Fast and Efficient Container Startup at the Edge via Dependency Scheduling

Silvery Fu[1], Radhika Mittal[2], Lei Zhang[3], Sylvia Ratnasamy[1]
(1: UC Berkeley, 2: UIUC, 3: Alibaba Group)

Container Technologies are Popular

- Adopted in 2,000+ companies
- 160+ million container images
- 86% of containers are deployed on kubernetes
- Emerging frameworks and use cases in edge computing

Slow Start

Transfer container image ⏱️

- fetch image from a repository

Decompress and set up ⏱️

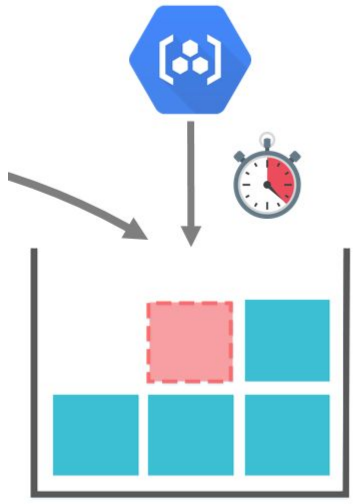
T: task time; S: startup time; R: running time

- $T = S + R; S \propto R$

λ

Short tasks suffer!

Startup Latency

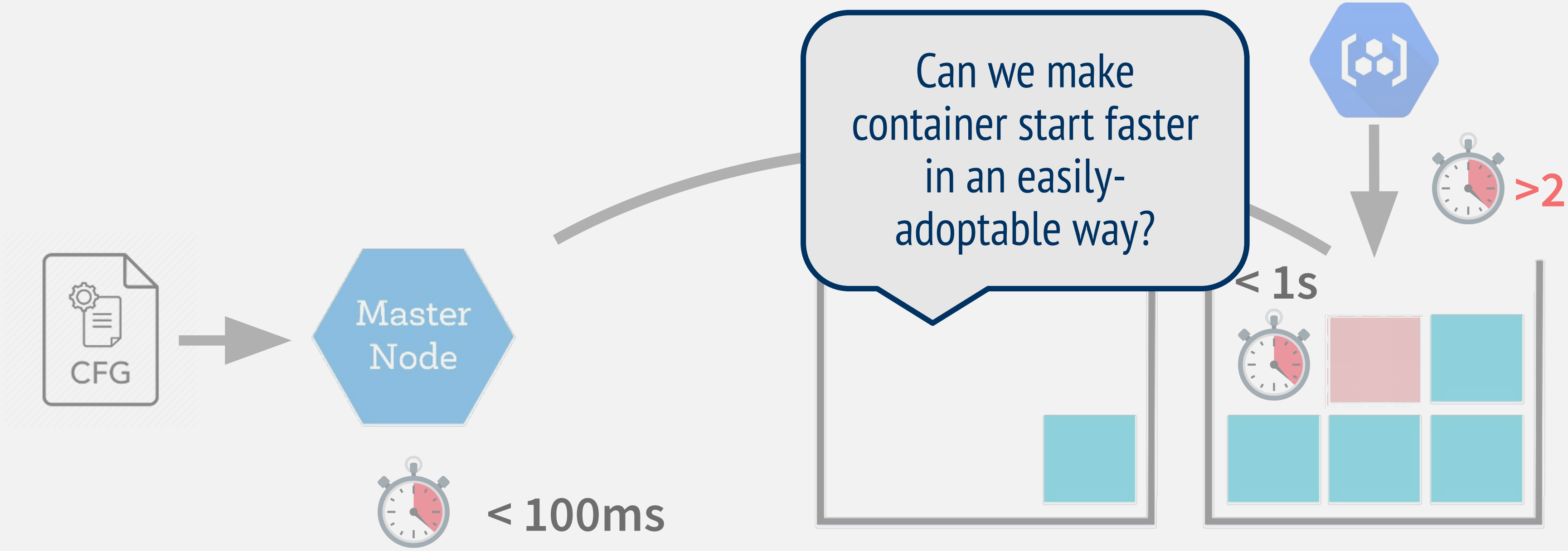


- Profile dependency pulling:
  - Trace: *56k, 33TB* images
  - Amazon ECR, m4.xlarge
  - Average *image pulling* latency is <u>19.2 seconds</u>

- An *<u>image</u>* includes all container dependencies, including binaries, code, configurations files.

