The ARM to z of Multi-Architecture Microservices

Christy Norman Perez
Open Source Developer, IBM

Chris Jones
Open Source Developer, IBM

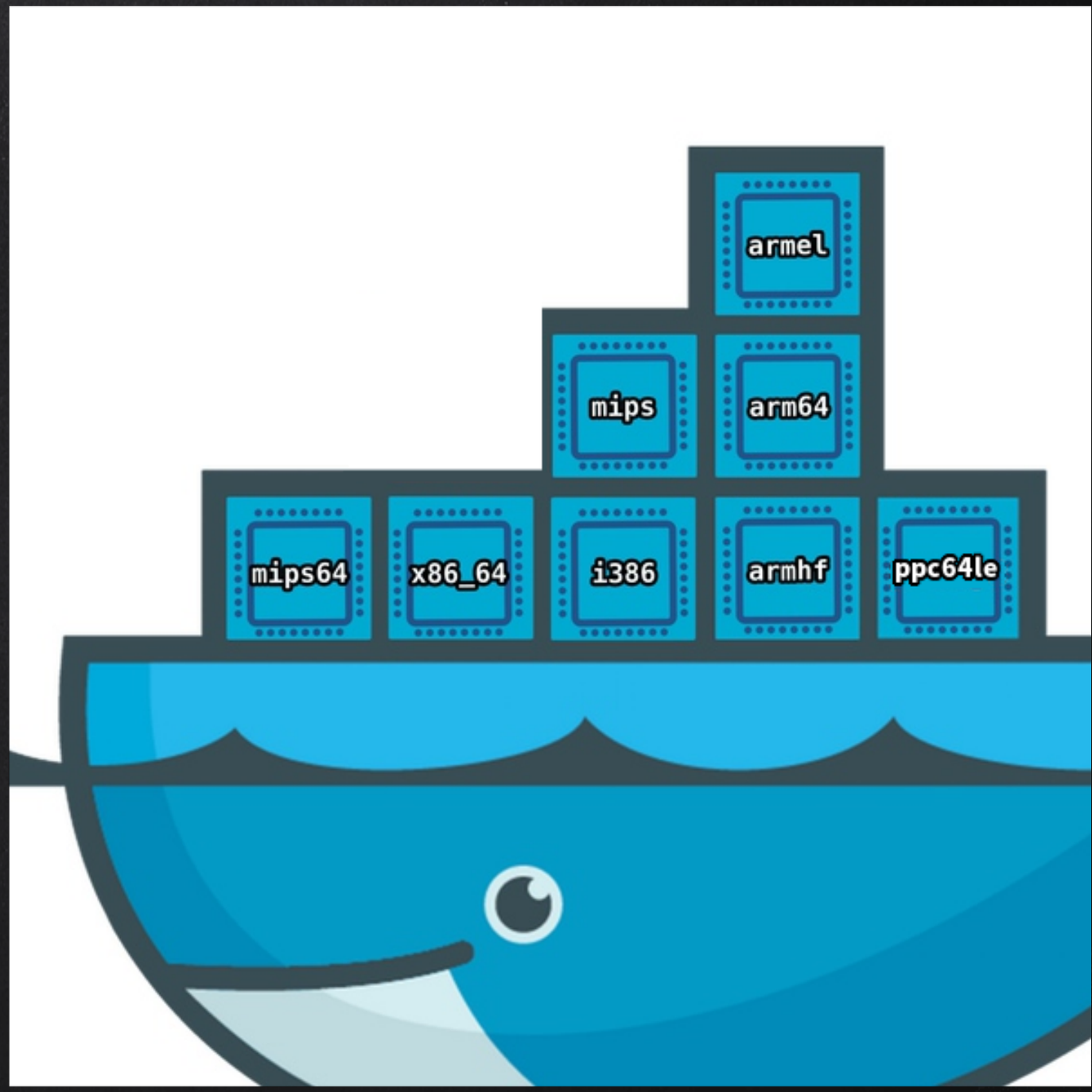
Agenda

- What is "multi-arch?"
- Challenges
- Benefits
- The path of multi-arch
- Demo – Building images
- Demo – Shipping images
- Demo – Running Swarm
- Q&A

What are we talking about?

