Booking.com

Luka Stout



Emir Beganović



When partitioning beats sharing

Ephemeral MicroVMs for CI at scale

Agenda

CPU throttling

What we built

Down to silicon

Did it work?

Reliability & limits

**This talk will not feature any AI*

CI at Booking.com

GitLab

Docker executor

~3,000

developers

10M+

jobs per month

3

data centers,
bare metal

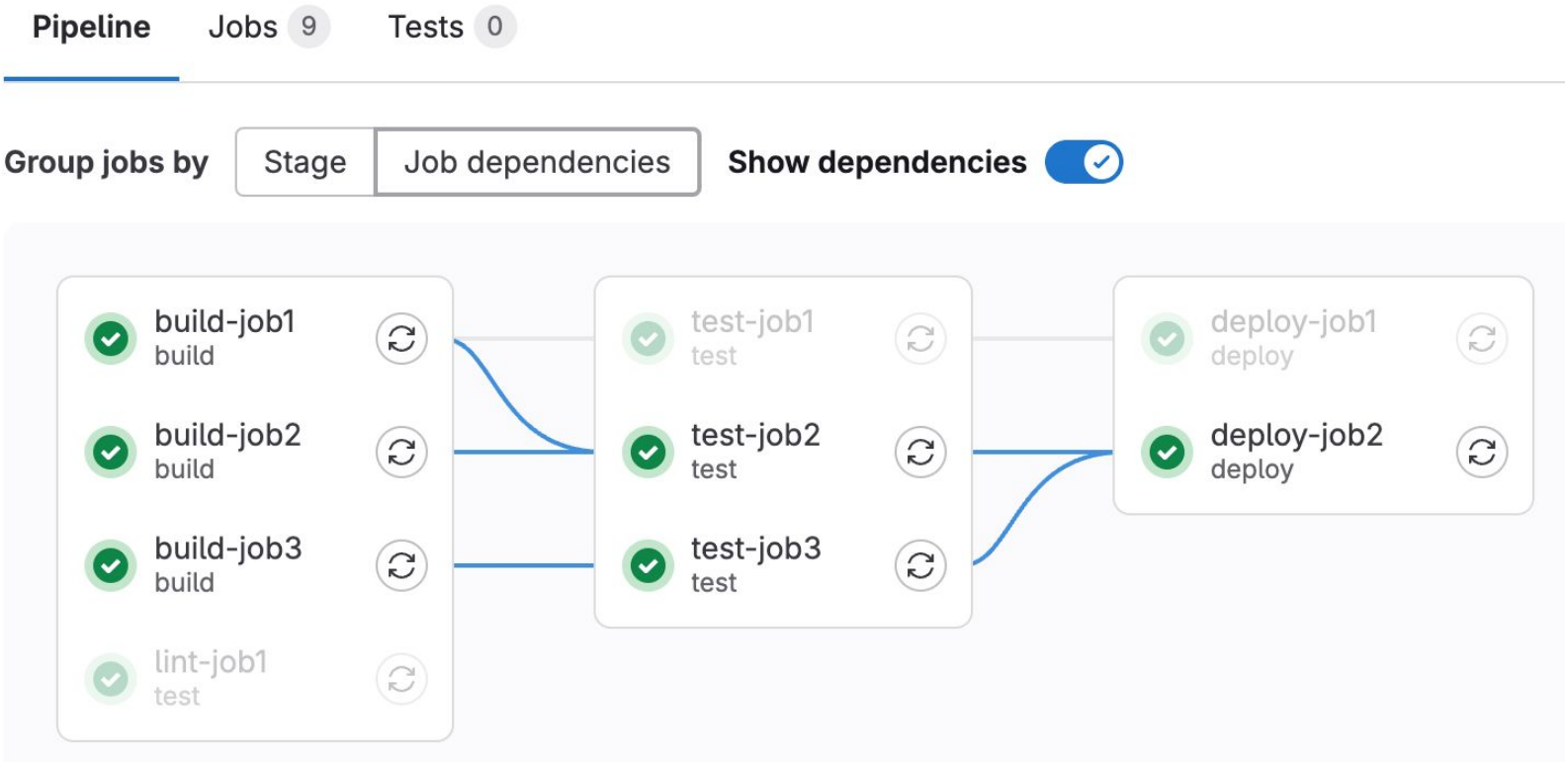
70

hosts

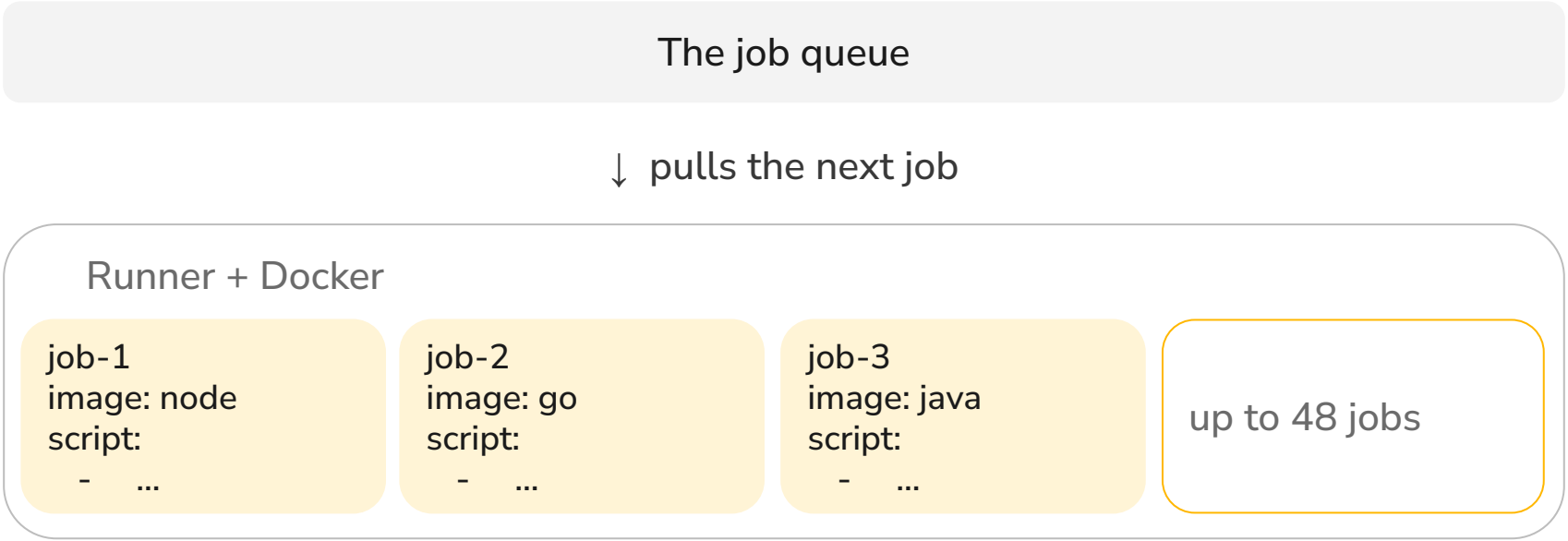
13,440

threads

GitLab CI: pipelines, stages, jobs



What we started out with



What we started out with

The job queue

↓ pulls the next job

