Learn the Go Programming Language

For experienced developers or
those of an adventurous nature

gotutorial.net
@GoTutorialNet

Matt Nunogawa
@amattn

Paarth Patel
@paarthp

Lesson 11

Deployment, Containers
& A Case Study

v0.1 draft

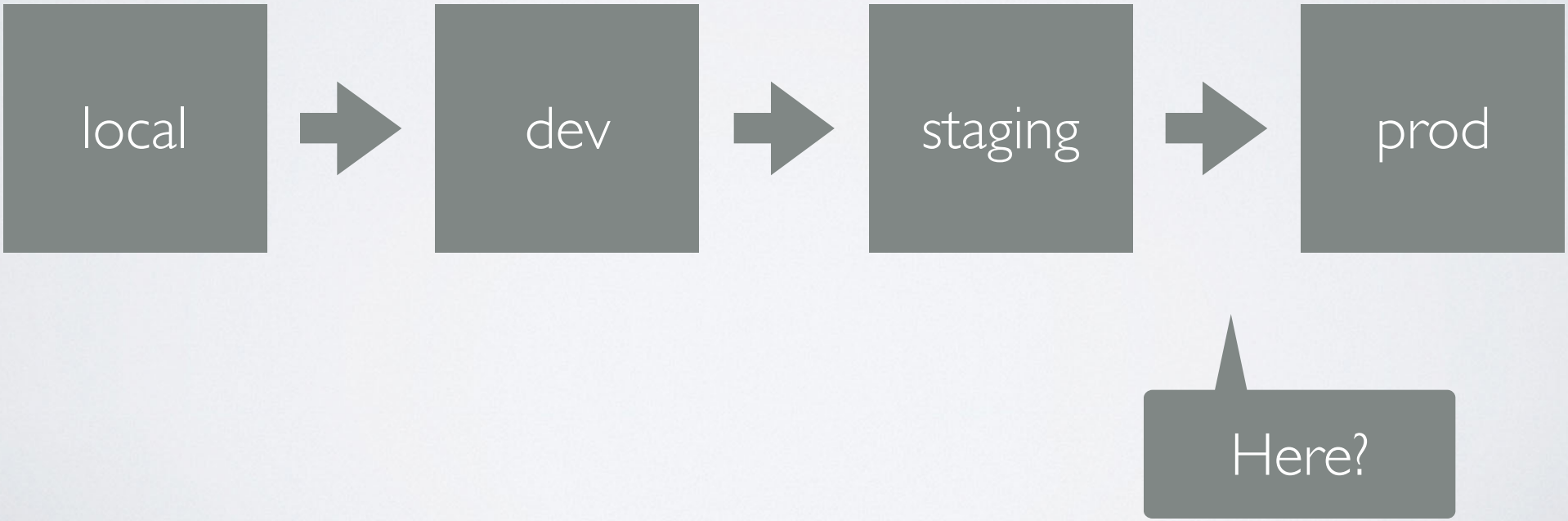
Prelude: Deploying Go Binaries

- build with go, gox or goxc
- scp
- upstart, systemd, supervisord, etc.
- optionally monitor process/endpoint/logs
  - mmonit, http://godrb.com, pingdom, etc.