Goals

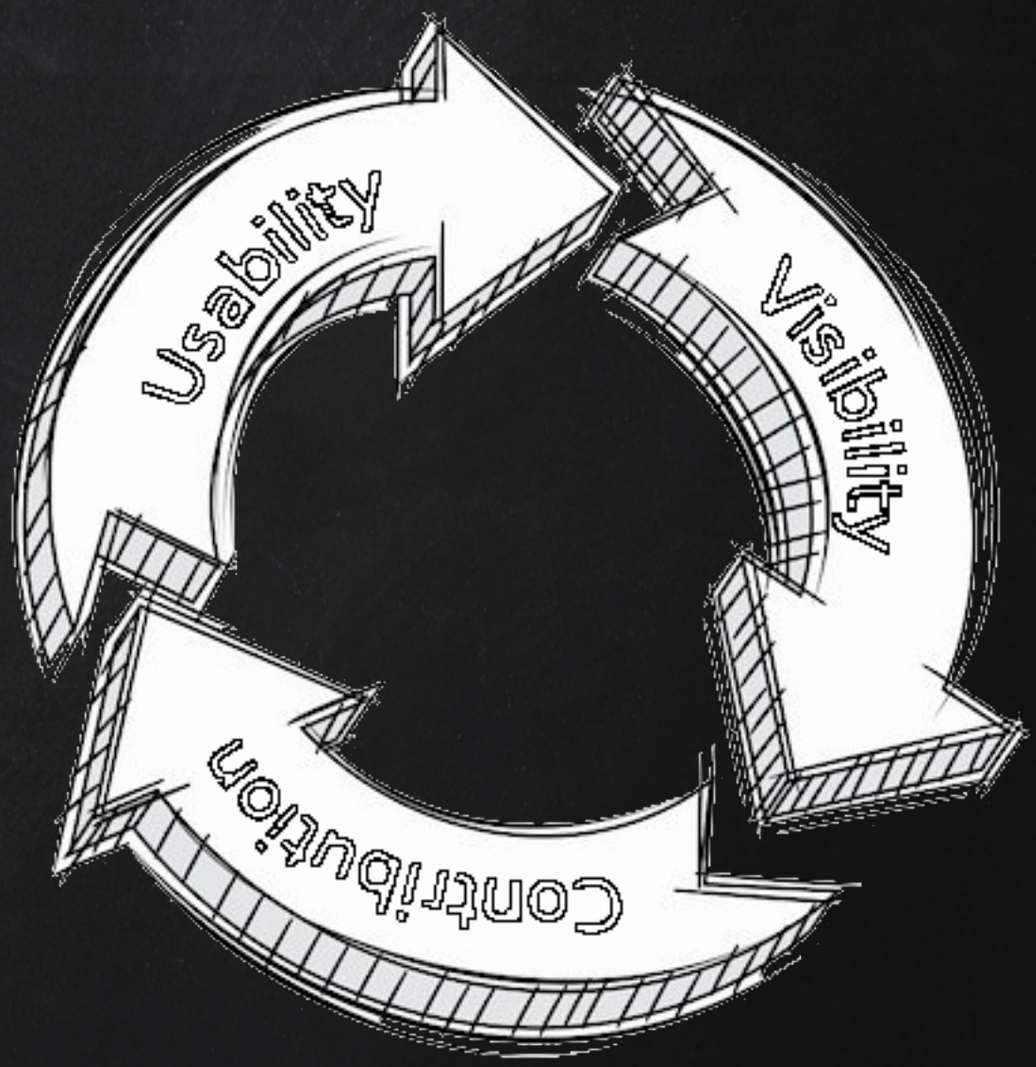
- User experience across architectures is the same
- Expand your project



Why should you care?