Runner + Docker

job-1
image: node
script:
  - ...

job-2
image: go
script:
  - ...

job-3
image: java
script:
  - ...

up to 48 jobs

```
config.toml
[[runners]]
    executor = "docker"
    [runners.docker]
        cpus = "4"
```

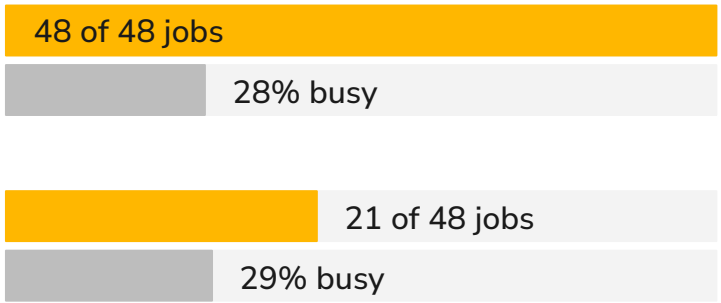
48 jobs × 4 CPUs = 192

every thread on the host is promised to a job

Idle host, slow builds

Host load

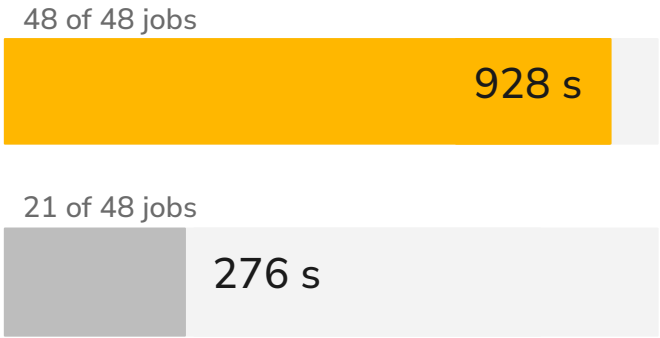
A host - 96 cores / 192 threads



Yellow: job slots in use. Grey: host CPU%.

Job duration

same host, same day



Why is a four-CPU job starved on a host that is ~70% idle?