Brief Interlude on the nature and notion of Deployment

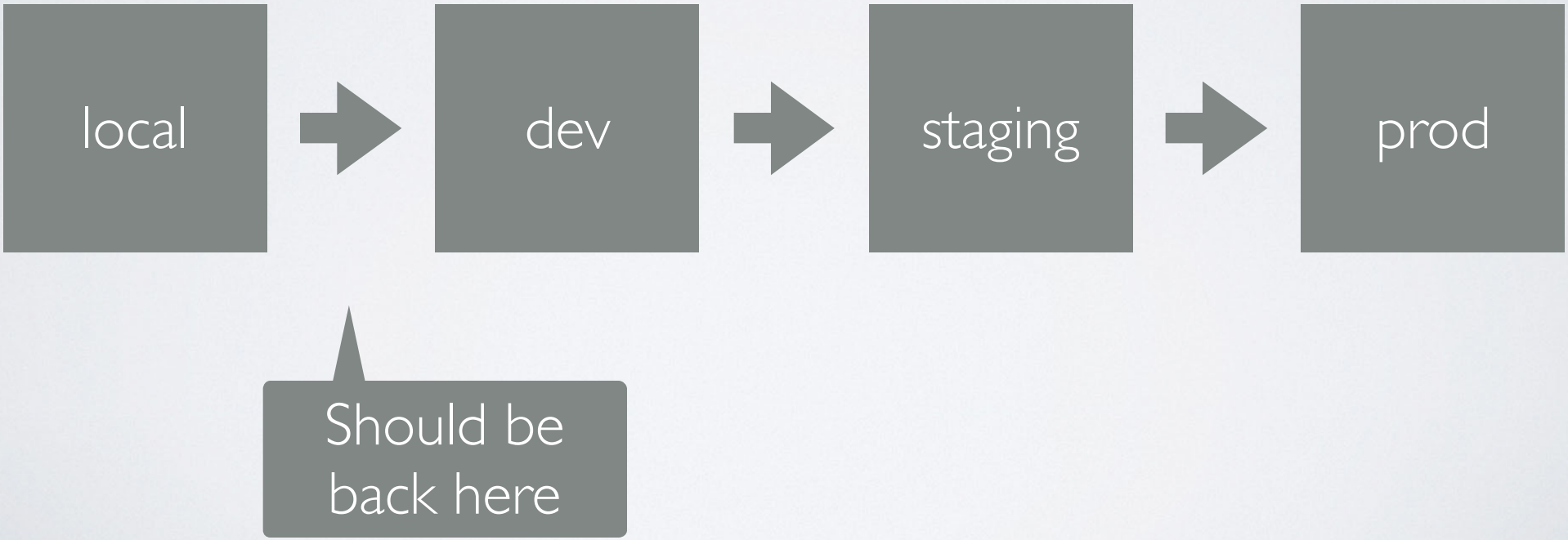
Typical Flow

When does "deployment" happen?



Conceptually

Every commit should create a potentially deployable artifact



"Every commit is deployable."

–The Incredible Hulk

Containers

Containers, General Info

- Gets us closer to our ideal concept of "every commit is deployable"
- Package context + actors in a single artifact
- Wild West
- Can get "Inception-y"

Containers

- Automate everything
- We use docker
  - Really like Dockerfiles
- We really like CoreOS toolset
  - (etcd, flanneld, et al.)

Basic Container Strategy

- At a high level:
  - Run containers in a VM on your Mac
  - Run those same containers with different config on multiple VMs in the cloud.

Container Issues

- Config Issues
- keeping them up to date
  - be prepared to be constantly rebuilding your containers. Even if your code doesn't change

Container Issues

- containers can't normally see each other
- IP Overlay Networks
  - flanneld or similar
  - relying on port forwarding is fragile, *fragile*, **fragile**

Container Issues

- hub.docker.com repo
  - is slow, not free for private repos
  - images (esp java ones) are > 1 GB
- self hosted private repo
  - operational overhead
  - we've settled here

Container Issues

- CI is a challenge
  - not all are docker aware
  - the exceptions kind of expect "one container per repo"
  - you will be writing up lots of helper scripts or play nicely with their limitations

Container Issues

- Deployment
  - the promise: containers run anywhere
  - the reality: fragility is normal and must be actively engineered against