Deploying Containers

Cloud experiment with high-speed networks and powerful machines!



Can we make container start faster in an easily-adoptable way?

Trace: *56k, 33TB* images
Amazon EC2

Scheduling latency

Booting Latency

Pulling Latency

Can we avoid pulling images?

Design 1: Image-aware Placement

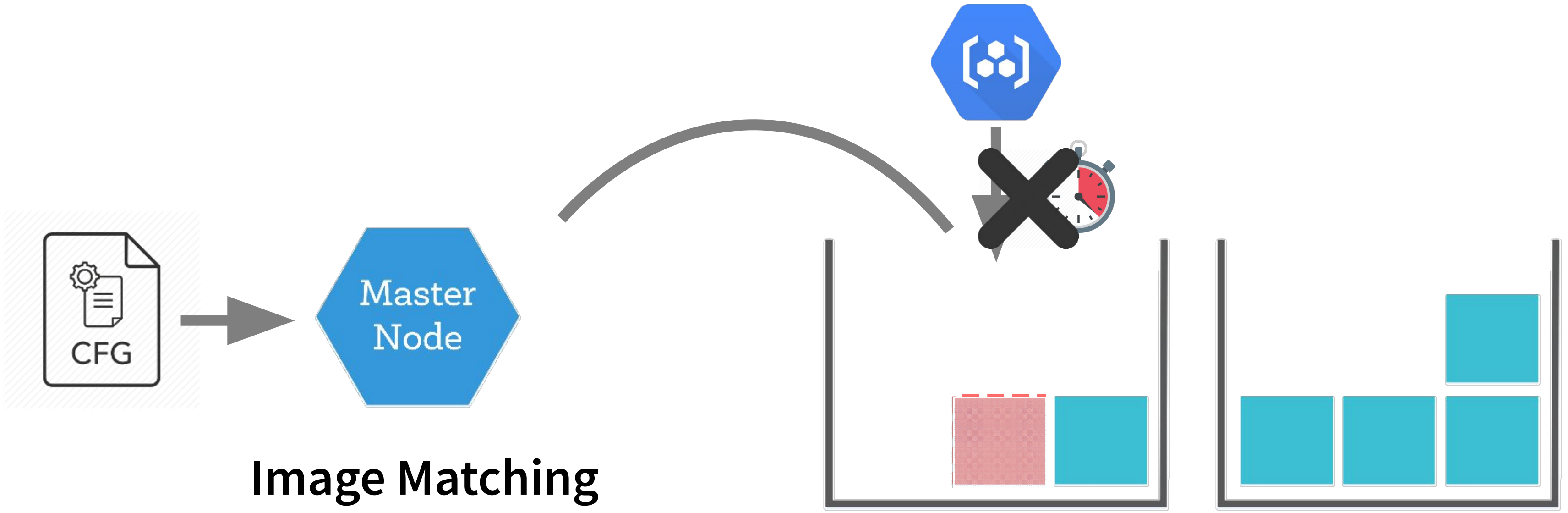
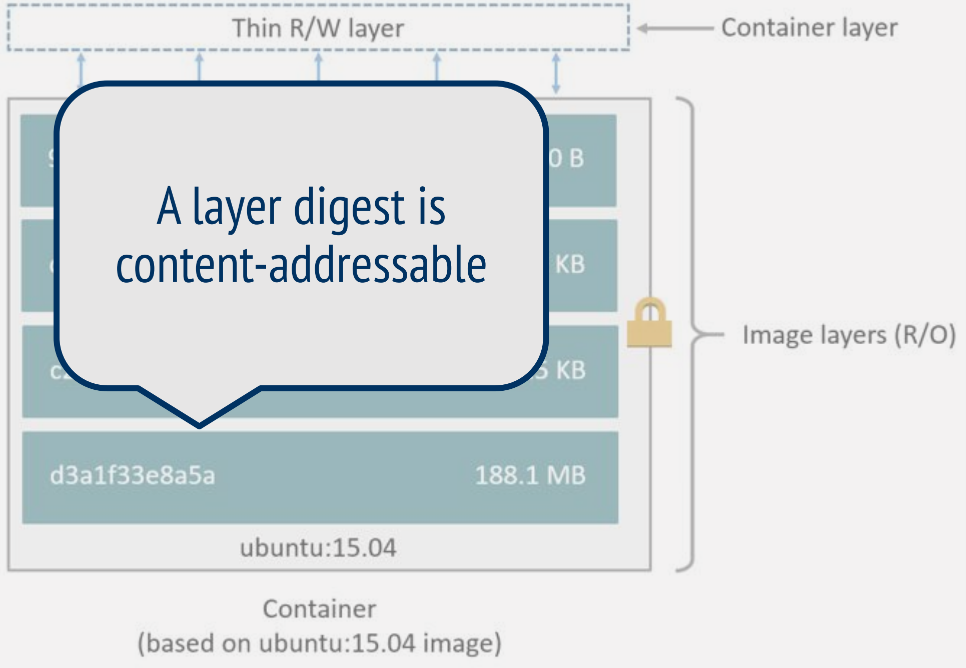


