# DevOps & Continuous Deployment

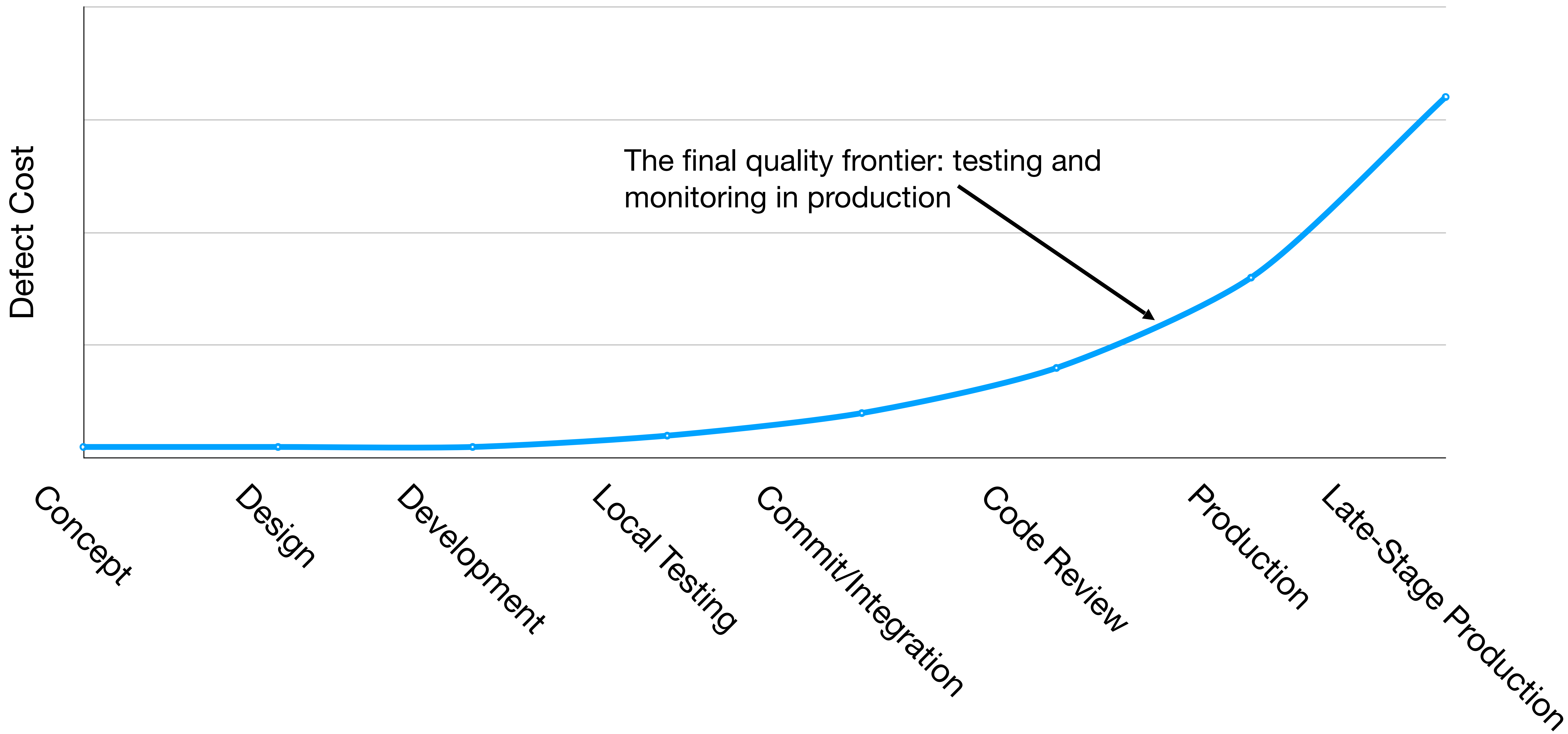
## Advanced Software Engineering
## Spring 2023

# Outline

1. Continuous delivery and devops: motivation
2. Infrastructure as code: concepts, and the mess of tools
3. Continuous delivery practices using infrastructure as code
4. Monitoring, telemetry and operations practices and tools

# Cost to Fix a Defect Over Time

## Rough estimate



# Case Study of a Failed Deployment: Knight Capital

## Knightmare: A DevOps Cautionary Tale

D7 DevOps April 17, 2014 6 Minutes

I was speaking at a conference last year on the topics of DevOps, Configuration as Code, and Continuous Delivery and used the following story to demonstrate the importance making deployments fully automated and repeatable as part of a DevOps/Continuous Delivery initiative. Since that conference I have been asked by several people to share the story through my blog. This story is true – this really happened. This is my telling of the story based on what I have read (I was not involved in this).

This is the story of how a company with nearly $400 million in assets went ban
minutes because of a failed deployment.

“In the week before go-live, a Knight engineer manually deployed the new RLP code in SMARS to its eight servers. However, the engineer made a mistake and did not copy the new code to one of the servers. Knight did not have a second engineer review the deployment, and neither was there an automated system to alert anyone to the discrepancy. “

https://www.henricodolfing.com/2019/06/project-failure-case-study-knight-capital.html

# What Could Knight Capital Have Done Better?

- Use capture/replay testing instead of driving market conditions in a test
- Avoid including “test” code in production deployments
- Automate deployments
- Define and monitor risk-based KPIs
- Create checklists for responding to incidents

# Deployment Philosophy: Instagram

## “Faster is safer”

“If stuff blows up it affects a very small percentage of people”

Instagram cofounder and CTO Mike Krieger

https://www.fastcompany.com/3047642/do-the-simple-thing-first-the-engineering-behind-instagram