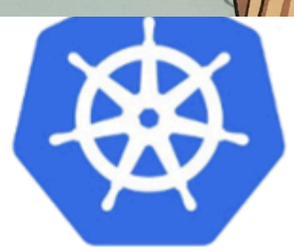
Benefits

- Grow your project
- Strengths of other platforms



Vendor lock-in

Benefits to others



kubernetes

ubuntu

Challenges

Usability

```
$ uname –m
ppc64le

$ docker run –it ubuntu
standard_init_linux.go:178: exec user process caused
“no such file or directory”

$ docker run –it ppc64le/ubuntu                    *
root@eb7051894530:/#
```

Availability

```
$ uname –m
arm64

$ docker run –it rethinkdb
standard_init_linux.go:178: exec user process caused
“no such file or directory”

$ docker run –it arm64/rethinkdb
docker: Error response from daemon: repository
arm64/rethinkdb not found: does not exist or no
pull access.
```

Summary

- Users have to remember they are on a different architecture
- Arch-dependent (Devs?) image names harm usability (ref frozen images script?)
- Siloed projects cause heartbreaks

Tools

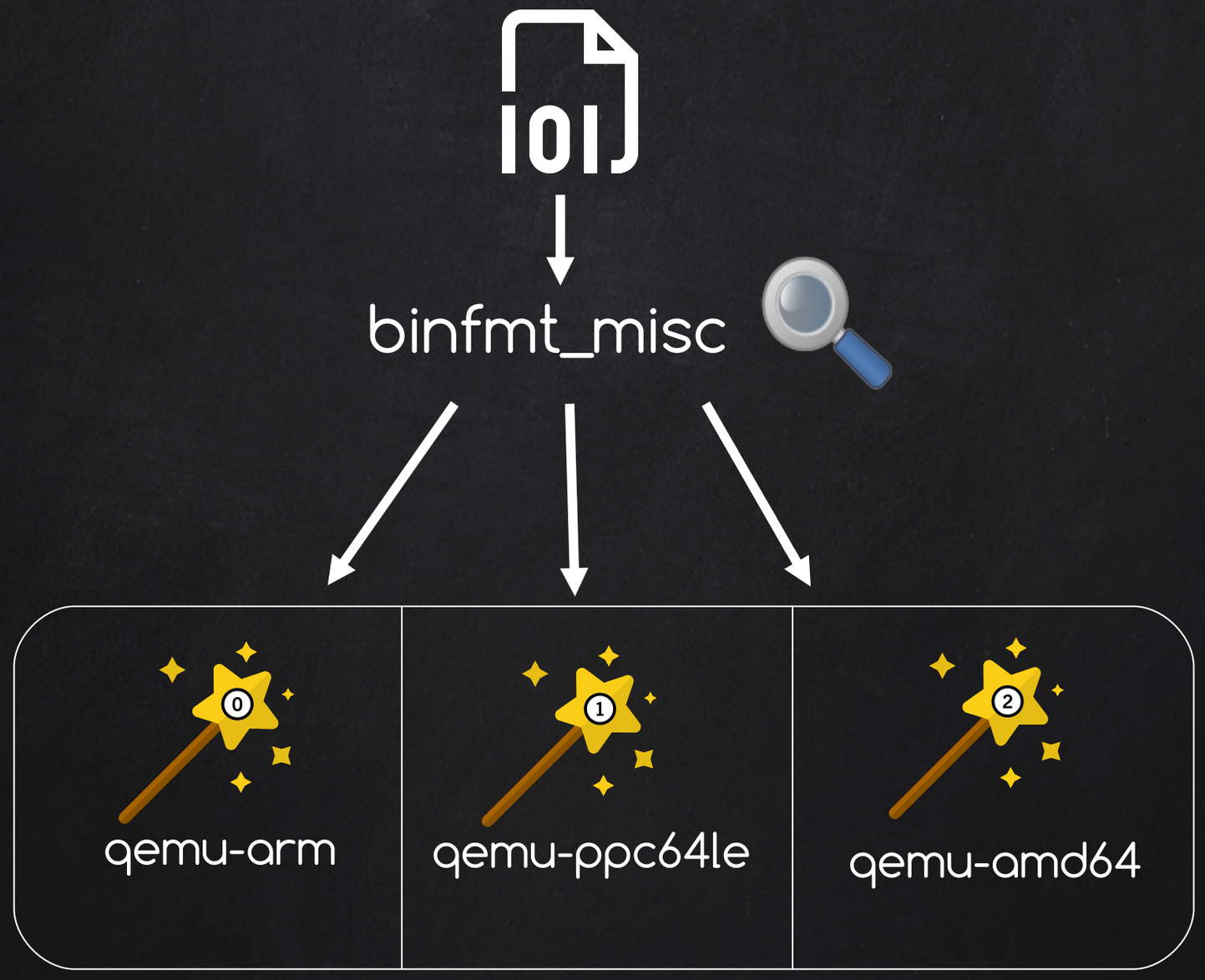
Building binaries and images

- actual hardware
- cross-compiling
- full-system emulation (QEMU)
- user-mode emulation (binfmt_misc)

binfmt_misc

- Allows you to run ELF binaries that weren't compiled on your host architecture
- Maps non-native binaries to arch-specific interpreters (e.g. QEMU)
- Pre-Configured in Docker for Mac!