Booking.com

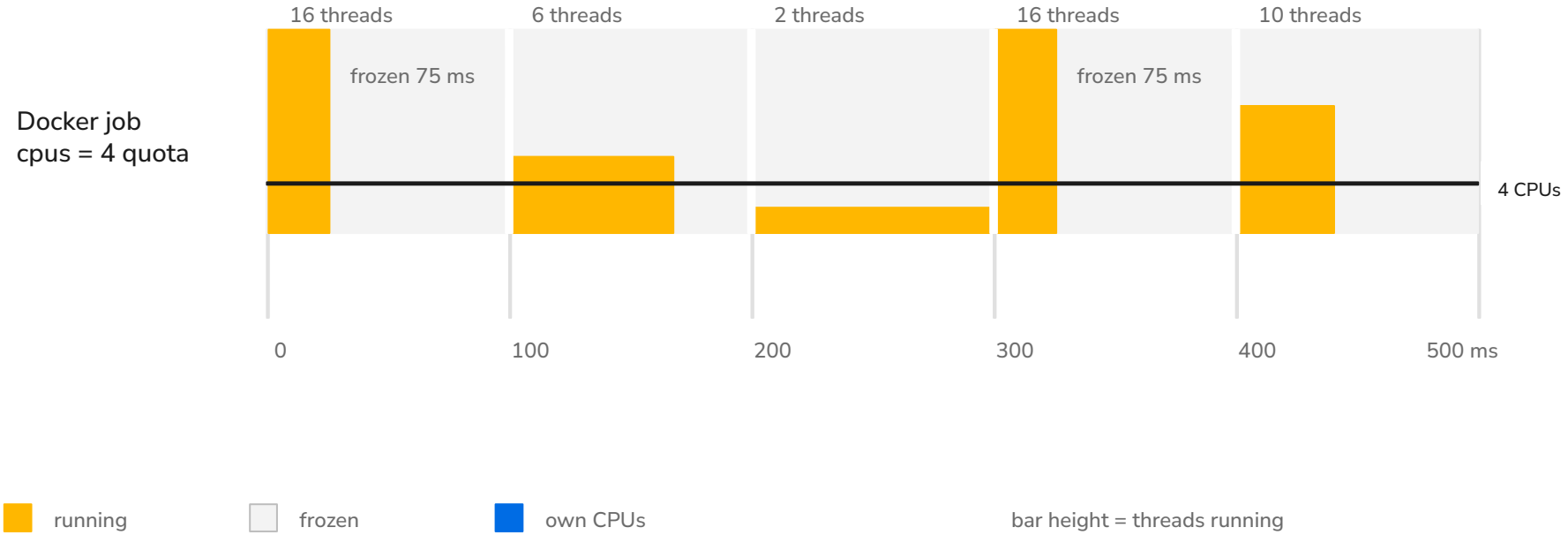
Booking.com

01

CFS bandwidth throttling

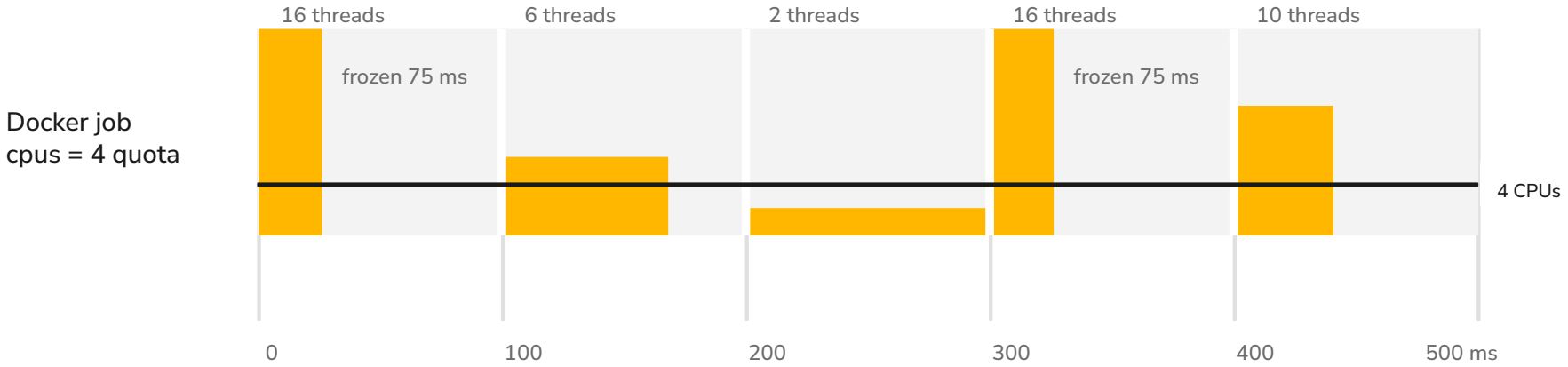
cpus = "4" means 400 ms of CPU time every 100 ms

One budget for the entire job, shared by every thread



cpus = "4" means 400 ms of CPU time every 100 ms

One budget for the entire job, shared by every thread

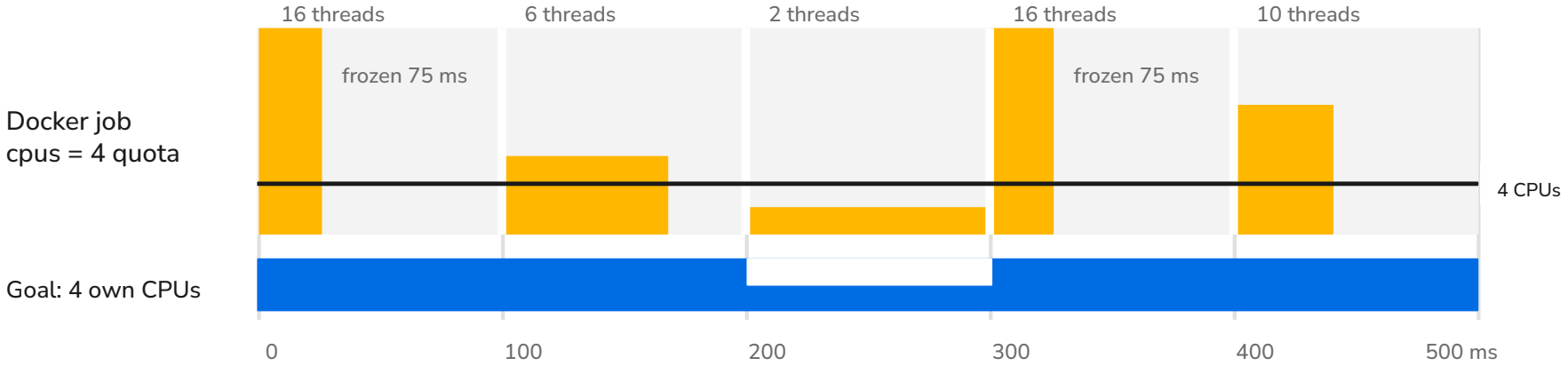


16 threads use up 400 ms in 25 ms and then all 16 wait 75 ms.

running frozen own CPUs bar height = threads running

cpus = "4" means 400 ms of CPU time every 100 ms

One budget for the entire job, shared by every thread



16 threads use up 400 ms in 25 ms and then all 16 wait 75 ms.

running frozen own CPUs bar height = threads running

Known unknowns: 1000s of repositories

Inside the job: theirs

build tools
thread pools
test runners

sized for the 192 threads they can see

Around the job: ours

cpus = 4 quota
throttling counters

one per job, kept by the kernel

So we measured what we control: how often the kernel froze each job.

