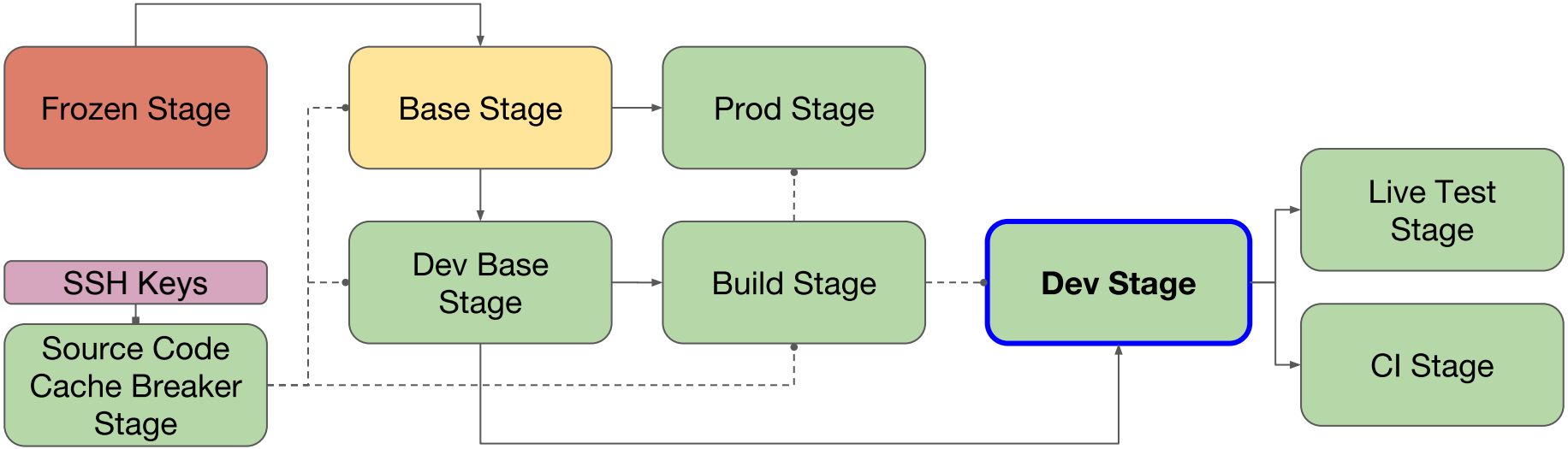
```
# Cache Breaking Stage
FROM ubuntu:18.04 as cache_breaker_stage

ARG SUPER_SECRET_PASSWORD
ARG RANDOMIZER

# Download remote file that may change between builds
RUN wget --user roscon --password $SUPER_SECRET_PASSWORD
https://legitexample.com/requirements-dev.txt
RUN git clone git@github.com:sourcecode/isthirteen.git

# Dev Base Stage
FROM base_stage as dev_base_stage

COPY --from=cache_breaker_stage /isthirteen /catkin_ws/src/isthirteen
# Install all of our build/dev requirements using pip, apt, curl, etc...
COPY --from=cache_breaker_stage /requirements-dev.txt /requirements-dev.txt
RUN pip install -r requirements-dev.txt
```

- Builds all of the source code, by the end of the stage you have all of your binaries prepared
- Gets cannibalized for binaries and then discarded
- Will only be copied from, so you can inject secrets here as well for things such as npm/bazel