binfmt_misc



Docker for Mac

```
$ uname -m
x86_64
```

```
# docker run ppc64le image
$ docker run -it --rm ppc64le/busybox:latest uname -m
ppc64le
```

Building using binfmt_misc

```
# build ppc64le image on non-native hardware
$ docker build -v qemu-static-ppc64le -t awesome-app-ppc64le -f Dockerfile.ppc64le .
```

https://github.com/multiarch/qemu-user-static

Cross-compiling

examples

```
# go
$ GOOS=linux GOARCH=arm go build
```

```
# java
$ javac HelloWorld.java
```

Cross-Compile Workflow

```
java-app>
 ├── Dockerfile.build
 ├── Dockerfile.arm
 ├── Dockerfile.amd64
 ├── Dockerfile.ppc64le
 ├── Dockerfile.s390x
 └── HelloWorld.java
```

Cross-Compile Workflow

```
# Dockerfile.build

FROM openjdk

COPY HelloWorld.java HelloWorld.java

RUN javac HelloWorld.java
```

Dev system (amd64):

```
$ docker build -t hwj -f Dockerfile.build .    # repeatable build env

$ docker run --name hello-world-temp hwj       # need container

$ docker cp hello-world-temp:HelloWorld.class . # copy out of container
```

Cross-Compile Workflow

```
# Dockerfile.arm

FROM arm32v7/openjdk

COPY HelloWorld.class HelloWorld.class

CMD ["java", "HelloWorld"]
```

Dev system (amd64):

```
$ docker build -t clnperez/hello-world-arm -f Dockerfile.arm .

$ docker push clnperez/hello-world-arm
```

ARM user:

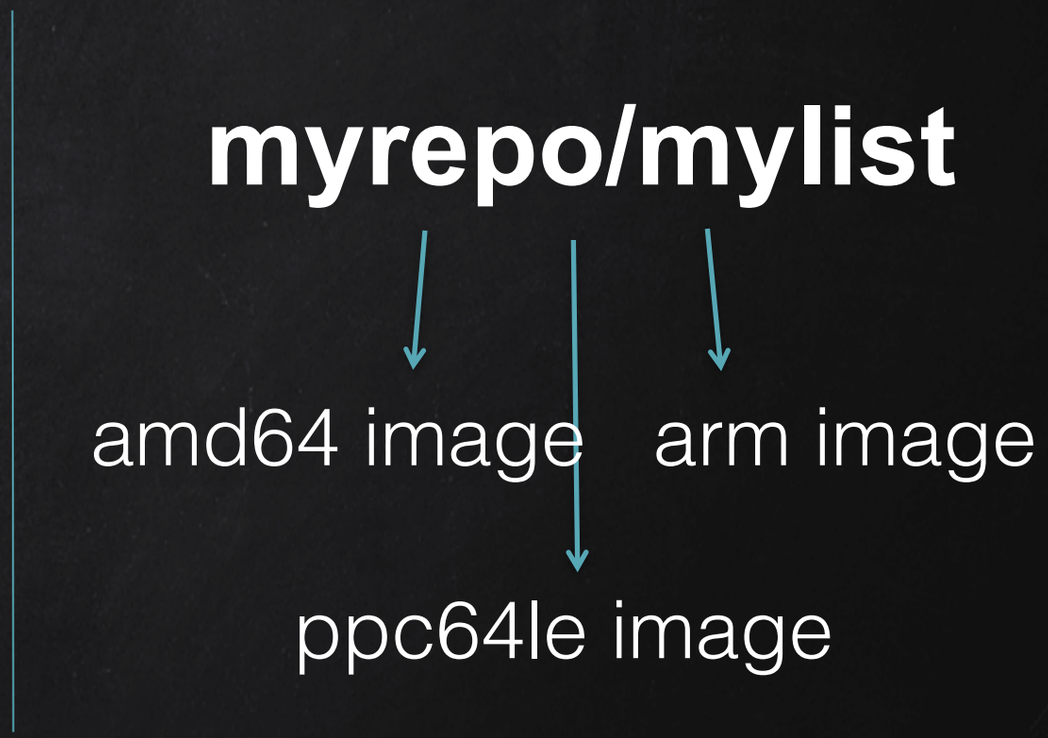
```
$ docker run clnperez/hello-world-arm

Hello World!
```