One third of our CPU periods were throttled!

Whole fleet, 7 days

36%

of all CFS periods throttled

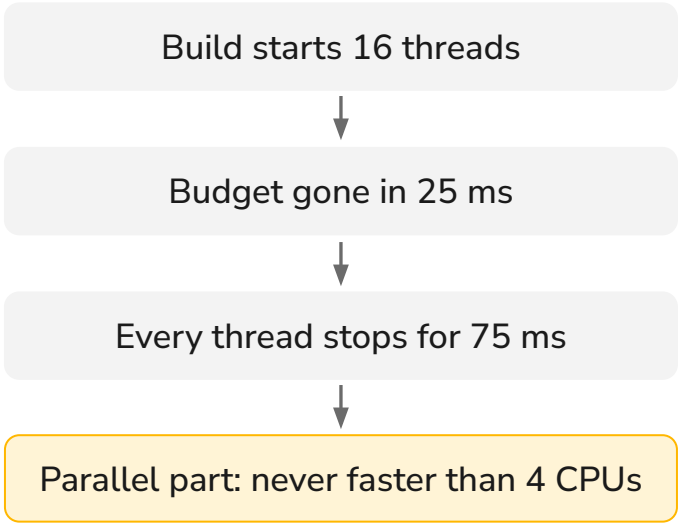
One Android monorepo

42%

of its CPU time spent frozen:
44 minutes for every hour of work

≈ 71,400 CPU hours a month frozen across the fleet

Amdahl's Law: more cores only help the parallel part



More CPUs only speed up the parallel part

Booking.com

02

Why not just tune it?

Config knobs that GitLab runners give us

`cpus`	throttles later
`cpu_shares`	noisy neighbours
`cpuset_cpus`	one set for all jobs
`limit`	more hosts

What we think a CI job needs

Not four CPUs of quota

Not a fair share of a busy host

Four CPUs nobody else is on.