# Continuous Delivery

## “Faster is safer”: Key values of continuous delivery

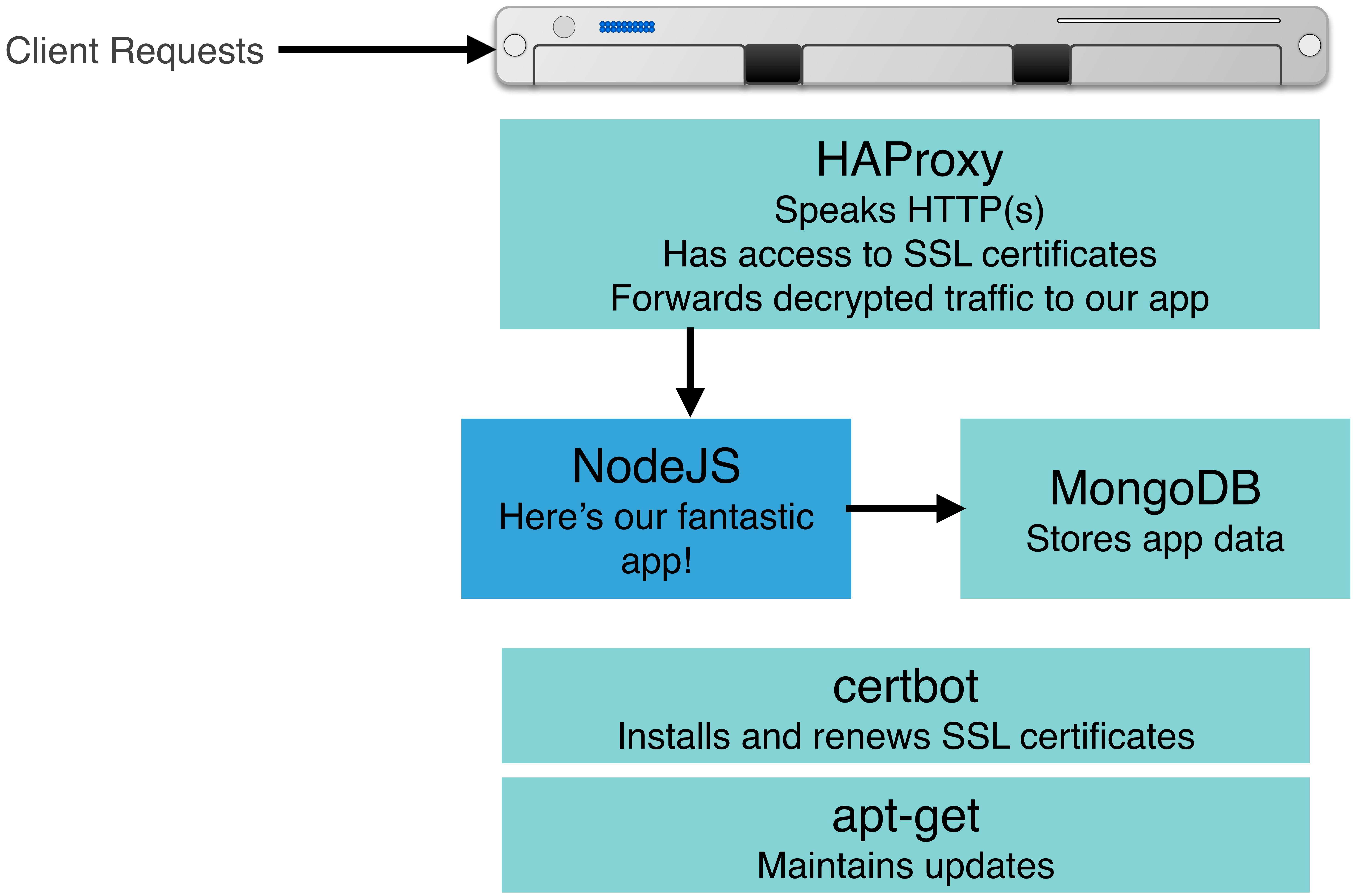
- Release frequently, in small batches
- Maintain key performance indicators to evaluate the impact of updates
- Phase roll-outs
- Evaluate business impact of new features

# Continuous Delivery Relies on Staging Environments

- As software gets more complex with more dependencies, it's impossible to simulate the whole thing when testing
- Idea: Deploy to a complete production-like environment, but don't have everyone use it
  - Examples:
    - “Eat your own dogfood”
    - Beta/Alpha testers
- Lower risk if a problem occurs in staging than in production

# What is non-continuous deployment?

## Deployment Example: NodeJS/MongoDB App



Client Requests

HAProxy
Speaks HTTP(s)
Has access to SSL certificates
Forwards decrypted traffic to our app

NodeJS
Here’s our fantastic app!

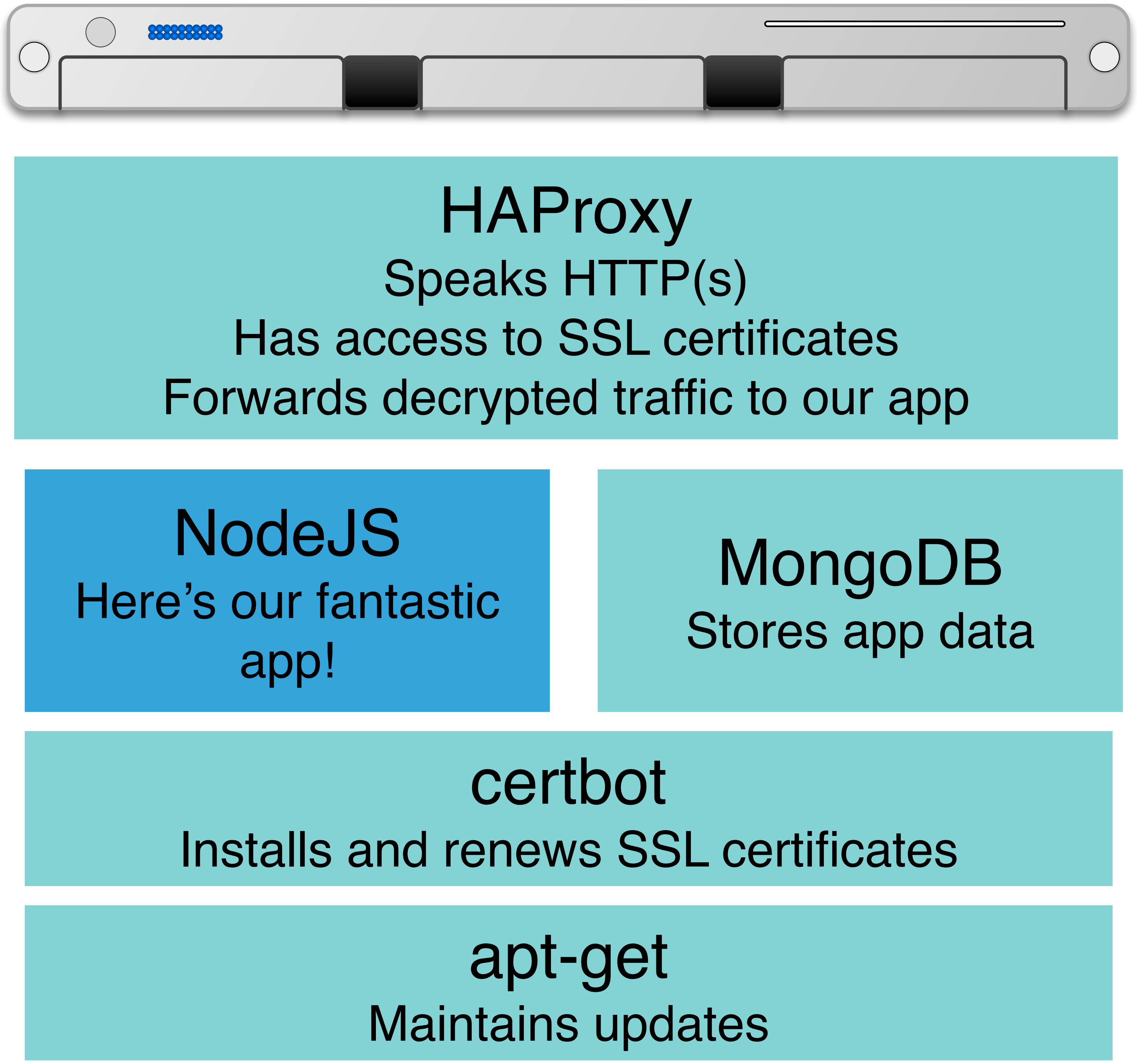
MongoDB
Stores app data

certbot
Installs and renews SSL certificates

apt-get
Maintains updates

# What is non-continuous deployment?

## Deployment Example: NodeJS/MongoDB App



HAProxy
Speaks HTTP(s)
Has access to SSL certificates
Forwards decrypted traffic to our app

NodeJS
Here's our fantastic app!