Image Matching

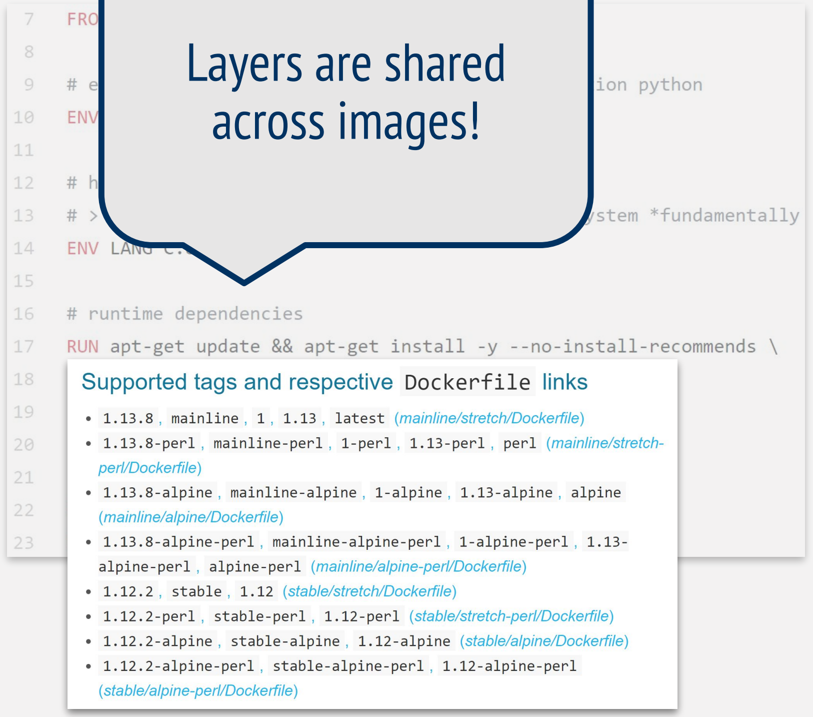
- Issues:
  - binary decision
  - image name changes

Can we do better than matching image?

