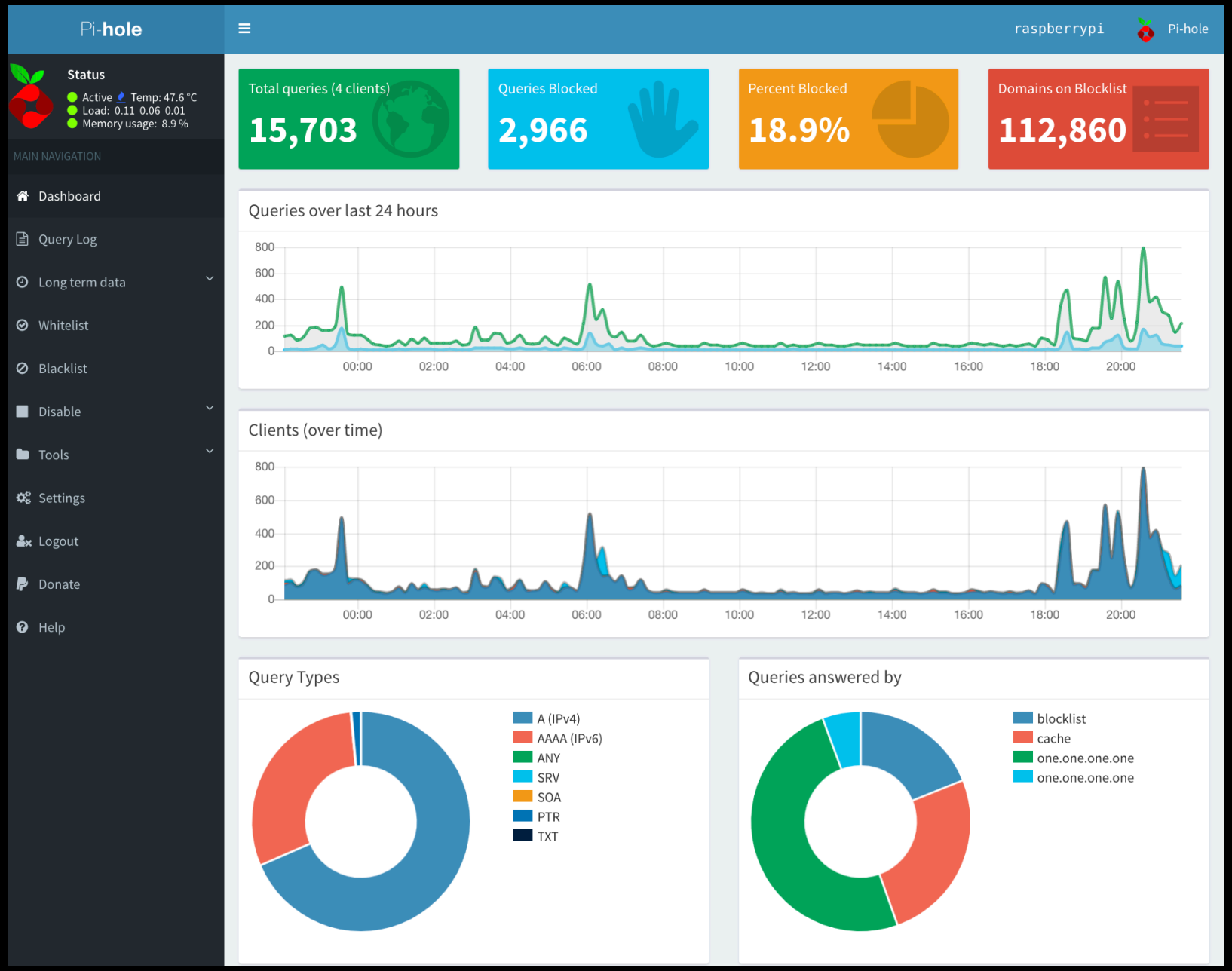
CONTAINER MONITORING & LOGGING WORKSHOP

dockercon19
SAN FRANCISCO

Goals of the Workshop

- [ ] Introduction
- [ ] Operations Models
- [ ] What to Monitor
- [ ] Build an ELK Stack
- [ ] Build a Prometheus Stack
- [ ] Best Practices & Recap

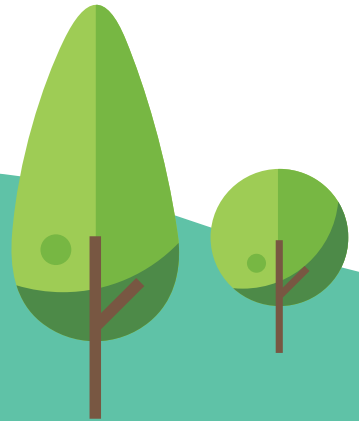
20%



DON'T LIMIT MONITORING & LOGGING TO INFRASTRUCTURE

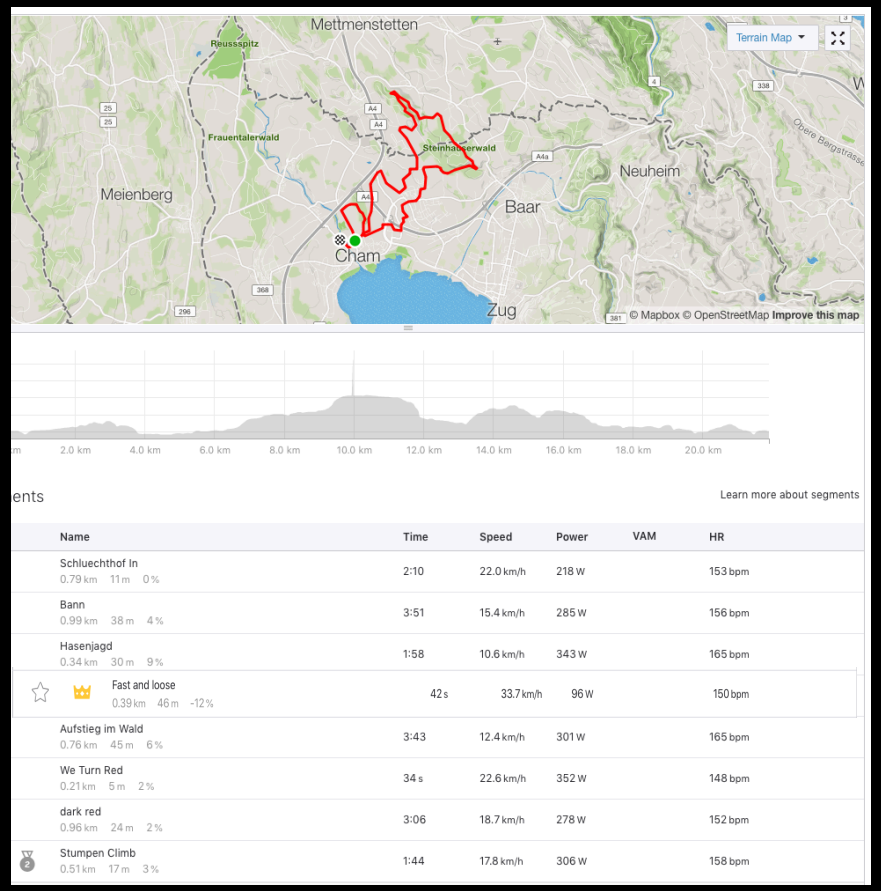
Brian Christner

SRE & Co-Founder
Docker Captain
56K.Cloud
brian@56k.cloud / @idomyowntricks




56K.
CLOUD
dockercon19
SAN FRANCISCO

Zugerberg, Zug, Switzerland



Name	Time	Speed	Power	VAM	HR
Schluechthof In 0.79 km 11 m 0 %	2:10	22.0 km/h	218 W		153 bpm
Bann 0.99 km 38 m 4 %	3:51	15.4 km/h	285 W		156 bpm
Hasenjagd 0.34 km 30 m 9 %	1:58	10.6 km/h	343 W		165 bpm
Fast and loose 0.39 km 46 m -12 %	42 s	33.7 km/h	96 W		150 bpm
Aufstieg im Wald 0.76 km 45 m 6 %	3:43	12.4 km/h	301 W		165 bpm
We Turn Red 0.21 km 5 m 2 %	34 s	22.6 km/h	352 W		148 bpm
dark red 0.96 km 24 m 2 %	3:06	18.7 km/h	278 W		152 bpm
Stumpen Climb 0.51 km 17 m 3 %	1:44	17.8 km/h	306 W		158 bpm

STRAVA Dashboard Training Explore Challenges Upgrade

Fast and loose / **Effort Comparison**

Effort Comparison

Share

Fast and loose

0.3km -12% Grade 46m

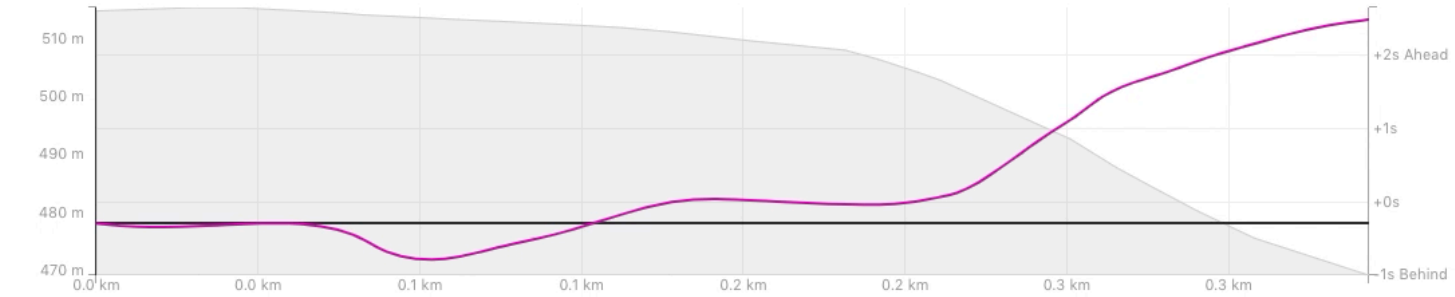
Your PR 42s
Brian Christner

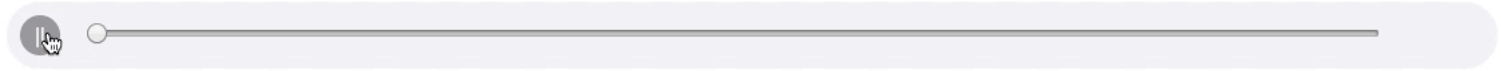
KOM 39s
Dani Mendler

Athletes	Time	Speed
You 10/20/18 - PR	0s	36.0 km/h
Dani Mendler 11/16/18 - KOM	0s	+4.0 km/h

Customize and compare up to five efforts and see where the race was won.