Case Study
Collective Health

LAYOUT

- Everyone has a Root dir
- Everyone has the same layout in that dir

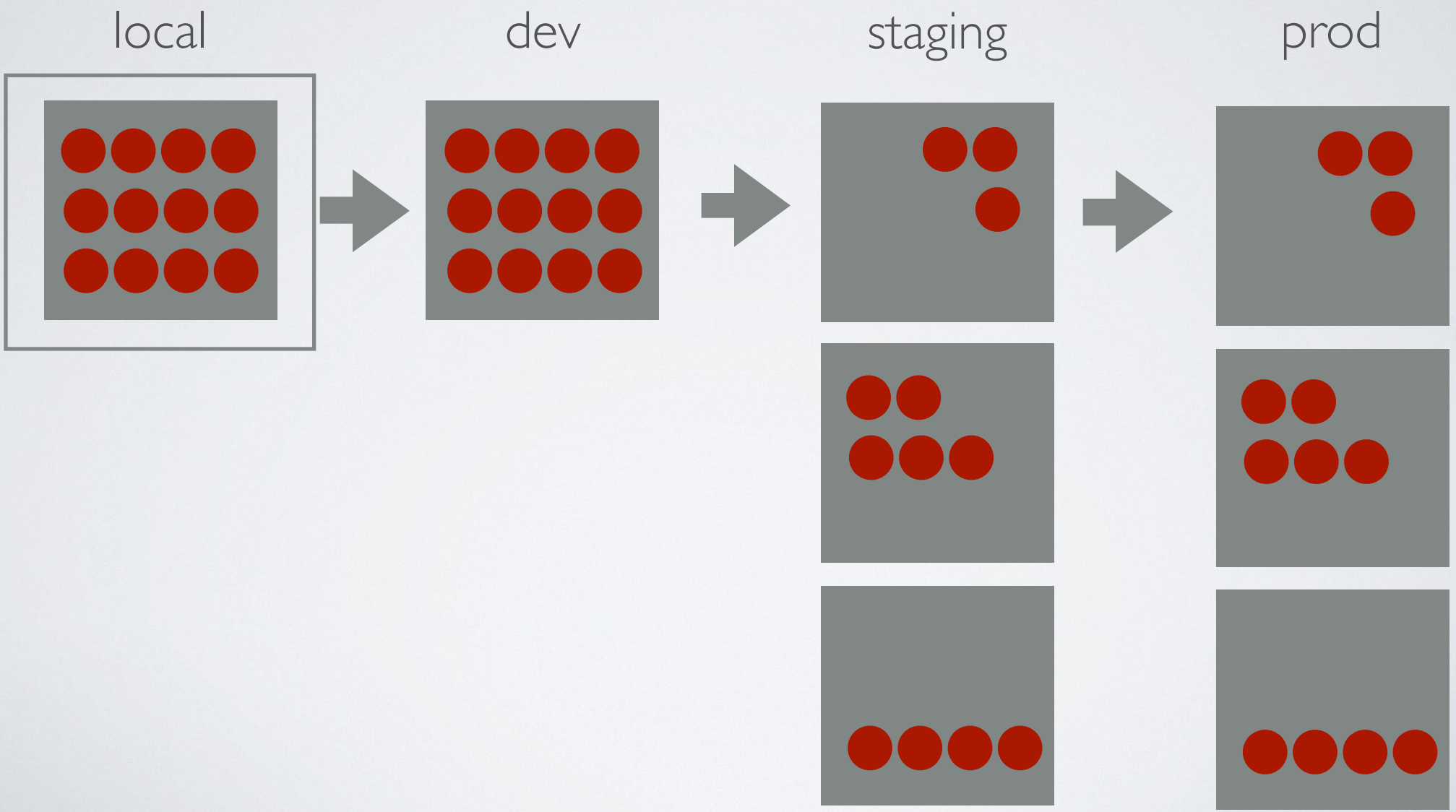
```
CHROOT="/Users/$USERNAME/cchh"
GOPATH="/Users/$USERNAME/cchh/gopath"

$CHROOT/bin
$CHROOT/repo1
$CHROOT/repo2
$CHROOT/gopath/src/github.com/collectivehealth/repo3
$CHROOT/gopath/src/github.com/collectivehealth/repo4
```

DEVENV

- Everyone has a shared repo of scripts, metadata, documentation etc.