MongoDB
Stores app data

certbot
Installs and renews SSL certificates

apt-get
Maintains updates

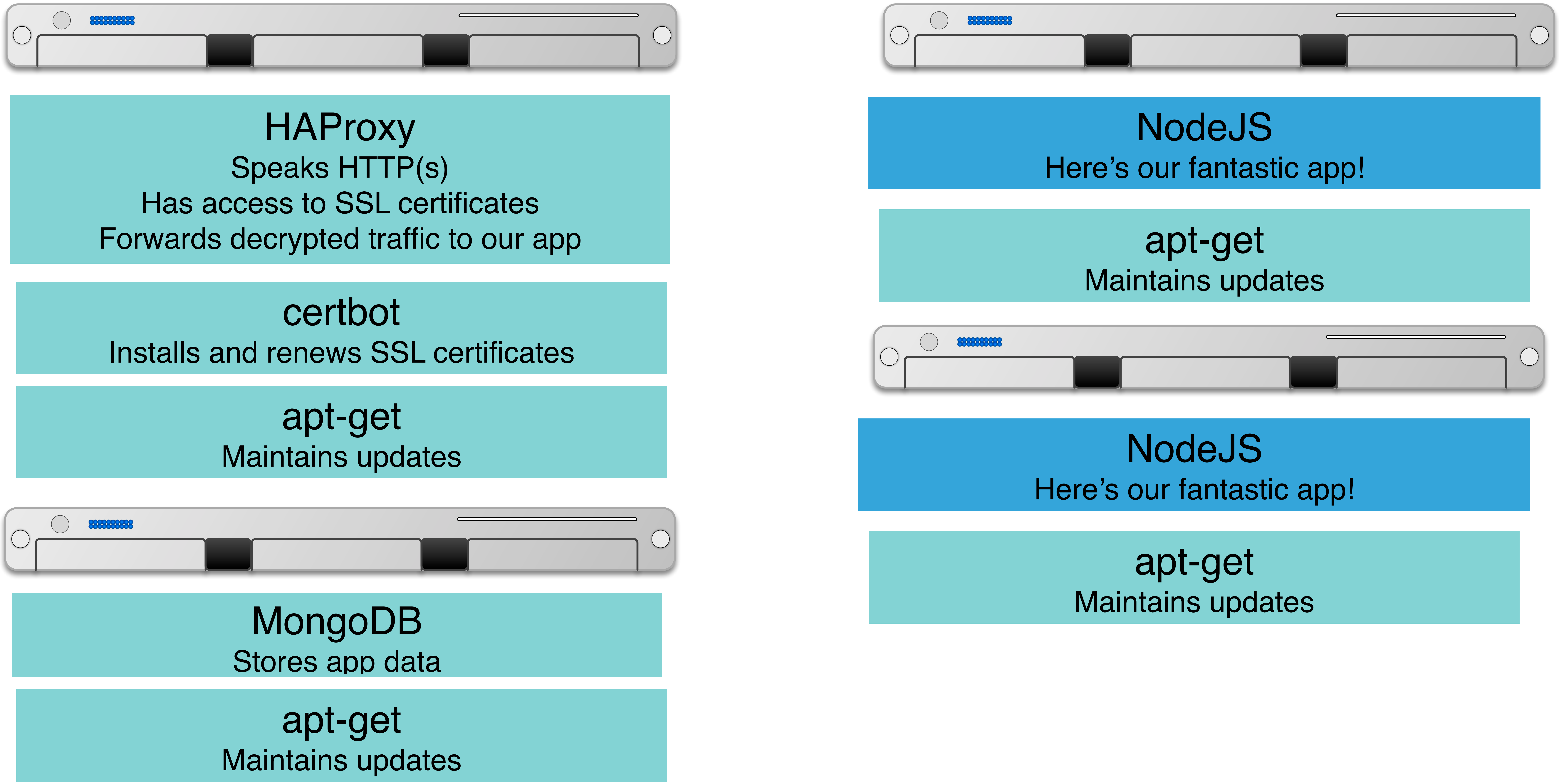
**First setup:**
Step 0: Install Ubuntu
Step 1: Install HAProxy
Step 2: Configure HAProxy
Step 3: Configure firewall
Step 4: Install certbot
Step 5: Configure SSL
Step 6: Install MongoDB
Step 7: Configure MongoDB, create database
Step 8: Install NodeJS
Step 9: Copy application to server
Step 10: Test application

**Updating app:**
Step 1: Copy updated app to server
Step 2: Restart app
Step 3: Check still working

**Updating infrastructure:**
Step 1: SSH to server
Step 2: apt-get upgrade?
Step 3: Hope that it works?

# What is non-continuous deployment?

## Deployment Example: NodeJS/MongoDB App, now scaling to multiple servers



HAProxy
Speaks HTTP(s)
Has access to SSL certificates
Forwards decrypted traffic to our app

certbot
Installs and renews SSL certificates

apt-get
Maintains updates

MongoDB
Stores app data

apt-get
Maintains updates

NodeJS
Here’s our fantastic app!

apt-get
Maintains updates

NodeJS
Here’s our fantastic app!

apt-get
Maintains updates

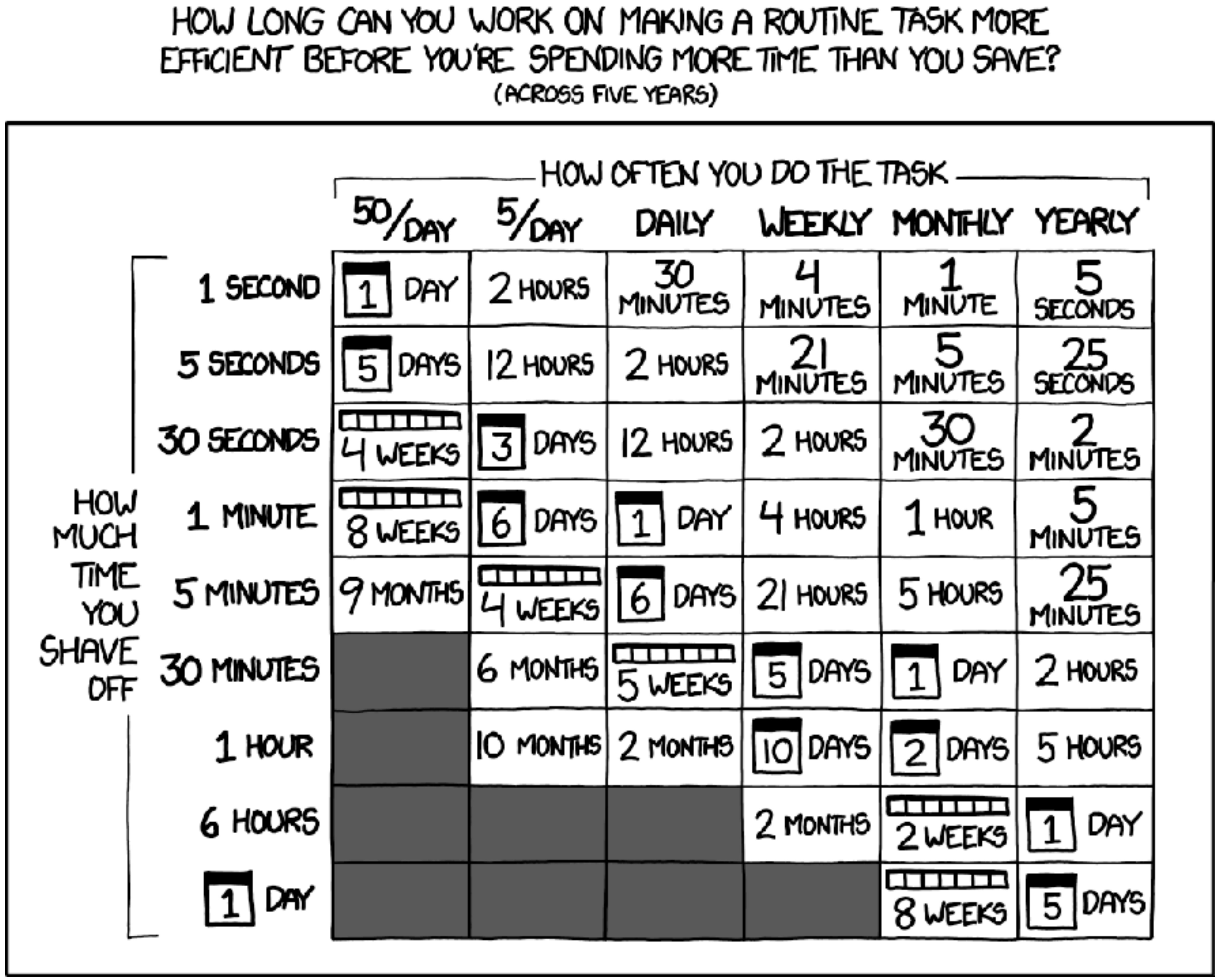
# DevOps Blends Operations and Development Responsibilities