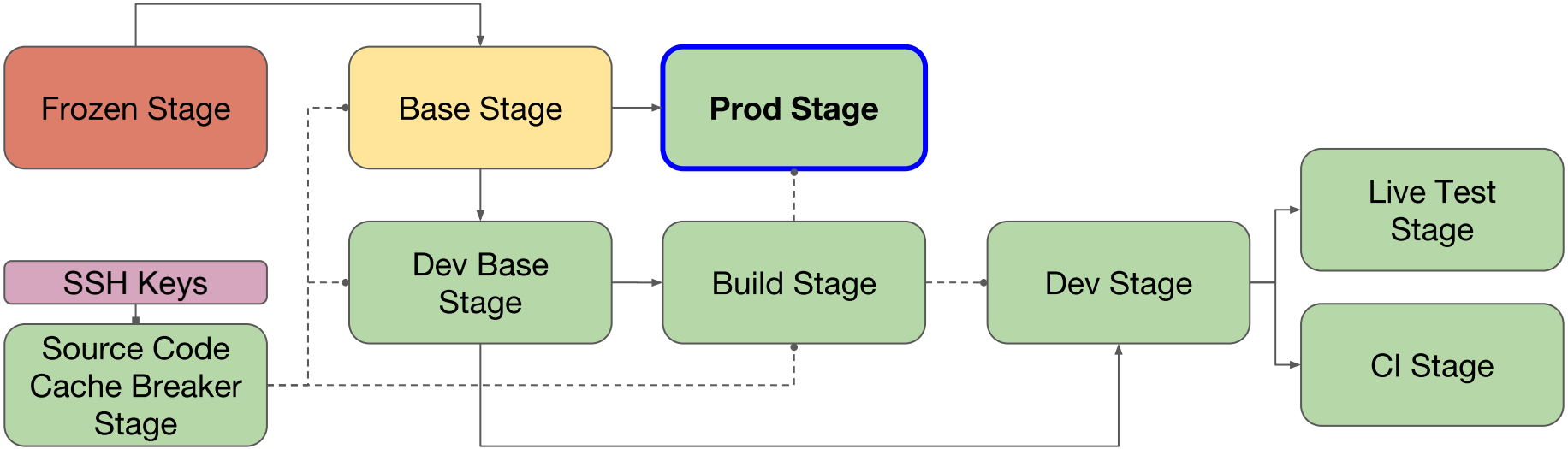
```
# Base Stage
FROM dev_base_stage as dev_build_stage

ARG SUPER_SECRET_PASSWORD

# Build the stuff!
RUN cd catkin_ws && catkin_make install
```

- Is your final dev image, you should be able to put this on a newly reimaged robot and get testing right away
- Can be used directly as a development environment, or as a base for CI/CD images
- Inherits from dev base stage so it has all of the runtime dependencies and build dependencies



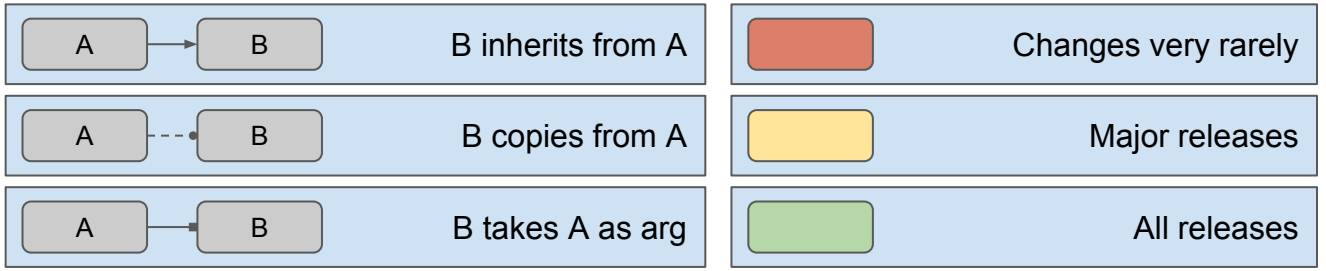
```
# Base Stage
FROM base_stage as prod_stage

# We're done!
COPY --from=dev_build_stage /catkin_ws/ /catkin_ws/
```