CH Flow



Config

Config Is A Pain

- Need to be able to inject config into containers
- keep track of per-container config
- secrets (passwords, tokens) are troublesome

How we do it

- Every repo has a container/ dir where we store various config, helper scripts, etc.

```
$CHROOT/repo2/container
$CHROOT/repo2/container/Dockerfile
$CHROOT/repo2/container/bin/pre_setup.sh
$CHROOT/repo2/container/etc/config.dev.toml
$CHROOT/repo2/container/etc/config.prod.toml
$CHROOT/repo2/container/etc/supervisord.conf
...
```

How we used to do it

- We tried an "all_config" repo, but we had fragility keeping disparate repos in sync

- Easy to version the "whole cluster"

```
$CHROOT/repo1/<src>
$CHROOT/repo2/<src>
$CHROOT/all_config/containers/repo1/<config files>
$CHROOT/all_config/containers/repo2/<config files>
```

Config, Misc

- Minimize work needed to "customize" a container for a specific stage
- Our containers contain contain config files for every environment
- Lots of config Injection options:
  - ENV vars, config files, kv store (etcd, etc.)
  - Use the same method for all environments (local dev, staging, prod, etc.)

SUGGESTION

- recommend some way to keep all config under version control than have a single ENV, or kv entry for CLUSTER_ENV or similar

```
# sudo docker run -e “CLUSTER_ENV.toml=staging"

# on fs:
/container/etc/config.local.toml
/container/etc/config.dev.toml
/container/etc/config.staging.toml
/container/etc/config.prod.toml

# from scripts
run.sh –config=config.$CLUSTER_ENV.toml

# in your app
switch os.GetEnv(“CLUSTER_ENV”) {
    case “prod":
        // ...
    case "dev":
        // ...
    case "local":
    default:
        // ...
}
```

Secrets

- Tokens, passwords, etc.
  - Definitely don't check into git
  - We currently have just a few flat files with kv pairs. Each dev copies the file from a secure location. Use slack or email to communicate updates. This works fine for small groups or if your secrets file doesn't change often.
  - We plan to eventually migrate to a specific tool such as:
    - Hashicorp's Vault, Square's KeyWhiz, Amazon's KMS, etc.
    - And store our master key in a HSM: http://en.wikipedia.org/wiki/Hardware_security_module because our Ops is paranoid

Local Development

Local Dev Goals

- Must be easy to do
  - edit file, run a script, test local cluster
- Must have quick build, run, test loops

Local Dev @ CH

- We currently spin up a single VM on a laptop with 10-15 containers:
  - all FE code, served by nginx
  - many backend components
  - messaging layer
  - DB and NoSQL containers
- takes one script to spin up a well configured VM and a second script to build and run all containers
- all data is considered ephemeral. we blow these away regularly.

Tried a bunch of different Processes

1. spin up containers
2. edit file(s)
3. run script that builds (gox), injects into running container, restarts appropriate process
4. test

CH Lessons learned

- Everyone has a shared repo of scripts, metadata, documentation etc.
- Break out of the inception
  - /Users is available to VM
  - specific folders mounted in containers as volumes

Currently Process:

1. Fetch latest from CI, mount local volume with all the pre-built binaries, config in all of the containers
2. edit file(s)
3. run script that builds (gox), overwrites local binary, restarts process
4. test

Dev Patterns

- CI does building, testing, shaming
- FE can point to dev environment or just fetch all CI built layers or finally, just build if they choose to
- BE can fetch latest layers and override interesting locations in the run context. Or, just build if they choose to

Build & Release

"We are in the business of shipping containers."