Its own kernel

Fresh for every job

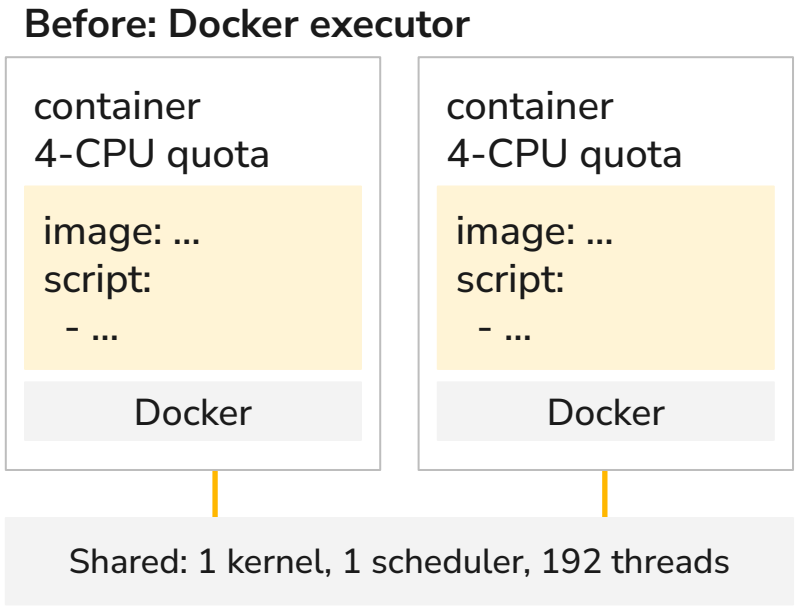
Full hardware isolation

= a virtual machine

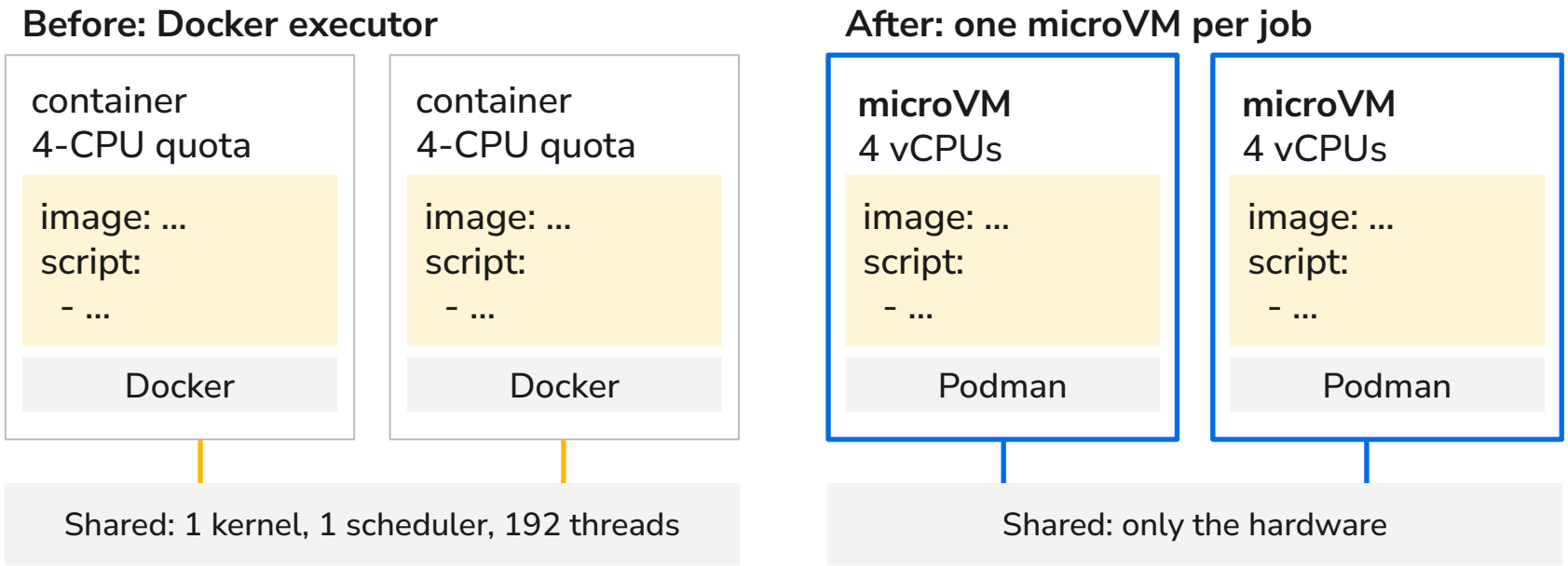
A microVM is a VM with the fat cut off

Full VM	MicroVM runners
Your workload	Your workload
Full guest OS: systemd, services	OCI Container
General-purpose kernel	Minimal kernel, built for our CI jobs
Bootloader	no bootloader: kernel loaded directly
BIOS / UEFI	no firmware
Emulated PC: PCI, VGA, USB	virtio disk + network only
~20 s to job	**~1.4 s to job**

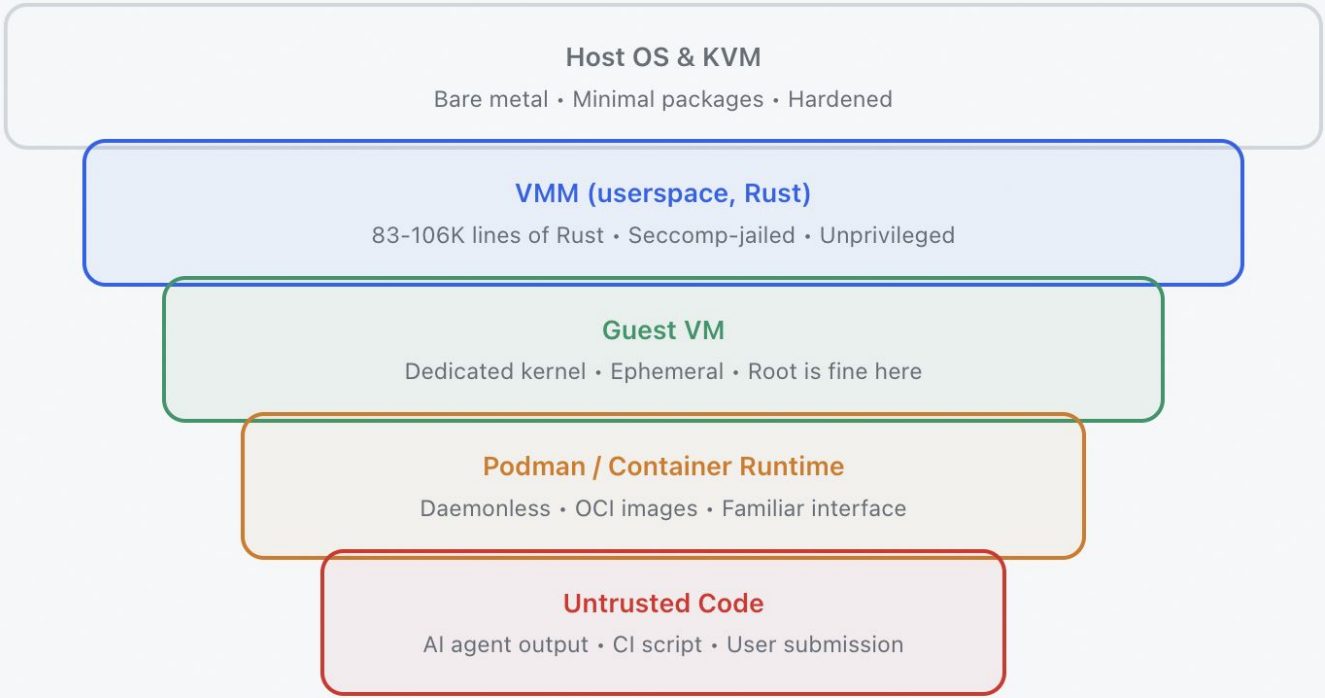
What we built



What we built



DEFENSE IN DEPTH



Booking.com

Developers must not notice

Hyrum's Law

“With enough users, every observable behaviour of your system becomes somebody's dependency.”

