
KubeCon
CloudNativeCon
North America 2024

KubeCon | CloudNativeCon

North America 2024

ARM-Wrestling

Overcoming CPU Migration Challenges to Reduce Costs

Eric Mountain & Laurent Bernaille

About Datadog

Over 700 integrations
Over 5,000 employees
Over 27,000 customers
Millions of hosts reporting
Trillions of data points per day

10,000s hosts in our infra
Everything runs in k8s
Dozens of k8s clusters
Multi-cloud
Rapid growth

Why we’re here today

- Why Datadog wanted to adopt ARM64
- How we bootstrapped the initiative
- How we scaled the migration
- Our strategy going forward

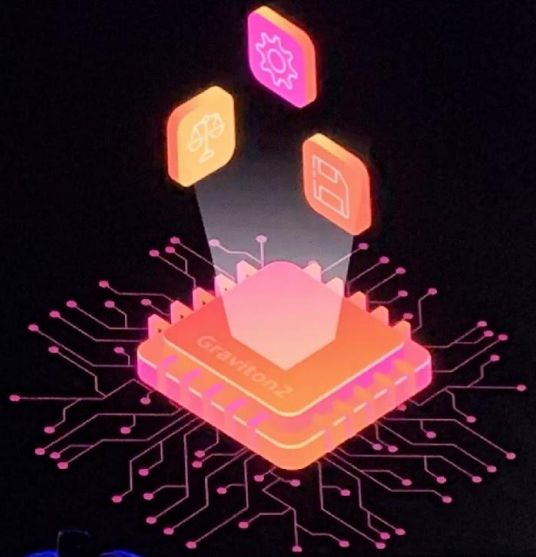
So... Why ARM Anyway?

POWERED BY AWS GRAVITON2 PROCESSORS

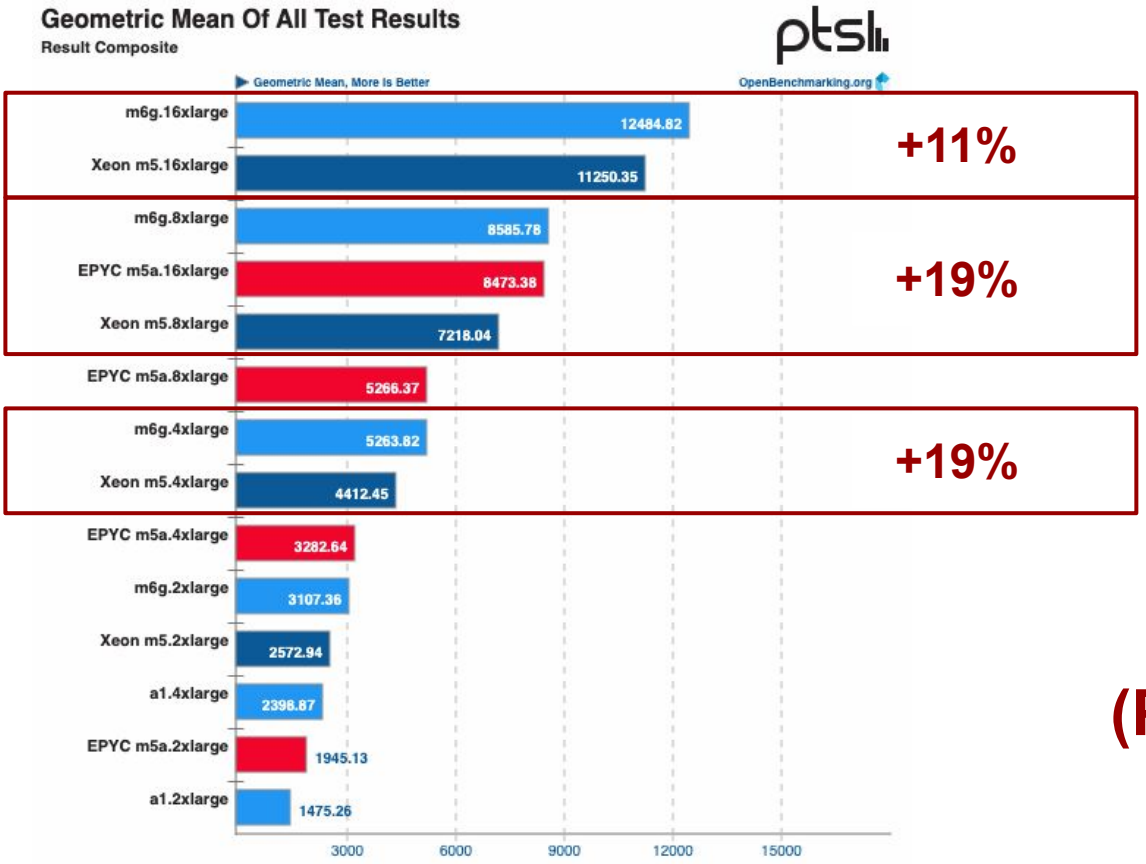
M6g, R6g, C6g instances for EC2

New generation of Arm-based instances powered by AWS Graviton2 processors offer 40% better price/performance than current x86-based instances

M6g	R6g	C6g
PREVIEW TODAY	COMING SOON	COMING SOON



Rationale: Benchmarks



10-20% faster
(Phoronix, May 2020)

https://www.phoronix.com/review/amazon-graviton2-benchmarks/

Rationale: Pricing

Region: US East (N. Virginia) | Pricing Unit: Instance | Cost: Monthly | Reserved: 3-year convertible - No Upfront | Visible: Columns | Compare | Clear Filters

Name	API Name	Instance Memory	vCPUs	On Demand	Linux Reserved cost	
Filter...	^[mcr][56]g?.4x	Min Mem: 0	Min vCPUs: 0	Filter...	Filter...	
C5 High-CPU Quadruple Extra Large	c5.4xlarge	32.0 GiB	16 vCPUs	$496.4000 monthly	$239.4400 monthly	-20%
C6G Quadruple Extra Large	c6g.4xlarge	32.0 GiB	16 vCPUs	$397.1200 monthly	$191.7710 monthly	
M5 General Purpose Quadruple Extra Large	m5.4xlarge	64.0 GiB	16 vCPUs	$560.6400 monthly	$283.9700 monthly	-18%
M6G Quadruple Extra Large	m6g.4xlarge	64.0 GiB	16 vCPUs	$449.6800 monthly	$233.3810 monthly	
R5 Quadruple Extra Large	r5.4xlarge	128.0 GiB	16 vCPUs	$735.8400 monthly	$383.9800 monthly	-20%
R6G Quadruple Extra Large	r6g.4xlarge	128.0 GiB	16 vCPUs	$588.6720 monthly	$307.7680 monthly	

~20% cheaper

Rationale: Quick Internal Benchmarks

Promising benchmarks on etcd, Redis, Kafka

		Monthly costs		
		$165.71	$133.73	
etcd metric	**Unit**	**m5d.2xlarge**	**m6gd.2xlarge**	**Improvement**
1k client linear read throughput	req/s	24,512	35,676	**45%**
1k client linear total time	seconds	4.0	2.8	**31%**
1k client linear p99	millisecs	90.7	68.0	**25%**

20% cheaper… and 25+% faster

Exec rationale and **sponsorship**

- We need to control our costs and ARM is cheaper
- ARM footprint growing fast (AWS, Apple)
- Effort mostly on a few platform teams (compute & build)
- Easy adoption criteria
  - Works / doesn't work
  - Performs better / worse ⇒ cost has the final word
- First applications will iron out most teething issues

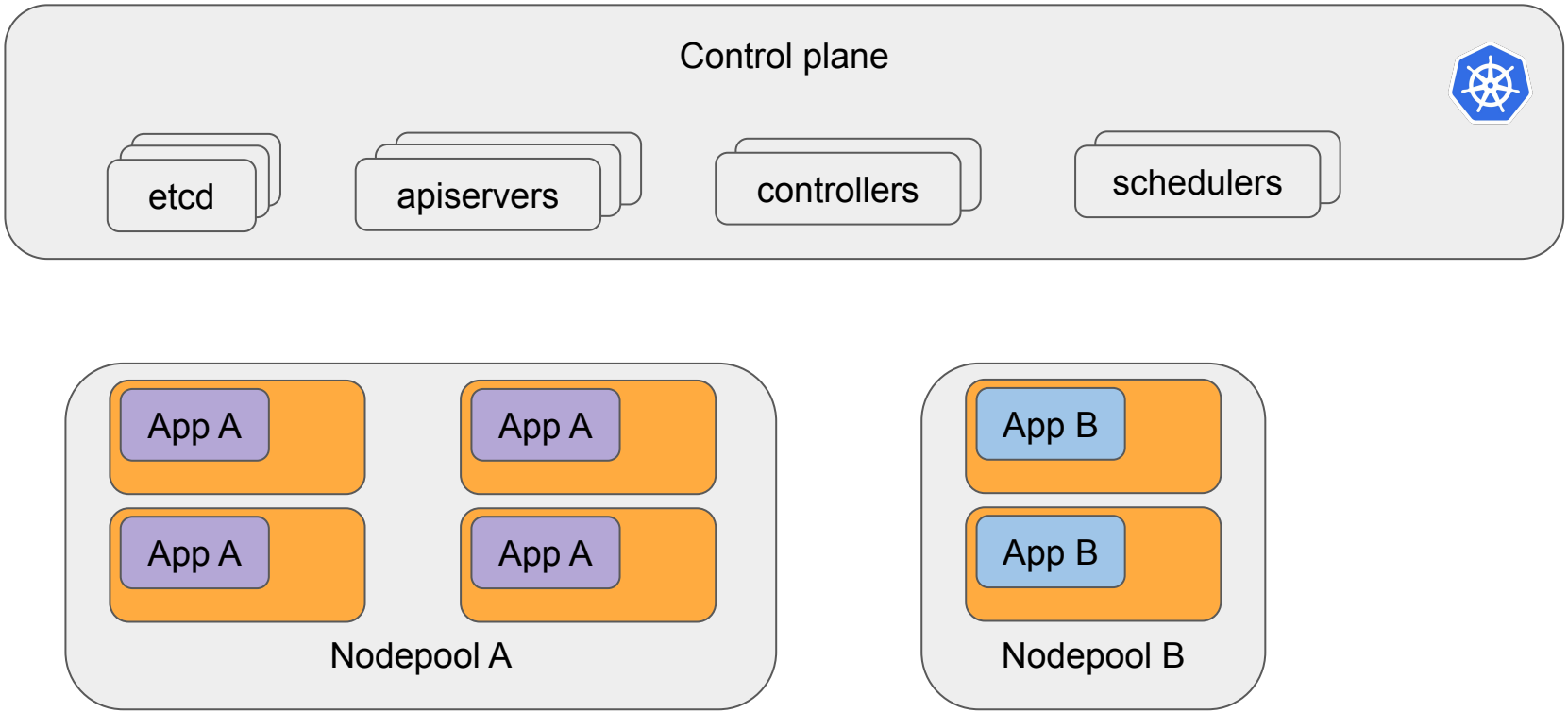
Alexis Lê-Quôc, CTO, Q1 2021

Starting point

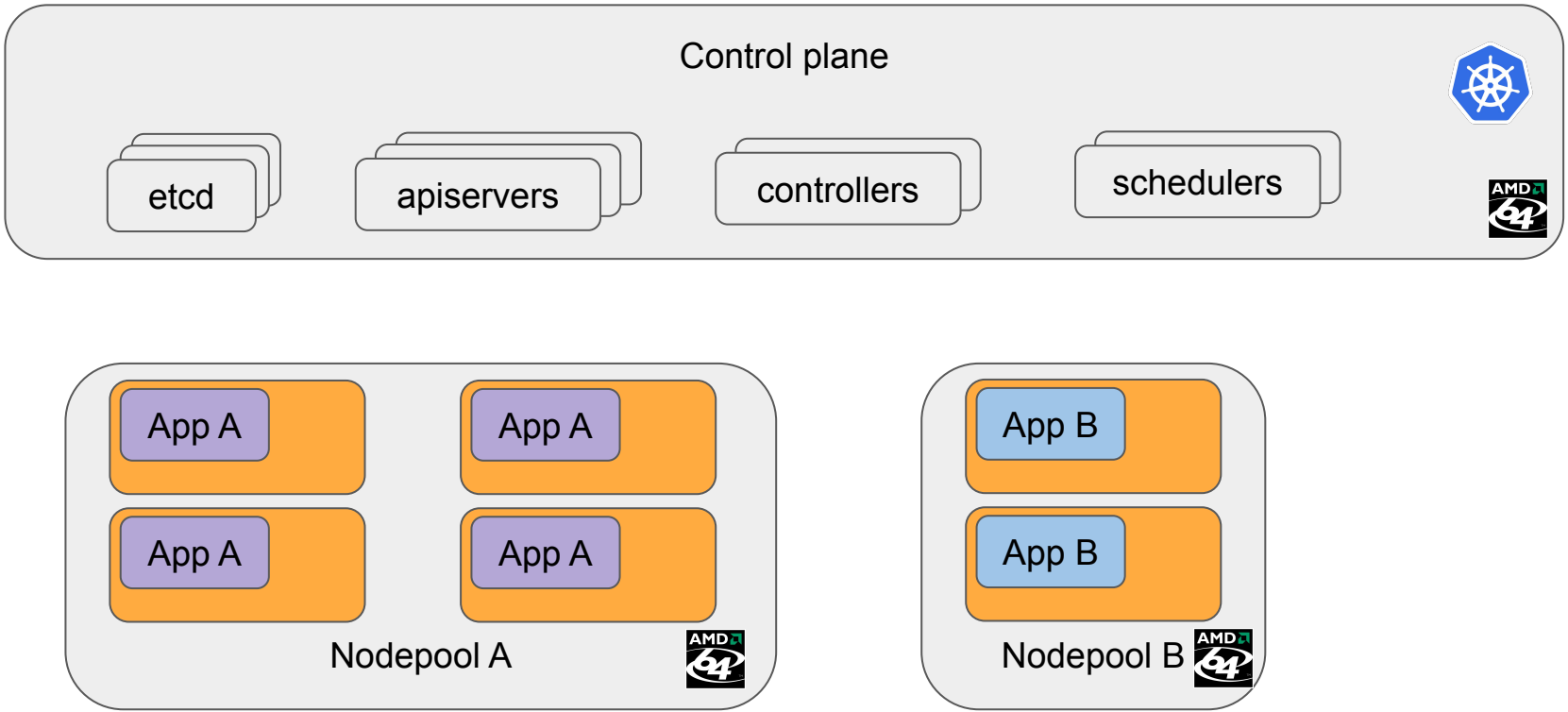
- Knowns
  - Graviton 2 are 20% cheaper than equivalent m5/c5
  - Some companies use ARM extensively already
- Unknowns
  - Performance for our applications
  - Requirements for x86 specifics
- Success == handling same workload at lower cost

Day 0: Bootstrapping ARM

Our clusters

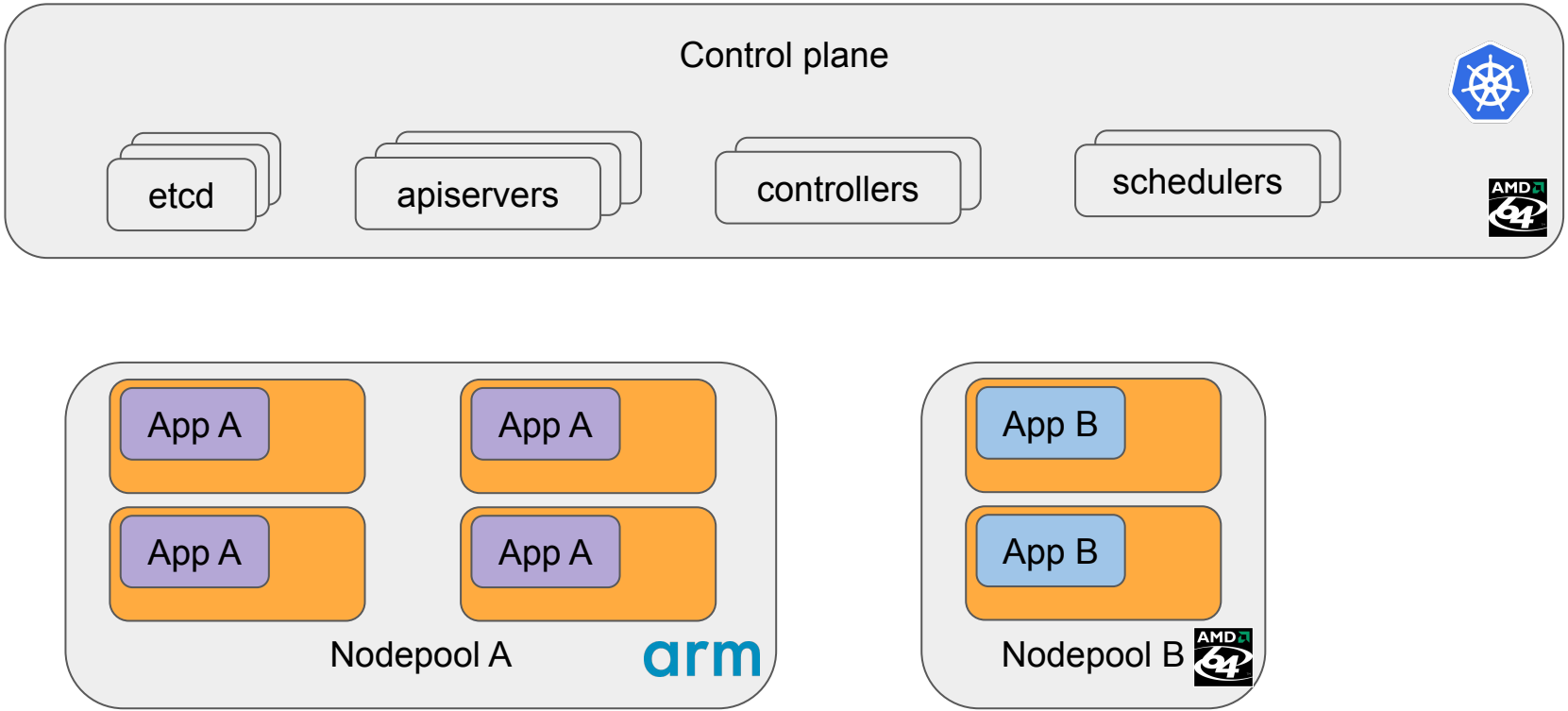


Our clusters

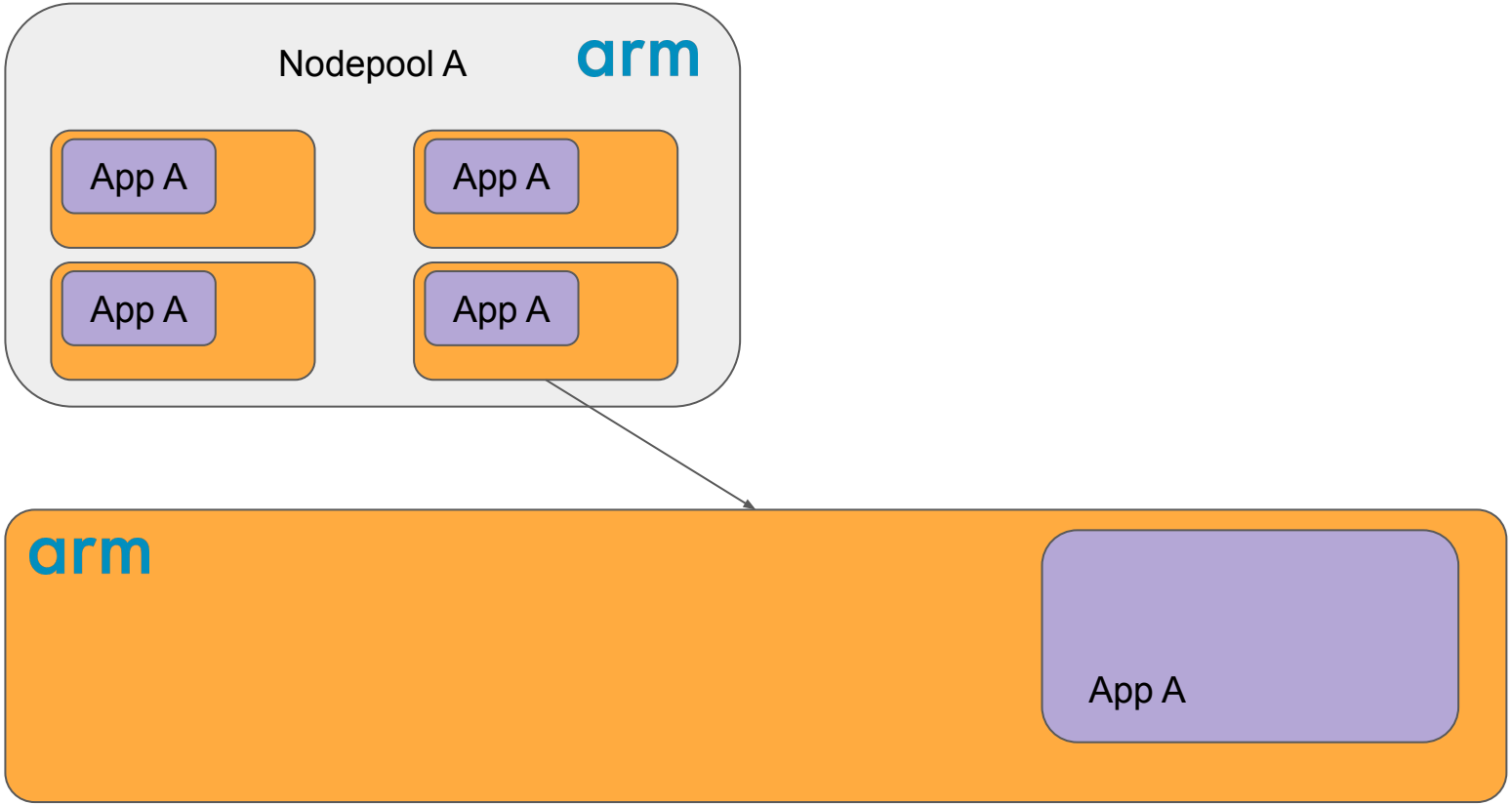


For this talk: x86 / x86_64 / amd64 are the same

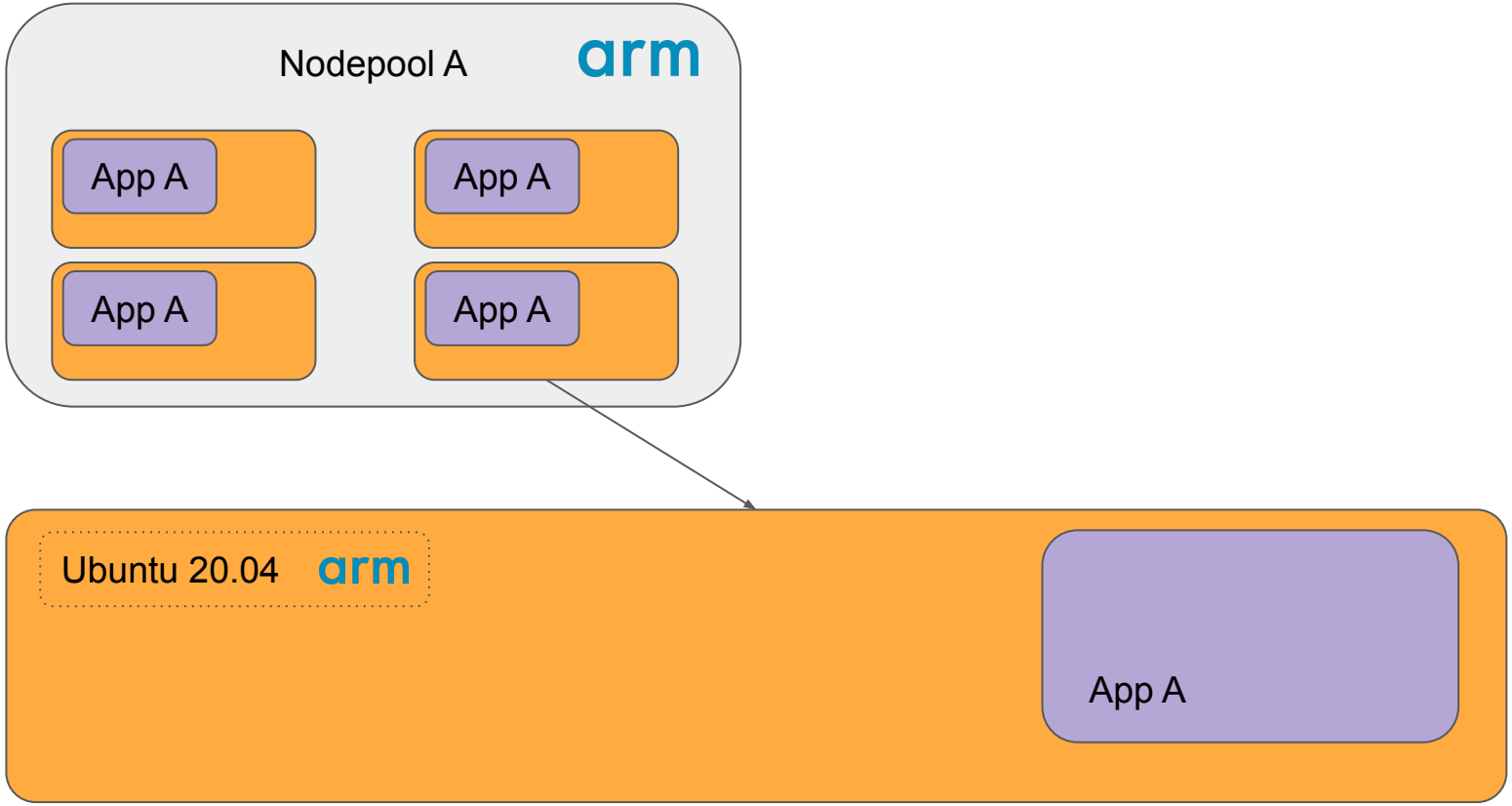
What we want


Control plane
etcd
apiservers
controllers
schedulers
AMD64
App A
App A
App A
App A
Nodepool A
arm
App B
App B
Nodepool B
AMD64