Layer View

- 

A layer digest is content-addressable



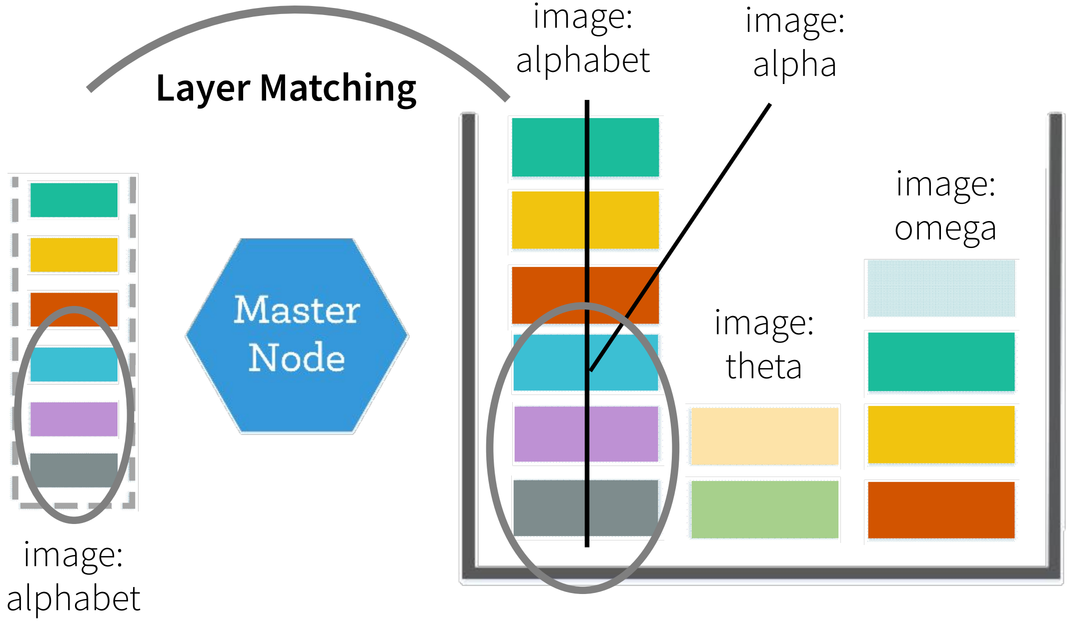
```
FRO

# e                                          ion python
ENV

# h
# >                                          ystem *fundamentally
ENV LANG C.

# runtime dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
```

Supported tags and respective `Dockerfile` links

- `1.13.8`, `mainline`, `1`, `1.13`, `latest` *(mainline/stretch/Dockerfile)*
- `1.13.8-perl`, `mainline-perl`, `1-perl`, `1.13-perl`, `perl` *(mainline/stretch-perl/Dockerfile)*
- `1.13.8-alpine`, `mainline-alpine`, `1-alpine`, `1.13-alpine`, `alpine` *(mainline/alpine/Dockerfile)*
- `1.13.8-alpine-perl`, `mainline-alpine-perl`, `1-alpine-perl`, `1.13-alpine-perl`, `alpine-perl` *(mainline/alpine-perl/Dockerfile)*
- `1.12.2`, `stable`, `1.12` *(stable/stretch/Dockerfile)*
- `1.12.2-perl`, `stable-perl`, `1.12-perl` *(stable/stretch-perl/Dockerfile)*
- `1.12.2-alpine`, `stable-alpine`, `1.12-alpine` *(stable/alpine/Dockerfile)*
- `1.12.2-alpine-perl`, `stable-alpine-perl`, `1.12-alpine-perl` *(stable/alpine-perl/Dockerfile)*

Layers are shared across images!