- Who manages infrastructure, deploys software, operates/monitors it?
- Pre-DevOps:
  - Entirely separate team (maybe a vendor operating under contract!) operates software
- DevOps:
  - Blended responsibilities between developers and operators

| | Developers | Operators |
|---|---|---|
| Waterfall | | Test Staging Production |
| Agile | Test | Staging Production |
| DevOps | Test Staging Production | Production |

# Continuous Deployment Relies on Infrastructure as Code

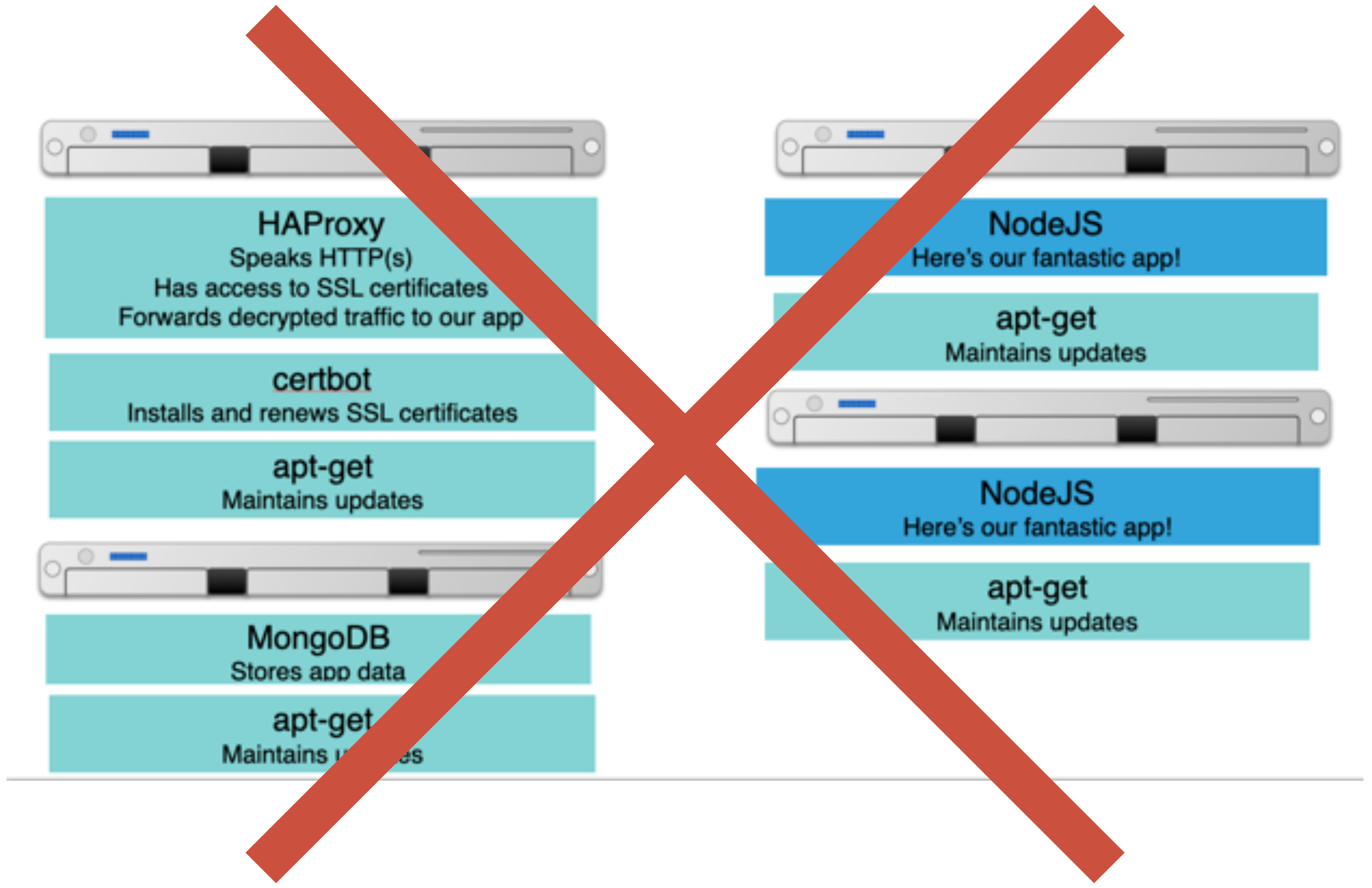
## Core DevOps tenet: Automate provisioning of infrastructure



HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE? (ACROSS FIVE YEARS)