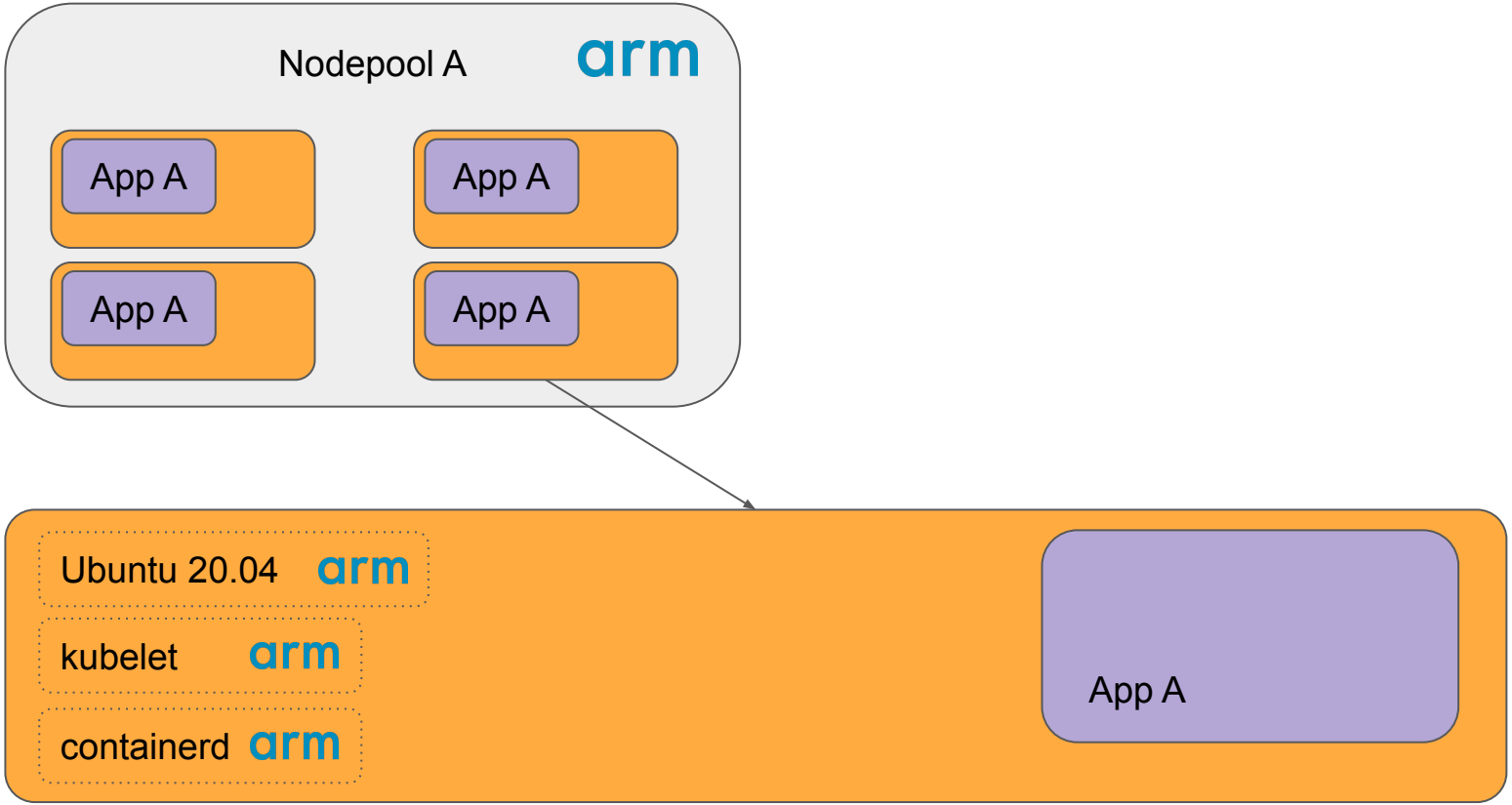
What does it require?



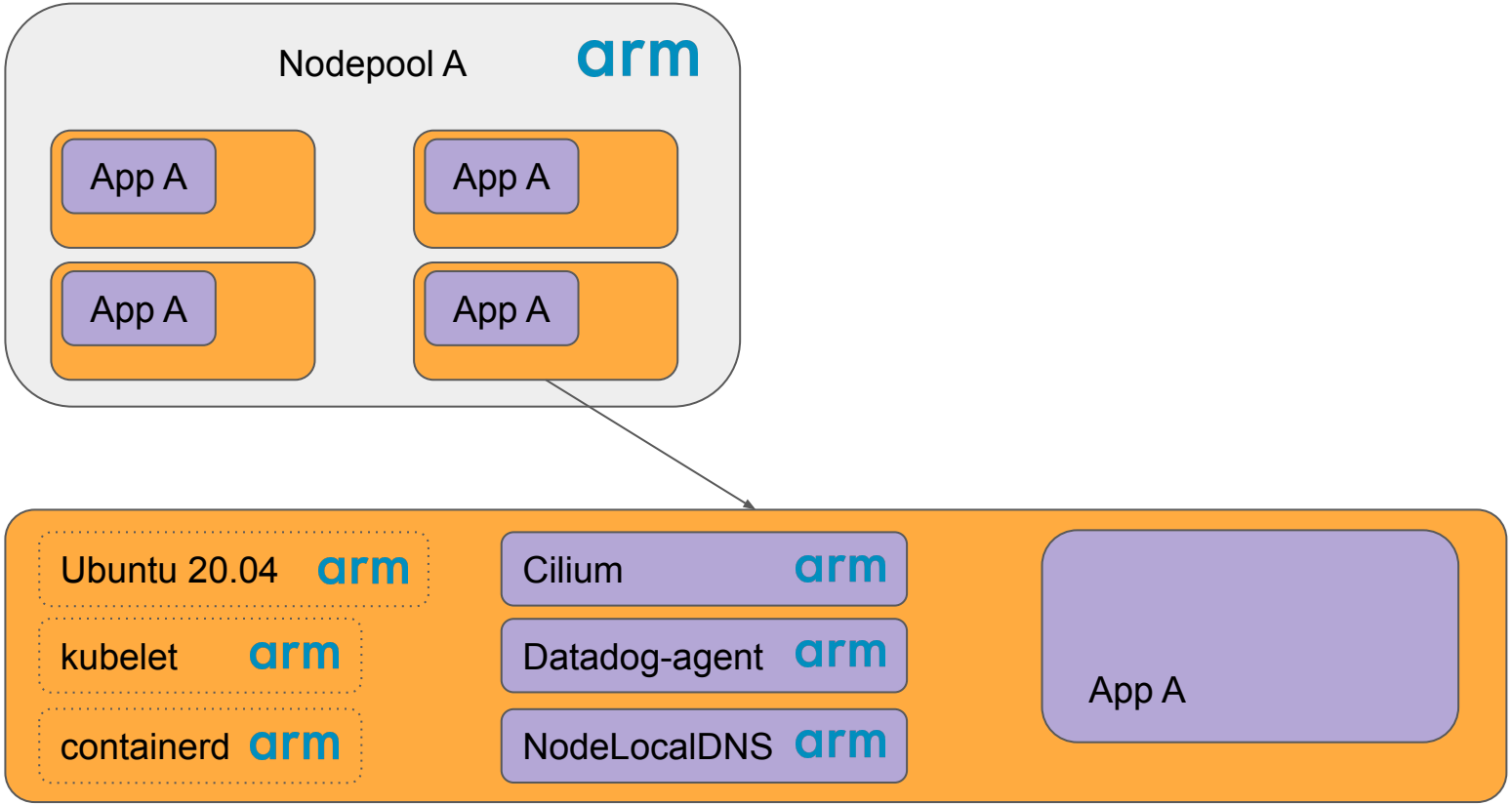
What does it require?



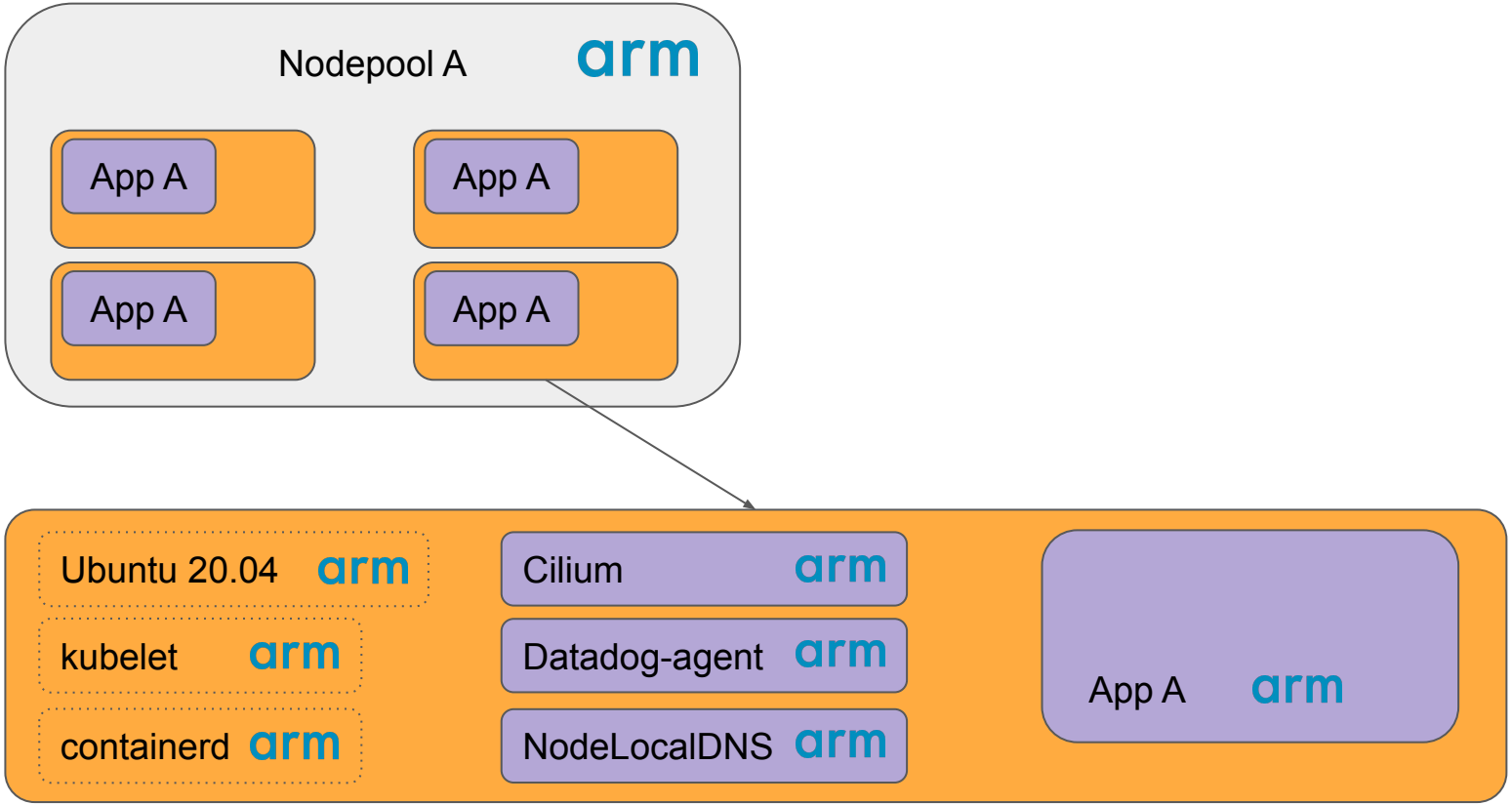
What does it require?



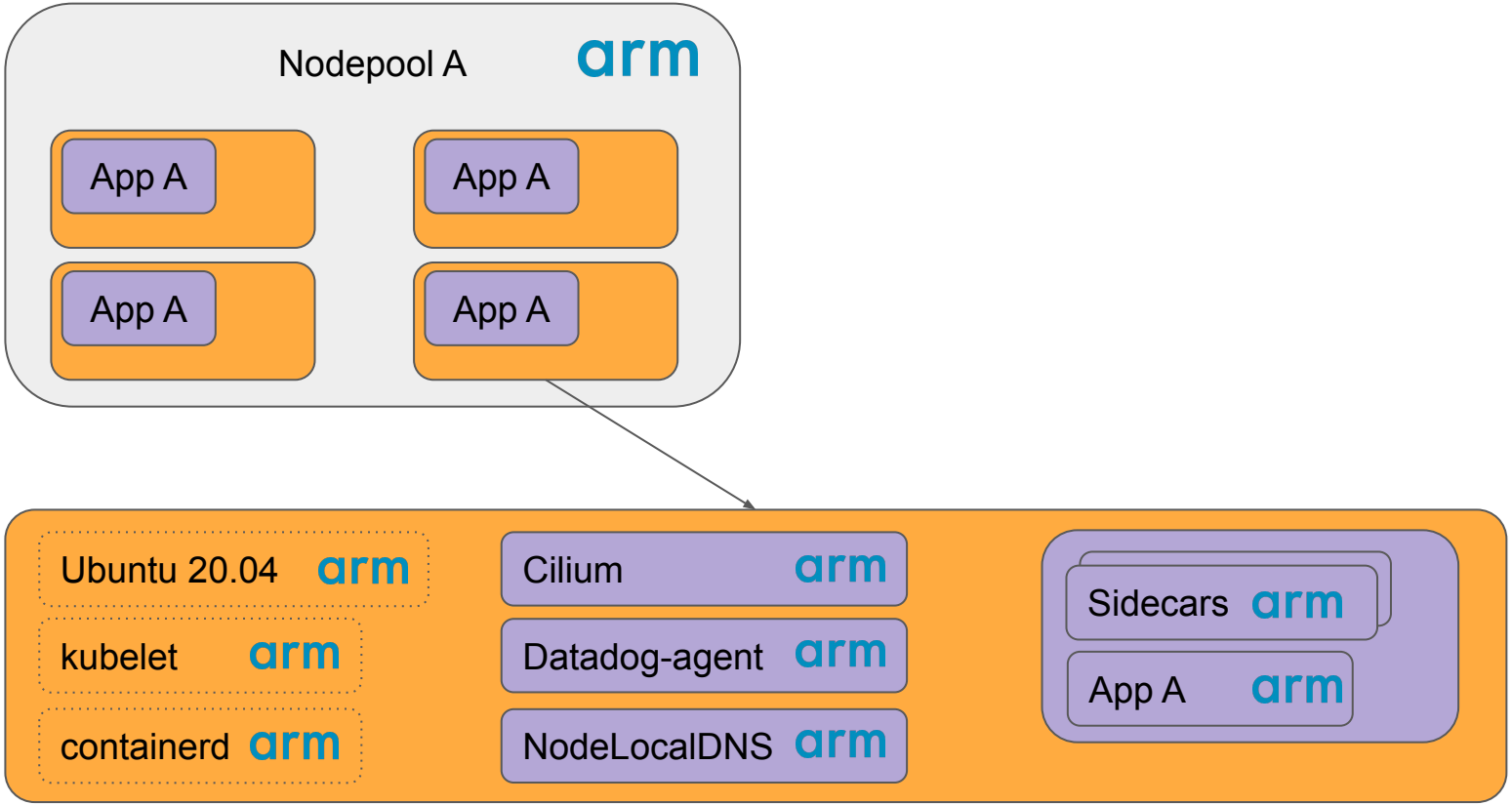
What does it require?



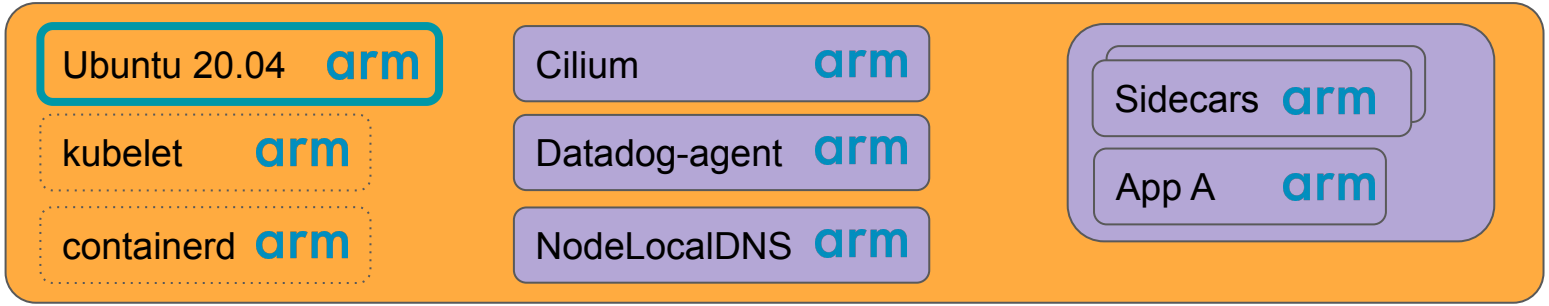
What does it require?



What does it require?



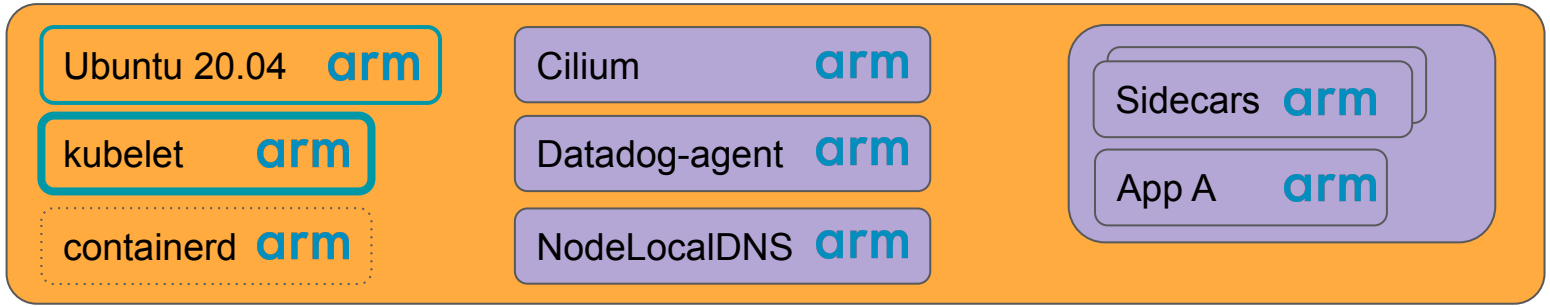
Make our platform ARM ready: AMI



2 packer/amazon-ebs.json

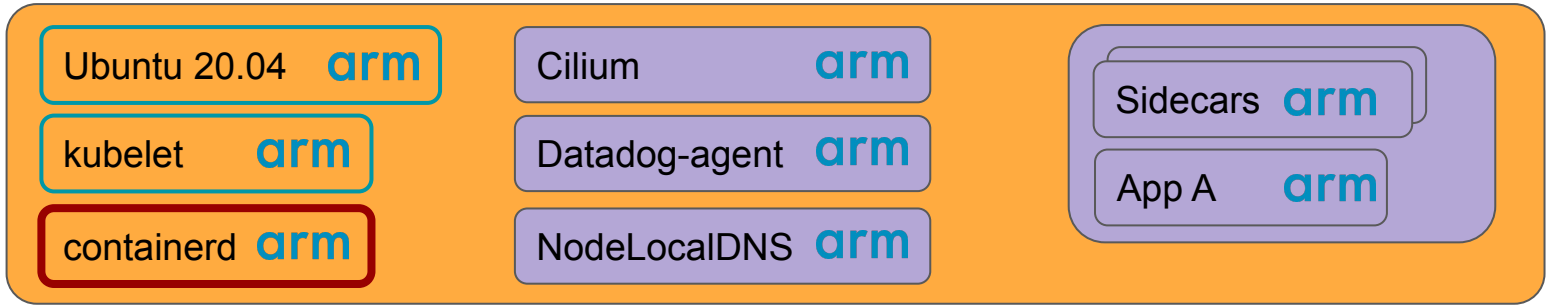
```
@@ -39,7 +39,7 @@
39              "source_ami_filter": {
40                "filters": {
41                  "virtualization-type": "hvm",
-                 "name": "ubuntu/images/hvm-ssd/ubuntu-{{ user `source_ami_basename` }}-amd64-server-*",
42  +               "name": "ubuntu/images/hvm-ssd/ubuntu-{{ user `source_ami_basename` }}-{{ user `arch` }}-server-*",
43                  "root-device-type": "ebs"
44                },
```

Make our platform ARM ready: kube binaries




```
    v1.19.7:
-     url: "https://github.com/DataDog/kubernetes/releases/download/v1.19.7-dd.1/kubernetes-server-linux-amd64.tar.gz"
-     cksum: "sha256:3602827d318f6dcb6aed3c2278da4b2d0112e619fccd4bf8007e597b6549ac44"
+     url: "https://github.com/DataDog/kubernetes/releases/download/v1.19.7-dd.1/kubernetes-server-linux-{{ arch }}.tar.gz"
+     cksum:
+       amd64: "sha256:3602827d318f6dcb6aed3c2278da4b2d0112e619fccd4bf8007e597b6549ac44"
+       arm64: "sha256:8580ea702ca46fd941c186a8b520a0a1b68b1aa38edd7fb09ce691b8f3dd8c2f"
```