What 3,000+ developers depend on

docker build

images built inside the job

Testcontainers

real databases started by the test suite

/.dockerenv

Podman provided `/run/.containerenv` instead

100KB+ CI variable

bigger than a shell argument

Booking.com

Building deterministic CI from YAML down to silicon

One CI job's journey

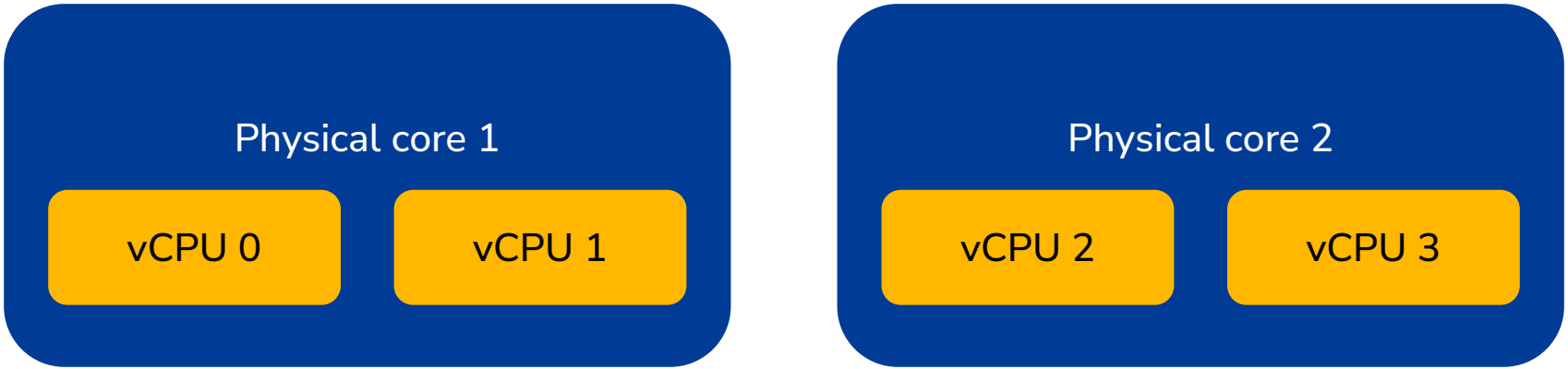


1. Pinned physical CPUs

2. Memory on the same NUMA node

Deterministic job placement in the scheduler

One job: two cores, four vCPUs



Guest sees the topology and there’s no CPU quota.

Universal Scalability Law

Contention and coordination can limit scaling.

Each job has its own:

Physical cores and threads
Fixed placement
No quota

Still shared:

L3 cache within a block
Memory bandwidth
Disk and network I/O

Less interference, but not a complete isolation.

A fresh VM, ready for a job in ~1.4 seconds

Direct kernel boot

No firmware or bootloader

XFS reflinks

Clone, don't copy

Pre-warmed

Images and caches ready

Kernel up: ~200 ms

Job ready: ~1.4 s

Cloud Hypervisor nested virtualisation for Android emulators

Booking.com

05

Did it work?

On an idle host, Docker wins

Docker, idle host



7,387

4 CPUs, each a full core

microVM, default size



3,903

4 CPUs, two cores plus their threads

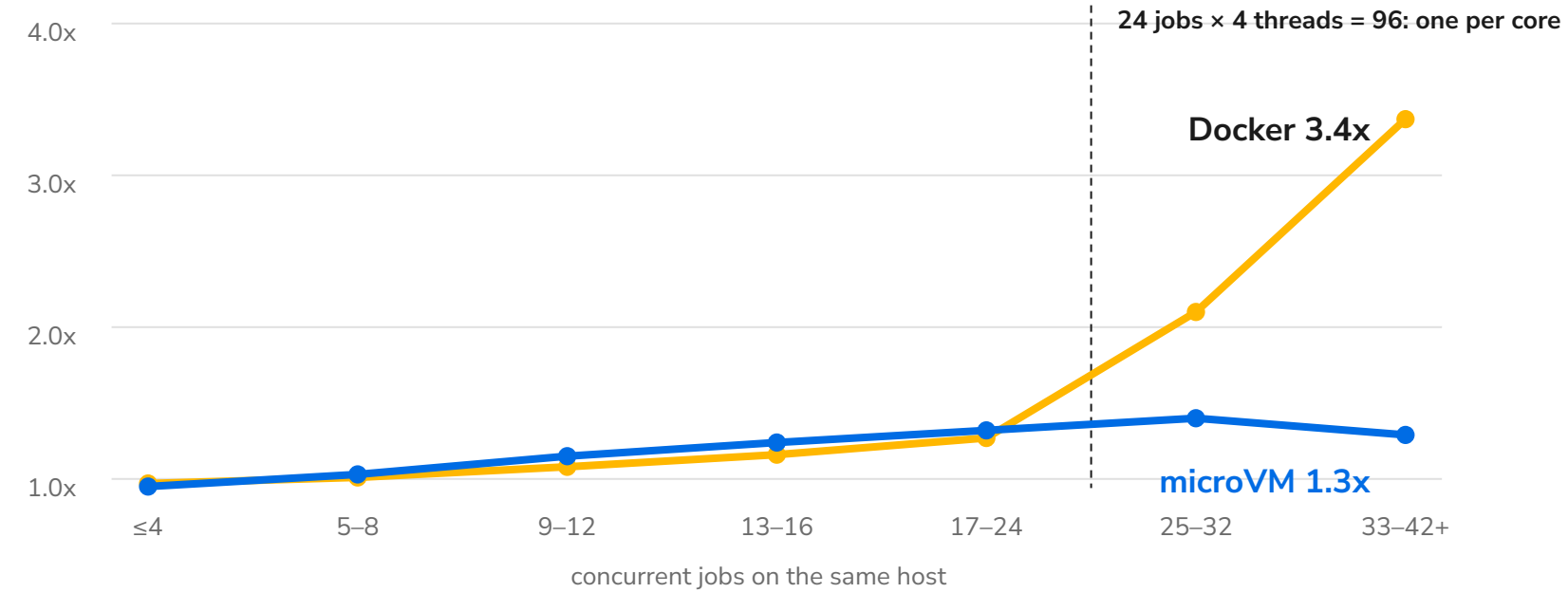
sysbench CPU, 4 threads, events/s

~47% less

Production hosts are not idle.