Get the Training Pack

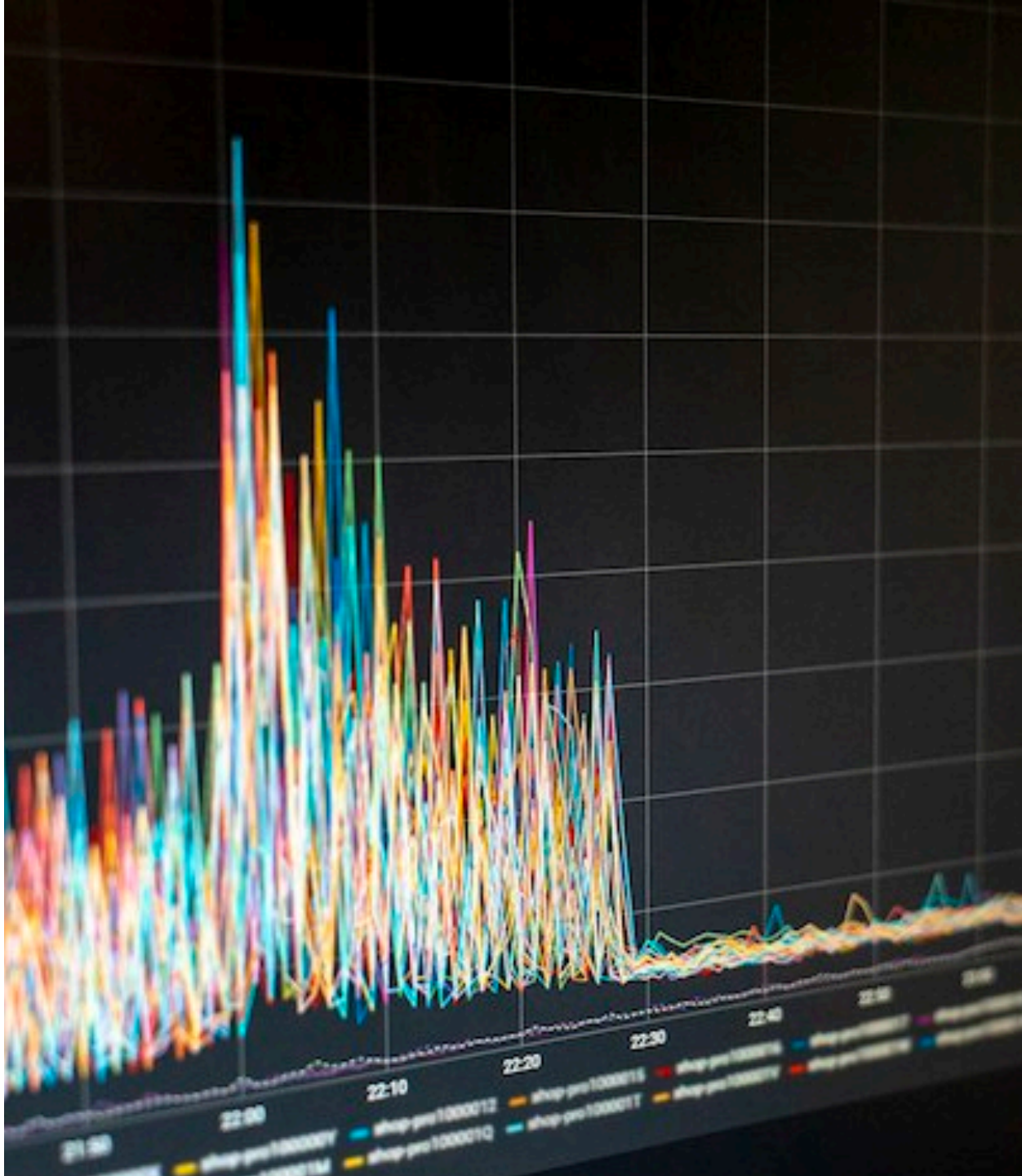




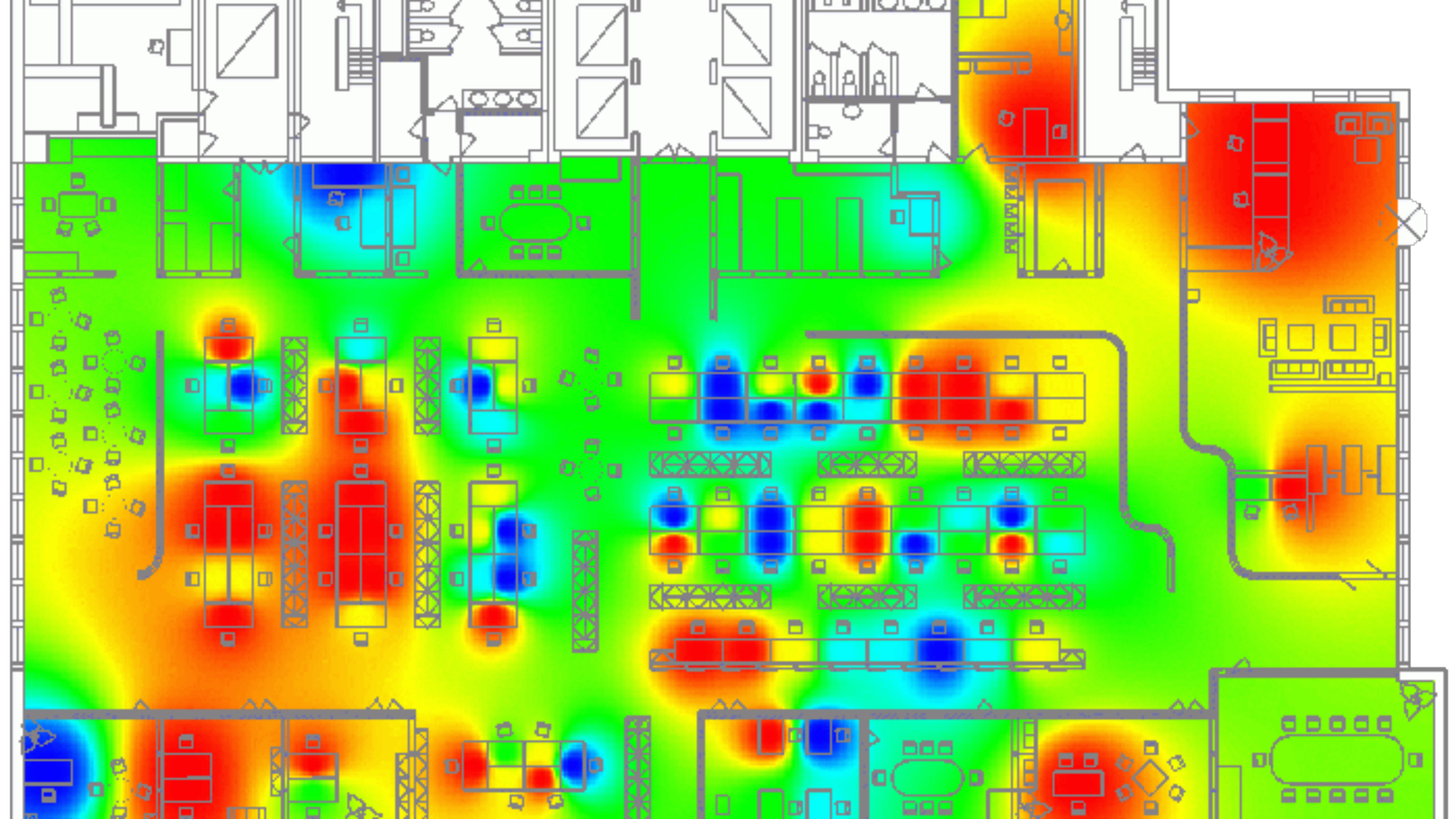
OK, give me a business example

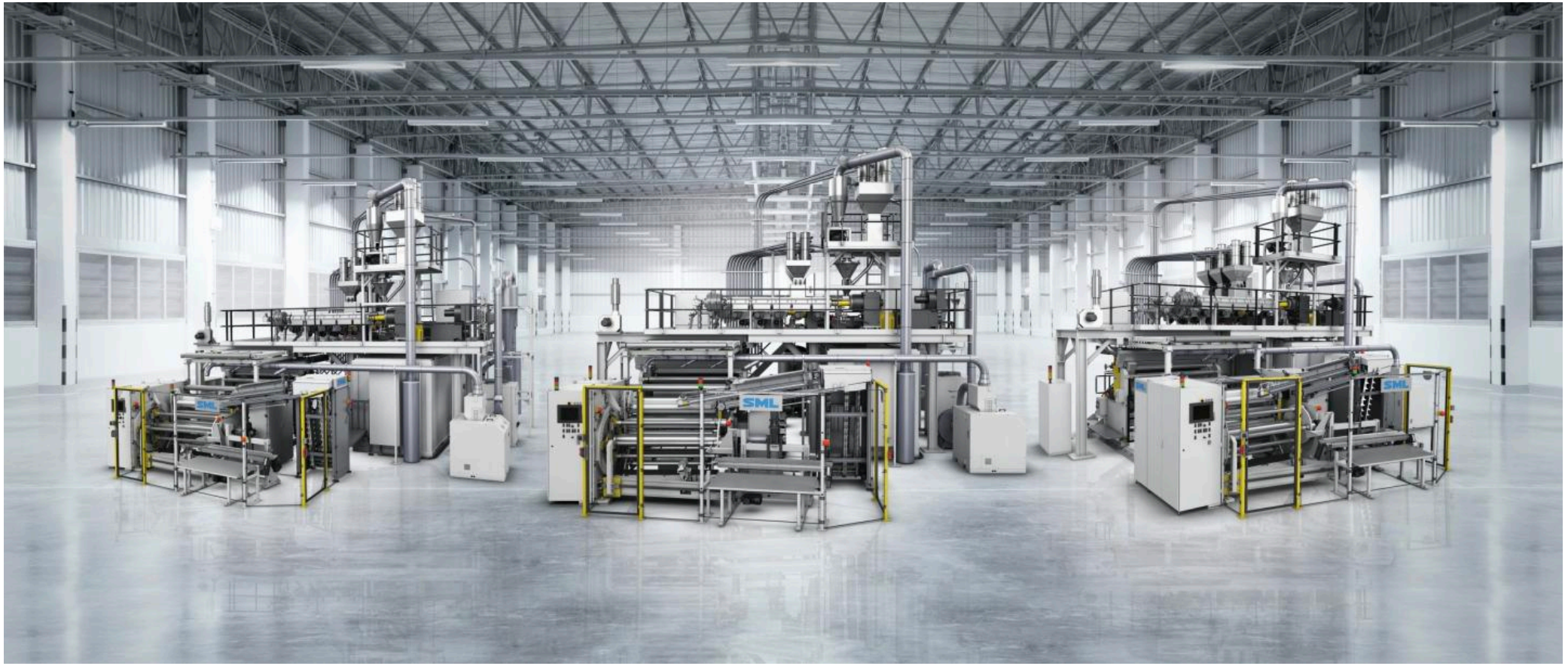
Website Down?

- Total Downtime: Just under 4 minutes
- 502 error messages total: 12,000
- People affected by the 502 error who did not get their bargain: 400



Casinos





Industrial machines

Goals of the Workshop

- [X] Introduction
- [ ] Operations Models
- [ ] What to Monitor
- [ ] Build an ELK Stack
- [ ] Build a Prometheus Stack
- [ ] Best Practices & Recap

Ops Paradise

- Everything is Automated
- Reduce Costs
- No support calls / support tickets

OPS FIREFIGHTING

OPERATIONAL MODELS

Manual	• User initiated • Interactive, command-line tools, simple scripts • Checklist and process driven
Reactive	• Hardware-centric data collection • Simple metric and log collection • Siloed tools and information • Manual analysis and remediation