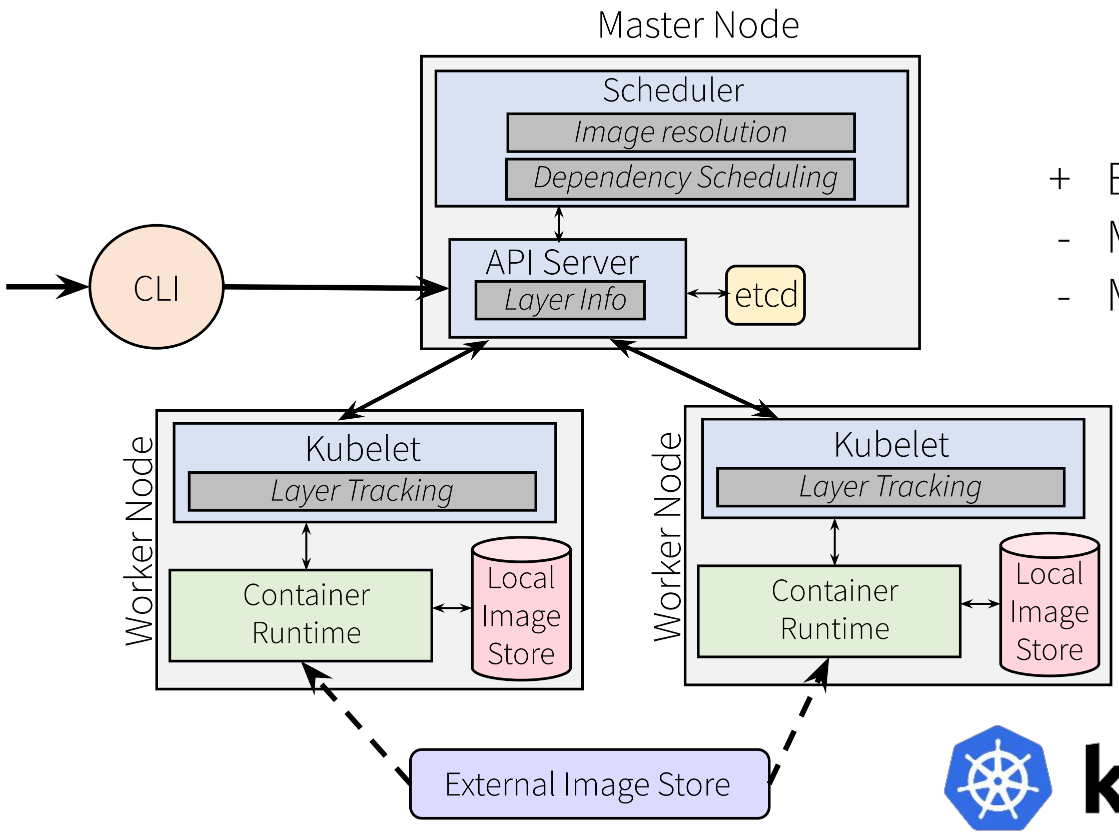
Design 2: Layer-aware Placement



Are the required changes easily adoptable?

kubernetes

k8s layer-aware



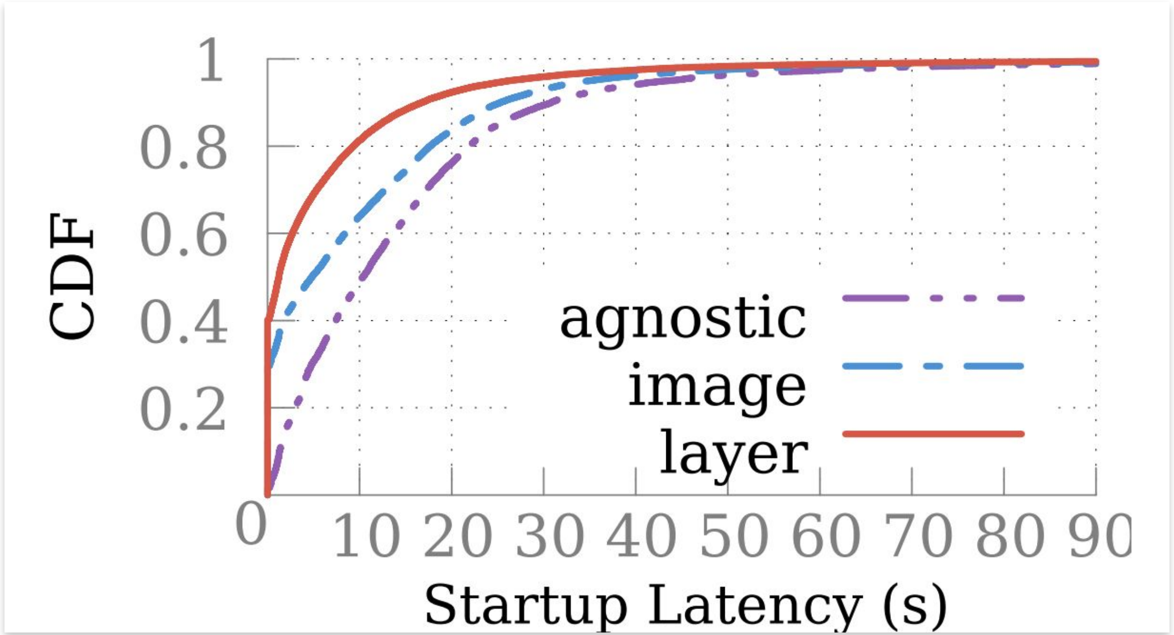
+ Better performance
- More API changes
- More overhead