On a full host, microVMs win

1.19M jobs, each compared to its own median



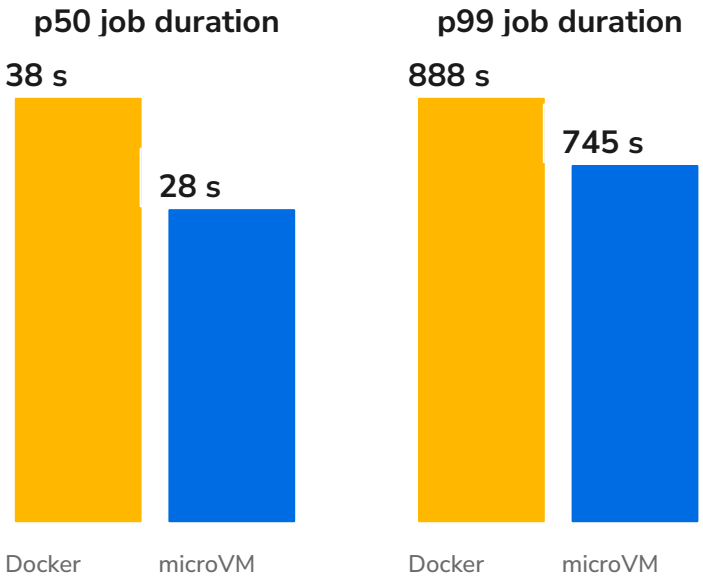
Little's Law: shorter jobs mean fewer jobs in flight

$$L = \lambda W$$

L	λ	W
jobs in flight	arrival rate	job duration

Shorter W means fewer jobs in flight, and fewer servers.

Fleet-wide, compute time fell 17%



−17%

compute time

2,000 job types, both fleets, same week

~11,700 CPU hours a month

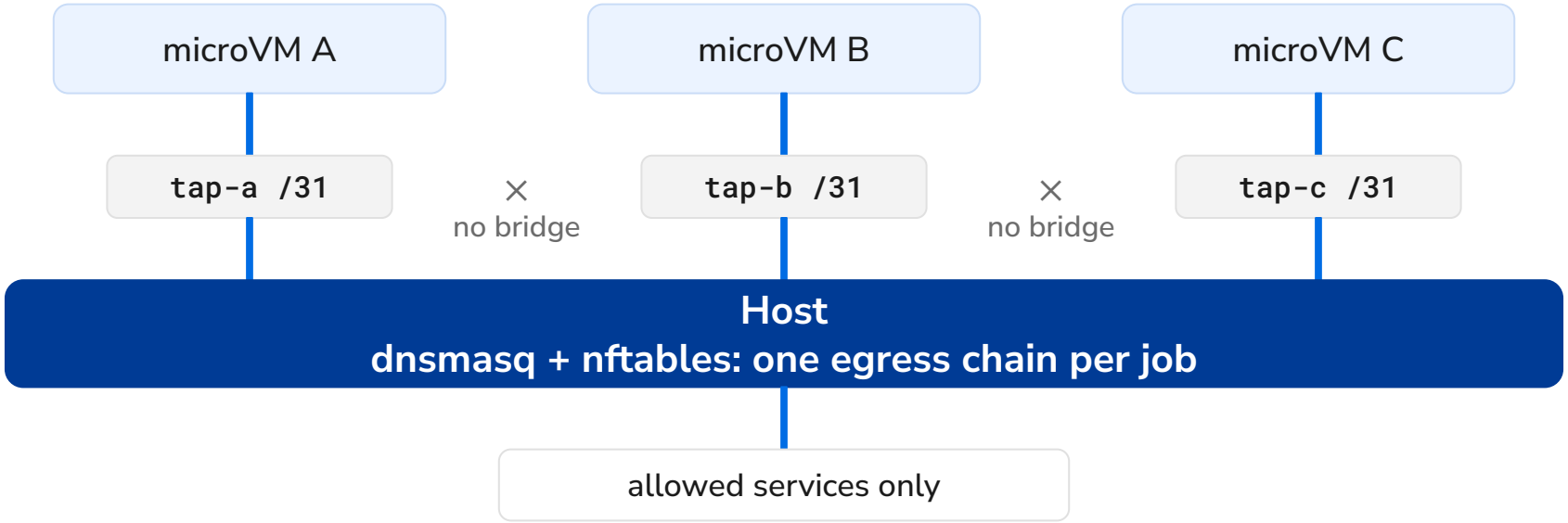
Booking.com

What else microVMs bought us

A kernel per job is a security boundary

Container escape	lands in a throwaway hardware isolated VM vs on the host
Root in the job	stays inside the job's VM, never on the host
Secrets and files	gone when the VM is thrown away
Attack surface	minimal kernel, virtio only, VMM in Rust