Proactive	• Application-centric data collection • End-to-end **observability** • **Key metrics** and thresholds well understood • Semi-automated analysis and remediation

USERS CARE ABOUT 3 THINGS

Availability - Is my System Online Yes/No

Latency - Does it take a long time to access applications x, y, z

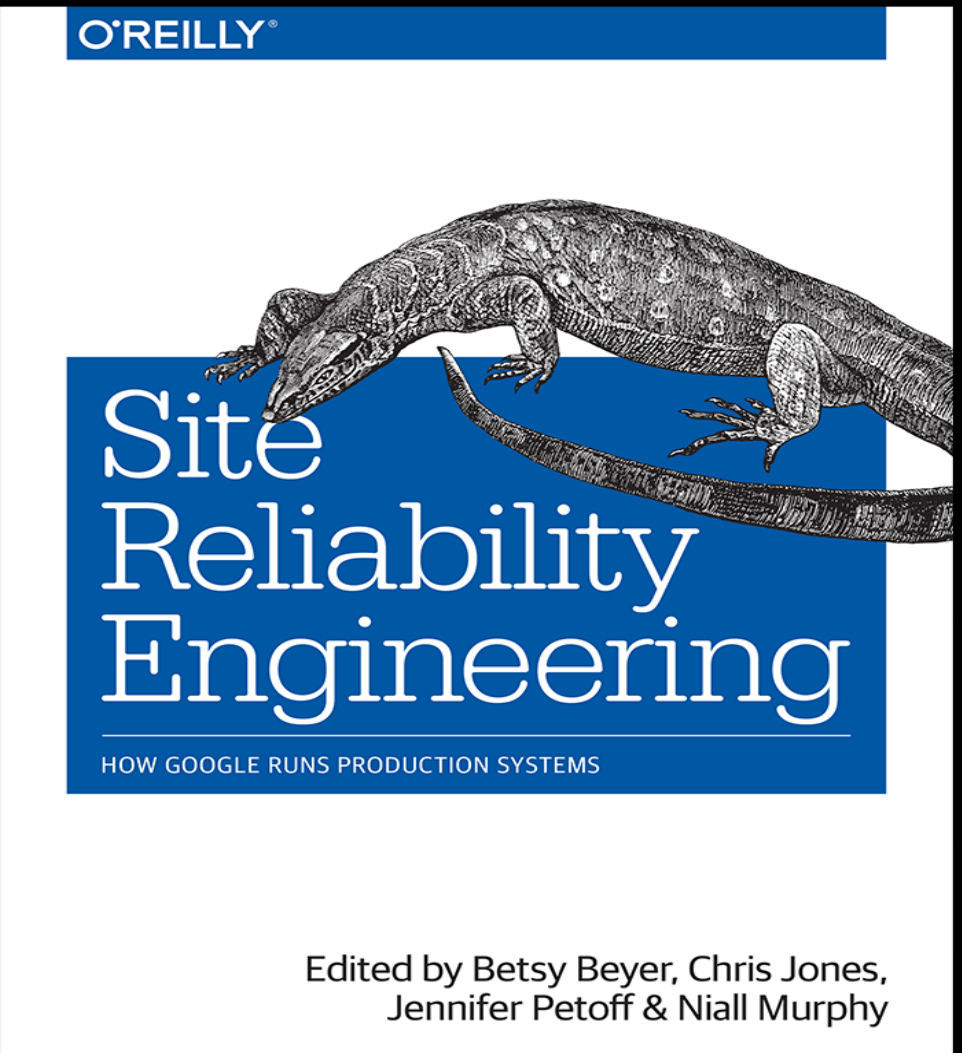
Reliability - Can the user rely on using the application

Brain Based Tools

- We can track 8 objects on average
- 4 Moving Objects
- Build Dashboards & Tools accordingly

SRE

SRE (Site Reliability Engineering)

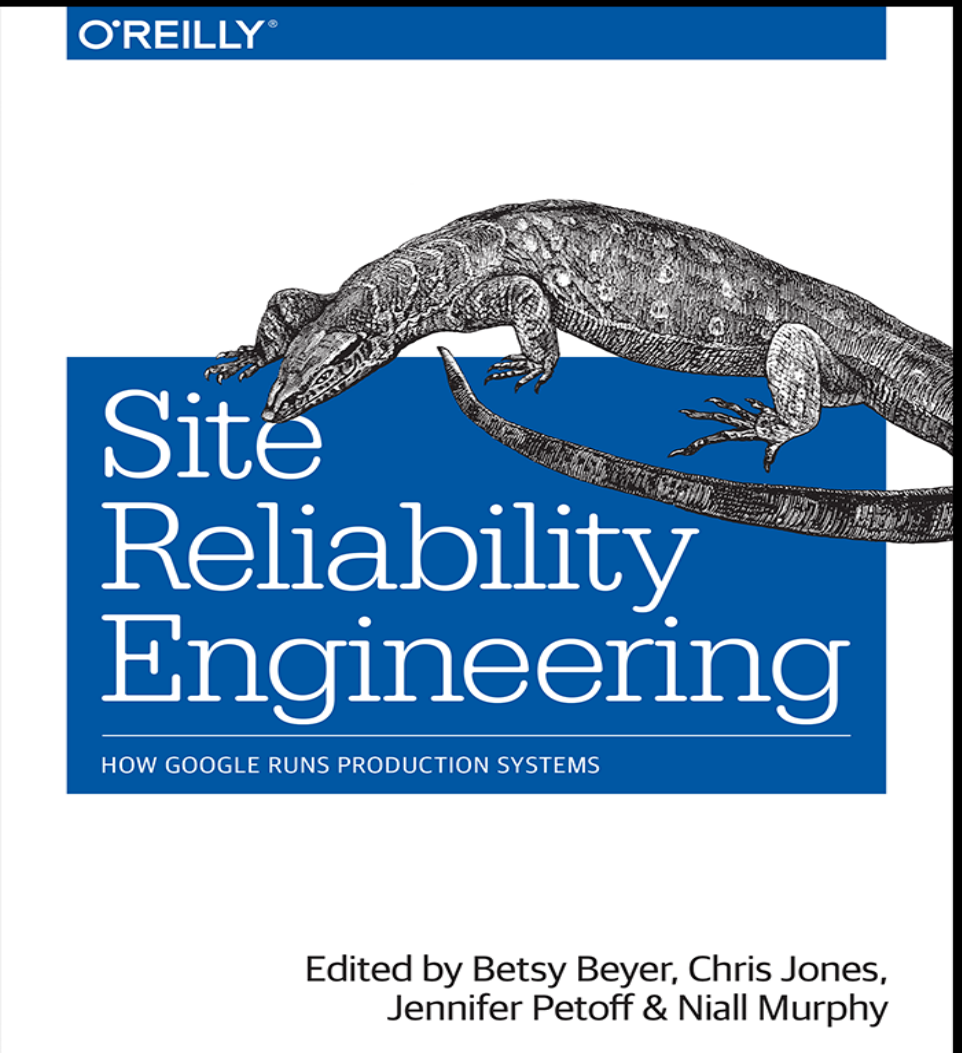


SRE is treats Operations as if it were a Software Problem

"Hope is not a strategy."
Traditional SRE saying

www.google.com/sre

TL;DR - SRE



4 Golden Signals

- Latency
- Traffic
- Errors
- Saturation

R.E.D (Microservice Level)