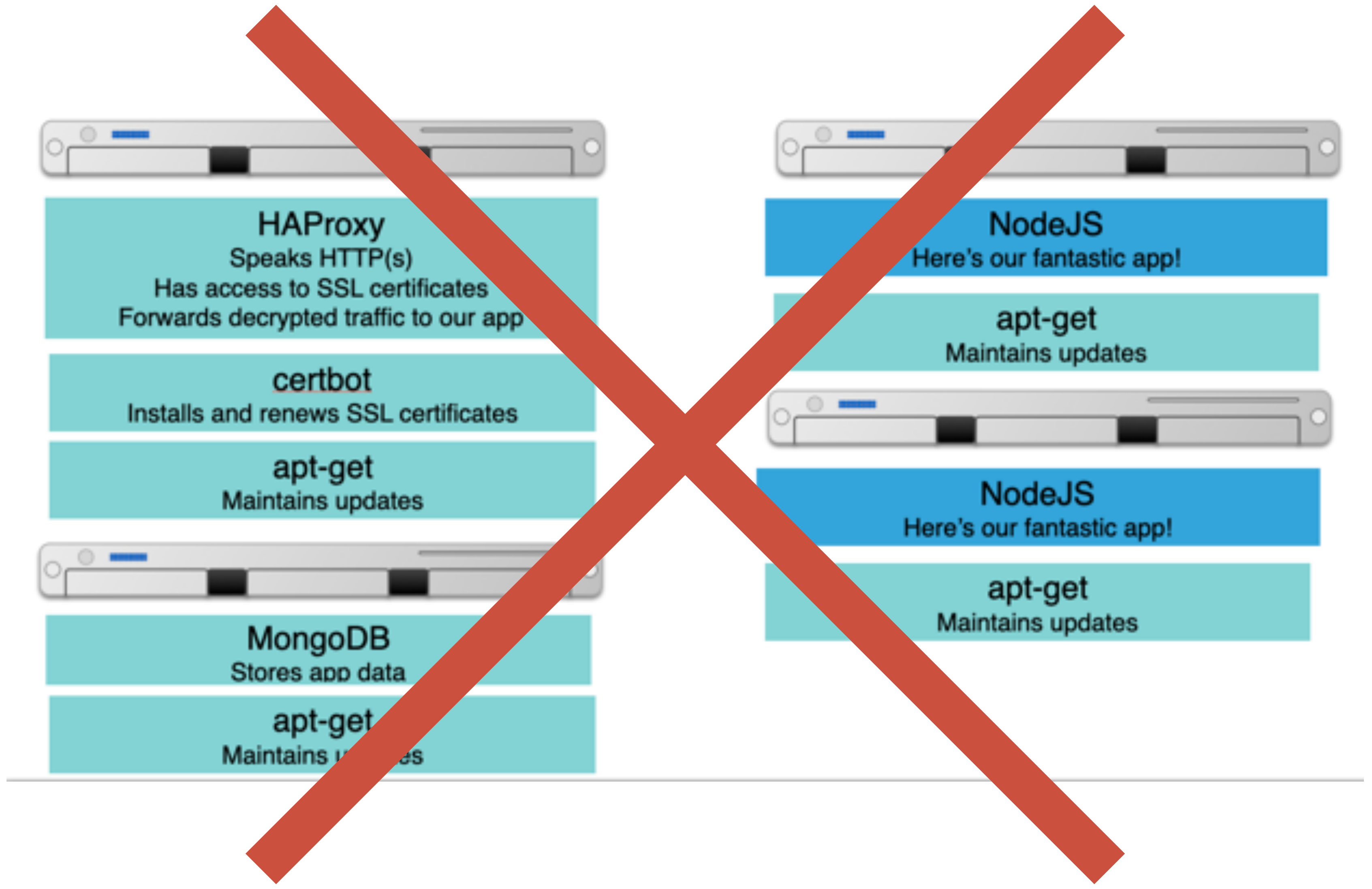
| HOW MUCH TIME YOU SHAVE OFF | HOW OFTEN YOU DO THE TASK | | | | | |
|---|---|---|---|---|---|---|
| | 50/DAY | 5/DAY | DAILY | WEEKLY | MONTHLY | YEARLY |
| 1 SECOND | 1 DAY | 2 HOURS | 30 MINUTES | 4 MINUTES | 1 MINUTE | 5 SECONDS |
| 5 SECONDS | 5 DAYS | 12 HOURS | 2 HOURS | 21 MINUTES | 5 MINUTES | 25 SECONDS |
| 30 SECONDS | 4 WEEKS | 3 DAYS | 12 HOURS | 2 HOURS | 30 MINUTES | 2 MINUTES |
| 1 MINUTE | 8 WEEKS | 6 DAYS | 1 DAY | 4 HOURS | 1 HOUR | 5 MINUTES |
| 5 MINUTES | 9 MONTHS | 4 WEEKS | 6 DAYS | 21 HOURS | 5 HOURS | 25 MINUTES |
| 30 MINUTES | | 6 MONTHS | 5 WEEKS | 5 DAYS | 1 DAY | 2 HOURS |
| 1 HOUR | | 10 MONTHS | 2 MONTHS | 10 DAYS | 2 DAYS | 5 HOURS |
| 6 HOURS | | | | 2 MONTHS | 2 WEEKS | 1 DAY |
| 1 DAY | | | | | 8 WEEKS | 5 DAYS |