kubernetes

Results

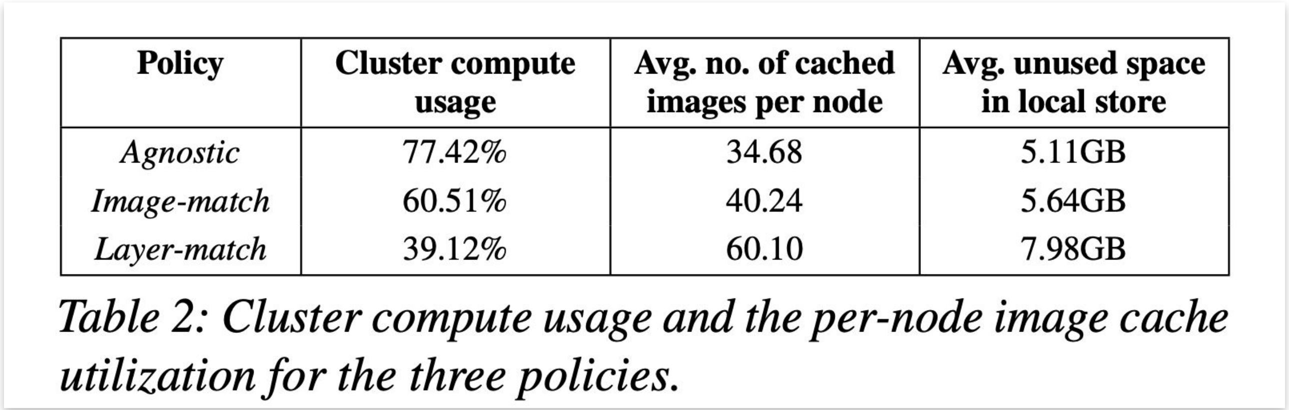
kubernetes

Faster Startup



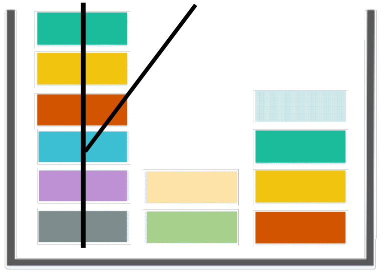
- Setup:
  - 200 nodes
  - 32GB image storage
  - 80% utilization
  - Zipf distribution
- Improvements on avg. startup latency:
  - 1.4x smaller (image)
  - 2.3x smaller (layer)

Resource Efficiency



Policy	Cluster compute usage	Avg. no. of cached images per node	Avg. unused space in local store
Agnostic	77.42%	34.68	5.11GB
Image-match	60.51%	40.24	5.64GB
Layer-match	39.12%	60.10	7.98GB

Table 2: Cluster compute usage and the per-node image cache utilization for the three policies.



- Smaller compute usage: 1.3x (image) and 2x (layer)
- More spare storage (excluding container images):
  - 1.1x (image) and 1.6x (layer)

Open questions

- T: task time; S = startup time; R = running time
  - $T = S + R$; $S \propto R$

  Short tasks suffer!

  in real-world? (..need categorization of edge workloads)

- What are the implications of resource efficiency gains and startup latency reductions?

- What are the (other) forms of data locality issues at the edge?

Open questions

System-wise:

- How to balance dep. scheduling and the other scheduling policies?
- How much overhead (e.g., on the node-master communication, the apiserver,)?
- ..

Summary

- Containers and container images are the emerging tools to facilitate software reuse in deployment.
- Such reuse can lead to substantial dependency sharing between containers.
- Dependency-aware scheduling exploits such sharing, and is highly effective in cutting container startup latency.

Thank you!

silvery@eecs.berkeley.edu