- **Request:** distributions of the amount of time each request takes
- **Errors:** the number of failed requests per second. Utilization: the average time that the resource was busy servicing work
- **Duration:** distributions of the amount of time each request takes

U.S.E (Low Level / Infrastructure)

For every resource, check Utilization, Saturation, and Errors

- **Resource:** all physical server functional components (CPUs, disks, busses, ...)
- **Utilization:** the average time that the resource was busy servicing work
- **Saturation:** the degree to which the resource has extra work which it can't service, often queued
- **Errors:** the count of error events

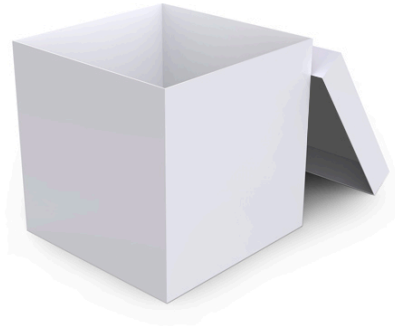
Black box vs. White box Monitoring

External App Metrics

Black Box Monitoring

HTTP, Ping, etc

Internal App Metrics



White Box Monitoring

App metrics, requests, responses, process times

Healthchecks

HEALTHCHECK

The `HEALTHCHECK` instruction has two forms:

- `HEALTHCHECK [OPTIONS] CMD command` (check container health by running a command inside the container)
- `HEALTHCHECK NONE` (disable any healthcheck inherited from the base image)

The `HEALTHCHECK` instruction tells Docker how to test a container to check that it is still working. This can detect cases such as a web server that is stuck in an infinite loop and unable to handle new connections, even though the server process is still running.

When a container has a healthcheck specified, it has a *health status* in addition to its normal status. This status is initially `starting`. Whenever a health check passes, it becomes `healthy` (whatever state it was previously in). After a certain number of consecutive failures, it becomes `unhealthy`.

The options that can appear before `CMD` are:

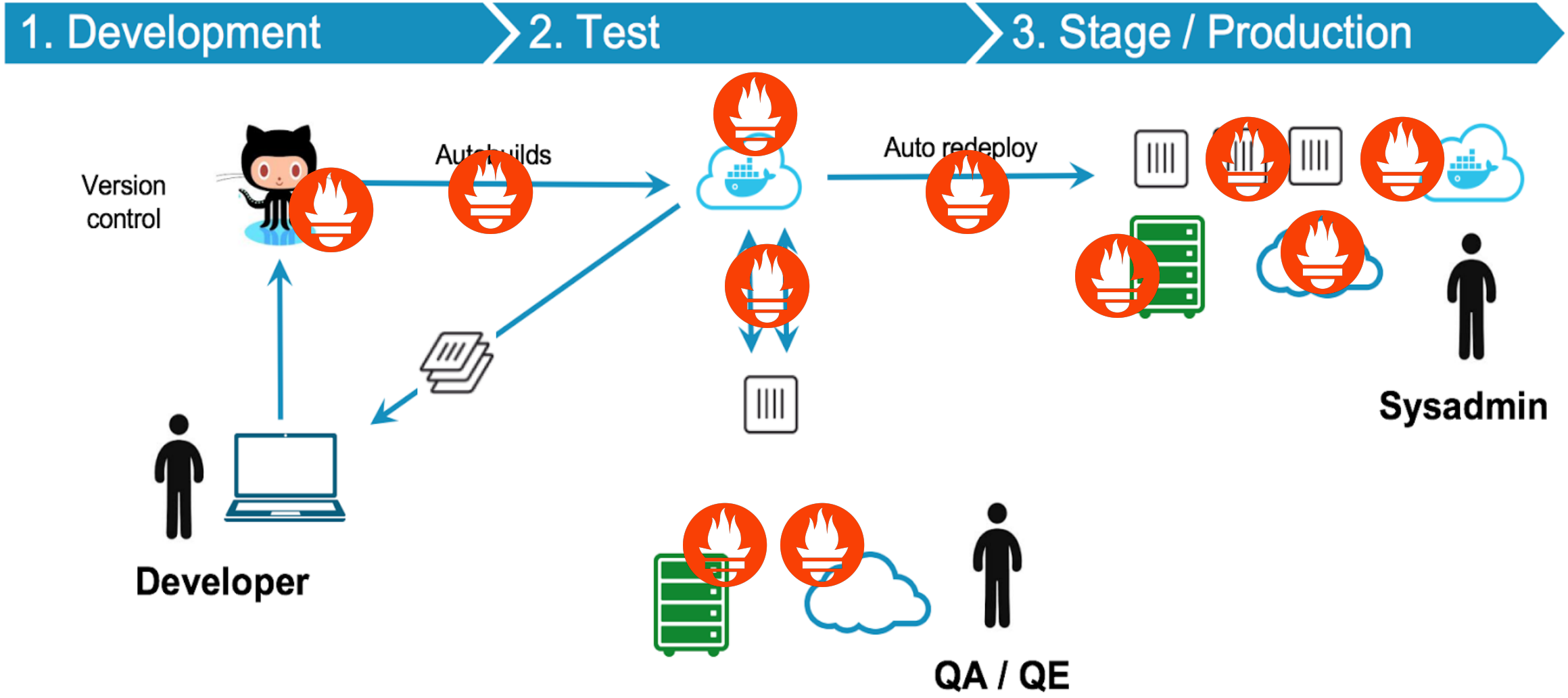
- `--interval=DURATION` (default: `30s`)
- `--timeout=DURATION` (default: `30s`)
- `--start-period=DURATION` (default: `0s`)
- `--retries=N` (default: `3`)