Make our platform ARM ready: kube binaries

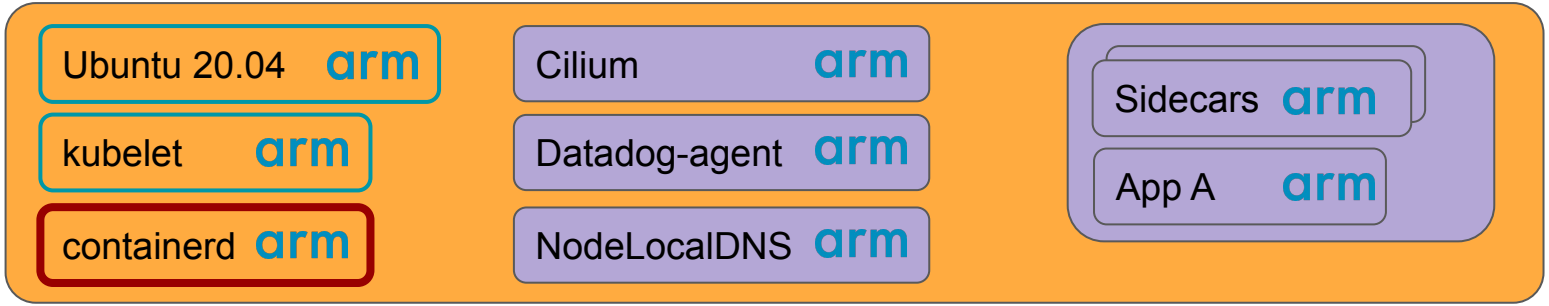


39 ansible/roles/node/defaults/main.yml

Viewed

```
...  @@ -1,25 +1,36 @@
 1   containerd_version: 1.2.14
-  - containerd_url: "https://github.com/containerd/containerd/releases/download/v{{ containerd_version }}/containerd-{{ containerd_version }}.linux-amd64.tar.gz"
-  - containerd_url_checksum: "sha256:307de3abdd16252746269cd8abd03a944a8c51e692711117efa0ecb6a13939ca"
 2  + containerd_url: "https://github.com/containerd/containerd/releases/download/v{{ containerd_version }}/containerd-{{ containerd_version }}.linux-{{ arch }}.tar.gz"
 3  + containerd_checksum:
 4  +   amd64: "sha256:307de3abdd16252746269cd8abd03a944a8c51e692711117efa0ecb6a13939ca"
 5  +   arm64: "Not available from GitHub releases"
```

Make our platform ARM ready: kube binaries



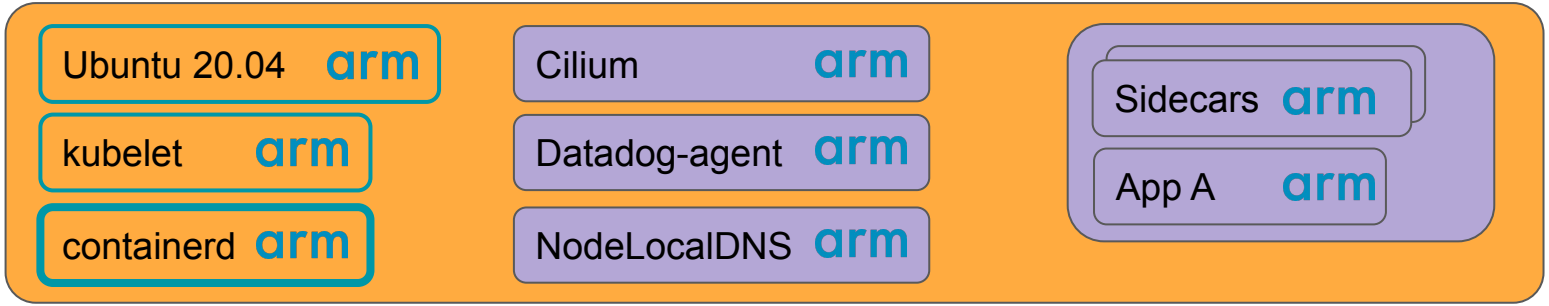
39 ansible/roles/node/defaults/main.yml

Viewed

```
...   @@ -1,25 +1,36 @@
1     containerd_version: 1.2.14
    - containerd_url: "https://github.com/containerd/containerd/releases/download/v{{ containerd_version }}/containerd-{{ containerd_version }}.linux-amd64.tar.gz"
    - containerd_url_checksum: "sha256:307de3abdd16252746269cd8abd03a944a8c51e692711117efa0ecb6a13939ca"
2   + containerd_url: "https://github.com/containerd/containerd/releases/download/v{{ containerd_version }}/containerd-{{ containerd_version }}.linux-{{ arch }}.tar.gz"
3   + containerd_checksum:
4   +   amd64: "sha256:307de3abdd16252746269cd8abd03a944a8c51e692711117efa0ecb6a13939ca"
5   +   arm64: "Not available from GitHub releases"
```

No ARM release for containerd

Make our platform ARM ready: kube binaries

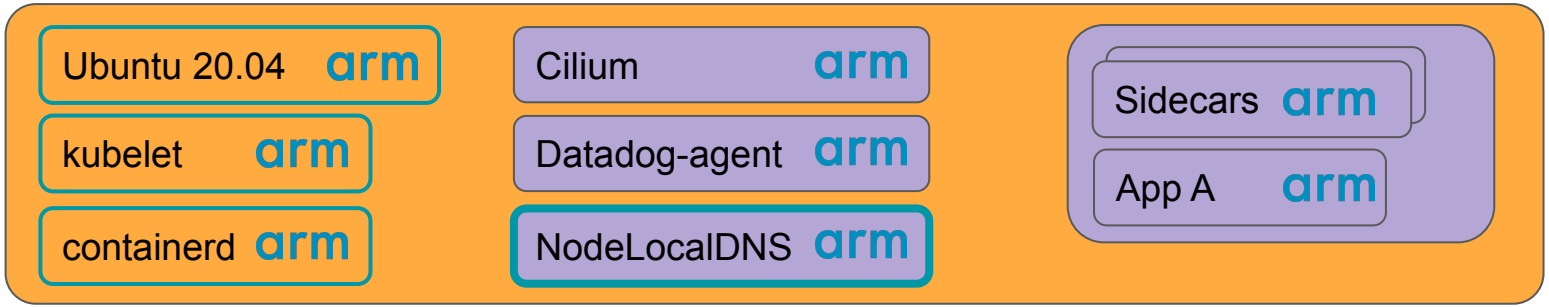



```
- include_tasks: containerd.yml
- include_tasks: containerd-{{ arch }}.yml
- include_tasks: containerd-common.yml
```

13 ■■■■■ ansible/roles/node/tasks/containerd-arm64.yml

```
...   @@ -0,0 +1,13 @@
1   + - name: "Add Docker's official GPG key"
2   +   ansible.builtin.apt_key:
3   +     url: https://download.docker.com/linux/ubuntu/gpg
4   +     state: present
5   +
6   + - name: Add Docker repository to source list
7   +   ansible.builtin.apt_repository:
8   +     repo: deb [arch=arm64] https://download.docker.com/linux/ubuntu {{ ansible_lsb.codename }} stable
9   +     state: present
10  +
11  + - name: Install containerd.io
12  +   apt:
13  +     name: containerd.io
```

Make our platform ARM ready: daemonsets



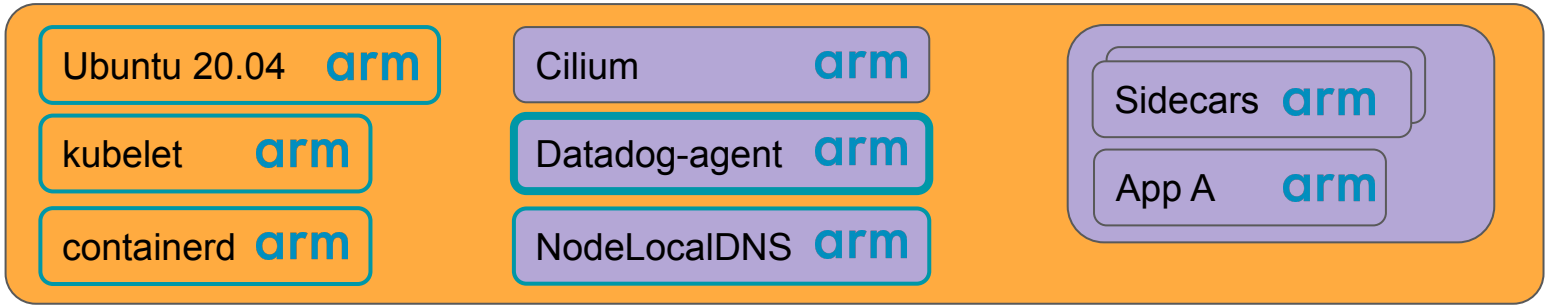
k8s-dns-node-cache already provides an ARM image

From the very beginning of https://github.com/kubernetes/dns/

PR #1, 2016

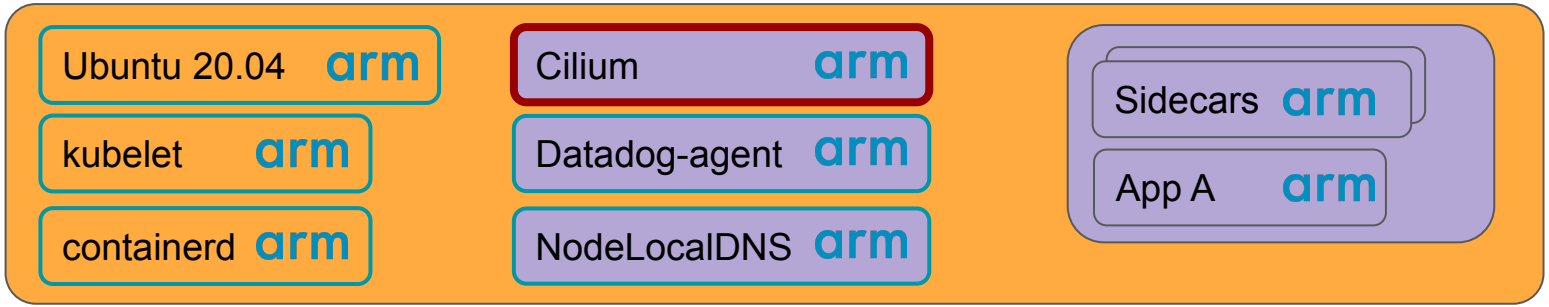
```
20    + ALL_ARCH := amd64 arm arm64 ppc64le
```

Make our platform ARM ready: daemonsets



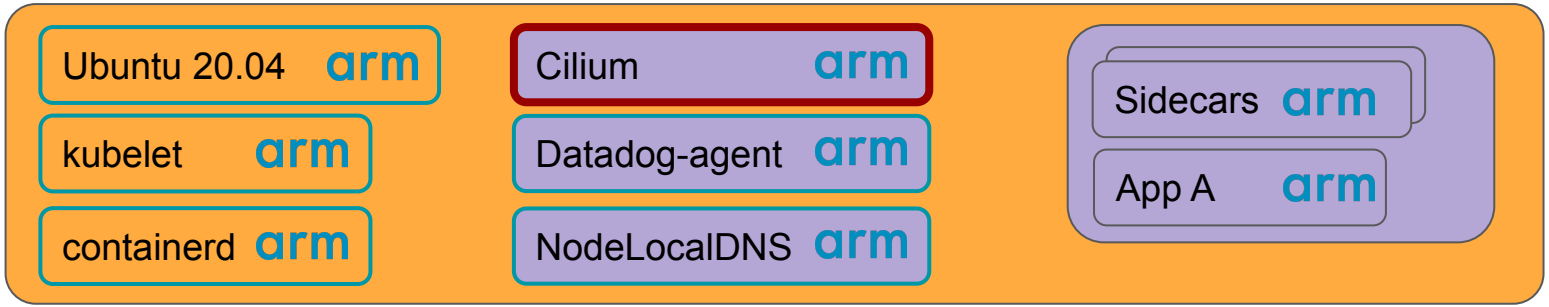
The public Datadog image supports ARM since 2019 (6.15 / 7.16)

Make our platform ARM ready: daemonsets



We run Cilium 1.8, which is x86 only

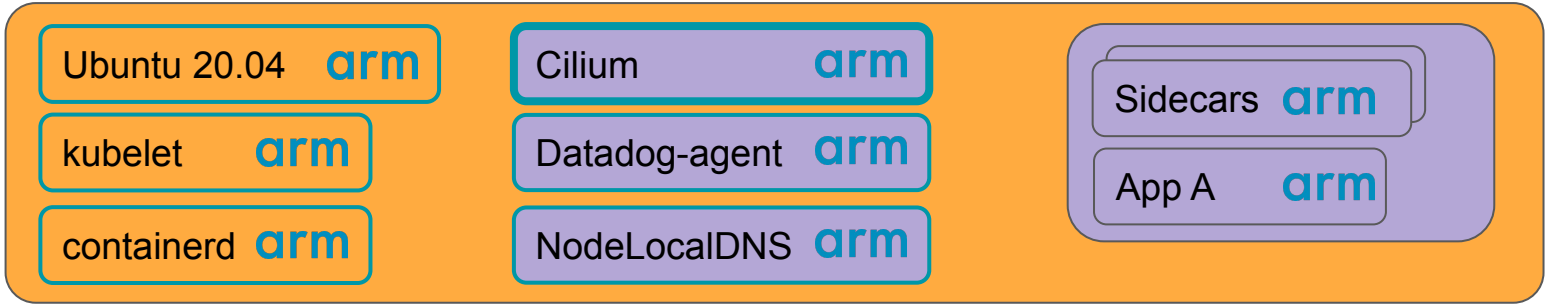
Make our platform ARM ready: daemonsets



We run Cilium 1.8, which is x86 only

Cilium 1.9-dev has ARM support

Make our platform ARM ready: daemonsets



We run Cilium 1.8, which is x86 only

Cilium 1.9-dev has ARM support

build: Experimental multi-platform images #12013

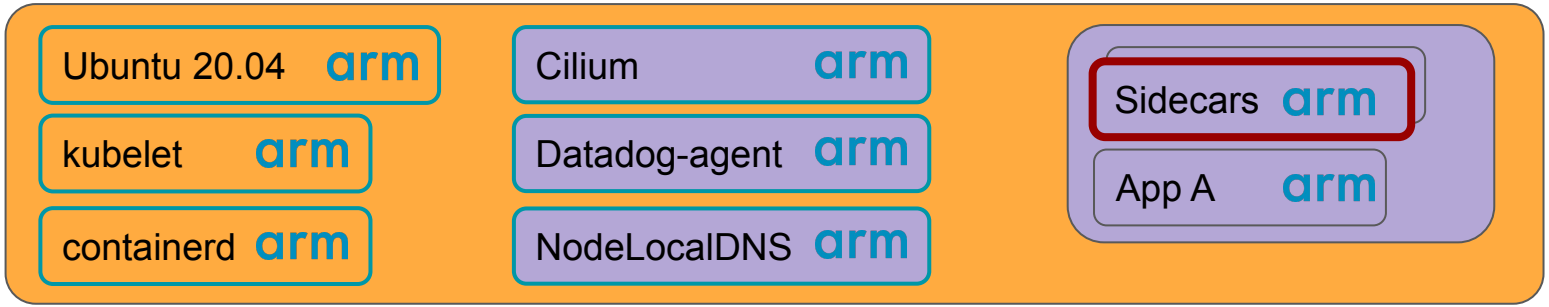
Merged aanm merged 11 commits into master from multi-platform-images

Conversation 116 Commits 11 Checks 0 Files changed 33

Commits on Jul 3, 2020

Many thanks to Hemanth Malla for the huge backport!

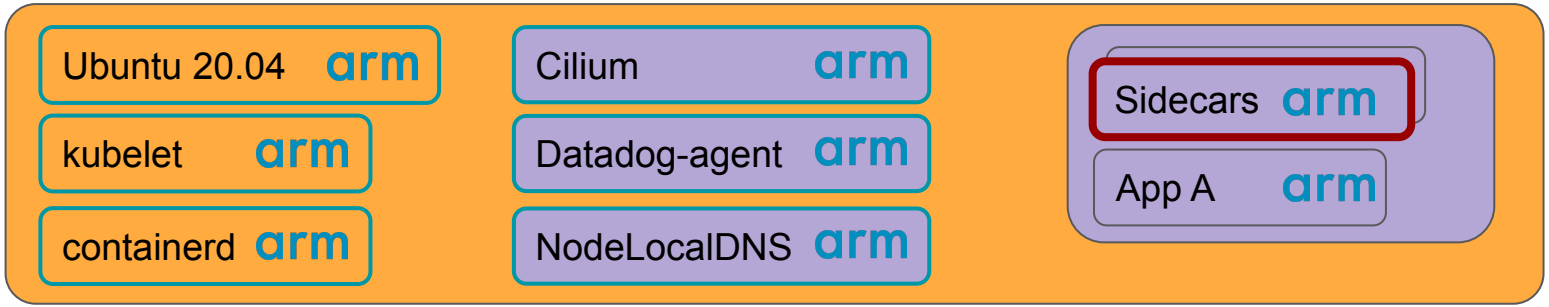
Make our platform ARM ready: daemonsets



We use consul-template to retrieve secrets from vault

No ARM image upstream but an ARM binary

Make our platform ARM ready: Sidecars

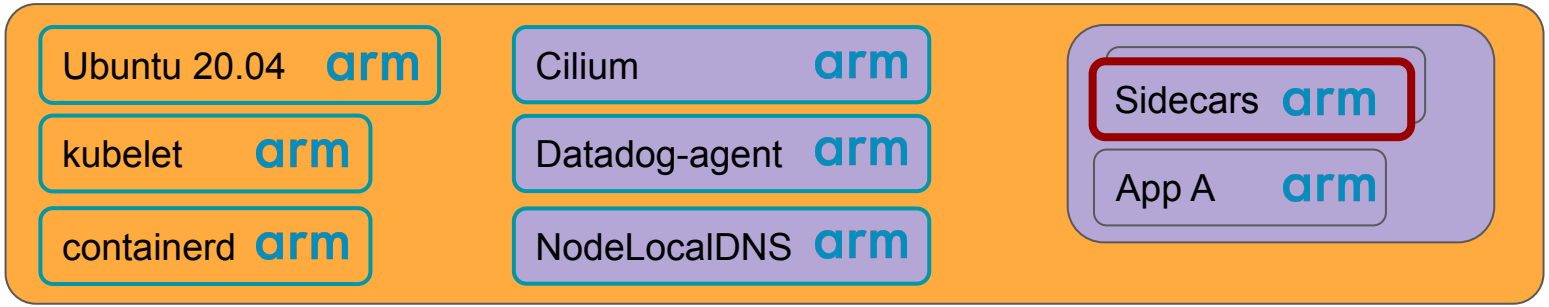


We use consul-template to retrieve secrets from vault

No ARM image upstream but an ARM binary => reuse x86 Dockerfile

```
FROM alpine
RUN curl -OL …/consul-template_${VERSION}_linux_${ARCH}.tgz && \
    tar -xzf … && mv …
```

Make our platform ARM ready: Sidecars

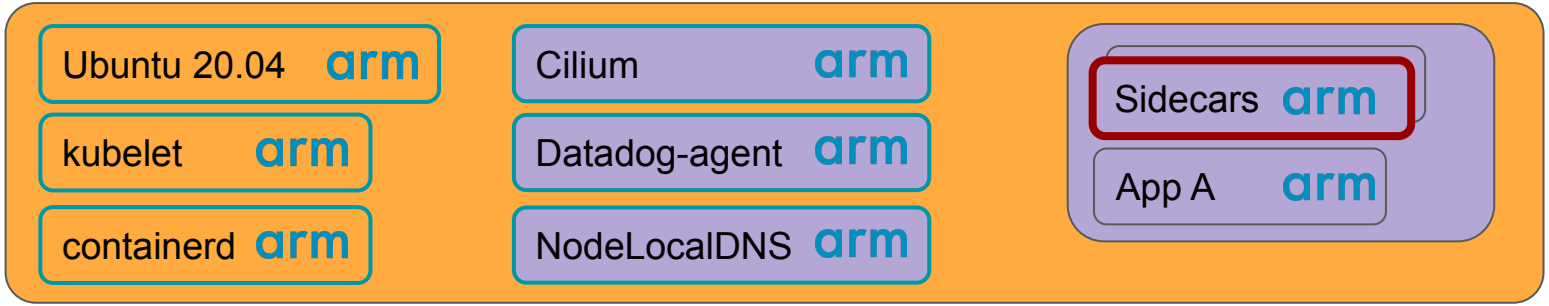


We use consul-template to retrieve secrets from vault

No ARM image upstream but an ARM binary => reuse x86 Dockerfile

ARM image does not work

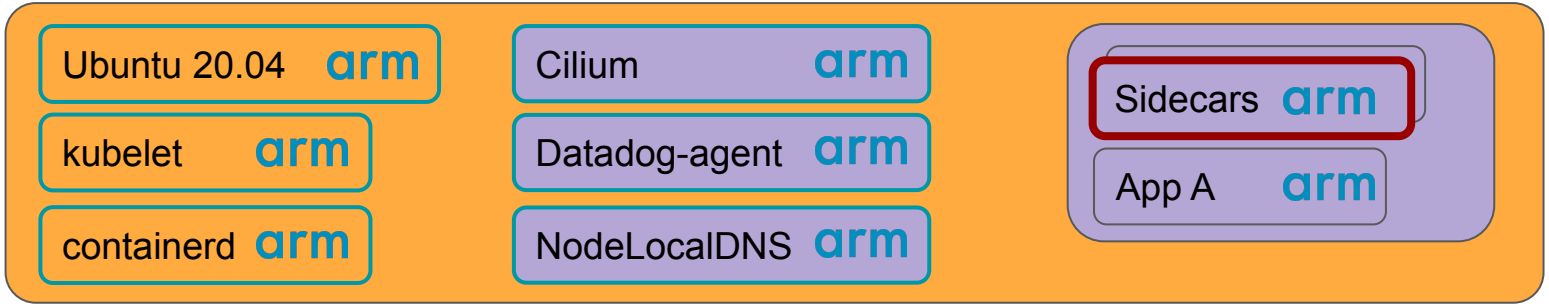
Make our platform ARM ready: Sidecars



ARM image does not work, let's debug

```
# ./consul-template-arm64
/bin/sh: ./consul-template-arm64: not found
```

Make our platform ARM ready: Sidecars

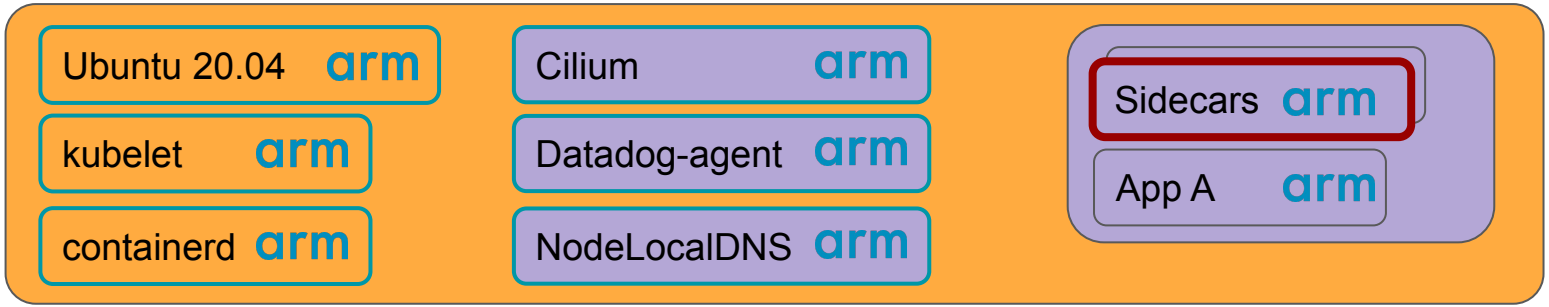


ARM image does not work => linked with libc **but alpine image**

```
# ./consul-template-arm64
/bin/sh: ./consul-template-arm64: not found

# ldd consul-template-arm64
    /lib/ld-linux-aarch64.so.1 (0xffff86517000)
    libpthread.so.0 => /lib/ld-linux-aarch64.so.1 (0xffff86517000)
    libc.so.6 => /lib/ld-linux-aarch64.so.1 (0xffff86517000)
```

```
FROM alpine
```

Make our platform ARM ready: Sidecars

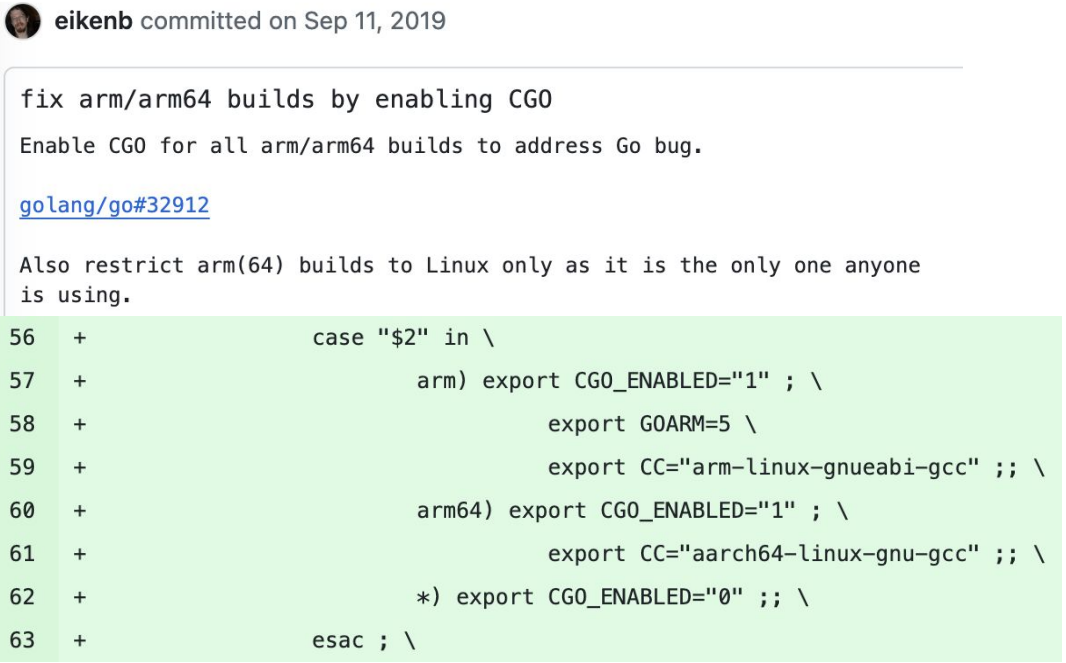


ARM image does not work => linked with libc **but alpine image**

```
# ./consul-template-arm64
/bin/sh: ./consul-template-arm64: not found

# ldd consul-template-arm64
    /lib/ld-linux-aarch64.so.1 (0xffff86517000)
    libpthread.so.0 => /lib/ld-linux-aarch64.so.1 (0xffff86517000)
    libc.so.6 => /lib/ld-linux-aarch64.so.1 (0xffff86517000)

# ldd consul-template-x86
/lib/ld-musl-aarch64.so.1: consul-template-x86: Not a valid dynamic program
```

What happened?