–Paarth

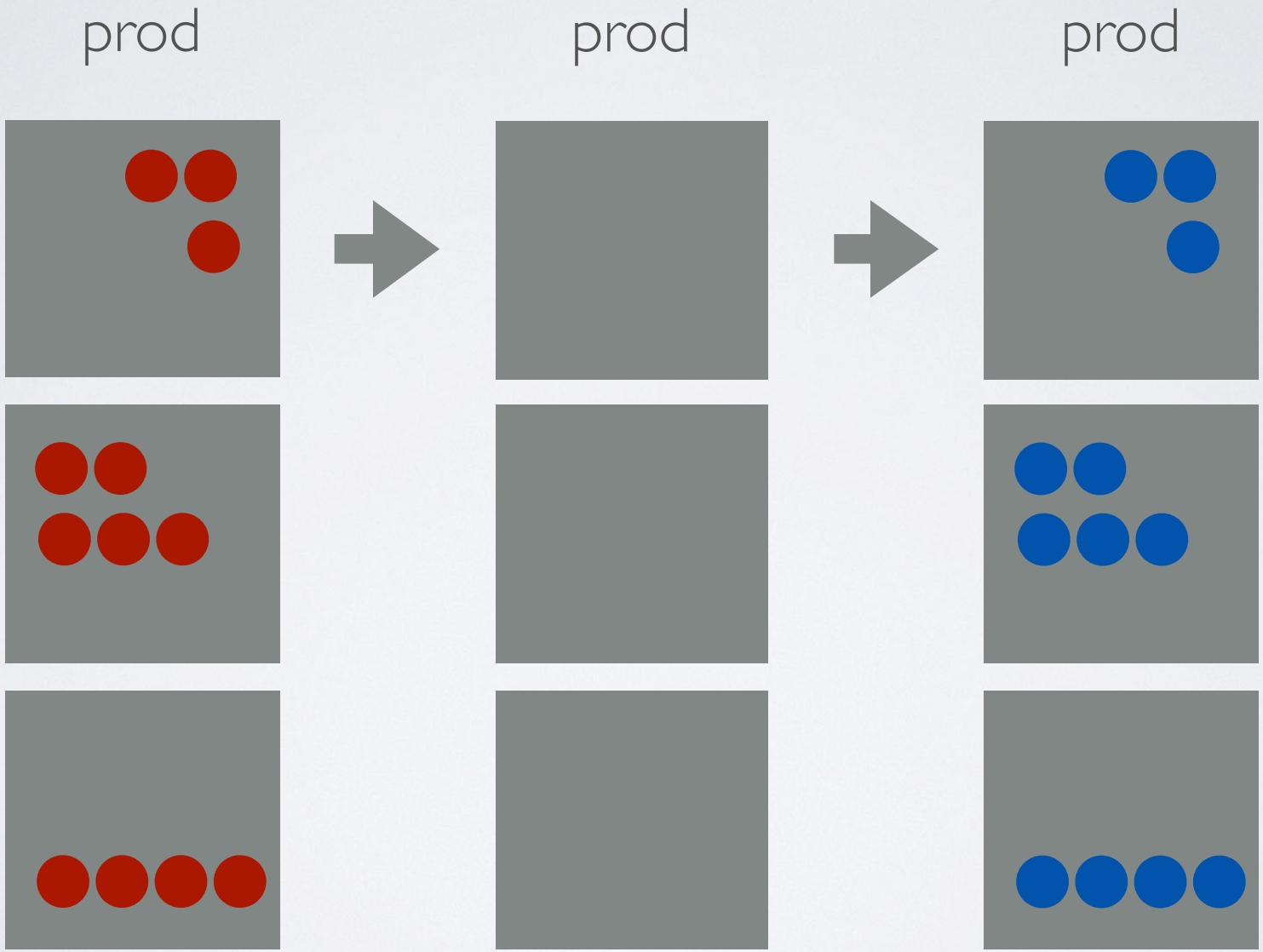
Continuous Integration

- We use Jenkins & dotCI plugin
- We don't do CD, yet
- We do CI up to test, then PBR for staging and prod
- Build and release begins early on along side development.