Readiness / Liveness Probes

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - args:
    - /server
    image: k8s.gcr.io/liveness
    livenessProbe:
      httpGet:
        # when "host" is not defined, "PodIP" will be used
        # host: my-host
        # when "scheme" is not defined, "HTTP" scheme will be used. Only "HTTP" and "HTTPS" are allowed
        # scheme: HTTPS
        path: /healthz
        port: 8080
        httpHeaders:
        - name: X-Custom-Header
          value: Awesome
      initialDelaySeconds: 15
      timeoutSeconds: 1
    name: liveness
```

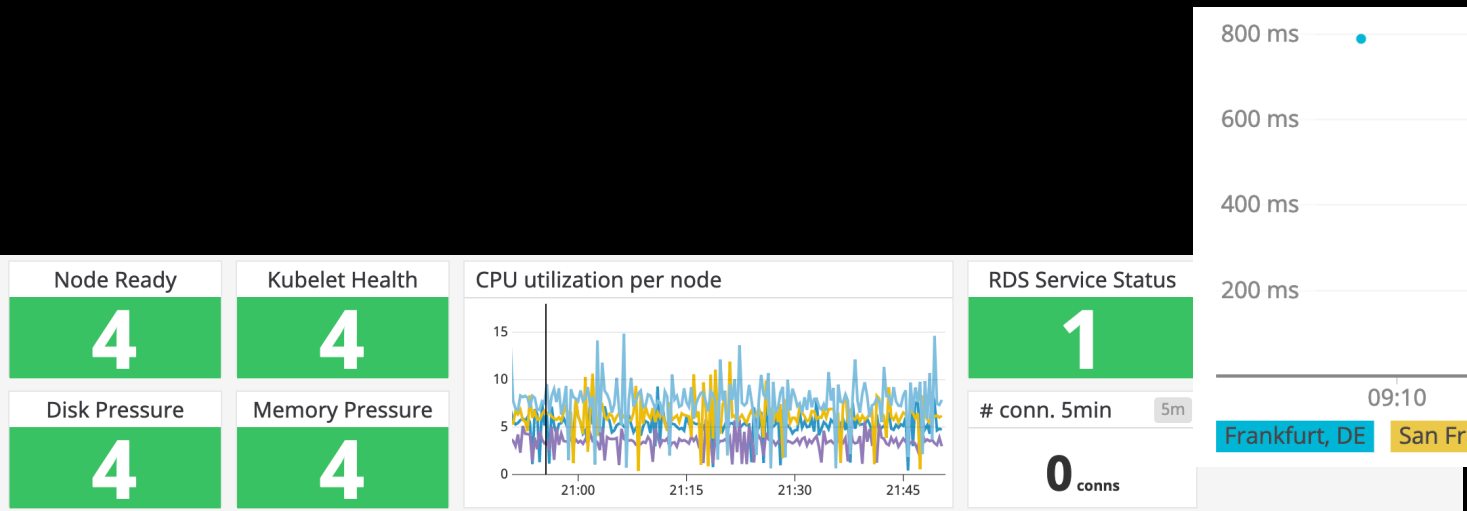
Goals of the Workshop

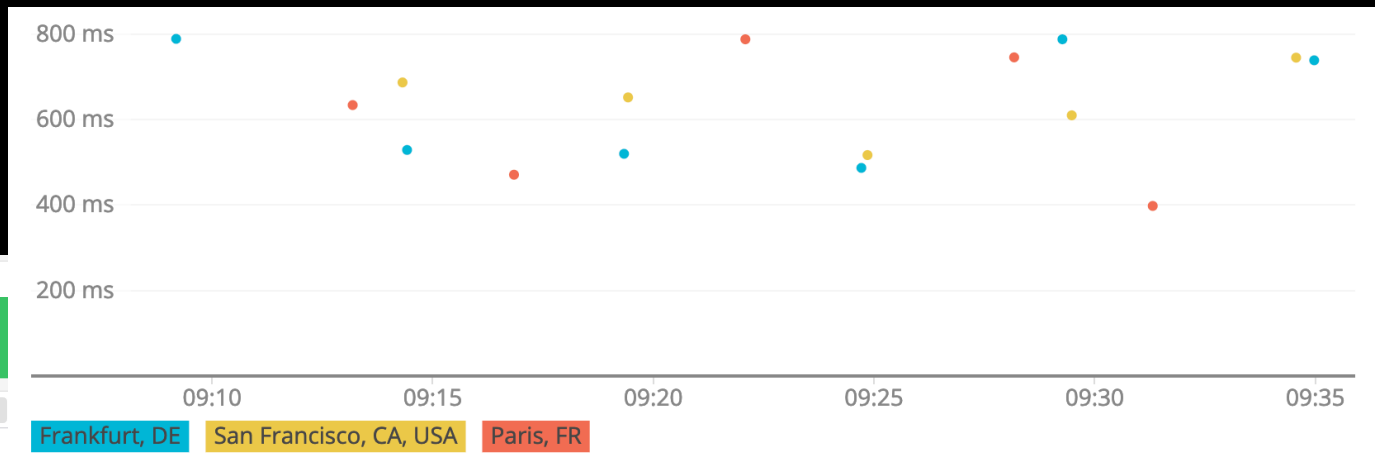
- [X] Introduction
- [X] Operations Models
- [ ] What to Monitor
- [ ] Build an ELK Stack
- [ ] Build a Prometheus Stack
- [ ] Best Practices & Recap


1. Development
2. Test
3. Stage / Production
Version control
Auto redeploy
Sysadmin
Developer
QA / QE

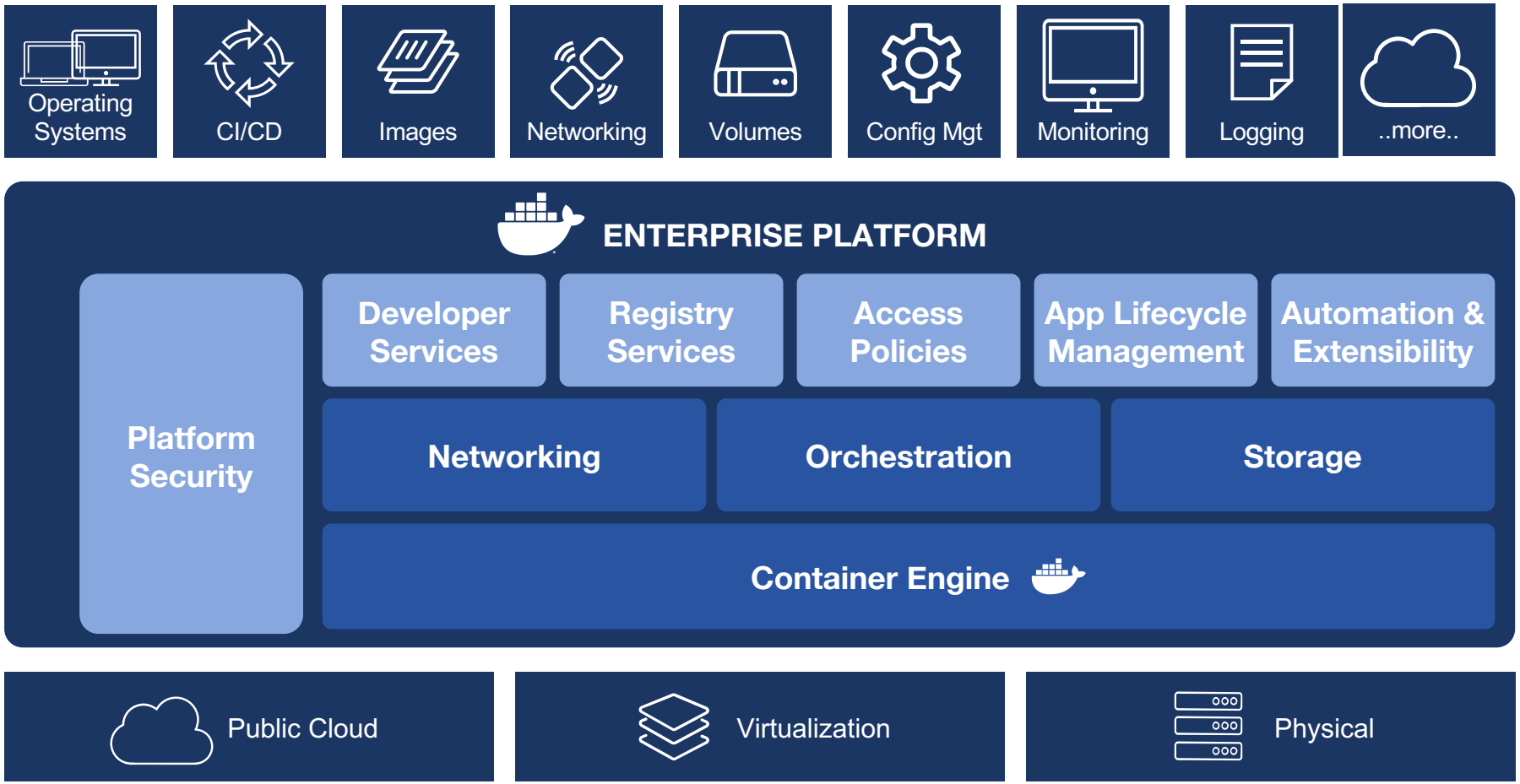
A joint collaboration between BMW & Daimler AG
www.your-now.com