- Consul-template moved to CGO because of a Go bug



eikenb committed on Sep 11, 2019

fix arm/arm64 builds by enabling CGO

Enable CGO for all arm/arm64 builds to address Go bug.

golang/go#32912

Also restrict arm(64) builds to Linux only as it is the only one anyone is using.

```
56  +                case "$2" in \
57  +                        arm) export CGO_ENABLED="1" ; \
58  +                                export GOARM=5 \
59  +                                export CC="arm-linux-gnueabi-gcc" ;; \
60  +                        arm64) export CGO_ENABLED="1" ; \
61  +                                export CC="aarch64-linux-gnu-gcc" ;; \
62  +                        *) export CGO_ENABLED="0" ;; \
63  +                esac ; \
```

What happened?

- Consul-template moved to CGO because of a Go bug

eikenb committed on Sep 11, 2019

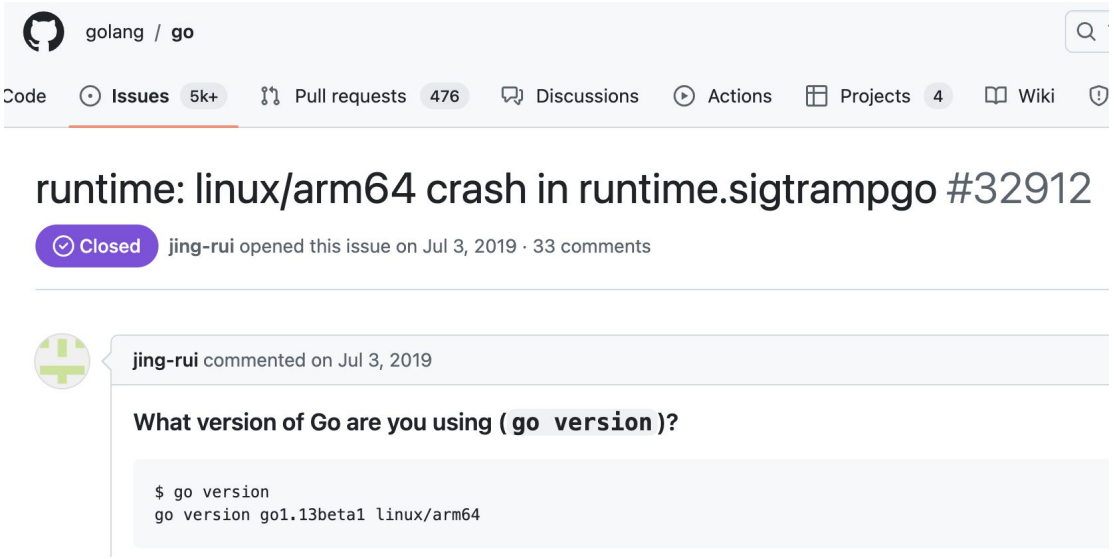
```
fix arm/arm64 builds by enabling CGO

Enable CGO for all arm/arm64 builds to address Go bug.

golang/go#32912

Also restrict arm(64) builds to Linux only as it is the only one anyone
is using.
```

```
56  +                case "$2" in \
57  +                        arm) export CGO_ENABLED="1" ; \
58  +                                export GOARM=5 \
59  +                                export CC="arm-linux-gnueabi-gcc" ;; \
60  +                        arm64) export CGO_ENABLED="1" ; \
61  +                                export CC="aarch64-linux-gnu-gcc" ;; \
62  +                        *) export CGO_ENABLED="0" ;; \
63  +                esac ; \
```

but Build image was glibc based, and runtime image was Alpine/musl

What was this go bug?

- Consul-template moved to CGO because of a Go bug
- Go bug was triggering crashes on ARM, *but not when using CGO*



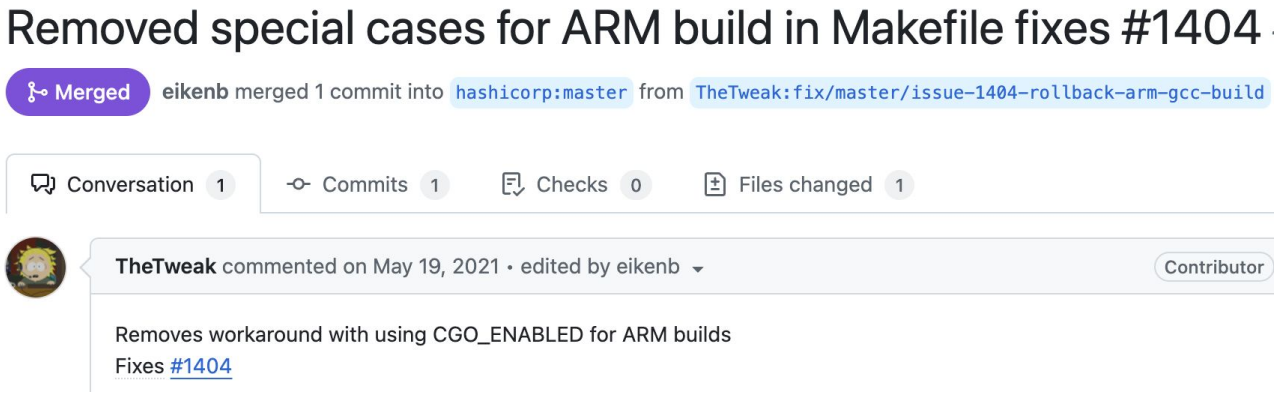
What happened?

- Consul-template moved to CGO because of a Go bug
- Go bug was triggering crashes on ARM
- Go Bug was fixed in Go 1.14

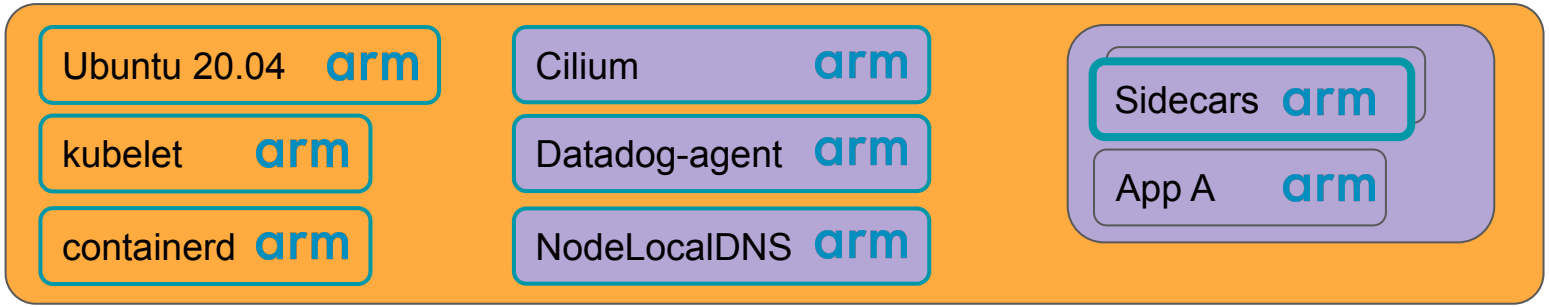
```
runtime: fix crash during VDSO calls on arm

As discussed in #32912, a crash occurs when go runtime calls a VDSO
function (say
__vdso_clock_gettime) and a signal arrives to that thread.
Since VDSO functions temporarily destroy the G register (R10),
Go functions asynchronously executed in that thread (i.e. Go's signal
handler) can try to load data from the destroyed G, which causes
segmentation fault.
```

What happened?

- Consul-template moved to CGO because of a Go bug
- Go bug was triggering crashes on ARM
- Go Bug was fixed in Go 1.14
- Change reverted in consul-template 0.26.0



Make our platform ARM ready: Sidecars



We use consul-template to retrieve secrets from vault

No ARM image upstream but an ARM binary => reuse x86 Dockerfile

But use ubuntu as base

```
FROM ubuntu
RUN curl -OL …/consul-template_${VERSION}_linux_${ARCH}.tgz && \
    tar -xzf … && mv …
```

Summary

- We have ARM nodes
- We have ARM images for our daemonsets
- We have ARM images for utility containers (sidecars / init-containers)

A few challenges but we're getting there!

What now?

- We have ARM nodes
- We have ARM images for our daemonsets
- We have ARM images for utility containers (sidecars / init-containers)
- **We need to build ARM binaries for our applications**

What now?

- We have ARM nodes
- We have ARM images for our daemonsets
- We have ARM images for utility containers (sidecars / init-containers)
- **We need to build ARM binaries for our applications**
- **We need ARM images for our applications**

Quick summary

- We have ARM nodes
- We have ARM images for our daemonsets
- We have ARM images for utility containers (sidecars / init-containers)
- **We need to build ARM binaries for our applications**
- **We need ARM images for our applications**
  - We want to make ARM as easy and transparent as possible

Quick summary

- We have ARM nodes
- We have ARM images for our daemonsets
- We have ARM images for utility containers (sidecars / init-containers)
- **We need to build ARM binaries for our applications**
- **We need ARM images for our applications**
  - We want to make ARM as easy and transparent as possible
    => No changes to application charts (No "**-arm**" in image names)

Quick summary

- We have ARM nodes
- We have ARM images for our daemonsets
- We have ARM images for utility containers (sidecars / init-containers)
- **We need to build ARM binaries for our applications**
- **We need ARM images for our applications**
  - We want to make ARM as easy and transparent as possible

    => No changes to application charts (No "**-arm**" in image names)

    => multiarch images : use the right image based on the node architecture

What are multiarch images? Let's look at a regular one


```
$ docker manifest inspect datadog/agent:6.14.1
{
     "schemaVersion": 2,
     "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
     "config": {
          "mediaType": "application/vnd.docker.container.image.v1+json",
          "size": 7719,
          "digest": "sha256:b96c197..."
     },
     "layers": [
          {
               "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
               "size": 27093738,
               "digest": "sha256:b8f262..."
          },
          {
               "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
               "size": 511793,
               "digest": "sha256:31c6f6..."
          },
          [....
```

metadata

layers

What are multiarch images


```
$ docker manifest inspect datadog/agent:6.15.1
{
   "schemaVersion": 2,
   "mediaType": "application/vnd.docker.distribution.manifest.list.v2+json",
   "manifests": [
      {
         "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
         "size": 2409,
         "digest": "sha256:b6b46b...",
         "platform": {
            "architecture": "amd64",
            "os": "linux"
         }
      },
      {
         "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
         "size": 2409,
         "digest": "sha256:783ba6...",
         "platform": {
            "architecture": "arm64",
            "os": "linux"
         }
      }
```

metadata

manifest for x86

manifest for arm64

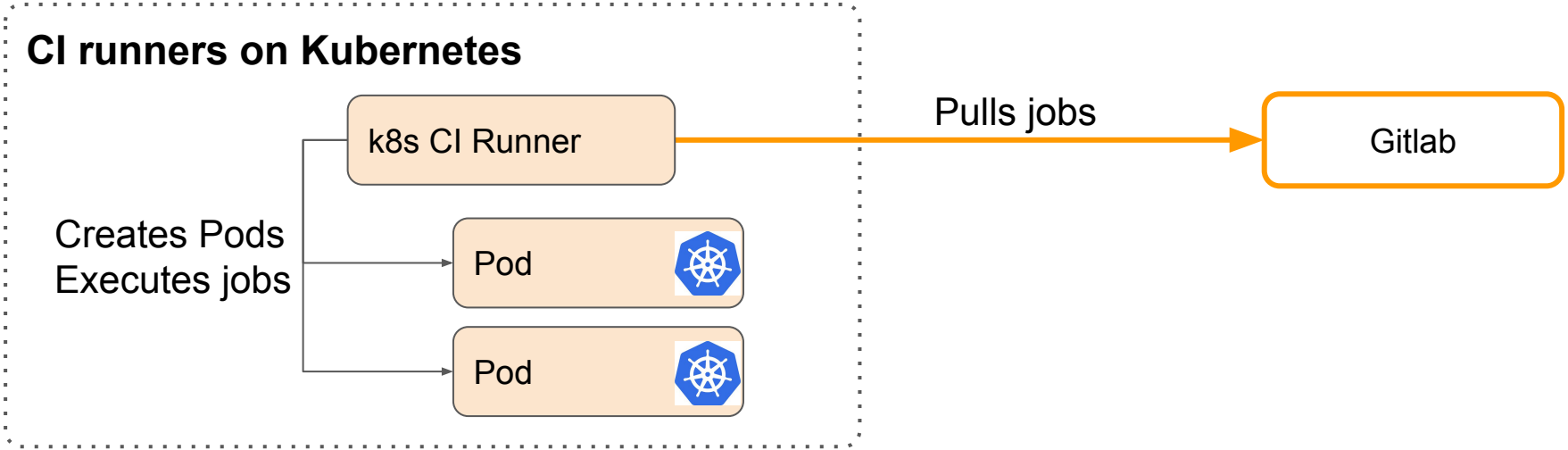
Quick summary

- We have ARM nodes
- We have ARM images for our daemonsets **(but want multiarch images)**
- We have ARM images for utility containers **(but want multiarch images)**
- **We need to build ARM binaries for our applications**
- **We need multiarch images for our applications**

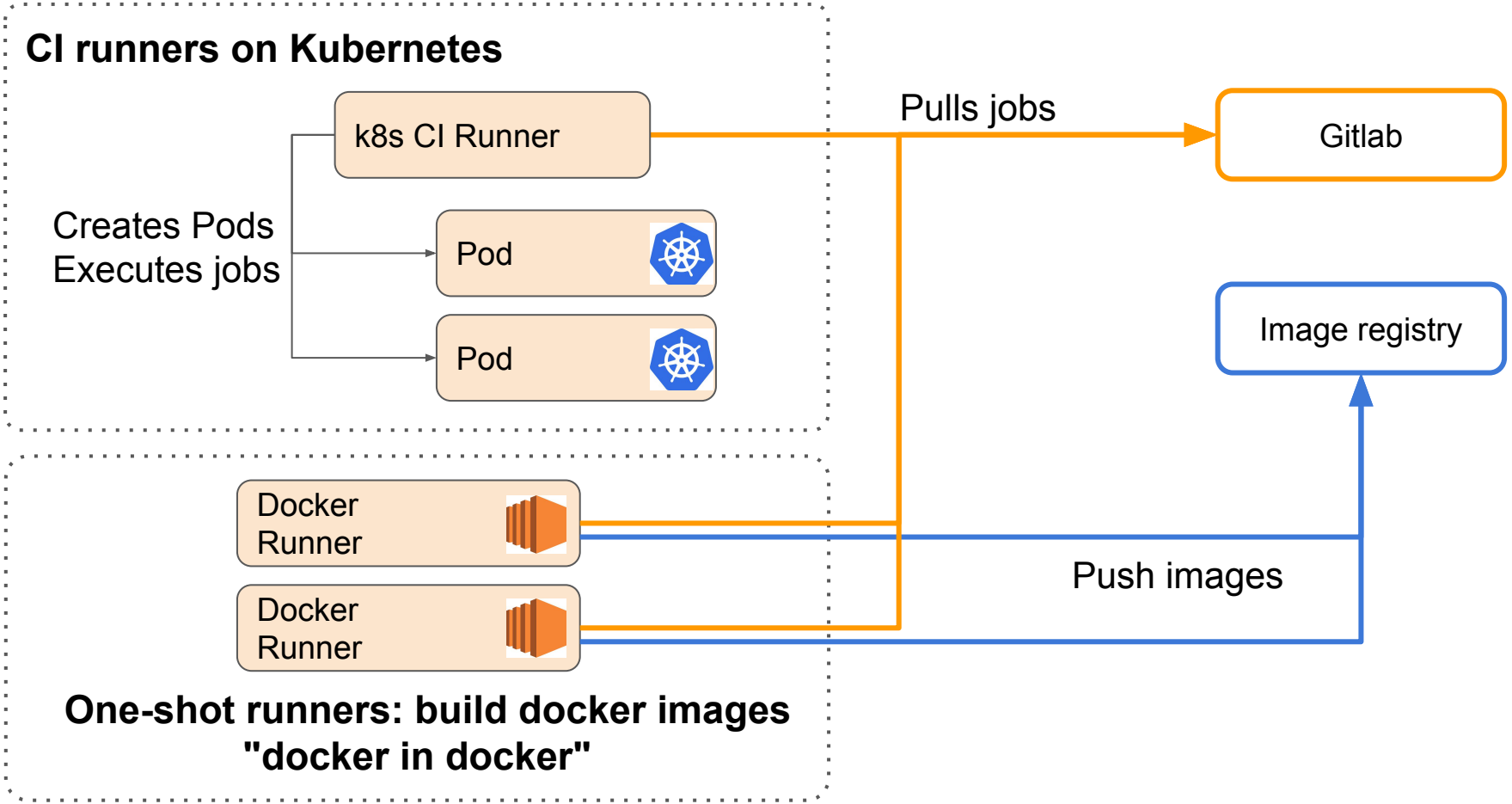
How can we do this?