© Randal Munroe/xkcd, licensed CC-BY-SA
https://xkcd.com/1205/



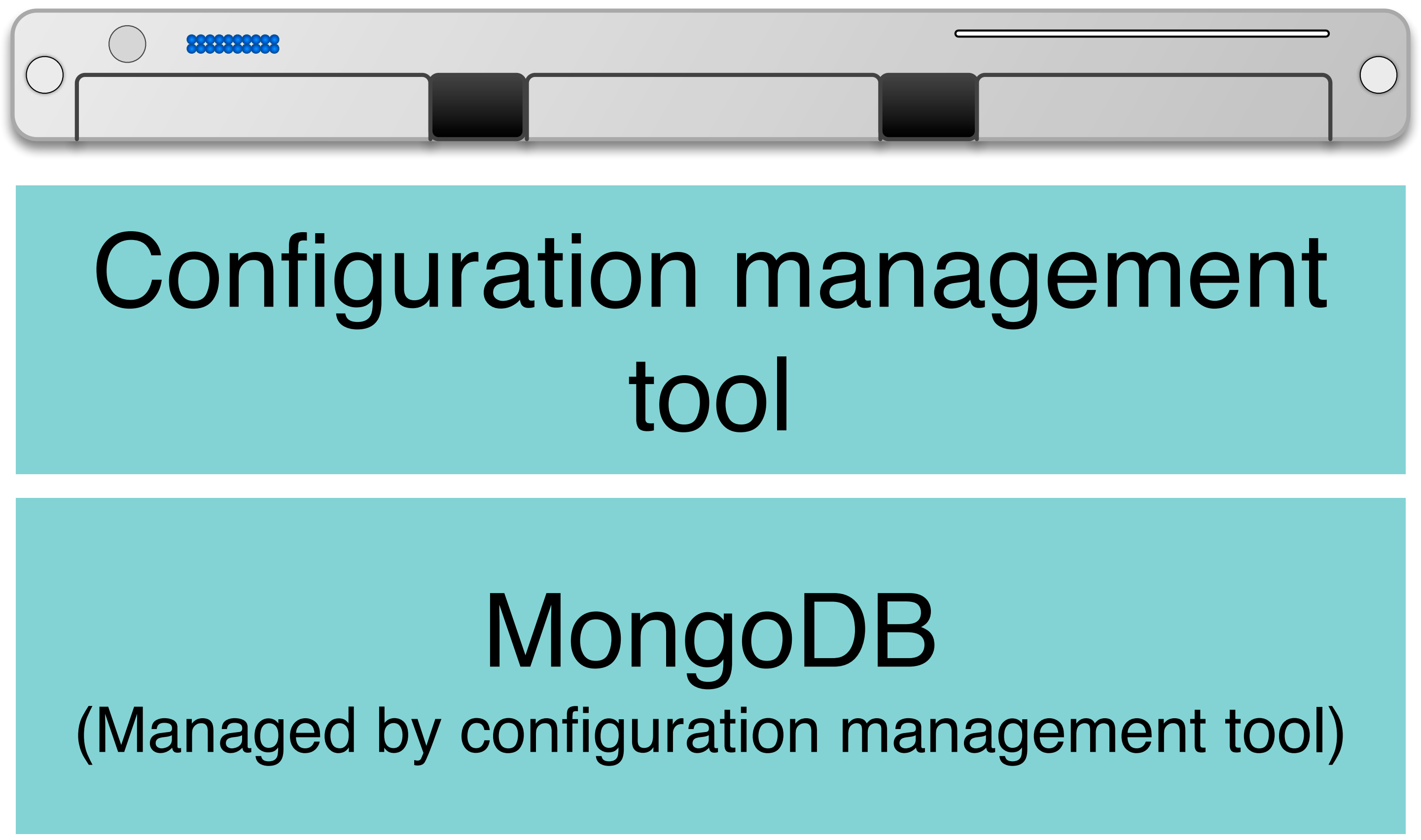
# Infrastructure As Code: Overview

- Provisioning servers is tedious and error prone
  - Deploy a VM, then ssh to it, install some packages, etc
- Keeping servers up-to-date is also a struggle
- Ideal:
  - “Give me HAProxy with some configuration file, and keep that configuration in a git repo, and when I change it, roll out an update”
  - “Give me some containers running my NodeJS app, and when I update my app, roll it out to those containers”
  - “Give me a bunch of servers with MongoDB set up in a cluster”



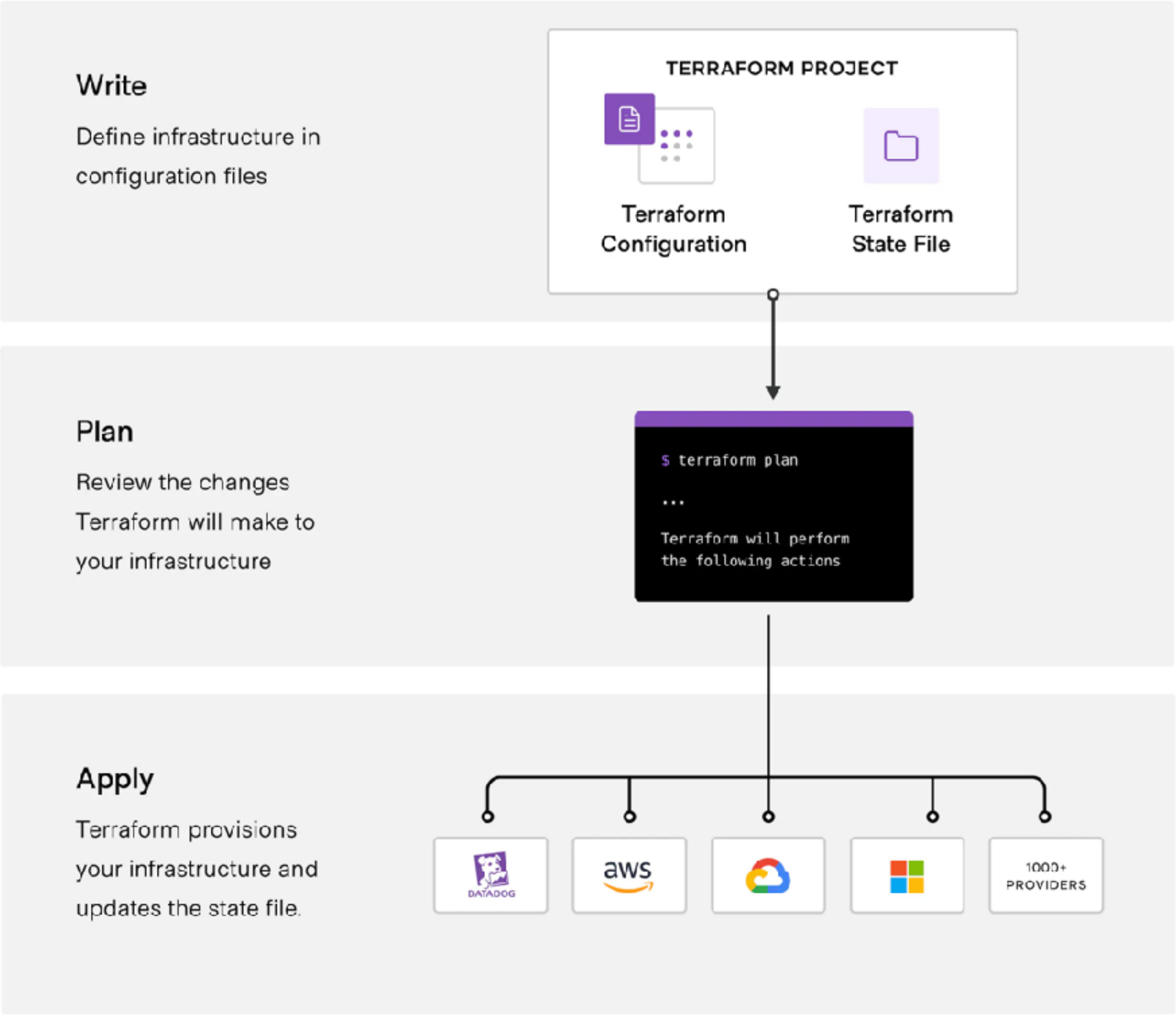
# Infrastructure as Code: Configuration Management

- Goal: Create a system that, when run, can automatically bring physical or virtual machines to some configured state
    - These configurations can then go into version control, code review, etc
- Metaphor: “Recipes” for configuring servers, organized into “cookbooks”
    - “Oh, this is how they do things at Amazon” - Inspiration for Chef, c 2009 (Apache License)
- Other tools with similar aims: Puppet (c 2005, Apache License), Ansible (c 2012, GPL)


Configuration management tool
MongoDB
(Managed by configuration management tool)

# Infrastructure as Code: Cloud Orchestration

- Goal: automatically provision public cloud resources on which we will then deploy software and configurations
- Again, those configurations are “code” in version control
- Terraform (HashiCorp, c. 2014, Mozilla Public License) as primary example
- Other configuration management tools support cloud now, management, too



Write

Define infrastructure in configuration files

Plan

Review the changes Terraform will make to your infrastructure

```
$ terraform plan
...
Terraform will perform
the following actions
```

Apply

Terraform provisions your infrastructure and updates the state file.