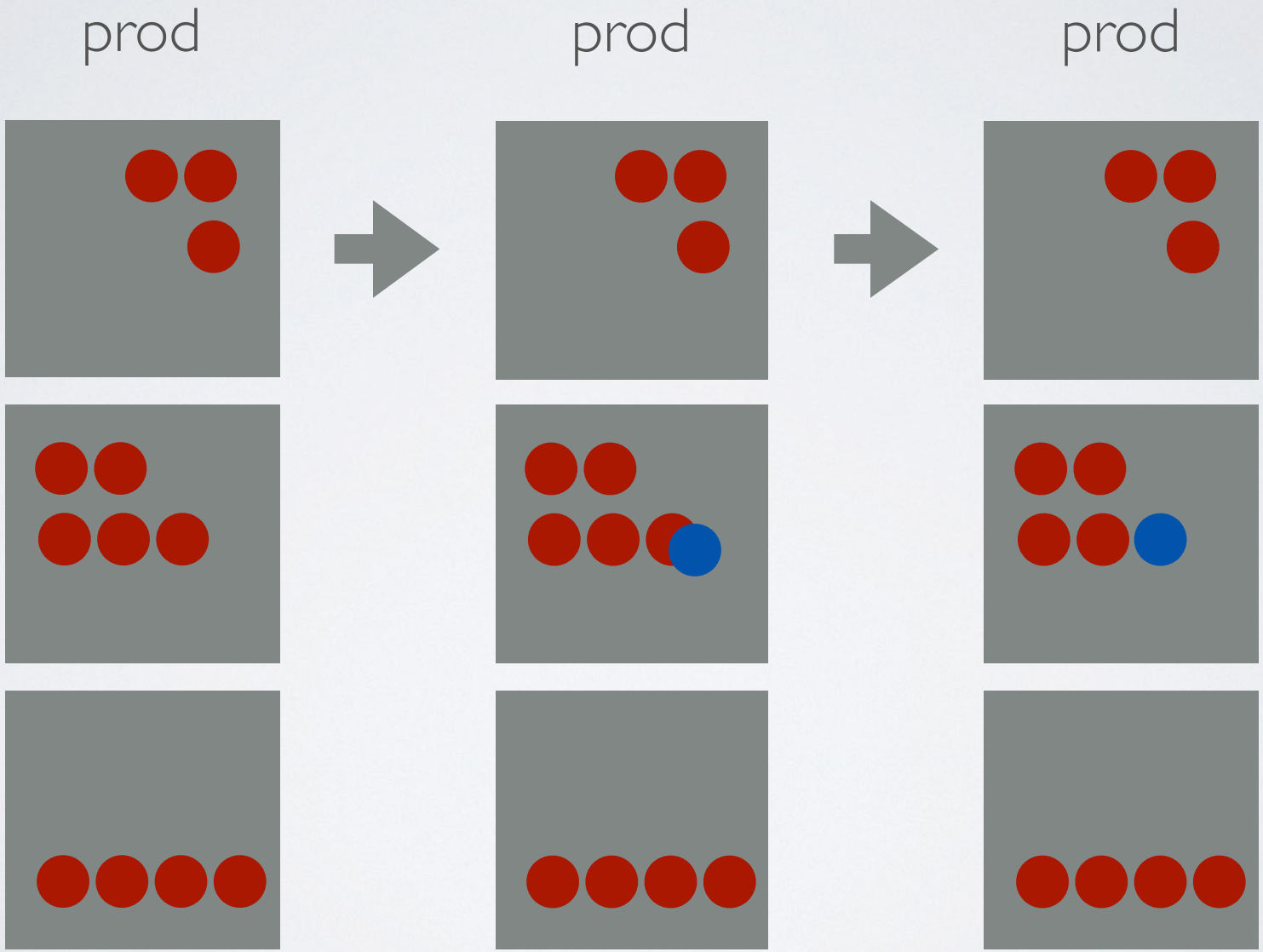
Deploy

- Currently doing "whole cluster" deploys
  - We can tolerate the downtime (couple of minutes)
- Goal: load balanced, per component deploys