Day -1: CI Before ARM

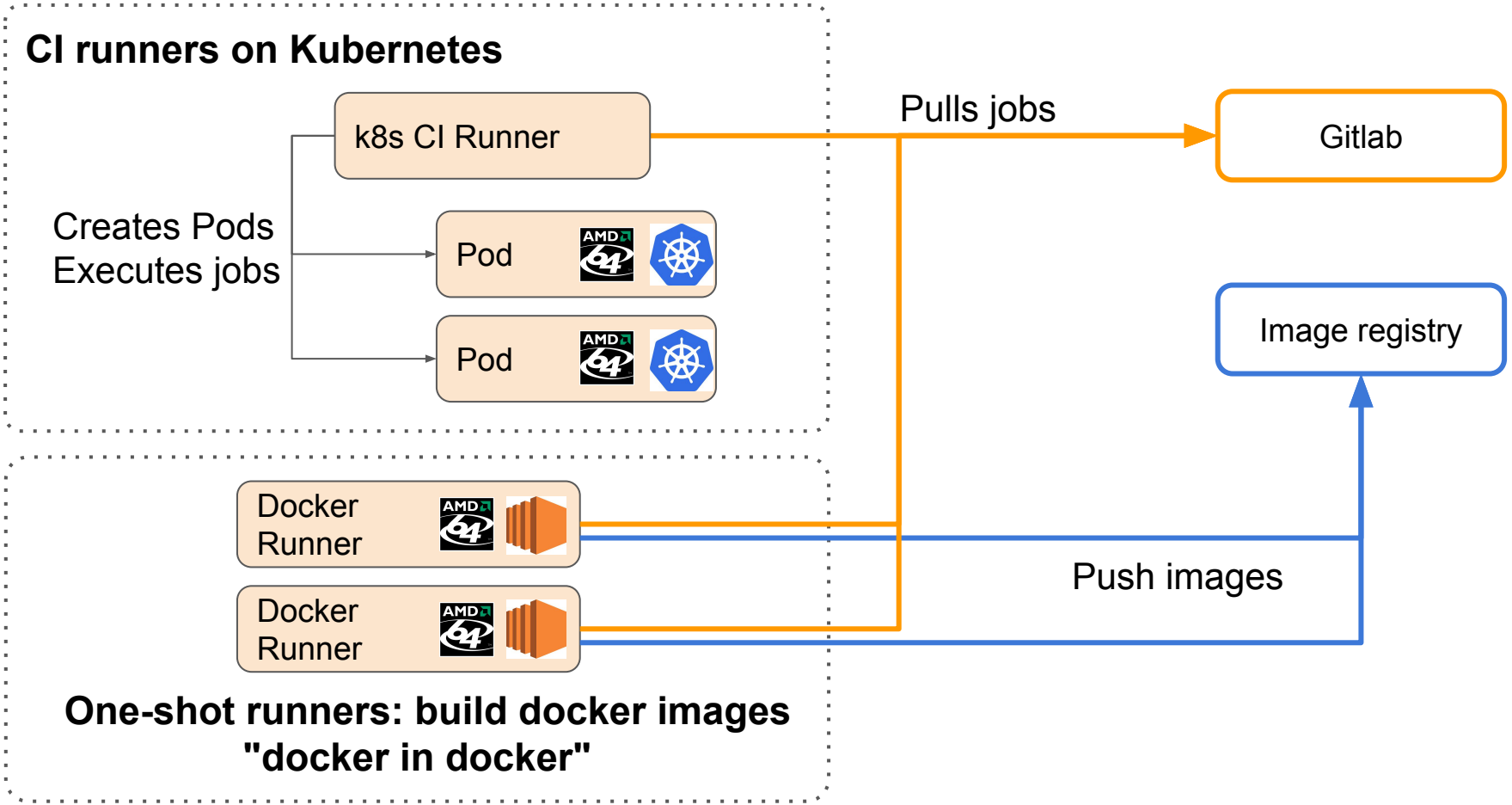
We use Kubernetes for CI



And ec2 instances to build images



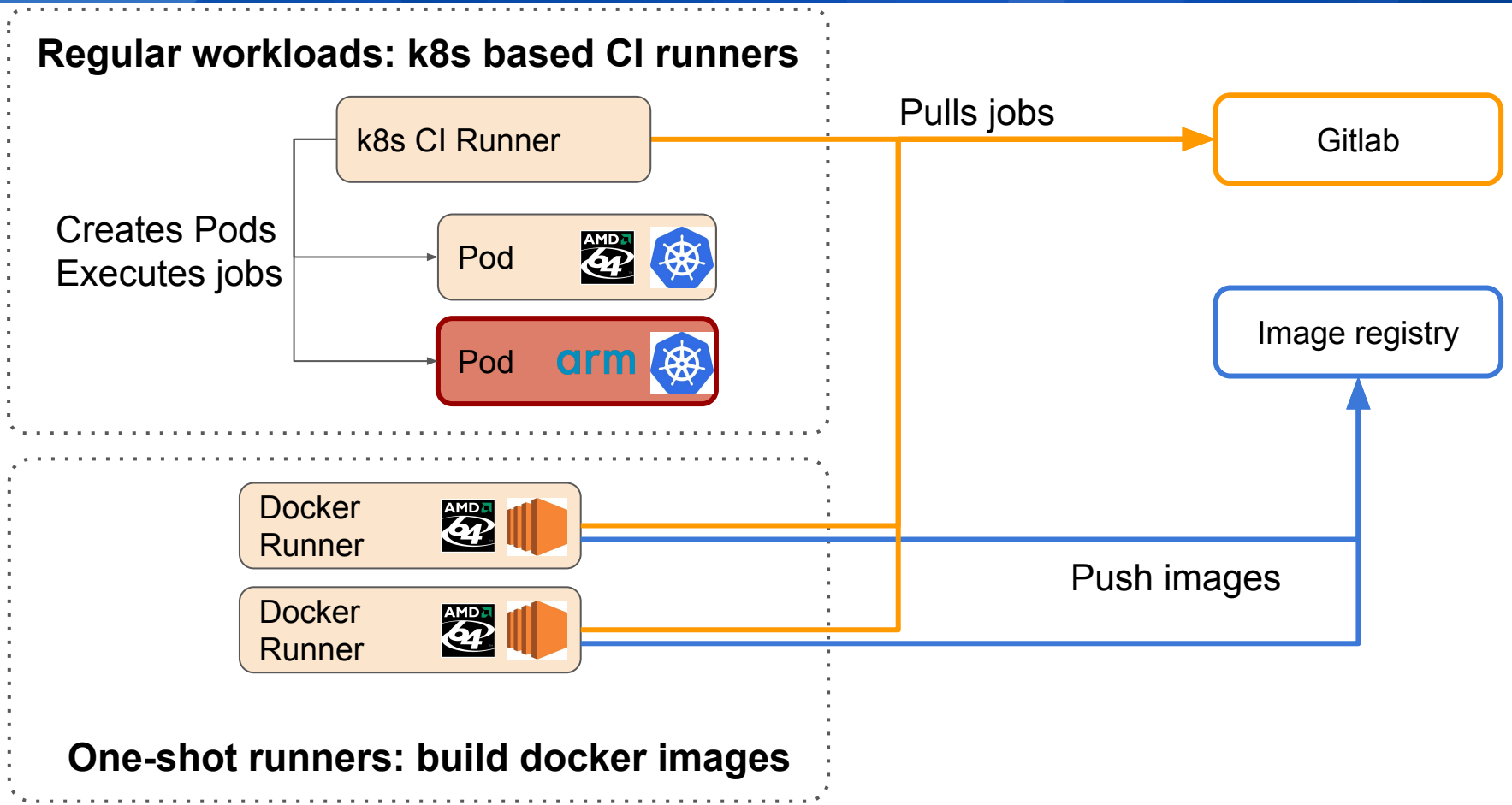
And of course, everything runs on x86



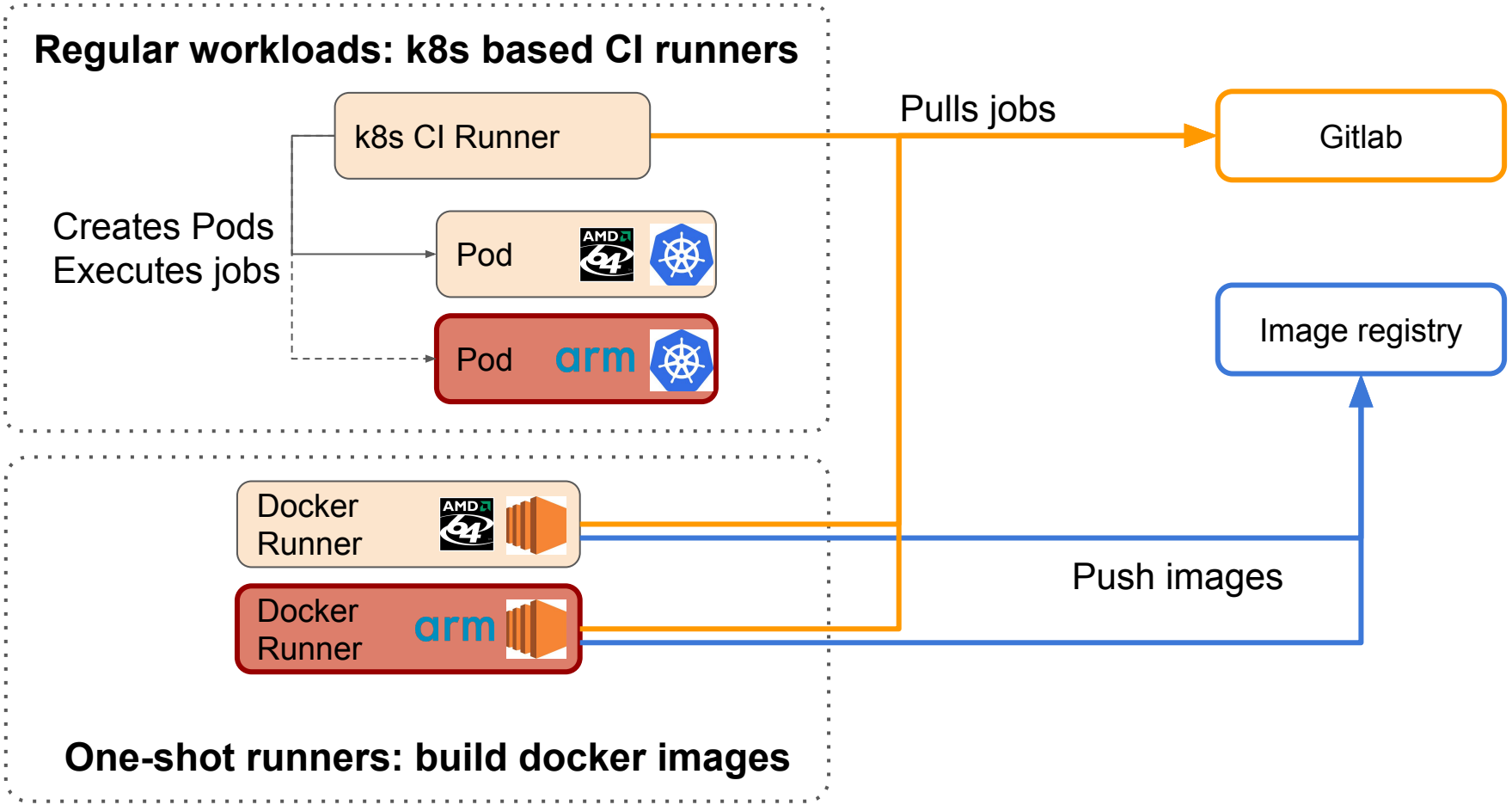
Day 0: Bootstrapping ARM

Making our CI ARM ready

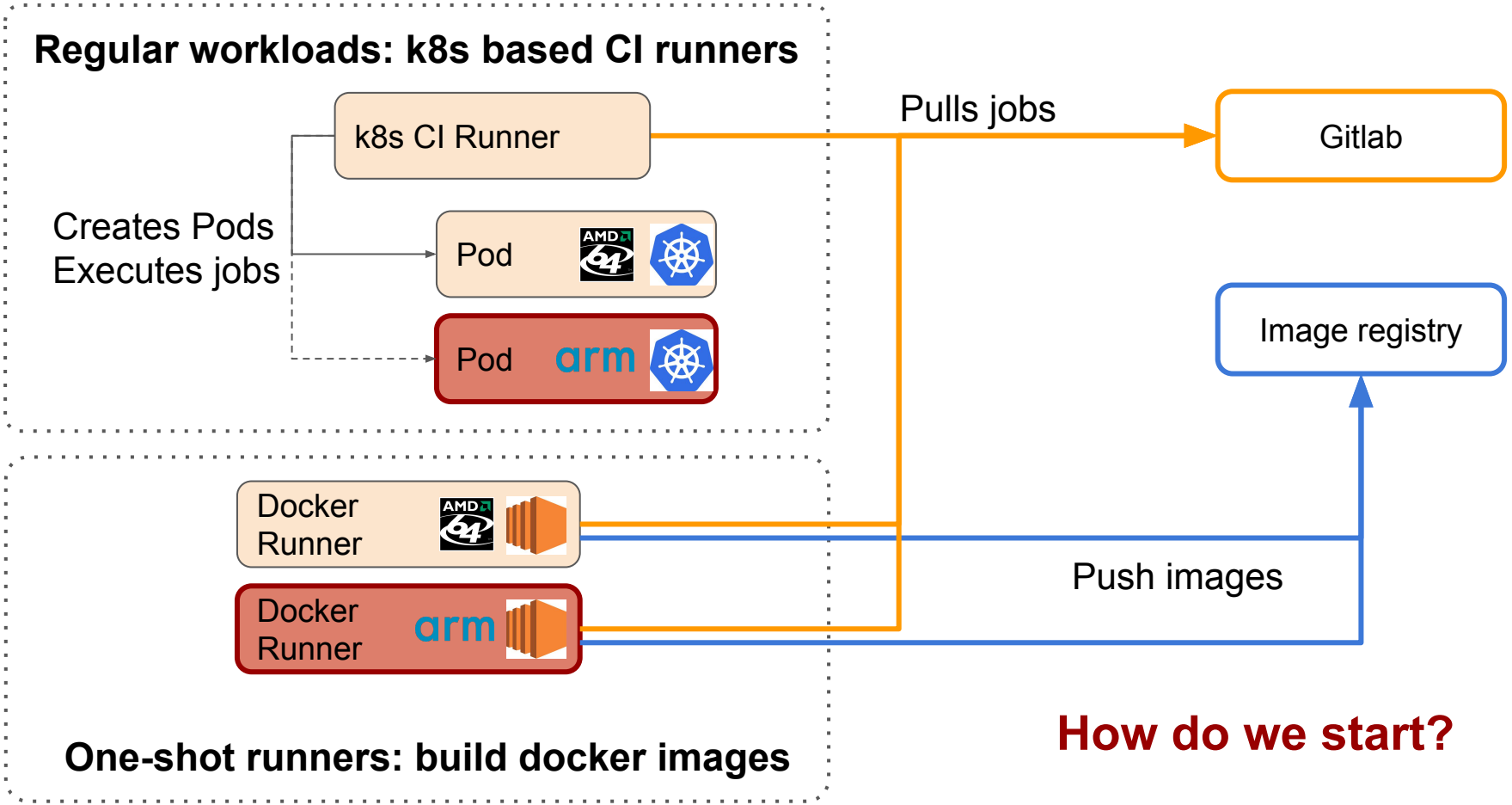
Building ARM binaries requires ARM node



ARM nodes require ARM binaries and images



ARM nodes require ARM binaries and images



Let's start with a simple example


```
FROM alpine

RUN apk add curl go
RUN curl -L https://api.github.com/dvdksn/buildme -o buildme.tgz
RUN tar -xzf buildme.tgz
RUN go build -o /server .

RUN curl -L https://github.com/krallin/tini-static -o /tini
RUN chmod +x /tini

ENTRYPOINT ["/tini"]

CMD ["/server"]
```

Build simple Go app

Get binary

Let's start with a simple example

Building the image

```
x86-node:~$ docker build --push -t lbernail/server .
```

Let's start with a simple example

Building the image

```
x86-node:~$ docker build --push -t lbernail/server .
```

Running the image on x86 => **OK**

```
x86-node:~$ docker run lbernail/server
```

Let's start with a simple example

Building the image

```
x86-node:~$ docker build --push -t lbernail/server .
```

Running the image on x86 => OK

```
x86-node:~$ docker run lbernail/server
```

Running the image on arm64 => Not OK

```
arm-node:~$ docker run lbernail/server .

Unable to find image 'lbernail/server:latest' locally
latest: Pulling from lbernail/server
docker: no matching manifest for linux/arm64/v8 in the manifest list entries.
```

Docker can build multiarch images, let's try

Building the image

```
x86-node:~$ docker buildx build --platform linux/amd64,linux/arm64 -t lbernail/server .
```

Docker can build multiarch images, let's try

Building the image

```
x86-node:~$ docker buildx build --platform linux/amd64,linux/arm64 -t lbernail/server .

 => [linux/amd64 internal] load metadata for docker.io/library/alpine:latest
 => [linux/arm64 internal] load metadata for docker.io/library/alpine:latest

[linux/amd64 1/7] FROM docker.io/library/alpine:latest
[linux/arm64 1/7] FROM docker.io/library/alpine:latest
[linux/amd64 2/7] RUN apk add curl go
 => ERROR [linux/arm64 2/7] RUN apk add curl go

------
 > [linux/arm64 2/7] RUN apk add curl go:
0.231 exec /bin/sh: exec format error
```

Docker can build multiarch images, let's try

Building the image

```
x86-node:~$ docker buildx build --platform linux/amd64,linux/arm64 -t lbernail/server .

 => [linux/amd64 internal] load metadata for docker.io/library/alpine:latest
 => [linux/arm64 internal] load metadata for docker.io/library/alpine:latest

[linux/amd64 1/7] FROM docker.io/library/alpine:latest
[linux/arm64 1/7] FROM docker.io/library/alpine:latest
[linux/amd64 2/7] RUN apk add curl go
 => ERROR [linux/arm64 2/7] RUN apk add curl go

------
 > [linux/arm64 2/7] RUN apk add curl go:
0.231 exec /bin/sh: exec format error
```

When building for ARM, docker uses an ARM alpine image
Binaries in ARM alpine image can't run on our x86 node

How can we address this? Back to the image


```
FROM alpine

RUN apk add curl go
RUN curl -L https://api.github.com/dvdksn/buildme -o buildme.tgz
RUN tar -xzf buildme.tgz
RUN go build -o /server .

RUN curl -L https://github.com/krallin/tini-static -o /tini
RUN chmod +x /tini

ENTRYPOINT ["/tini"]

CMD ["/server"]
```

How can we address this? Back to the image


```
FROM alpine

RUN apk add curl go
RUN curl -L https://api.github.com/dvdksn/buildme -o buildme.tgz
RUN tar -xzf buildme.tgz
RUN go build -o /server .

RUN curl -L https://github.com/krallin/tini-static -o /tini
RUN chmod +x /tini

ENTRYPOINT ["/tini"]

CMD ["/server"]
```

Maybe we can be more clever when building
=> multistage build?

How can we address this? Multistage builds


```
FROM --platform=$BUILDPLATFORM alpine AS build
ARG TARGETOS
ARG TARGETARCH

RUN apk add curl go
RUN curl -L https://api.github.com/dvdksn/buildme -o buildme.tgz
RUN tar -xzf buildme.tgz
RUN GOOS=${TARGETOS} GOARCH=${TARGETARCH} go build -o /server .

RUN curl -L https://github.com/krallin/tini-static-${TARGETARCH} -o /tini
RUN chmod +x /tini
```

Build phase

Runs on x86
- for x86
- for arm64

```
FROM alpine
COPY --from=build /server /server
COPY --from=build /tini /tini
ENTRYPOINT ["/tini"]
CMD ["/server"]
```

Final image

Assemble layers
No execution

It works!

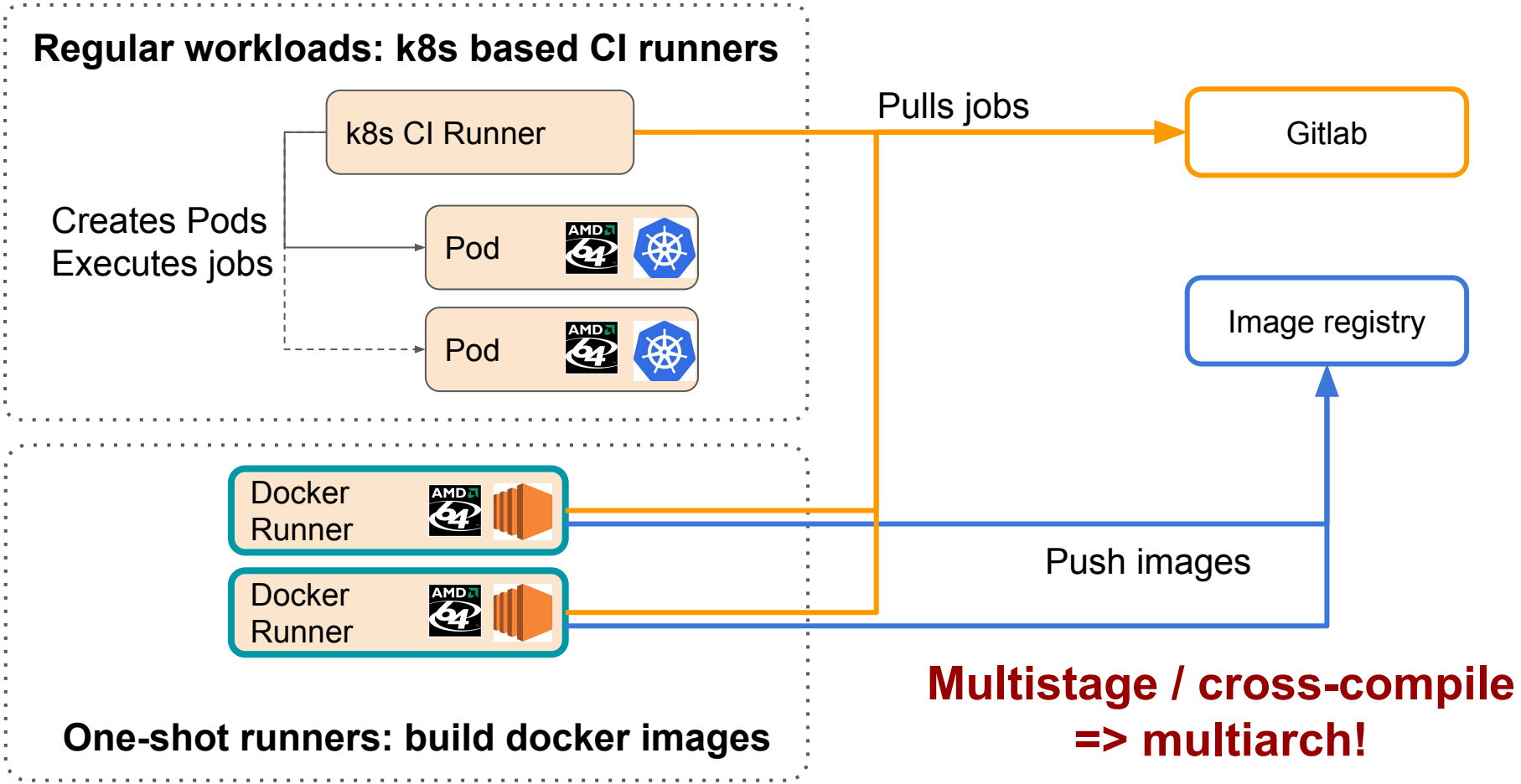
Running the image on x86 => **OK**

```
x86-node:~$ docker run lbernail/server
```

Running the image on arm64 => **OK**

```
arm-node:~$ docker run lbernail/server
```

Multiarch images using cross-compilation



Multiarch images using cross-compilation

- Works great
- Used by many applications (Cilium for instance)

Multiarch images using cross-compilation

- Works great
- Used by many applications (Cilium for instance)
- **Requires significant image refactoring**
- **Doesn't always work**
  - **Some images require RUN (example: install packages)**

Multiarch images using cross-compilation

- Works great
- Used by many applications (Cilium for instance)
- **Requires significant image refactoring**
- **Doesn't always work**
  - **Some images require RUN (example: install packages)**

Do we have alternatives?

Multiarch images using cross-compilation

- Works great
- Used by many applications (Cilium for instance)
- **Requires significant image refactoring**
- **Doesn't always work**
  - **Some images require RUN (example: install packages)**

Do we have alternatives?