Screenshot: https://developer.hashicorp.com/terraform/intro

# Infrastructure as Code: Auto-Scaling Clouds

- Goal: Maximize resource utilization and application performance under dynamic workloads

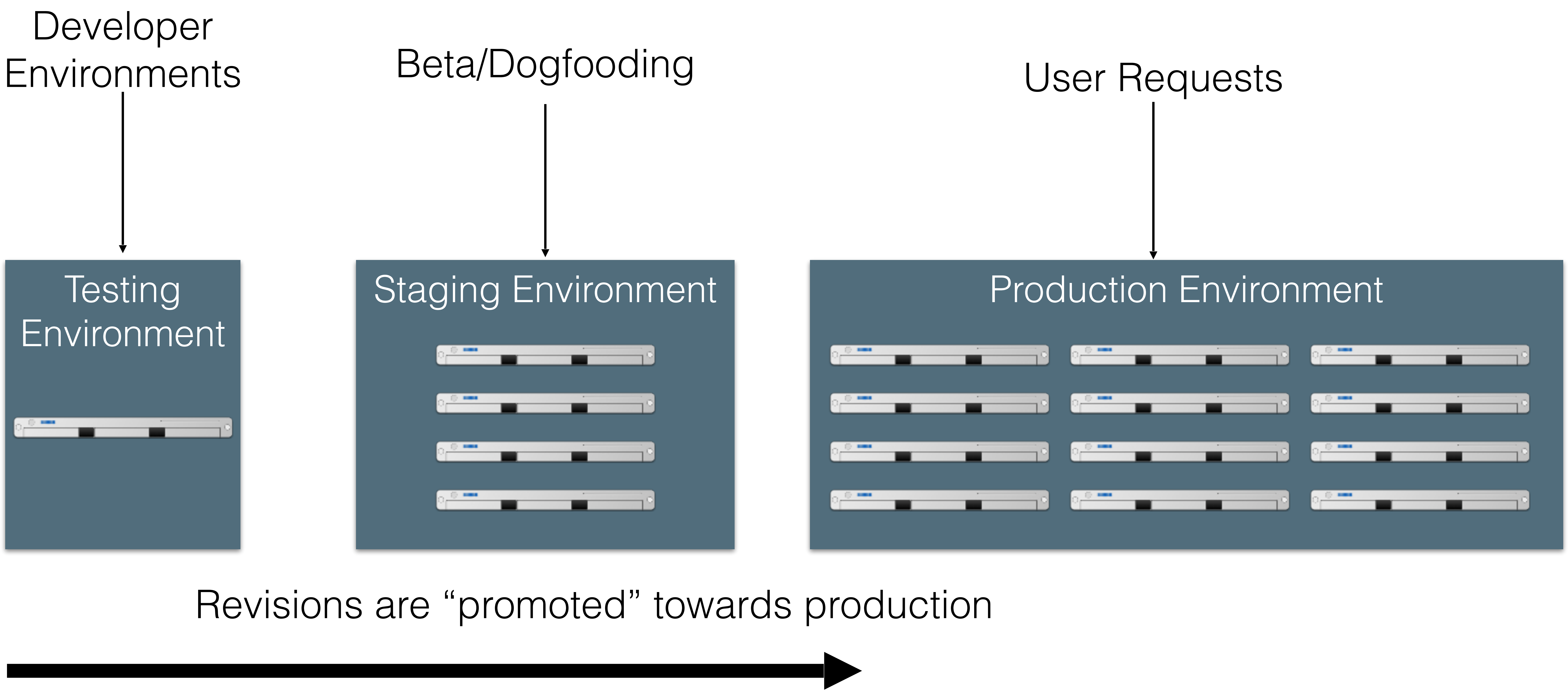
- Architecture: Each application runs in a group of containers on a large cluster (many applications on one cluster)

- Autoscale number of containers based on CPU, memory, or custom metrics

- "Borg" (Google, c 2014, proprietary), evolves into Kubernetes (Google, c 2014, Apache License)

# Continuous Delivery Leverages IaC

## IaC enables “easy” staging deployment



Q/A takes place in each stage (including production!)

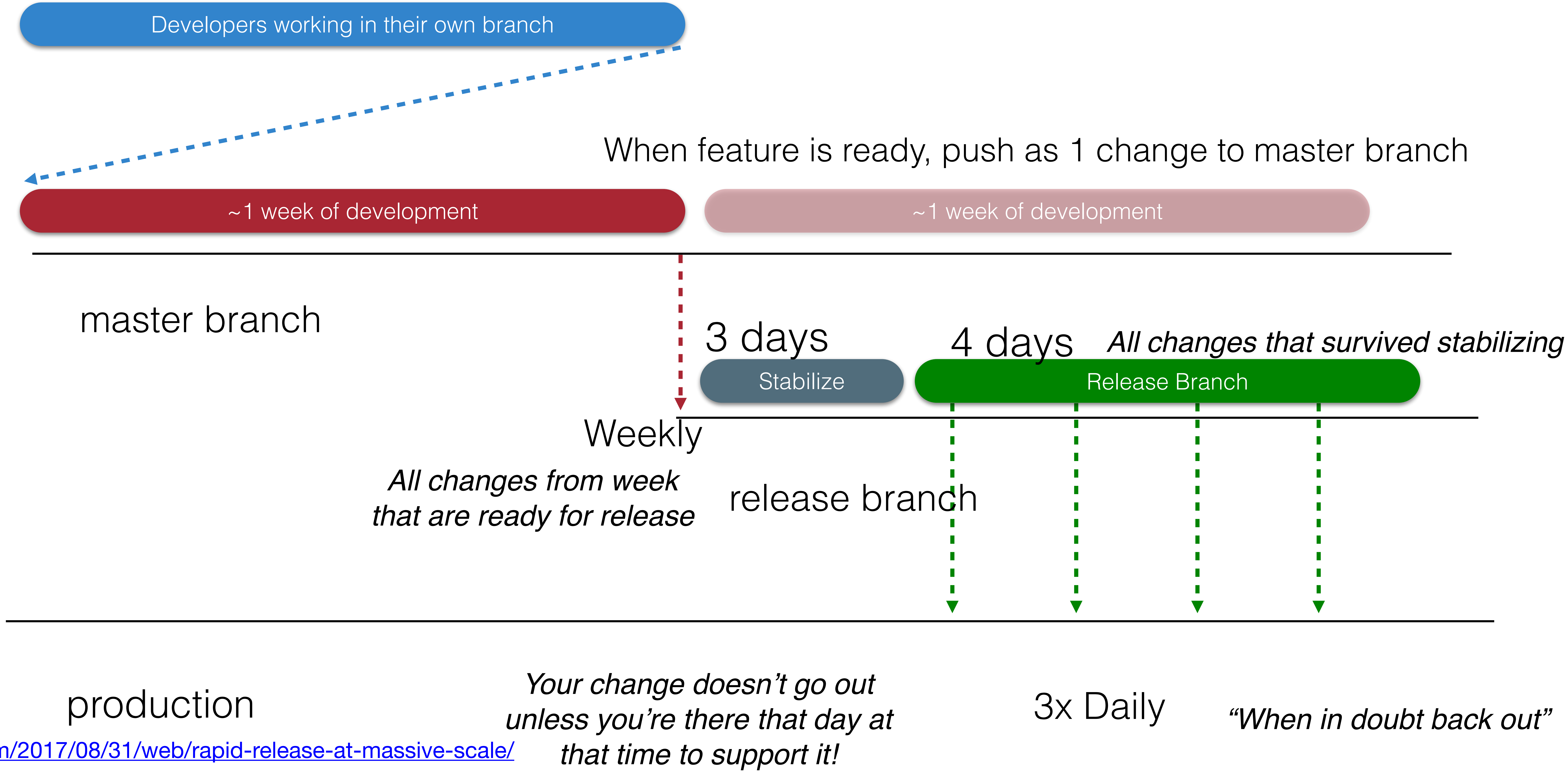
# Continuous Delivery Practices

## Release Pipelines

- Even if you are deploying every day (“continuously”), you still have some latency
- A new feature I develop today won't be released today
- But, a new feature I develop today can begin the **release pipeline** today (minimizes risk)
- **Release Engineer**: gatekeeper who decides when something is ready to go out, oversees the actual deployment process