Whole Cluster



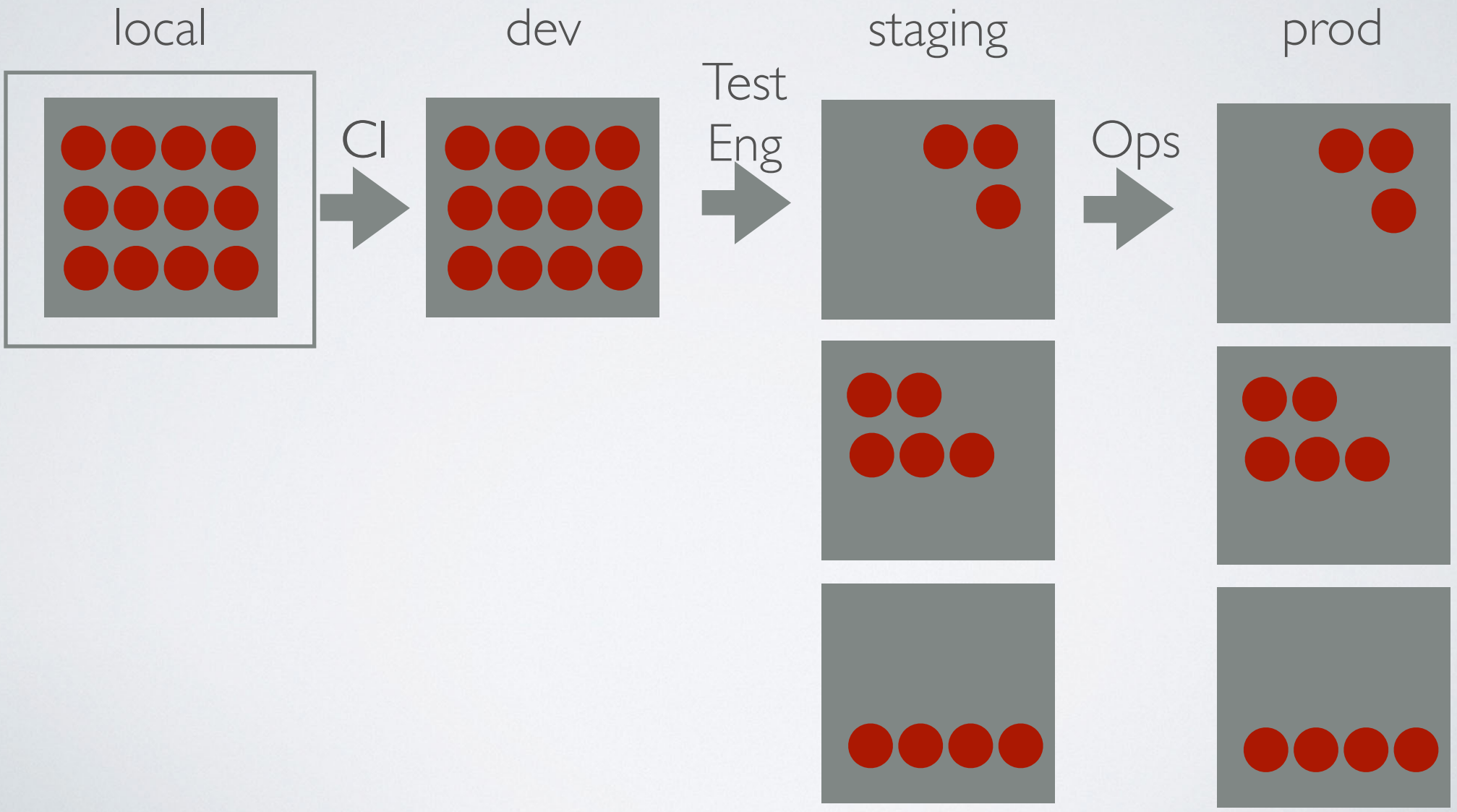
LOAD BALANCED, PER COMPONENT



Artifact Promotion v1

- Test engineering driven via CI
- Ops takes over when a RC reaches prod maturity

CH Flow



Lessons Learned

- Containers don't really contain
- Invest early on in making containers first class citizens in your network stack
- Ship containers

- Overload runtime context for your artifact customizations
- Invest early on in versioning strategy
- Ship containers

- Static builds make your life easy
- Invest in base container strategy, choose base image wisely, bias towards smallest possible
- Ship containers

Misc

Saving Time

- Our bash scripts spend a lot of LOC checking if we can skip a download or two.
  - Build times are sped up considerably
  - If downloads are up to date, entire cluster can launch in tens of seconds.