=> Emulation

Back to our original image


```
FROM alpine

RUN apk add curl go
RUN curl -L https://api.github.com/dvdksn/buildme -o buildme.tgz
RUN tar -xzf buildme.tgz
RUN go build -o /server .

RUN curl -L https://github.com/krallin/tini-static -o /tini
RUN chmod +x /tini

ENTRYPOINT ["/tini"]

CMD ["/server"]
```

Back to our original image


```
FROM alpine

RUN apk add curl go
RUN curl -L https://api.github.com/dvdksn/buildme -o buildme.tgz
RUN tar -xzf buildme.tgz
RUN go build -o /server .

RUN curl -L https://github.com/krallin/tini-static -o /tini
RUN chmod +x /tini

ENTRYPOINT ["/tini"]

CMD ["/server"]
```

Emulation to run arm binaries when building on x86

Back to our original image


```
FROM alpine

RUN apk add curl go
RUN curl -L https://api.github.com/dvdksn/buildme -o buildme.tgz
RUN tar -xzf buildme.tgz
RUN go build -o /server .

RUN curl -L https://github.com/krallin/tini-static -o /tini
RUN chmod +x /tini

ENTRYPOINT ["/tini"]

CMD ["/server"]
```

Emulation to run arm binaries when building on x86

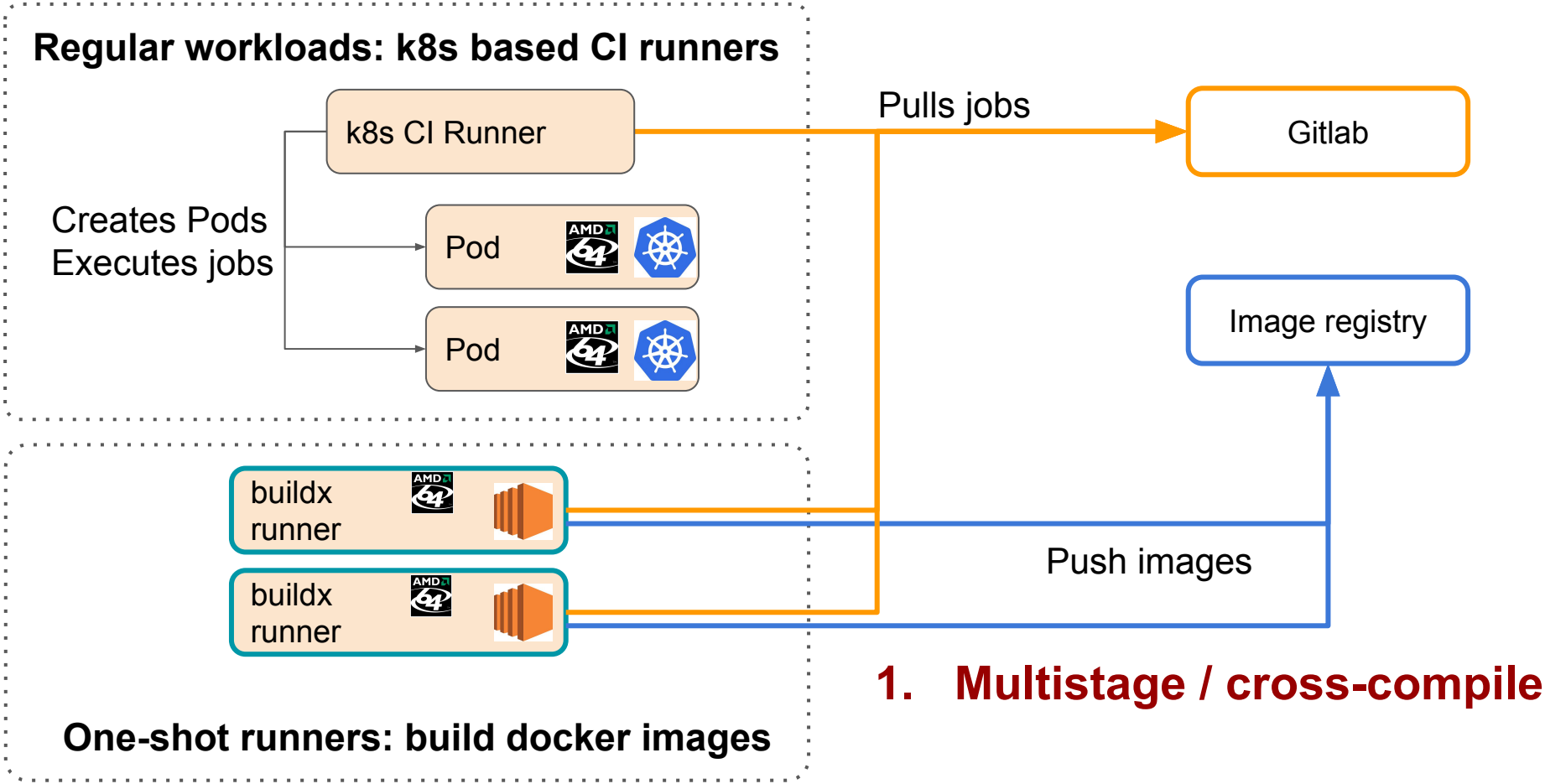
How it works

- ARM is emulated with qemu
- ARM binaries auto-detected with binfmt

Our image still requires some small changes

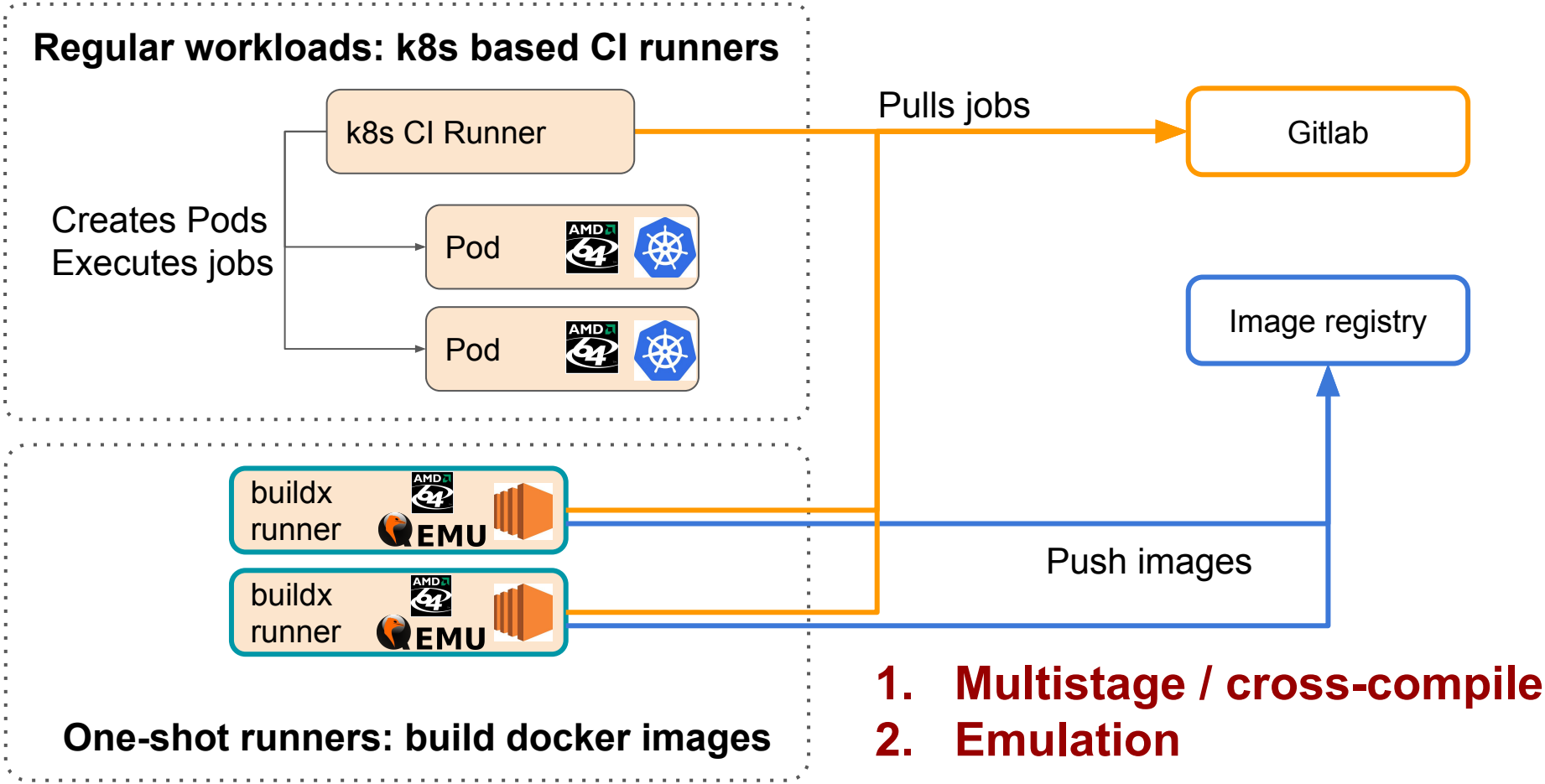

```
FROM alpine

ARG TARGETARCH

RUN apk add curl go
RUN curl -L https://api.github.com/dvdksn/buildme -o buildme.tgz
RUN tar -xzf buildme.tgz
RUN go build -o /server .

RUN curl -L https://github.com/krallin/tini-static-${TARGETARCH} -o /tini
RUN chmod +x /tini

ENTRYPOINT ["/tini"]

CMD ["/server"]
```

Our initial solutions



1. **Multistage / cross-compile**

Our initial solutions



1. **Multistage / cross-compile**
2. **Emulation**

Is emulation the perfect solution?

Requires the ability to install and configure qemu

Complex images can take a long time to build

Image Name	Emulated Build
hello-world	1 min 4 sec
toolbox	92 min

Is emulation the perfect solution?

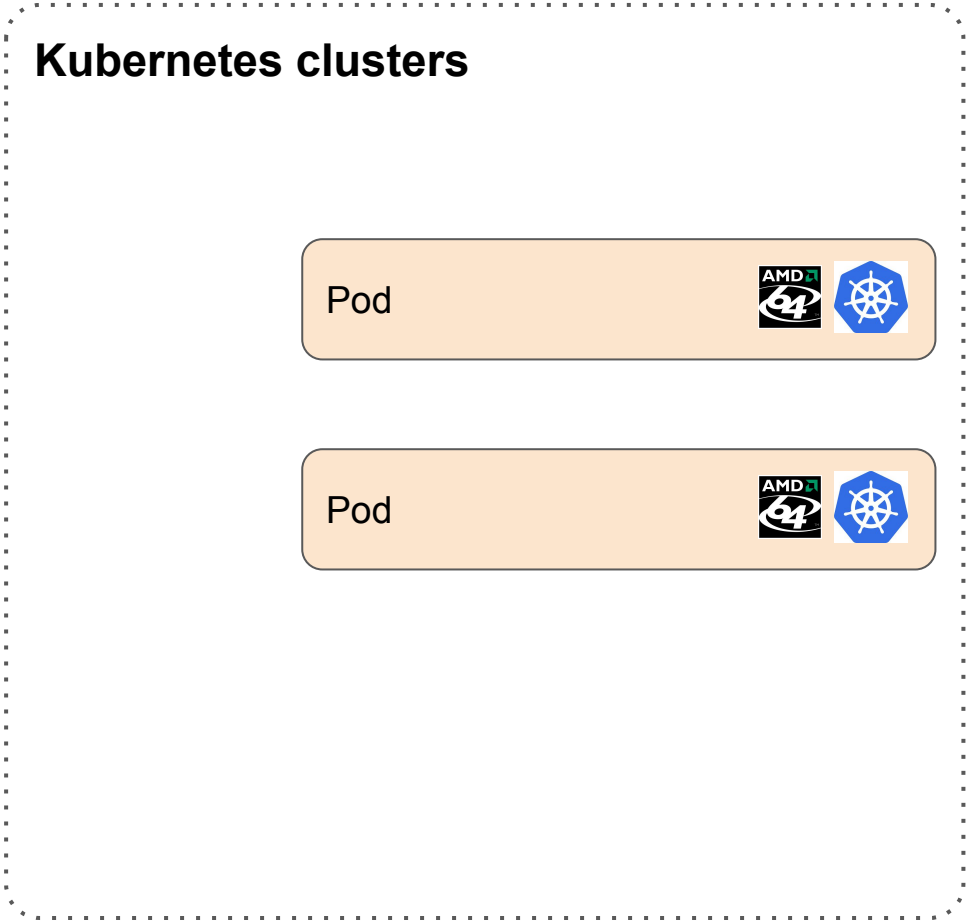
Requires the ability to install and configure qemu

Complex images can take a long time to build

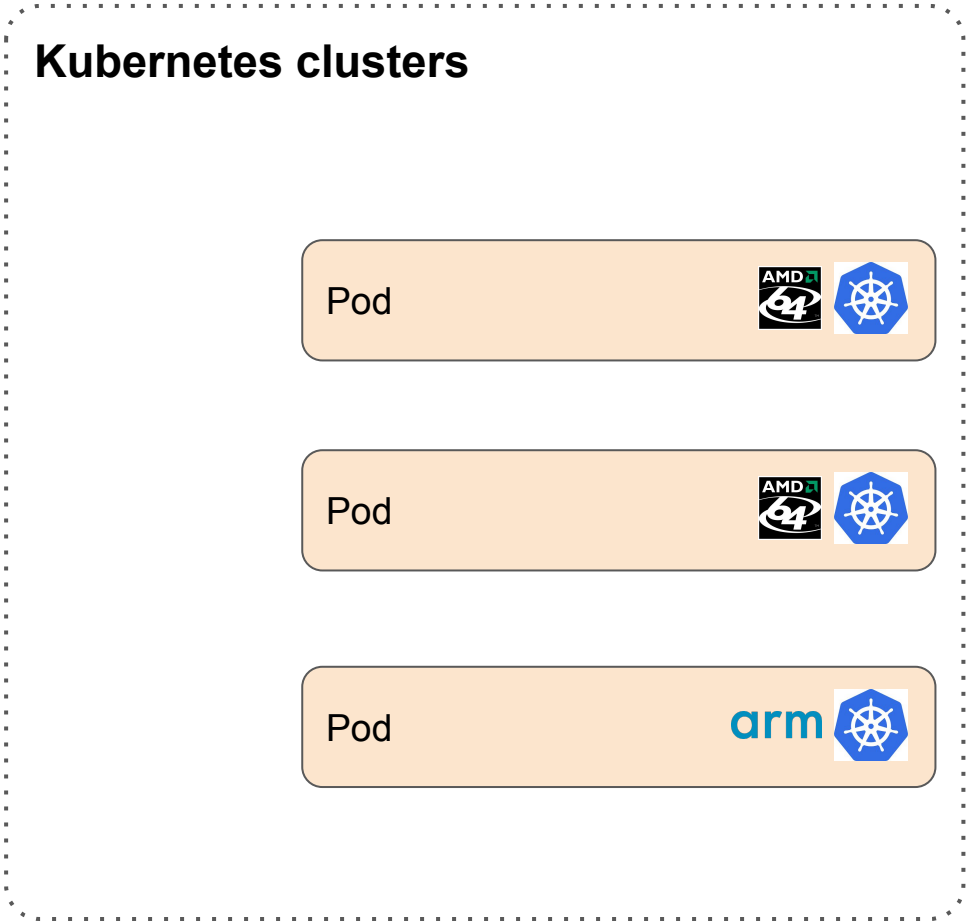
Image Name	Emulated Build
hello-world	1 min 4 sec
toolbox	92 min

What if we could build images in Kubernetes and use ARM nodes for ARM?