Manifest lists

Manifest list

- A multi-arch "image" *
- Engine decides which to pull
- Extra image detail
- Use in place of image name anywhere



*but not

`docker manifest` command

- Create & push manifest list to registry
- "Shallow pull" (inspect) of image
- Inspect manifest lists

docker manifest

```
# create interactively using cli
$ docker manifest create ubuntu /
  arm/ubuntu /
  amd64/ubuntu /
  ppc64le/ubuntu /
  s390x/ubuntu

# push to registry
$ docker manifest push ubuntu
```

docker manifest - yaml (soon)

```
# or create using yaml
$ docker manifest push
-f clnperez_hw.yaml
```

```yaml
image: docker.io/clnperez/hello-world:latest
manifests:
  -
    image: docker.io/hello-world:latest
    platform:
      architecture: amd64
      features:
        - sse
      variant: v_a
      os: linux
-
  image: s390x/hello-world:latest
  platform:
    architecture: s390x
    features:
    os: linux
```

docker manifest

```
$ docker manifest inspect ubuntu | grep “arch”

                "architecture": "amd64",
                "architecture": "arm",
                "architecture": "arm64",
                "architecture": "386",
                "architecture": "ppc64le",
                "architecture": "s390x",
```

Project structure

Makefiles, Dockerfiles & build scripts

- Multiple Dockerfiles (e.g. Dockerfile.arm32v7)
- FROM can be passed in as an ARG
- Can use manifest inspect to get arch-specific FROM
- Be mindful of hardcoding arch values

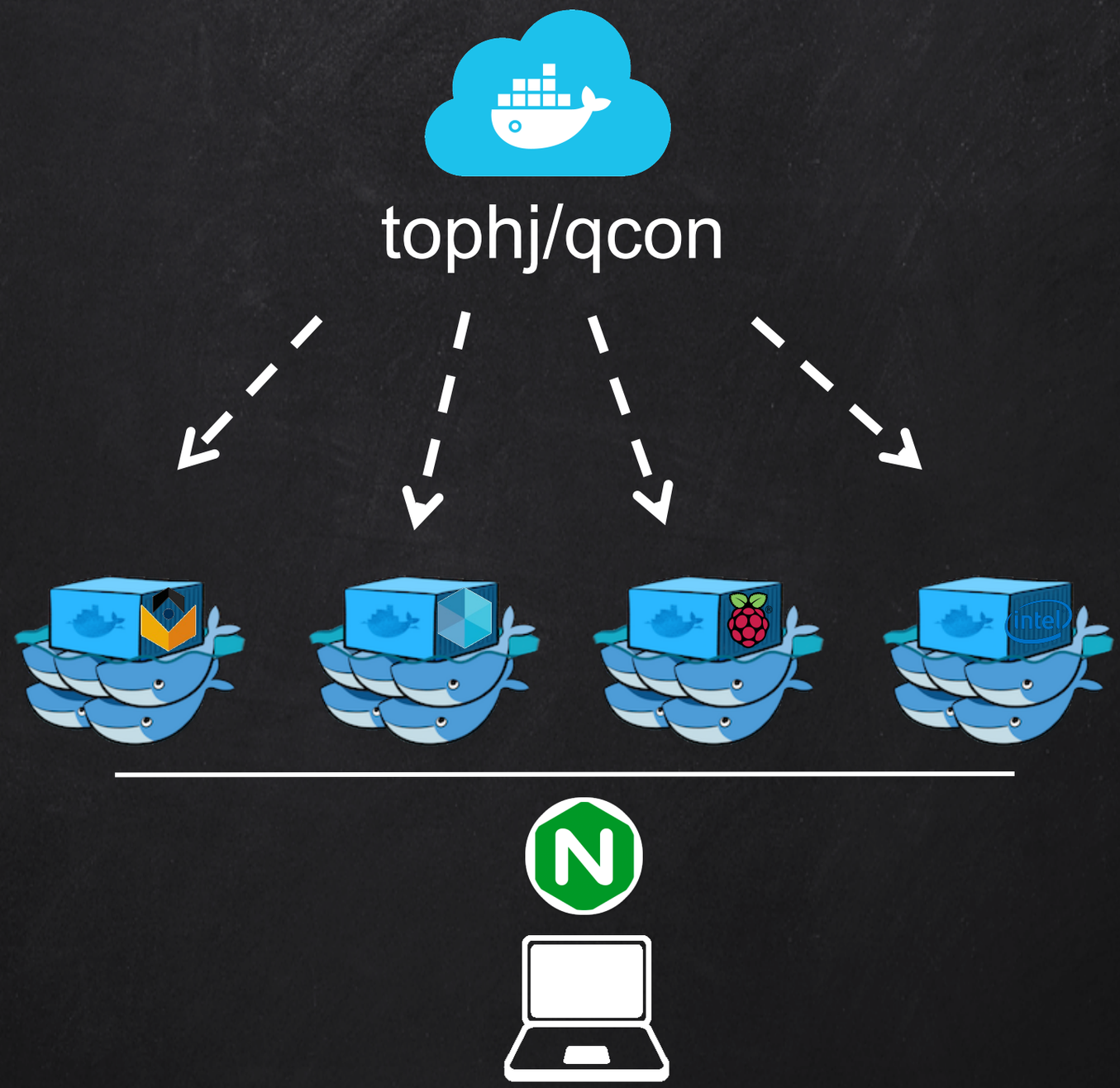
```
├── Dockerfile.amd64
├── Dockerfile.arm
├── Dockerfile.ppc64le
├── Dockerfile.s390x
└── Dockerfile.windows
```