Continuous monitoring of all infrastructure and application components already during the development process is key to a successful launch.

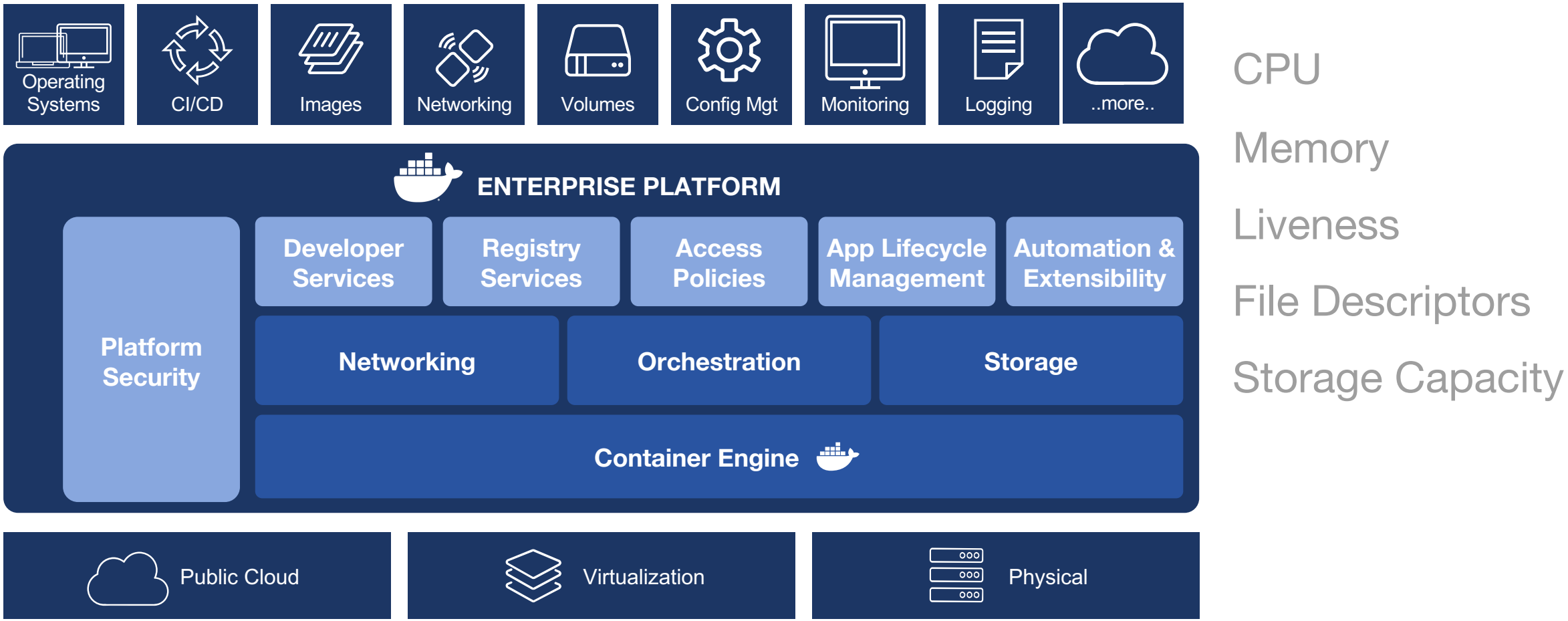




Understanding Failure Modes



Host / Hardware



CPU

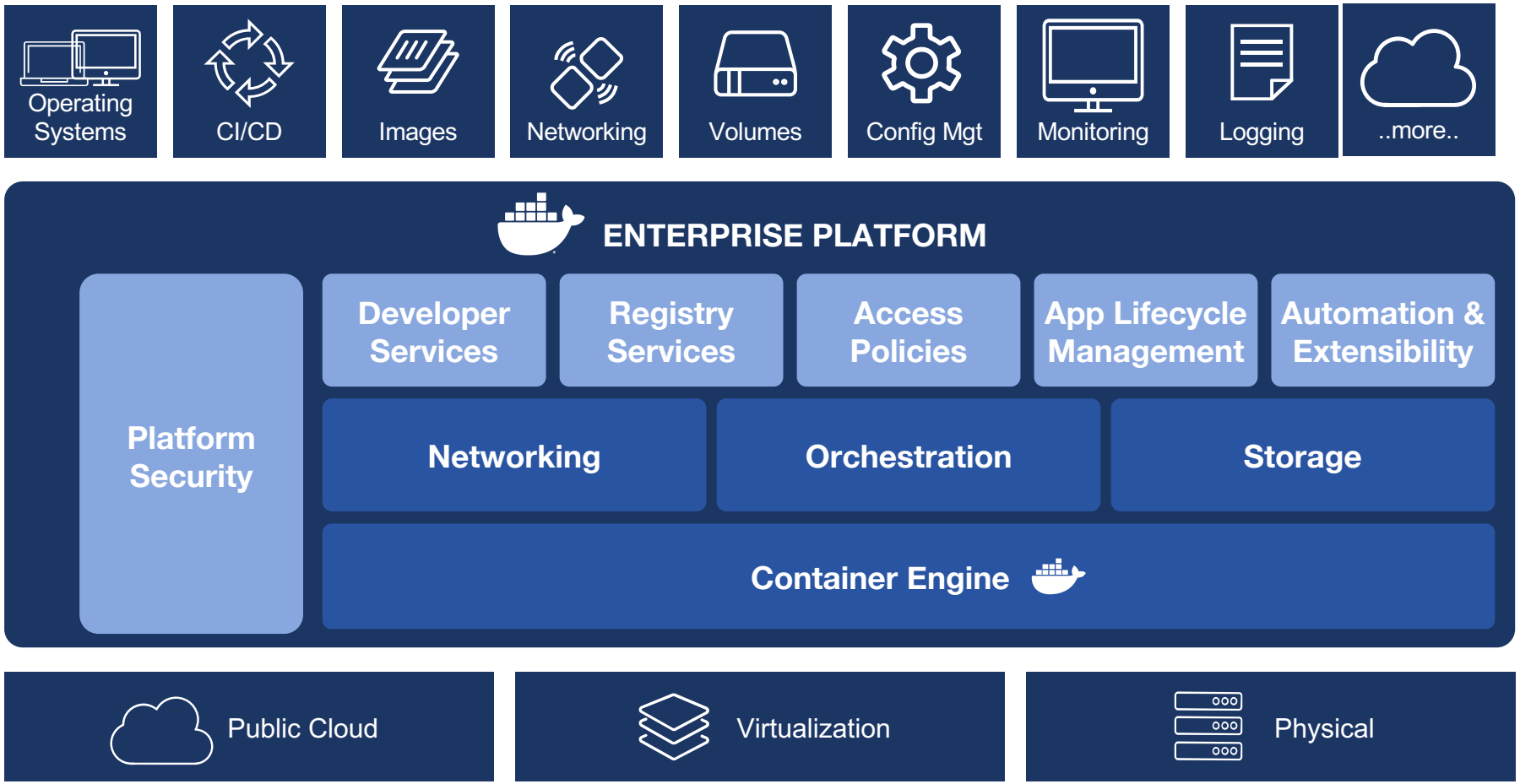
Memory

Liveness

File Descriptors

Storage Capacity

Networking

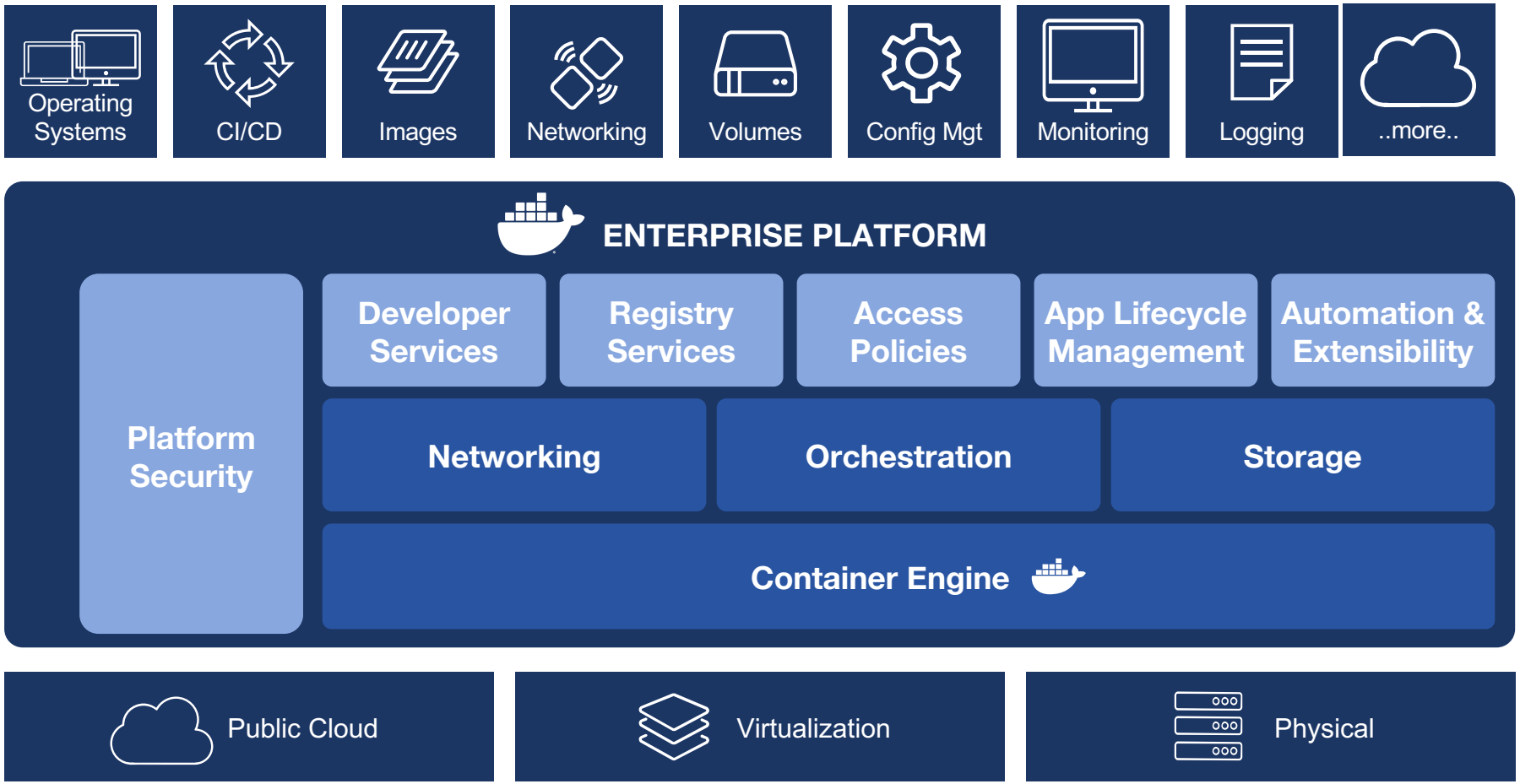


Reachability

Link Utilization

Dropped Packets

Orchestration



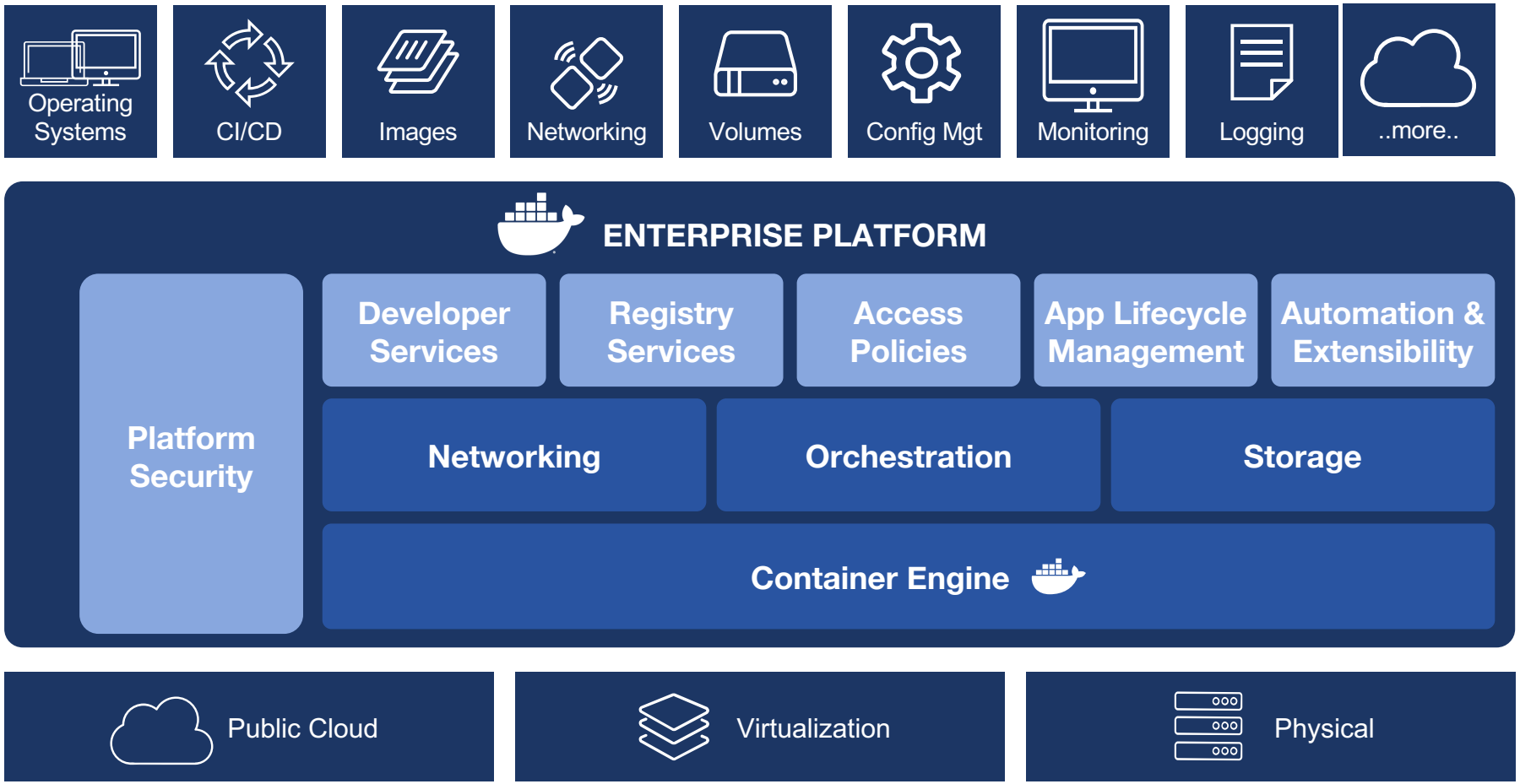
State

Deployment Rates

Capacity

Scheduling Events

Applications



Response Times

Error Rates

App Specific Metrics

Availability

Goals of the Workshop

- [X] Introduction
- [X] Operations Models
- [X] What to Monitor
- [ ] Build an ELK Stack
- [ ] Build a Prometheus Stack
- [ ] Best Practices & Recap

Logging Tools Overview

github.com/56kcloud/Training

https://dockr.ly/brian

Logging Challenges

- What should we log?
- Where and how long should we store logs
- Analysis

Logging Tools

- docker logs
- docker-compose logs
- docker service logs
- log drivers

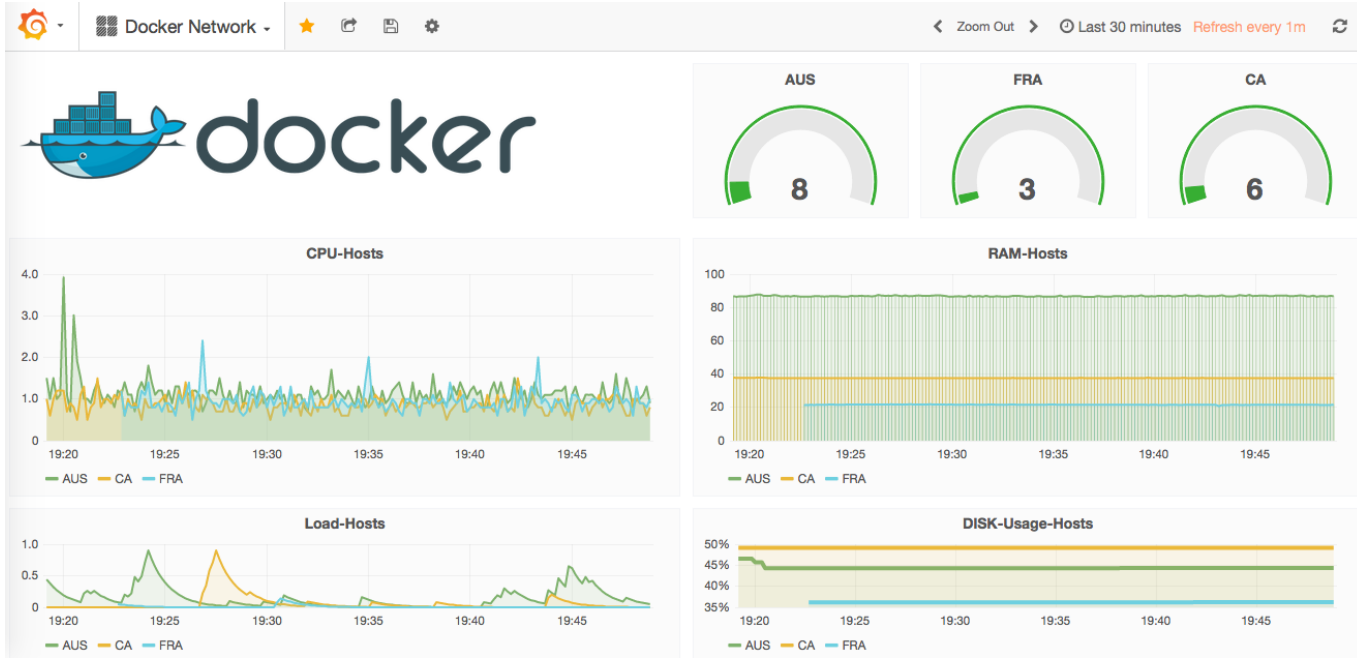
Logging Workshop

Goals of the Workshop