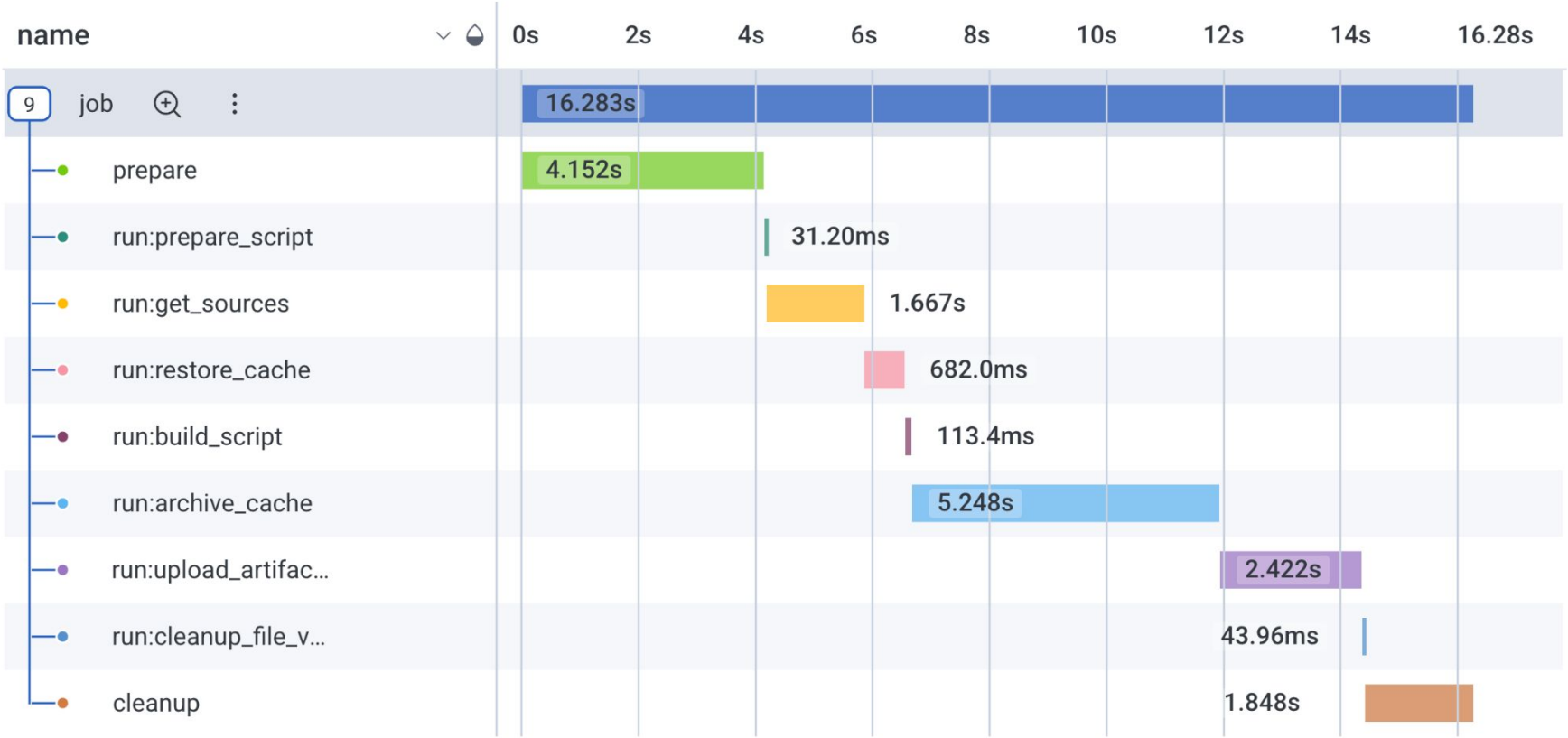
Network isolation by topology



Booking.com

07

Making it nice & reliable


name
0s
2s
4s
6s
8s
10s
12s
14s
16.28s
9
job
16.283s
prepare
4.152s
run:prepare_script
31.20ms
run:get_sources
1.667s
run:restore_cache
682.0ms
run:build_script
113.4ms
run:archive_cache
5.248s
run:upload_artifac...
2.422s
run:cleanup_file_v...
43.96ms
cleanup
1.848s

Full receipt of every job

```
23:38:20  VM resource usage
23:38:20  CPU: 3.5s total (2.0s user + 1.6s sys), 4 cores, 8% utilization over 12s
23:38:20  Peak Memory: 306 MB
23:38:20  Block I/O: 454.8 MB read, 66.7 MB written (2142 read ops, 121 write ops)
23:38:20  Network: 9.4 MB download, 9.7 MB upload
23:38:20  DNS queries
23:38:20  4 unique domains
23:38:20       5  gitlab.com
23:38:20       4  xxx.s3.dualstack.eu-central-1.amazonaws.com
23:38:20       1  xxx.booking.com
23:38:20       1  xxx.booking.com
23:38:20  DNS cache: 11/15 hits (73.3%), 4 forwarded
23:38:20  DNS upstream usage:
23:38:20    10.252.13.   .: 4 queries
23:38:20  11 total queries
```

Lessons from production

1. Empty cache is better than corrupted cache
2. Useless writes still land on real disks
3. Isolation doesn’t stop at VM level

Tradeoffs

You now operate

Guest kernel + hypervisor
CPU placement
VM networking

Keep containers for

Long-running services
Steady, cooperative workloads
Teams without VM capacity

+1.4 seconds of startup per job

Takeaways

1. Developers love predictable systems
2. Optimise for peaks, not for idle
3. Check if your containers are getting throttled!

Thank you!

Emir Beganović & Luka Stout

Questions?
#26emea-day3-track2

Booking.com