Optimizations

- If you must...
- ifdefs & build constraints

```
zfs.go:// +build linux freebsd solaris
zfs_unsupported.go:// +build !linux,!freebsd,!solaris
```

Putting it all together


tophj/qcon

tophj/qcon

- tophj/qcon:arm64
- tophj/qcon:amd64
- tophj/qcon:ppc64le
- tophj/qcon:s390x

docker swarm demo

```
├── Dockerfile.armhf
├── Dockerfile.ppc64le
├── Dockerfile.s390x
├── Dockerfile.x86_64
├── img
│   ├── captain_logo.png
│   ├── captain.png
│   ├── christy_logo.png
│   ├── christy.png
│   ├── pink.png
│   └── tophj.png
└── server.go
```

docker swarm demo

```
# create our tophj/demo manifest list which points to our
# architecture specific images
$ docker manifest create tophj/demo tophj/x86_64-demo \
tophj/armhf-demo tophj/ppc64le-demo tophj/s390x-demo
```

docker swarm demo

```
# annotate to change armhf to arm
$ docker manifest annotate tophj/demo tophj/armhf-demo \
--os linux --arch arm
```

```
# finally push the manifest list
$ docker manifest push tophj/demo
```

docker swarm demo

```
$ docker node ls
ID                            HOSTNAME            STATUS   AVAILABILITY   MANAGER
STATUS
3nisms9qfi27ko0683bylxyud     s390                Ready    Active
7jwo7d5braat1l6n63j8xfue6 *   x86_64              Ready    Active         Leader
n5tqx2yk77123sjiapqwmu14e     armhf               Ready    Active
u30pwzlt5hthwbbsdok8nxyh8     ppc64le             Ready    Active
```