- [X] Introduction
- [X] Operations Models
- [X] What to Monitor
- [X] Build an ELK Stack
- [ ] Build a Prometheus Stack
- [ ] Best Practices & Recap

The Basis of Monitoring

- What's Broken?
- Why is it Broken?
- How Long has it been broken?



Monitoring Tools

- Docker Stats
- Docker Top
- Docker df
- cAdvisor
- Prometheus

- Live container resources
- All containers or single
- Very basic but useful info

Docker Stats

1. docker

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
35605185bf52	cadvisor	0.84%	43.18MiB / 1.952GiB	2.16%	1.31MB / 606MB	19.1MB / 0B	11
d4bd451f0b68	demo3_worker_1	0.65%	16.4MiB / 1.952GiB	0.82%	47.7kB / 54.7kB	43MB / 0B	17
1aaa67a5a5a8	redis	0.23%	1.367MiB / 1.952GiB	0.07%	26.2kB / 14.6kB	2.77MB / 0B	4
d6d556507c0f	demo3_vote_1	0.45%	31.66MiB / 1.952GiB	1.58%	2.73kB / 0B	13.5MB / 0B	3
511eb5d3a5f4	db	0.29%	9.766MiB / 1.952GiB	0.49%	34.6kB / 35.2kB	27.6MB / 90.1kB	8
7c726e569b7b	demo3_result_1	0.08%	43.22MiB / 1.952GiB	2.16%	5.35kB / 3.29kB	34.2MB / 4.1kB	20

Docker Top

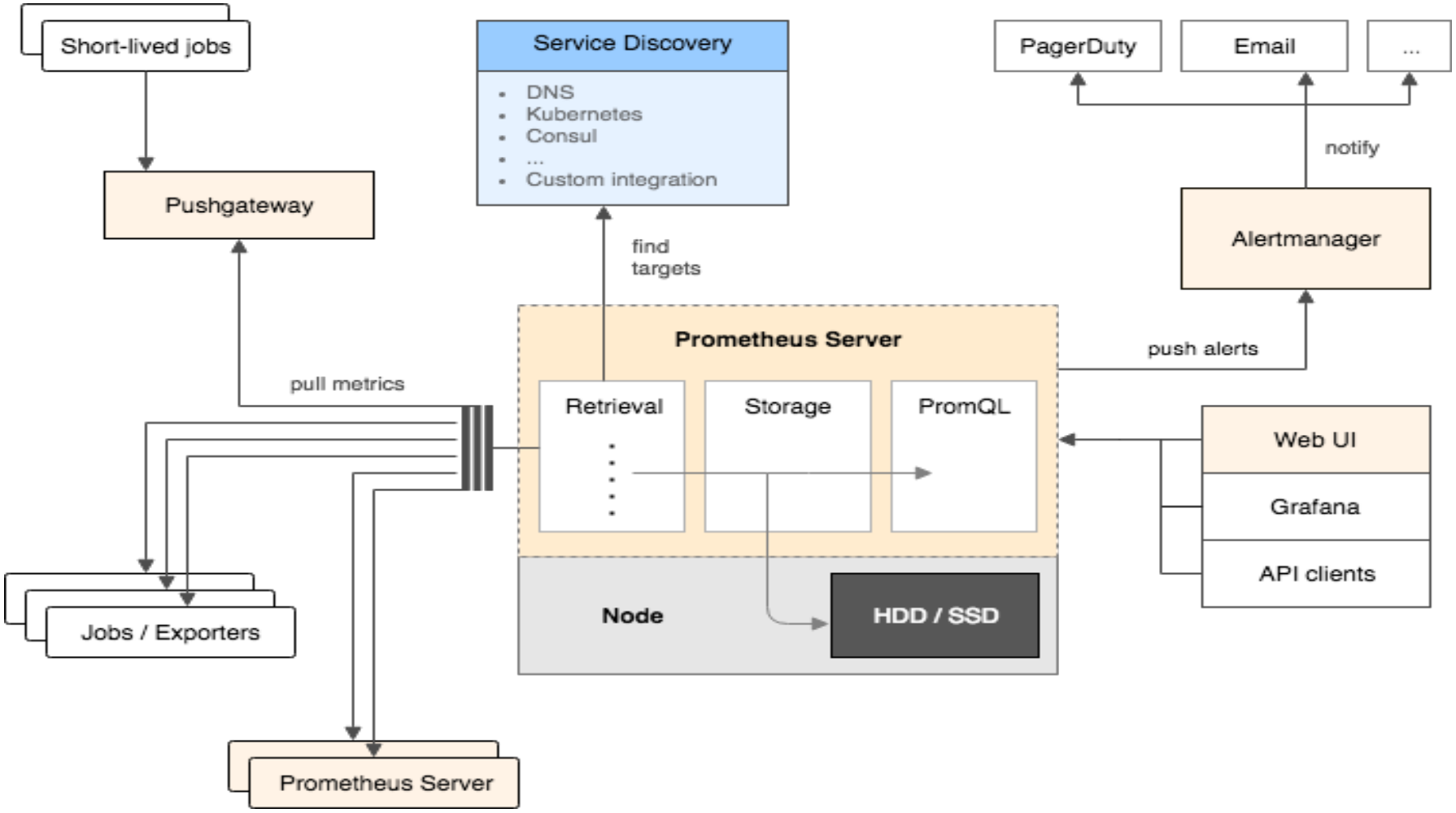
- Display running process in a container