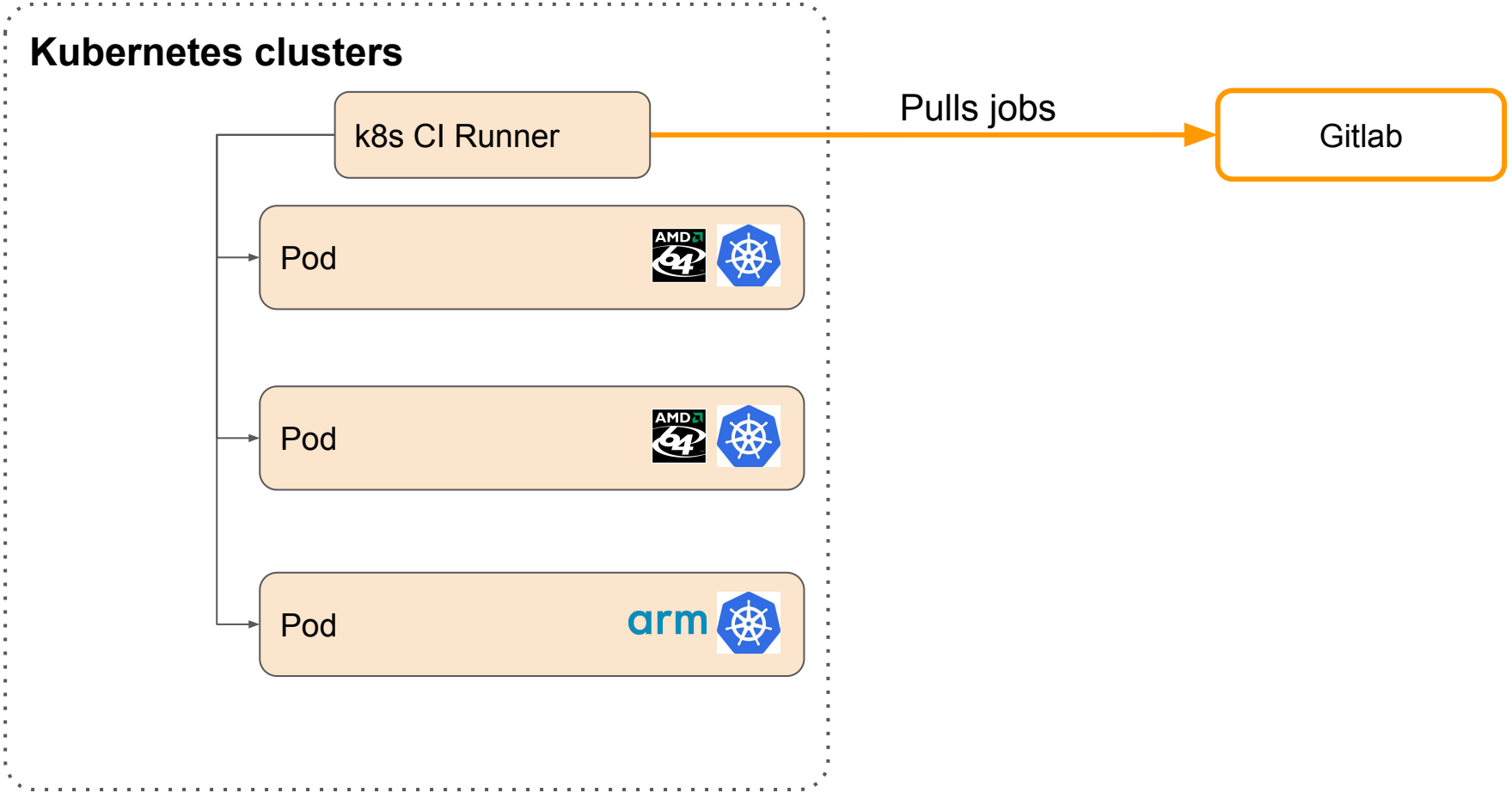
Back to our Kubernetes clusters



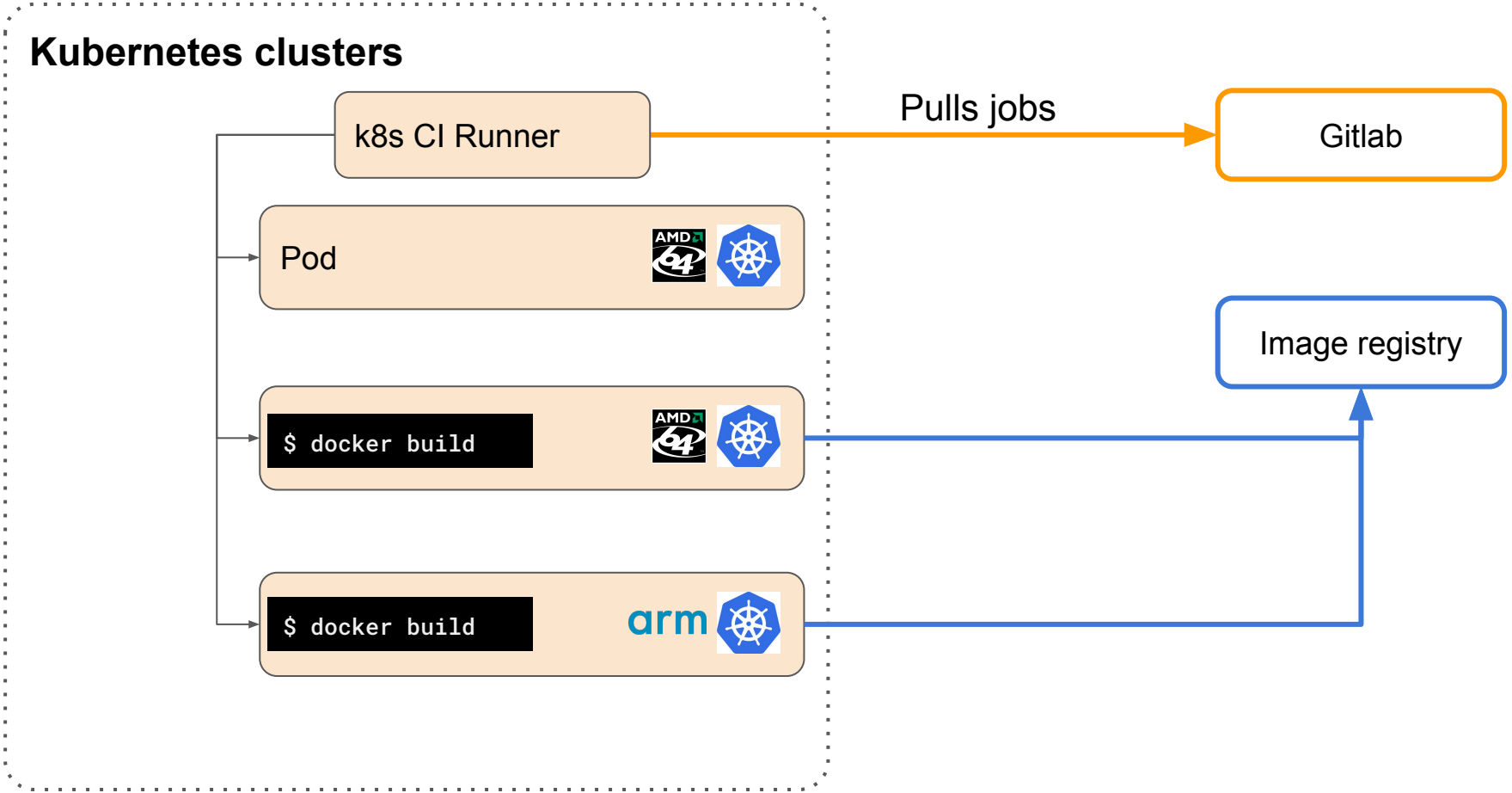
Clusters can have ARM nodes



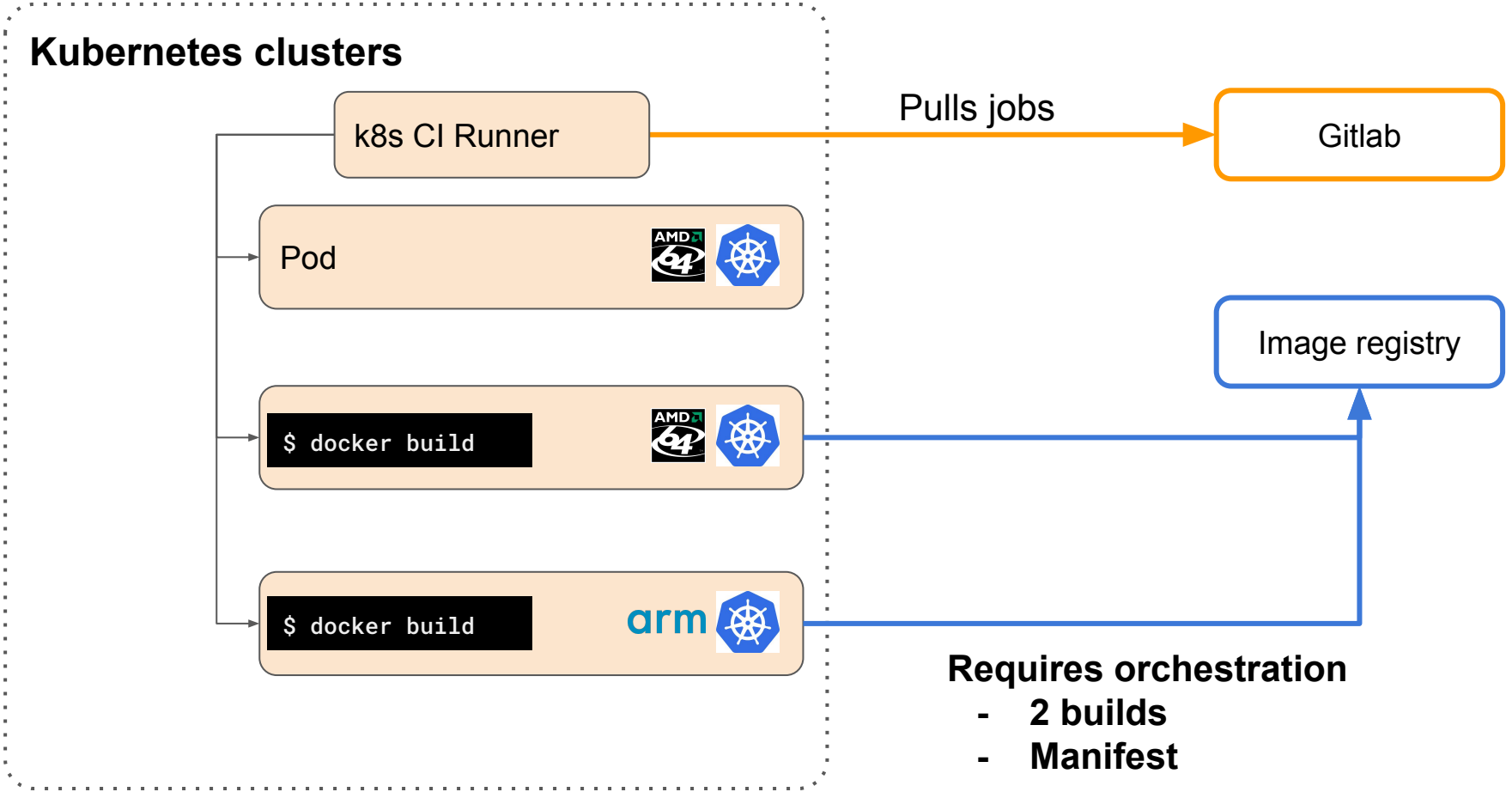
Native ARM builds in Kubernetes



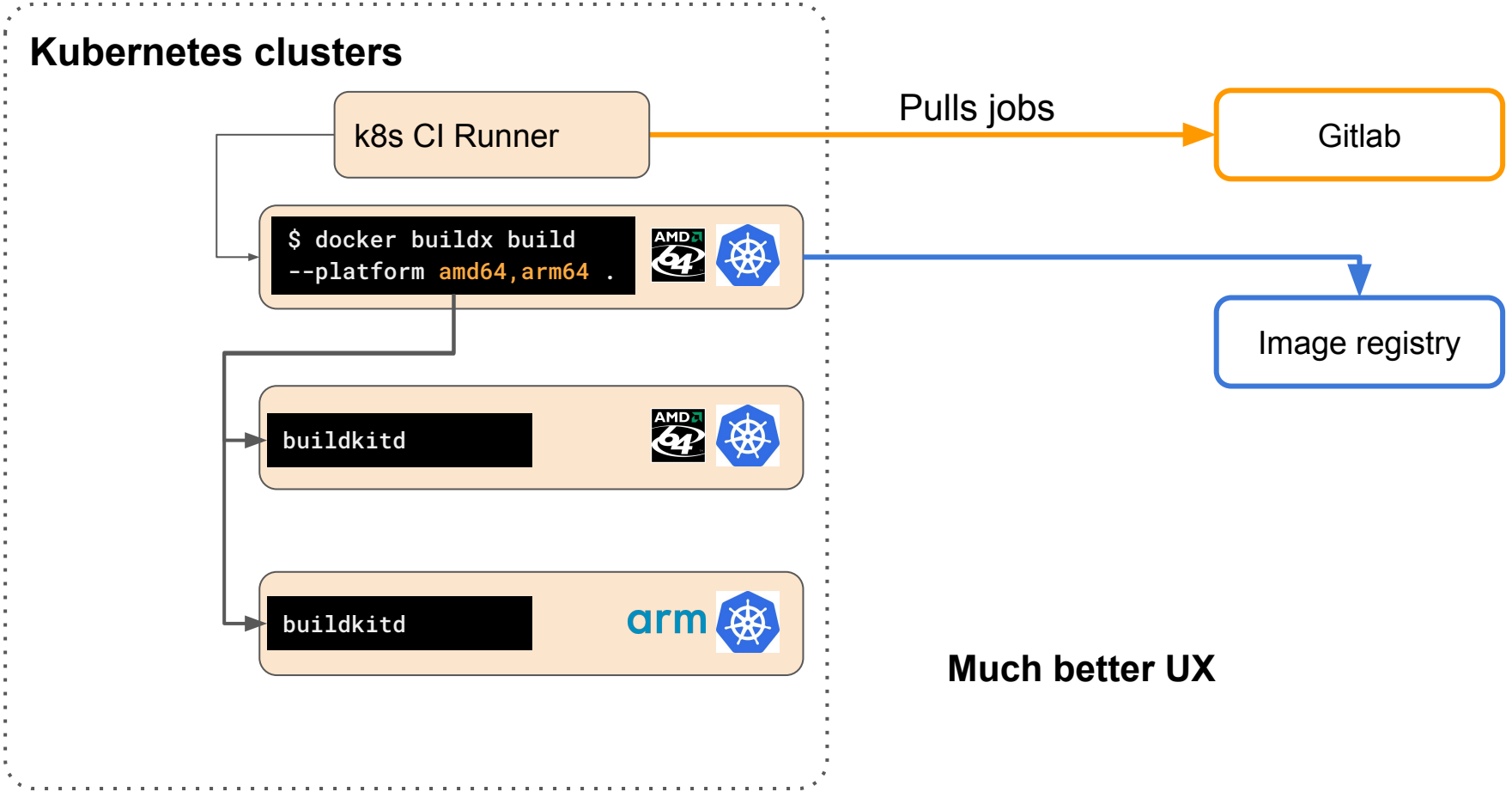
What if we built images in Kubernetes?



What if we built images in Kubernetes?



Buildkit remote builders



Native builds?

Much, much better!

Image Name	Emulated Build	Native Build
hello-world	1 min 4 sec	23 sec
toolbox	92 min	16 min

Image Builds on Kubernetes

- Works great! (but wasn't easy)
- Likely the best option but a few gotchas
  - Maybe easier to start with the other options

Early Results

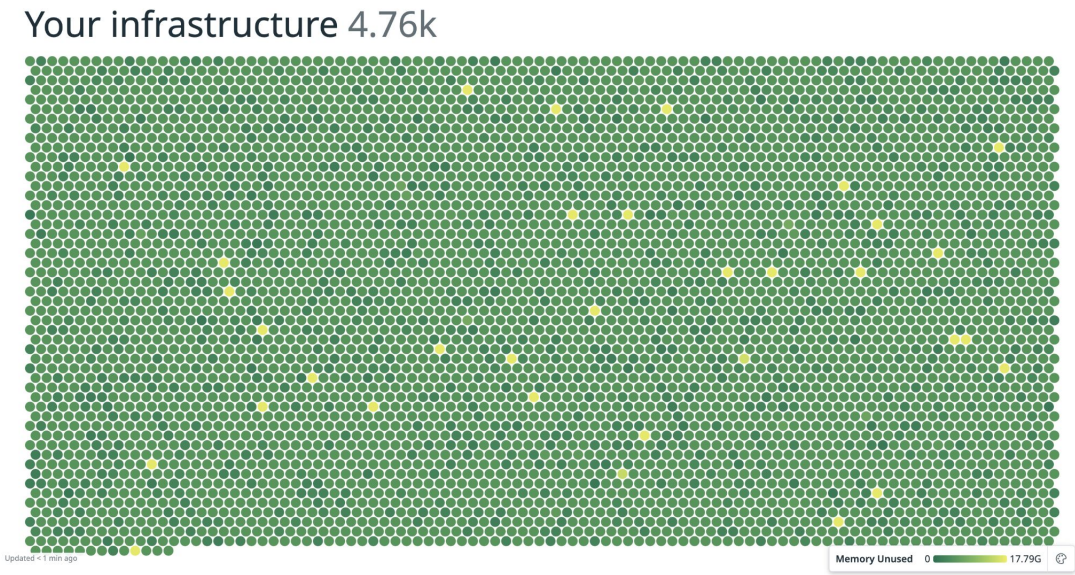
Begin adoption

- Start with high-potential “easy” applications
  - Large scale
  - Managed languages

⇒ Focus on large Golang repository
+ some early-adopters in Rust and Java

First application

- Takes part in metrics intake
  - High traffic
- Go
- Load-balanced
- Stateless



First application: rollout Jan 2022

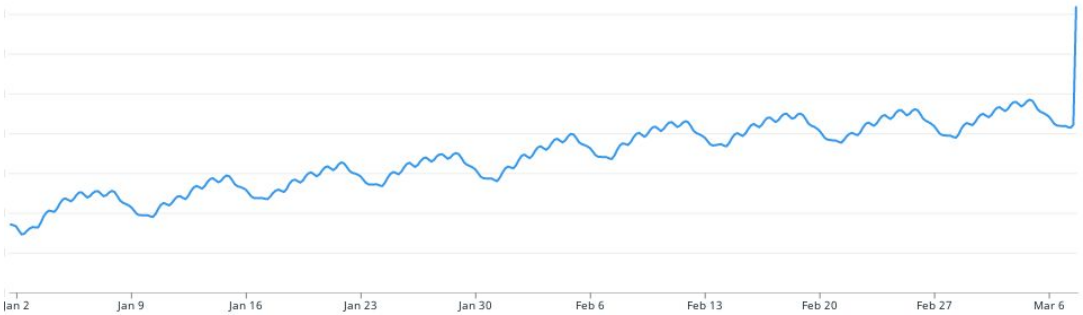
- Helm chart update
  - 2 deployments → 2 instance types
  - Extra headroom in case of issue

- Canary, then ~⅓ at a time

Proportion of ARM nodes

First application: results

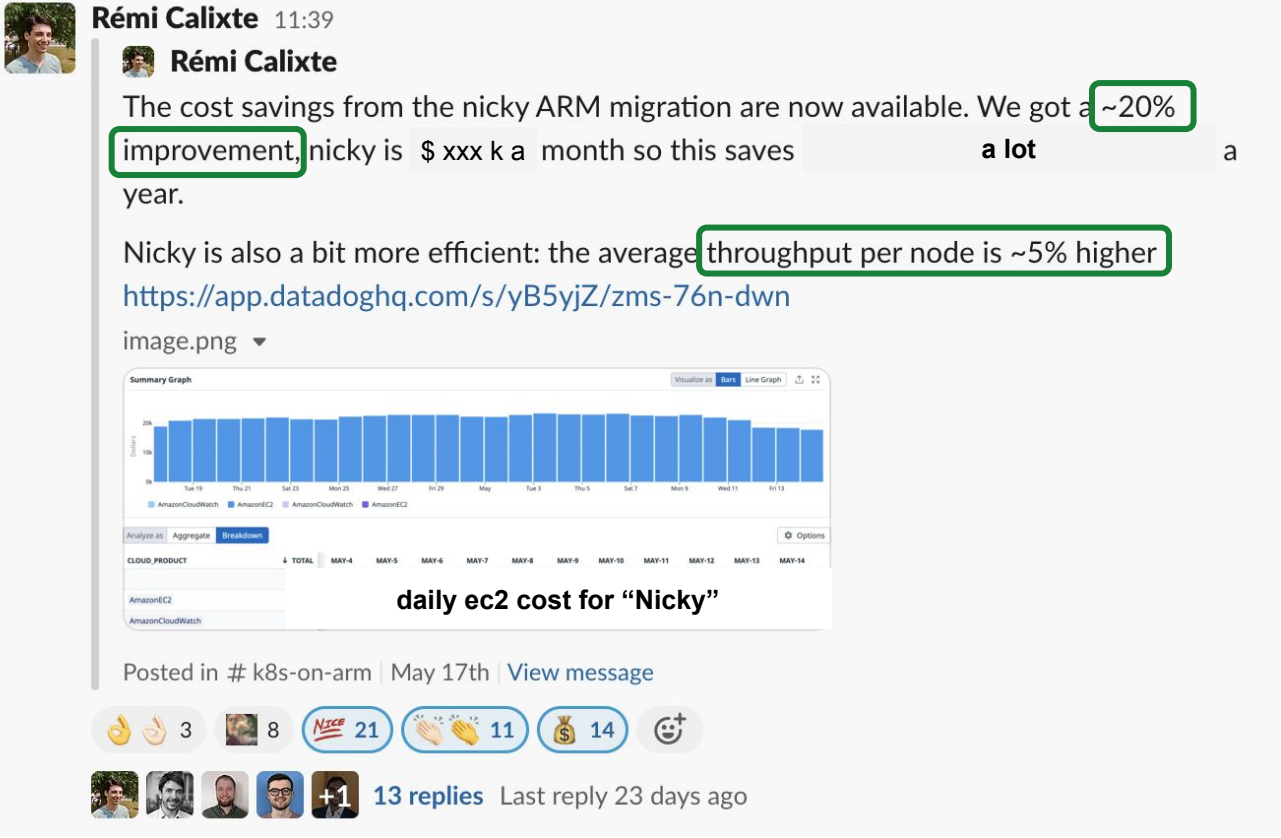
- Traffic increases
- Cost goes down! 🎉
- **3-5% slower**
  But **20% cheaper**



Requests handled per second

Infra cost per day

Second large application: Q2



Second large app in metrics processing

20% savings

5% better throughput

A Few "Fun" Bugs

Slow boot with local disks

- Instances with local disks taking **minutes** to boot

Slow boot with local disks

- Instances with local disks taking **minutes** to boot
- Traced to LUKS volume creation
- Random password generated from /dev/random
  - … blocking due to lack of entropy

Slow boot with local disks

- Instances with local disks taking **minutes** to boot
- Traced to LUKS volume creation
- Random password generated from /dev/random
  - … blocking due to lack of entropy
- Switched to /dev/urandom
  - Non-blocking
- Fixed in Ubuntu kernel 5.8+
  - Early-boot entropy seed passed via EFI

Extremely slow Rust app

- **Abysmal** performance in a Rust app
  - Not 10-20%, rather **10x slower**

Extremely slow Rust app

- **Abysmal** performance in a Rust app
- Profiling traced the problem to `Instant::now()`
  - Simple reproducer: 70x slower

What happened

std: Force `Instant::now()` to be monotonic #56988

Merged **bors** merged 1 commit into `rust-lang:master` from `alexcrichton:monotonic-instant` on Jan 8, 2019

Conversation 52 | Commits 1 | Checks 0 | Files changed 7

alexcrichton commented on Dec 19, 2018

This commit is an attempt to force `Instant::now` to be monotonic through any means possible. We tried relying on OS/hardware/clock implementations, but those seem buggy enough that we can't rely on them in practice. This commit implements the same hammer Firefox recently implemented (noted in #56612) which is to just keep whatever the lastest `Instant::now()` return value was in memory, returning that instead of the OS looks like it's moving backwards.

Buggy CPUs had time going back

⇒ panic in applications

⇒ software implementation with a mutex if not x86

Fix

1 library/std/src/sys/unix/time.rs

```
@@ -303,6 +303,7 @@ mod inner {
303             pub fn actually_monotonic() -> bool {
304                 (cfg!(target_os = "linux") && cfg!(target_arch = "x86_64"))
305                     || (cfg!(target_os = "linux") && cfg!(target_arch = "x86"))
306 +                   || (cfg!(target_os = "linux") && cfg!(target_arch = "aarch64"))
```

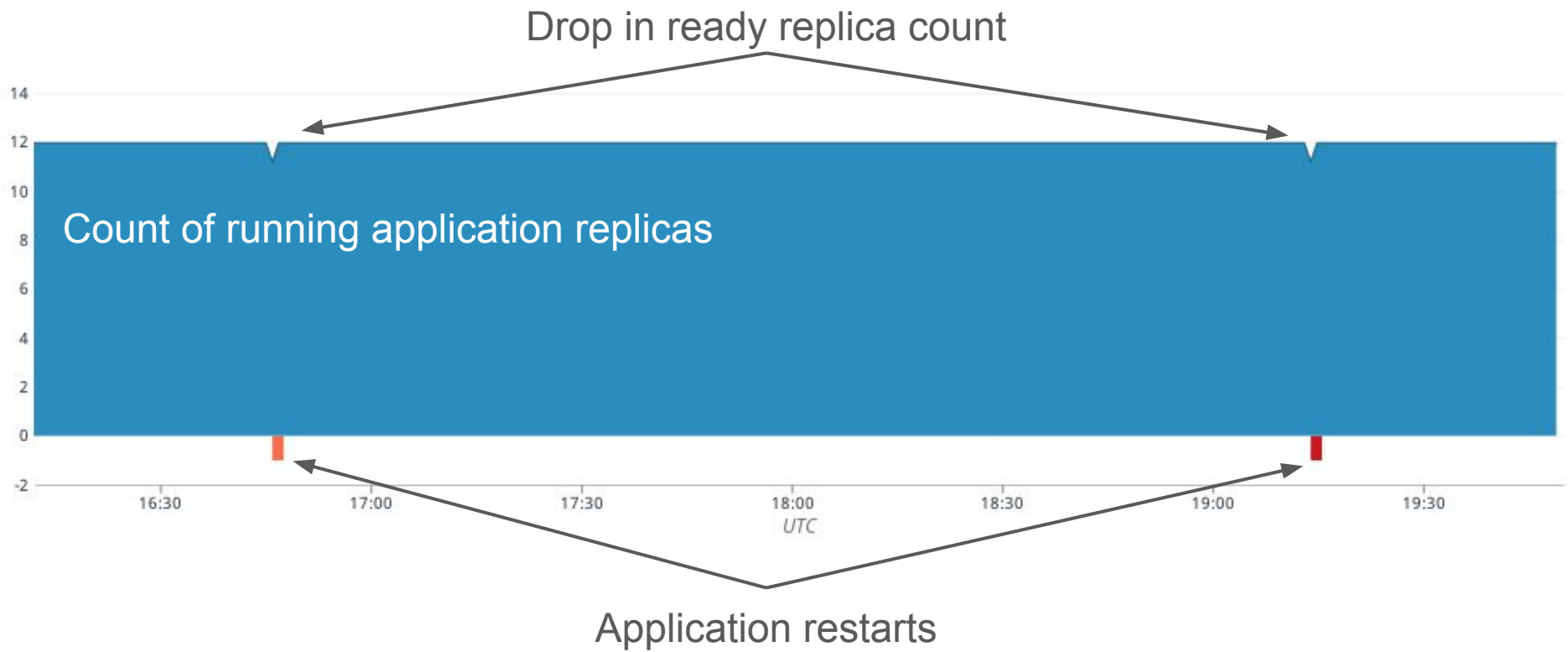
ARM CPUs with time issues mitigated in recent kernels

⇒ Add ARM64 to the list of monotonic architectures

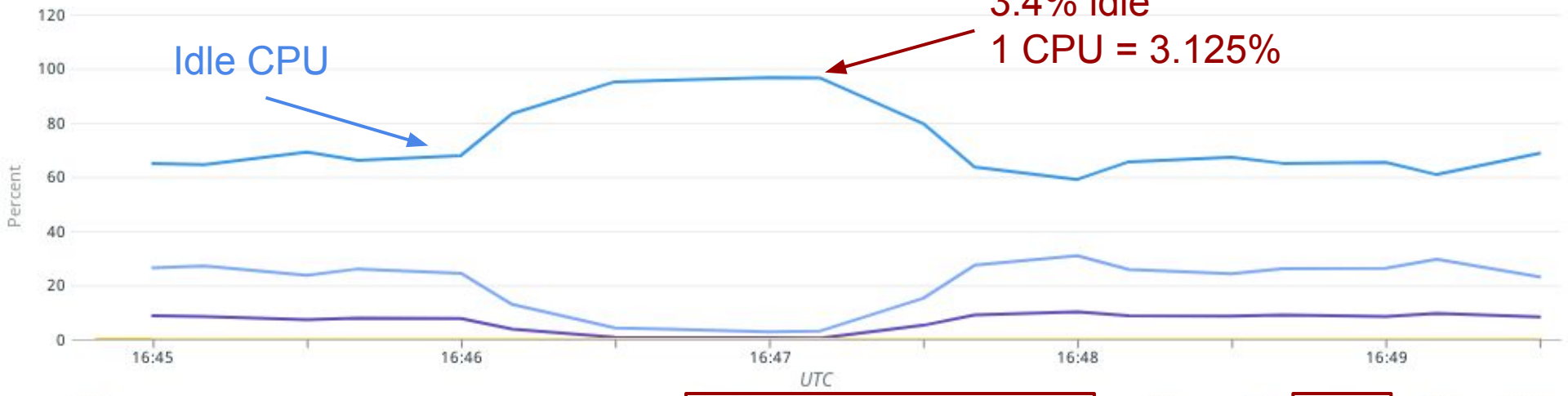
Many thanks to Ali Saidi from the AWS Graviton team for his help!

https://github.com/rust-lang/rust/pull/88652

Application stalls/lock-ups



Application stalls/lock-ups



Tags	Metric	Avg	Min	Max	Sum	Value
*	avg:system.cpu.idle{host:i-0146b21cc95936a9d}	73.41 %	59.03 %	96.66 %	1,395 %	68.66 %
*	avg:system.cpu.system{host:i-0146b21cc95936...	6.45 %	0.42 %	10.14 %	123 %	8.29 %
*	avg:system.cpu.iowait{host:i-0146b21cc95936a...	4e-3 %	2e-3 %	8e-3 %	0.078 %	—
*	avg:system.cpu.user{host:i-0146b21cc95936a9d}	20.14 %	2.79 %	30.82 %	383 %	23.05 %
*	avg:system.cpu.stolen{host:i-0146b21cc95936a...	0 %	0 %	0 %	0 %	0 %
*	avg:system.cpu.guest{host:i-0146b21cc95936a...	0 %	0 %	0 %	0 %	0 %

Application stalls/lock-ups

- 1 CPU burning away
- 4 apps encountering this pattern
  - Go 1.18.1
  - Only on ARM64

Application stalls/lock-ups

- 1 CPU burning away
- 4 apps encountering this pattern
  - Go 1.18.1
  - Only on ARM64
- runtime: gentraceback() dead loop on arm64 caused the process hang

Application stalls/lock-ups

- 1 CPU burning away
- 4 apps encountering this pattern
  - Go 1.18.1
  - Only on ARM64
- runtime: gentraceback() dead loop on arm64 caused the process hang
  - We enable profiling almost everywhere

Application stalls/lock-ups

- 1 CPU burning away
- 4 apps encountering this pattern
  - Go 1.18.1
  - Only on ARM64
- runtime: gentraceback() dead loop on arm64 caused the process hang
  - We enable profiling almost everywhere
- Locally patched Go with the fix until released upstream
  - Gotcha: needed MacOS builds for x86 and arm64
  - Weren’t really equipped to maintain local Go builds

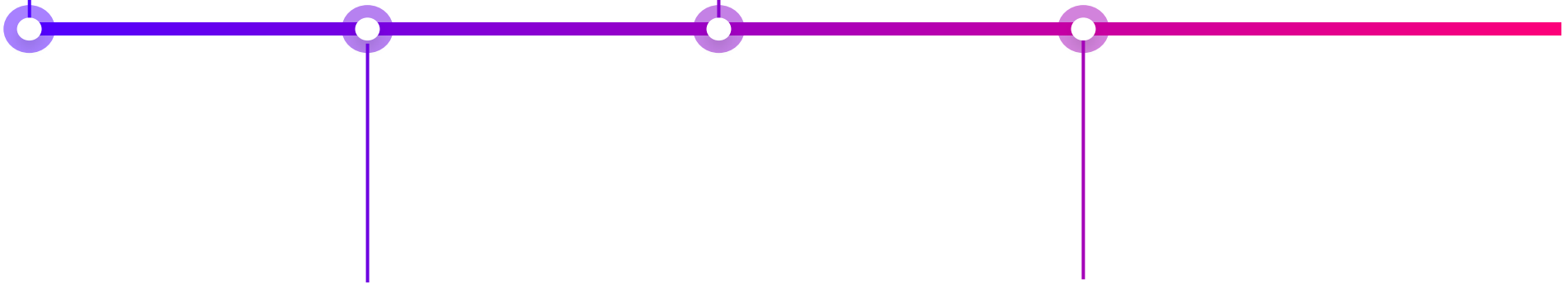
Bootstrapping ARM

- Significant efforts to prepare the platform
- Lack of maturity in ecosystem (especially in 2021, 2022)
- Performance not always as good as hoped
- **Promising results when considering cost**