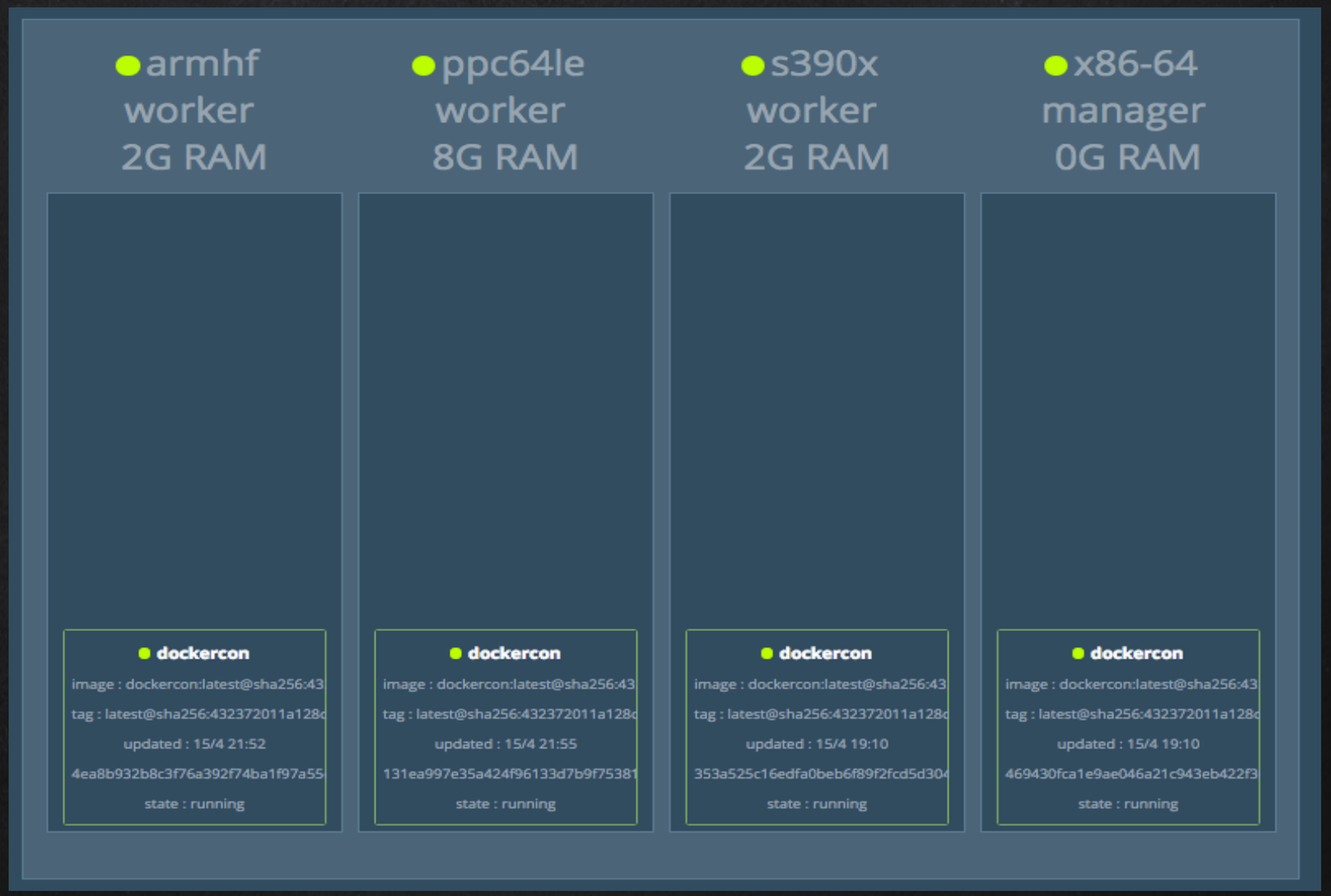
docker swarm demo

```
# start the swarm service using the multi-arch image name
$ docker service create  --mode global --name dockercon -p 8082:8080 tophj/demo

# start a simple load-balancer for fun
$ docker run -itd -p 80:81 --name nginx -v /christy/nginx:/etc/nginx nginx

# visit the IP of your load balancer in your browser
# be sure to refresh for multi-arch fun
```

docker swarm demo



References & Legal

- Server image: https://c1.staticflickr.com/4/3519/3462607995_150a6b2624_b.jpg
- HtcpcpTeapot image: https://commons.wikimedia.org/wiki/File:Htcpcp_teapot.jpg
- Raspberry Pi is a trademark of the Raspberry Pi Foundation.
- https://github.com/dockersamples/docker-swarm-visualizer
- ARMHF VM from Scaleway
- X86 VM from DigitalOcean
- Gophers: http://gopherize.me

Thank You!

Go forth and multi-arch!

@qcon #qcon