```
vegasbrianc@Brian-56K:~$docker top loving_hofstadter
PID                 USER                TIME                COMMAND
97904               root                0:00                ping 8.8.8.8
vegasbrianc@Brian-56K:~$
```

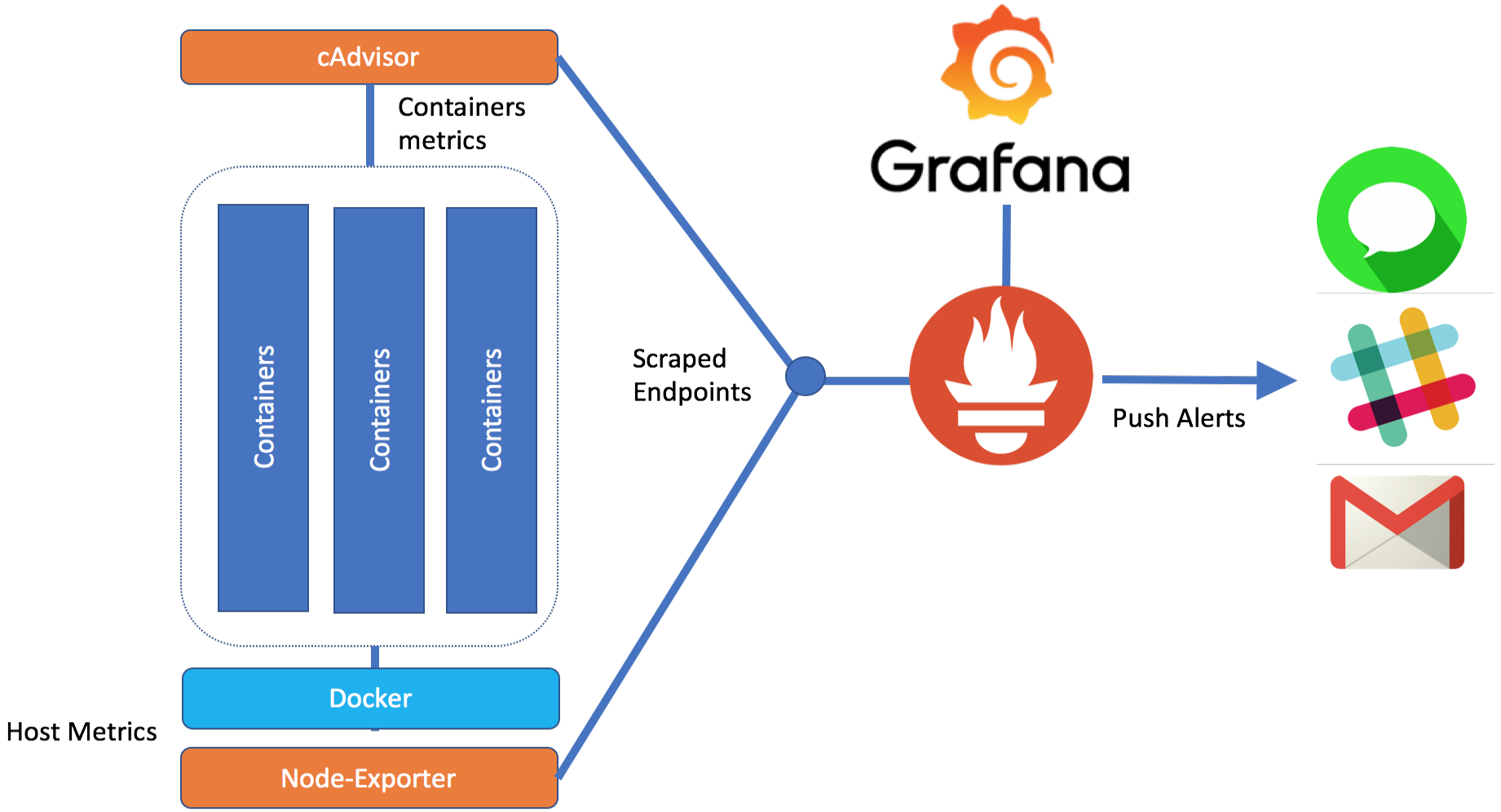
cAdvisor

- Developed by Google
- Real-time Data
- Clean UI
- Exposes Metrics
- Integrates well

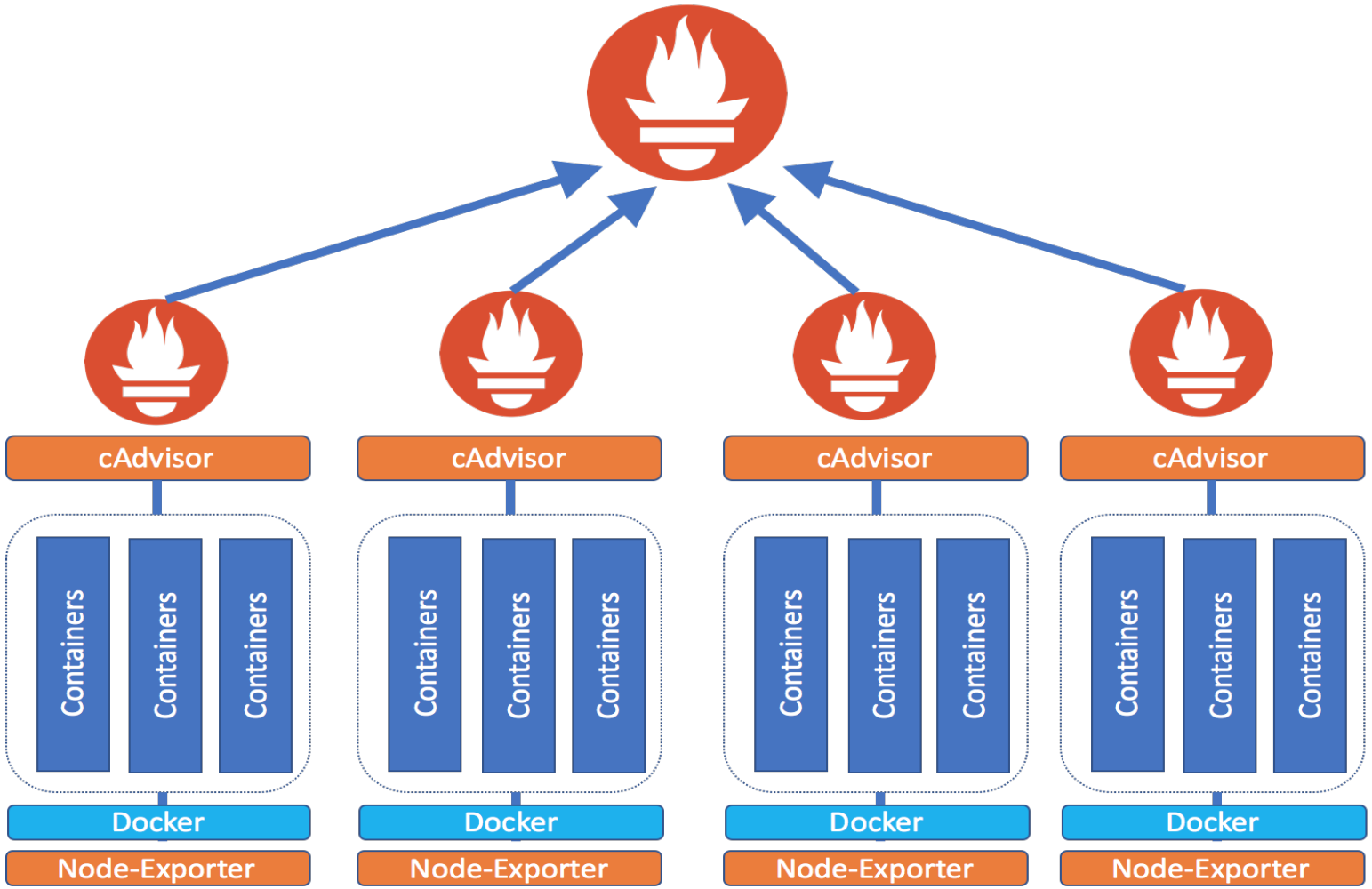
Prometheus



Prometheus Stack



Federated Configuration



github.com/56kcloud/Training

Goals of the Workshop

- [X] Introduction
- [X] Operations Models
- [X] What to Monitor
- [X] Build an ELK Stack
- [X] Build a Prometheus Stack
- [ ] Best Practices & Recap

Improve incrementally

Best Practices

- Start small & increment
- Don't Overlert yourself
- Set Resource Limits
- Aim for actionable Information
- Run separate from Workload
- Test for Failures
- Know your Failure Models

Resources

- 56K.Cloud - https://56K.Cloud
- Prometheus - https://github.com/vegasbrianc/prometheus
- ELK - https://github.com/deviantony/docker-elk
- Labs – github.com/56kcloud/Training/tree/master/DockerCon
- Docker Resource Link - https://awesome-docker.netlify.com

Even More Resources

- GitLab Dashboards - https://monitor.gitlab.net
- Grafana Dashboards - https://grafana.com/dashboards

Goals of the Workshop

- [X] Introduction
- [X] Operations Models
- [X] What to Monitor
- [X] Build an ELK Stack
- [X] Build a Prometheus Stack
- [X] Best Practices & Recap

56K. CLOUD

Thank You

Brian Christner
brian@56k.cloud
@idomyowntricks

dockercon19 SAN FRANCISCO