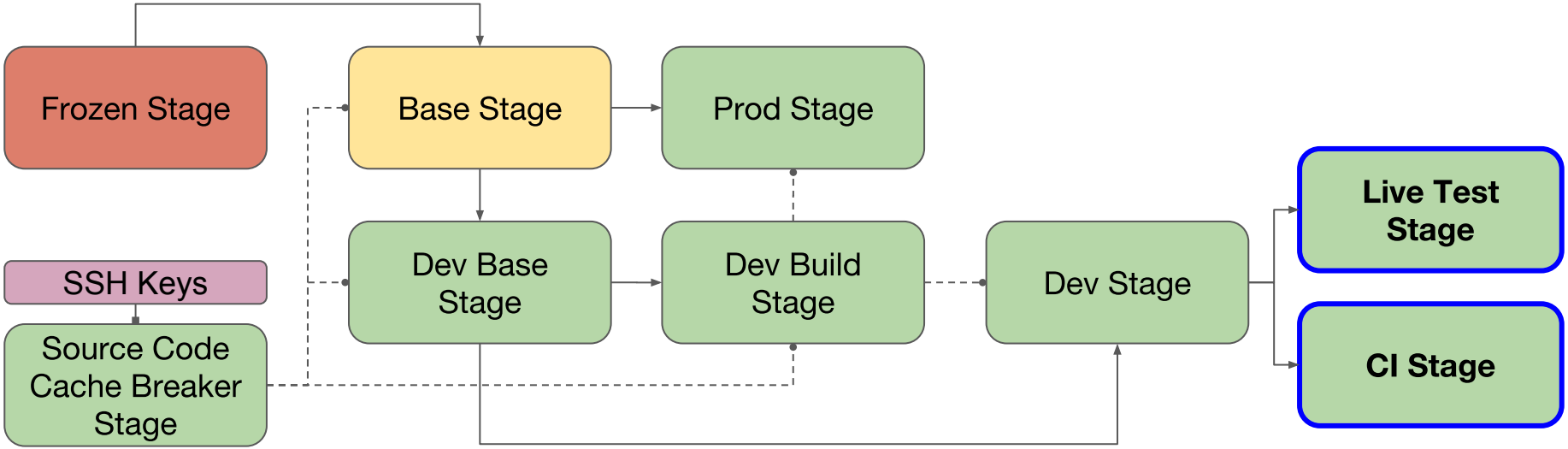
- Is your final production image, you should be able to put this on a newly reimaged robot and call it a day
- Copies over only the binaries, no need for the source code
- Inherits from base stage so it has all of the runtime dependencies



- Is your final production image, you should be able to put this on a newly reimaged robot and call it a day
- Copies over only the binaries, no need for the source code
- Inherits from base stage so it has all of the runtime dependencies

```
# Base Stage
FROM base_stage as prod_stage

# We're done, but actually!
COPY --from=dev_build_stage /catkin_ws/install /catkin_ws/install

# Set a command to automatically run when the container is started
CMD /bin/bash -c "source /catkin_ws/install/setup.bash && roslaunch isthirteen
is_it_thirteen.launch"
```


A
B
B inherits from A
A
B
B copies from A
A
B
B takes A as arg
Changes very rarely
Major releases
All releases


Frozen Stage
Base Stage
Prod Stage
SSH Keys
Source Code Cache Breaker Stage
Dev Base Stage
Dev Build Stage
Dev Stage
Live Test Stage
CI Stage

- Developers can have different versions with mutually exclusive dependencies running on the same robot
- No more worries about robots other developers have snowflaked, your own environment is portable and hermetic

- Entire build pipeline for all production robot types is now just ~300 lines of Dockerfile
- Releases are now nightly and on-demand, take under two hours instead of a day
- Build pipeline is transparent and easy to understand and modify
- Building a release candidate just takes github ssh keys and docker, no other tooling required

- Worry free testing in the field. If something goes wrong, just start a new container!
- Upgrades/downgrades are one shot, no failed intermediate states

- Layer and image size is now a first class concern and can affect update and dependency management strategies
- Changing a layer high up in the chain can cause large updates, up to 2GB in our case
  - But, images can be downloaded in the background without interrupting operation
- Dev images with unstripped binaries can be very large (~20GB), pulling/pushing to robots can take a while if you don’t have good wifi (still faster than pulling source and recompiling, though!)
- Requires new infrastructure (local and cloud registries)

Questions?