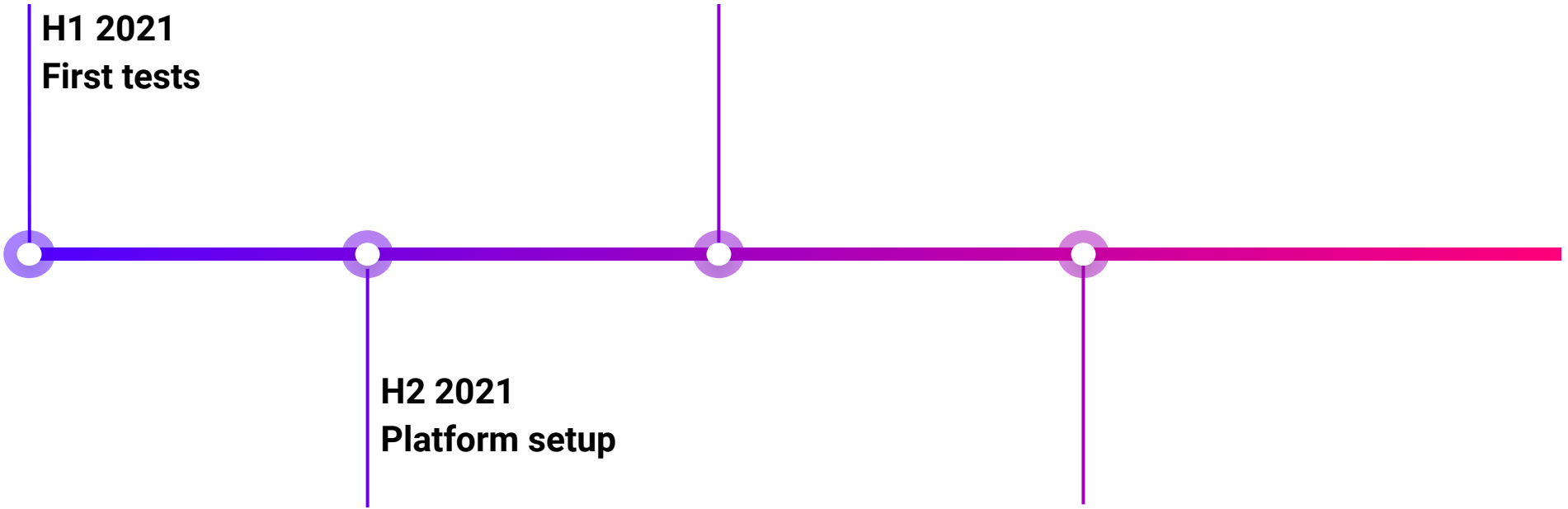
Day 1: Boosting ARM adoption

Timeline

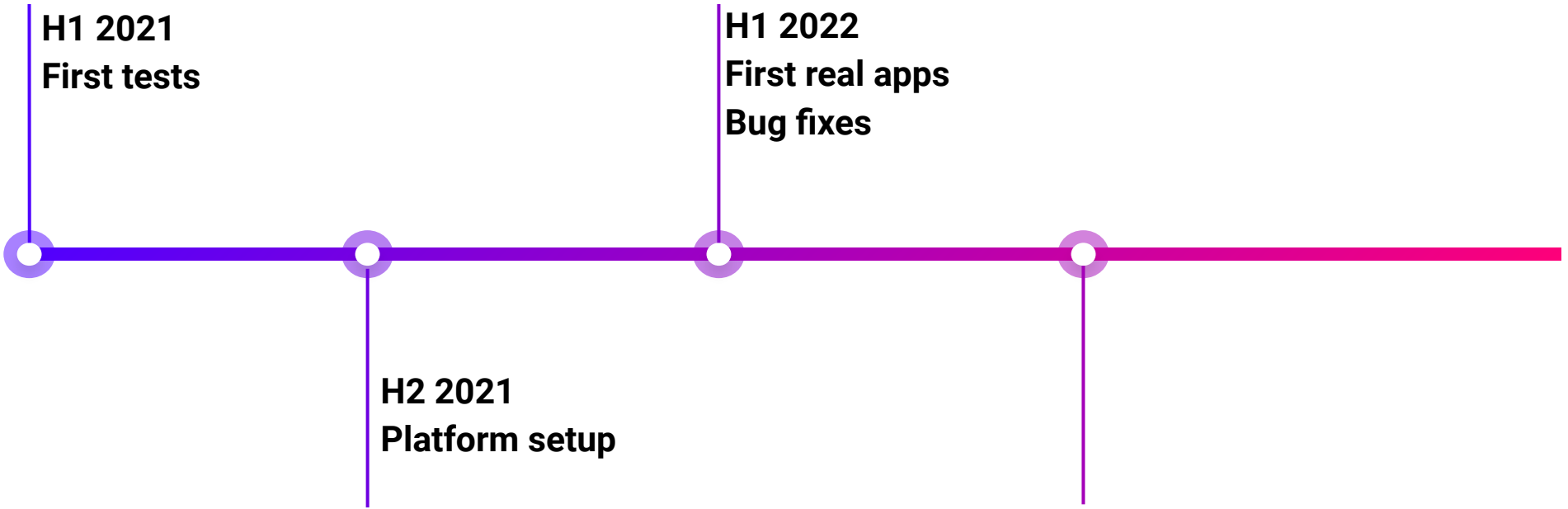
H1 2021
First tests



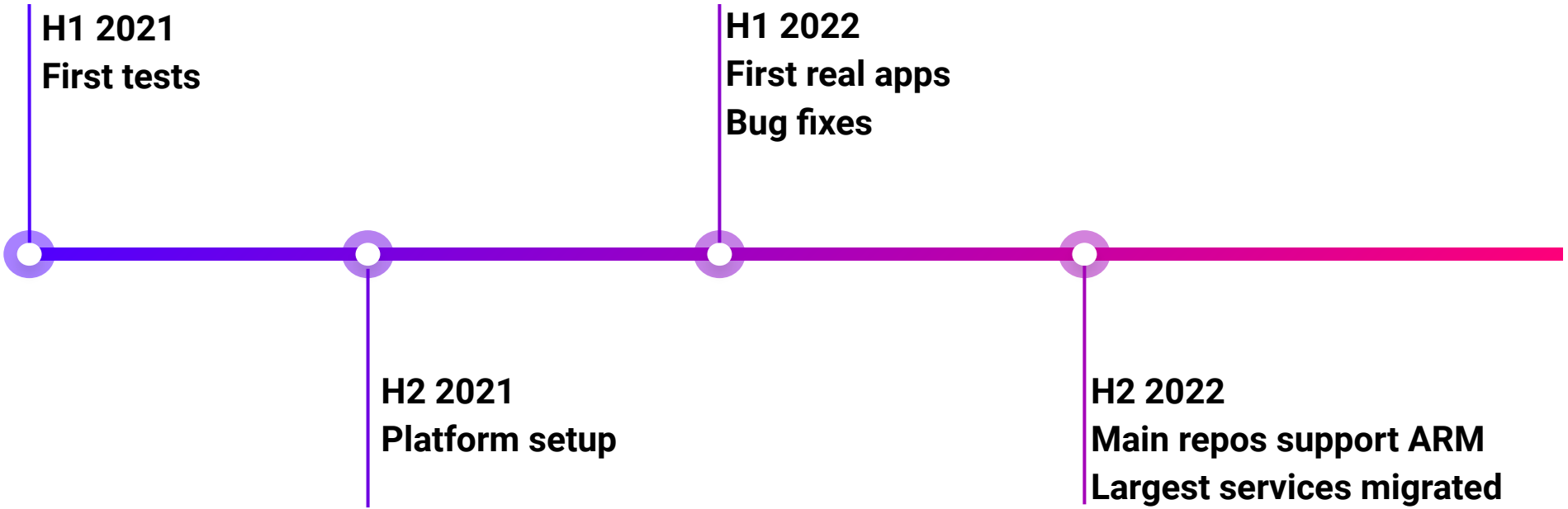
Timeline



Timeline


H1 2021
First tests
H2 2021
Platform setup
H1 2022
First real apps
Bug fixes

Timeline

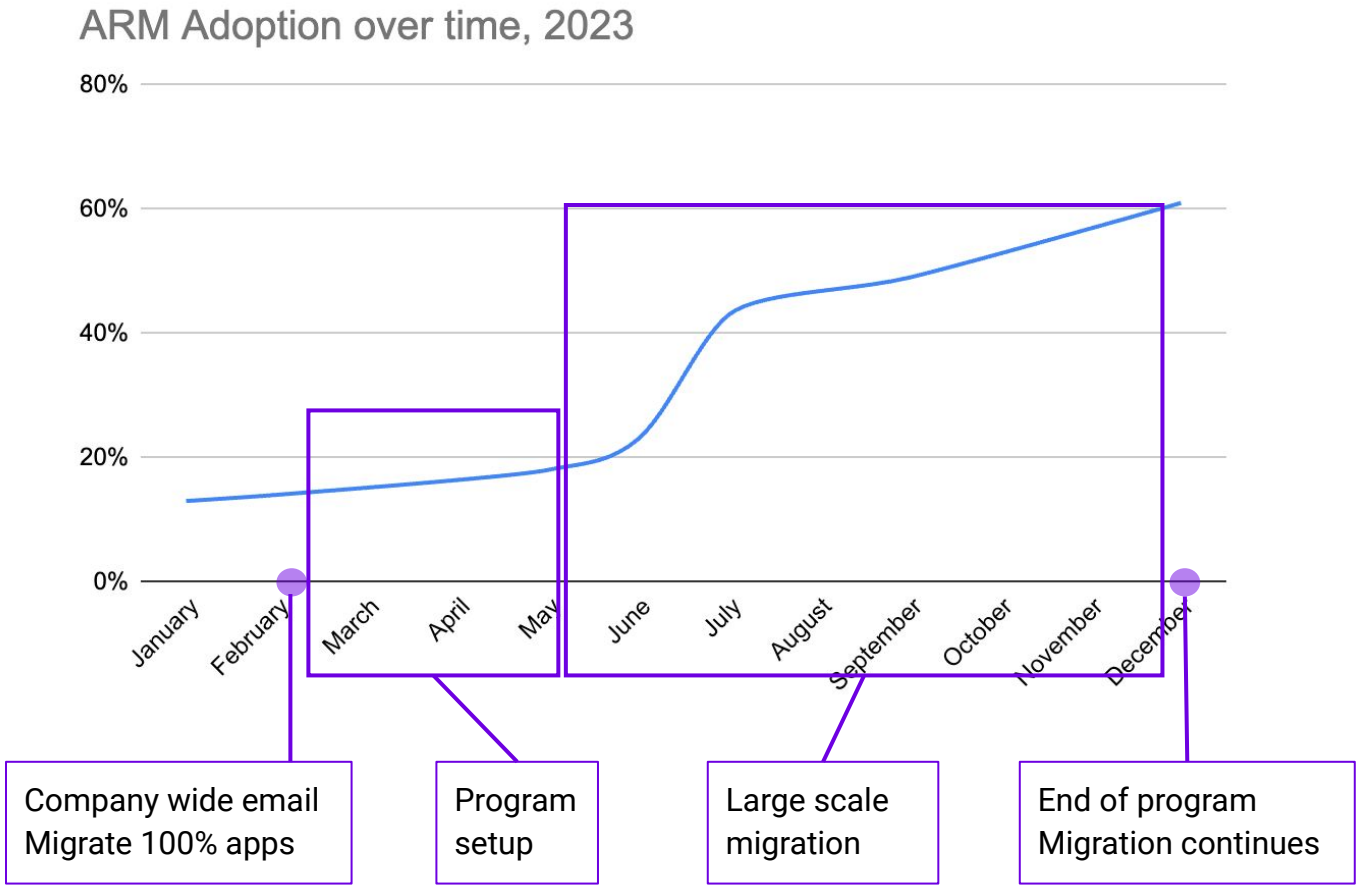

H1 2021
First tests
H2 2021
Platform setup
H1 2022
First real apps
Bug fixes
H2 2022
Main repos support ARM
Largest services migrated

Large scale migration program

> Our goal is to migrate 100% of services on AWS workloads to Arm by the **end of 2023**, with any exceptions documented and approved.

Early 2023

ARM migration timeline



As of mid 2024 (AWS)

Proportion of ARM cores: **~70%**

Savings (compute): **~10%**

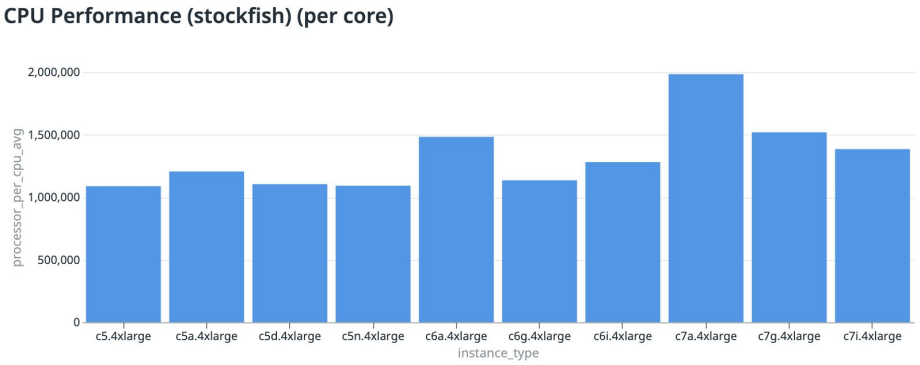
Day 2: What’s next?

Picking the best instance types

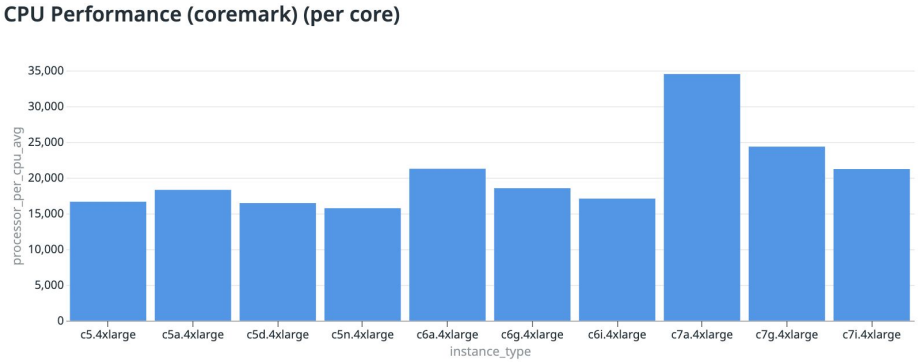
Initial migration is done but

- What about newer instance types?
- What about AMD?
- What about other cloud providers?

Automated synthetic benchmarks



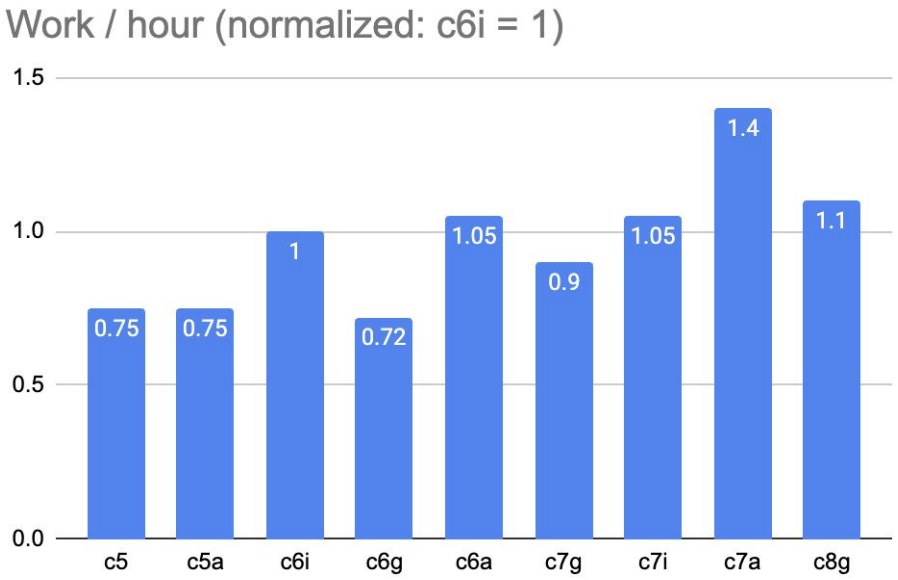
Stockfish



Coremark

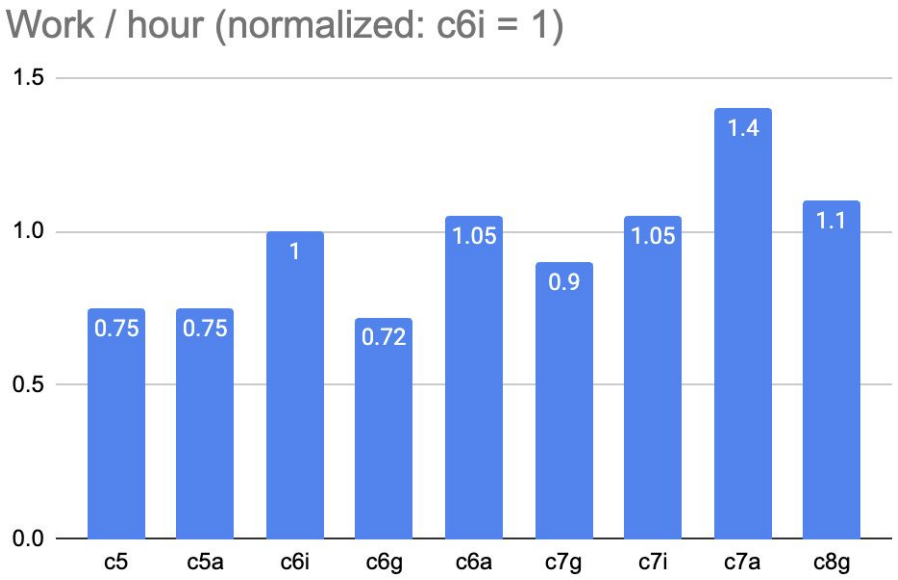
Comparison of 4xlarge instance types

What about real applications?

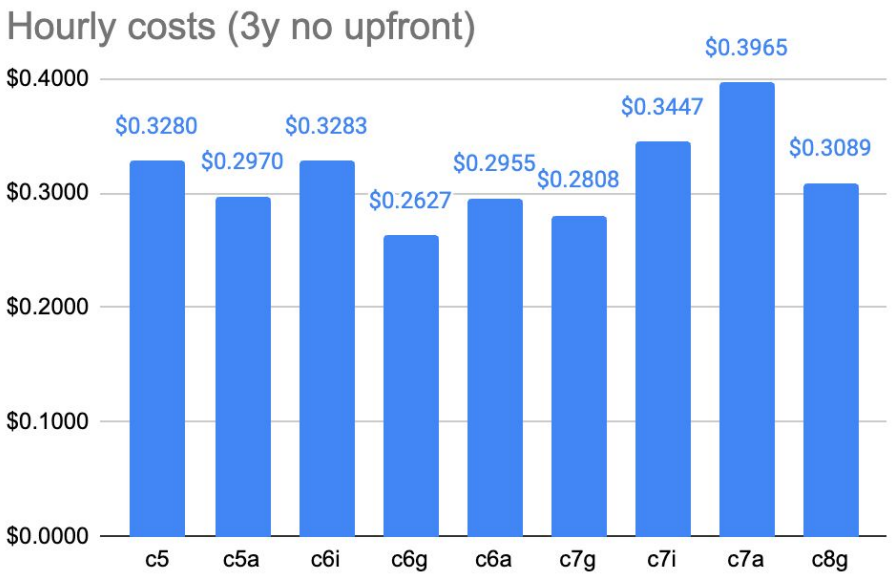


Performance of a
specific application

What about real applications?

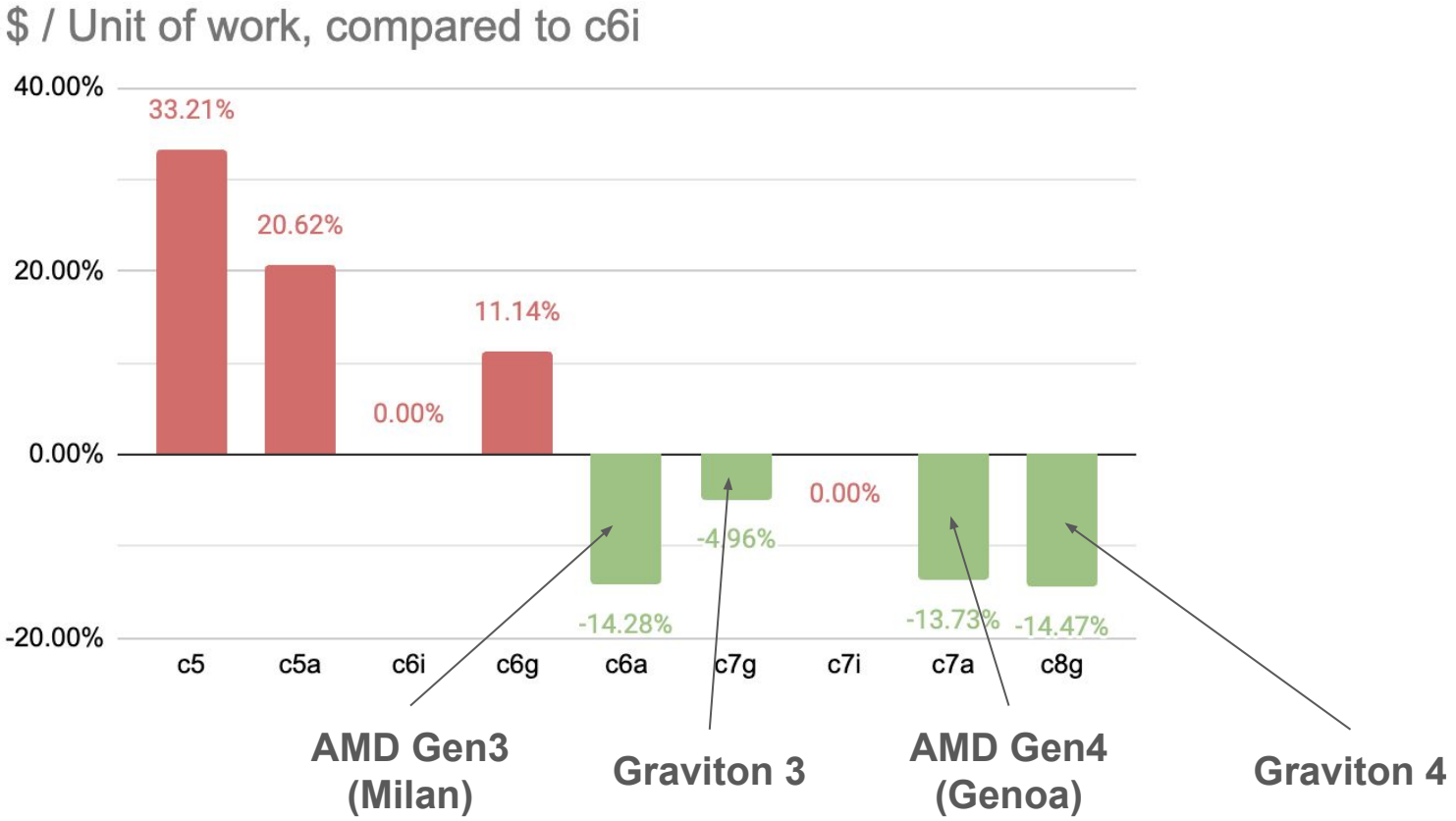


Performance of a [specific application]



List prices

Find the most cost efficient instance **for this app**



Simple high-level guidelines

- Not CPU bound

  ⇒ **Graviton 2**

- CPU bound & can autoscale

  ⇒ **AMD Gen 3/4 or Graviton 4**

- Need highest single core performance

  ⇒ **Intel**

Parting words

Takeaways

- Invest in Build/CI early
- Test your applications, synthetic benchmarks only go so far
- Be careful with capacity planning
- ARM is here to stay
  - GCP announced Axion CPUs
- ARM ecosystem has been improving fast
  - *But* we found a crazy LLVM bug in PostgreSQL (soon on our blog)

KubeCon | CloudNativeCon

North America 2024

Thank you!

We’re hiring
https://www.datadoghq.com/careers/

eric.mountain@datadoghq.com
laurent@datadoghq.com

Please scan the QR Code above to leave feedback on this session

Q&A