BASH FRAGMENT

```
# for files that don’t change very often use checksum

ValidateChecksumOrDownload() {
    if [ $# -eq 3 ]; then
        local_filename=$1
        expected_md5=$2
        remote_url=$3
    else
        inner_stdout "$script_name: ValidateChecksumOrDownload: Invalid argument $@"
    fi

    local candidate_md5=$(GetChecksumForFile $local_filename)

    # inner_stdout "candidate_md5 $candidate_md5"
    # inner_stdout "expected_md5 $expected_md5"

    if [[ "$candidate_md5" == "$expected_md5" ]]
    then
        inner_stdout "checksum for $local_filename matches $expected_md5, skipping download"
    else
        inner_stdout "checksum for $local_filename expected: $expected_md5, got $candidate_md5"
        inner_stdout "starting download $remote_url"
        "$CURL" -L# -o $local_filename $remote_url
    fi
}
```

BASH FRAGMENT

```
# for files that do change often, we just use the existence of the file to determine whether to
download or not.

ValidateFileExistsOrDownload() {
    local local_filename="<UNDEFINED_LOCAL_FILENAME>"
    local remote_url="<UNDEFINED_REMOTE_URL>"
    if [ $# -eq 2 ]; then
        local_filename=$1
        remote_url=$2
    else
        inner_stdout "$script_name: ValidateFileExistsOrDownload: Invalid argument $@"
    fi

    if [ -f "$local_filename" ]
    then
        inner_stdout "$local_filename found, skipping download"
    else
        inner_stdout "$local_filename not found, downloading from $remote_url"
        # -L location redirect (because github does lots of redirects)
        # -H github sends assets as octet-stream ("arbitrary binary data", per RFC2046), we need
to be able to handle that.
        "$CURL" -L# --output "$local_filename" -H 'Accept: application/octet-stream' $remote_url
    fi
}
```

Constantly Pulling

```
find_all_repos_in_dir() {
parent_dir=$1

echo ""
echo "_____________________________________________"
echo "Fetching all repos in parent dir:$parent_dir"

subdirs=$(find $parent_dir -type d -depth 1)
parallel --no-notice -j64 --keep-order --verbose
pull_ffonly.sh ::: $subdirs
}

find_all_repos_in_dir "$CHROOT"
find_all_repos_in_dir "$GOPATH/src/github.com/amattn"
find_all_repos_in_dir "$GOPATH/src/github.com/collectivehealth"
```

Constantly Pulling

```
# We want to fetch and update ONLY if we can cleanly do so.  The choice of rebase
or merge is up to the developer.
pull_ff_only() {
local subdir=$*
# printf "subdir $subdir"
# check if the .git directory exists
if [ -d "${subdir}/.git" ]; then
  printf "%s: " "$subdir"
  git --git-dir=${subdir}/.git --work-tree=$subdir pull --ff-only
  rc=$?
  if [[ $rc != 0 ]] ; then
    printf "\n${red}============================================$NC\n"
    printf "${red}rc: $rc$NC\n"
    printf "${red}Failure to fetch and or cleanly fast forward:$NC\n"
    printf "${red}$subdir $NC\n"
    printf "${red}============================================$NC\n"
    exit $rc
  fi
fi
}
```

Have a Dummy Component

- We use a component called books
- basically a book, author, title app, model and REST API
- helpful to test SOA, latencies, etc.

Thank You, Credits & License

http://gotutorial.net
@GoTutorialNet

Matt Nunogawa
@amattn

- I owe many many, thanks to the many authors of Go.
- These slides are Copyright 2013-2015 Matthew Nunogawa