# Deployment Example: Facebook.com

## Pre-2016



Developers working in their own branch

When feature is ready, push as 1 change to master branch

~1 week of development

~1 week of development

master branch

3 days

Stabilize

4 days

*All changes that survived stabilizing*

Release Branch

Weekly

*All changes from week that are ready for release*

release branch

production

*Your change doesn't go out unless you're there that day at that time to support it!*

3x Daily

*"When in doubt back out"*

https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

# Deployment Example: Facebook.com

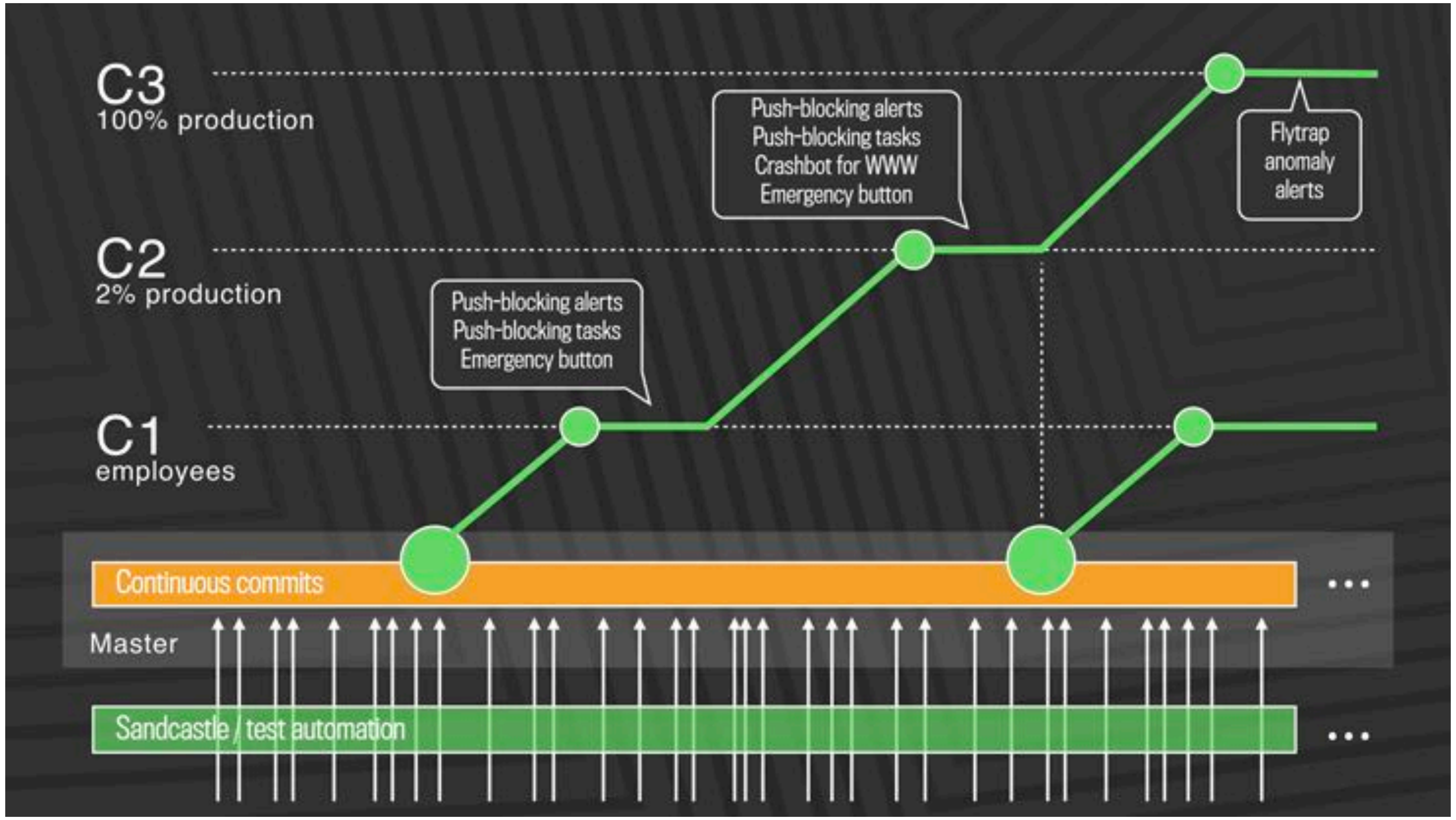
## Chuck Rossi, Director Software Infrastructure & Release Engineering @ Facebook

“Our main goal was to make sure that the new system made people’s experience better — or at the very least, didn’t make it worse. After almost exactly a year of planning and development, over the course of three days in April 2017 **we enabled 100 percent of our production web servers to run code deployed directly from master.**”

“Rapid release at massive scale” https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

# Deployment Example: Facebook.com

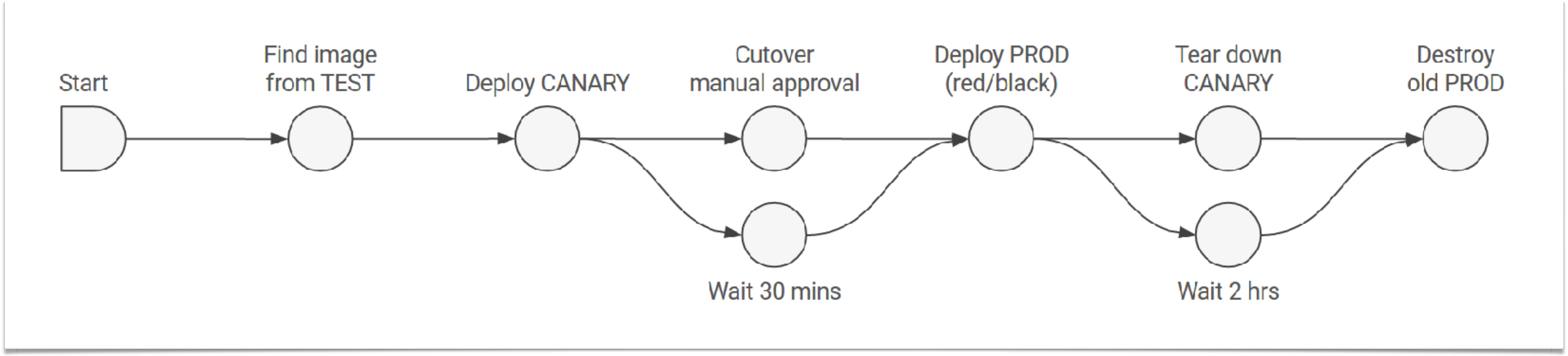
## Post-2016: Truly continuous releases from master branch



https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

# Continuous Delivery Tools

- Auto-deploys from version control to a staging environment + promotes through release pipeline
- Monitors key performance indicators to automatically take corrective actions
- Example: “Spinnaker” (Netflix, c 2015, MIT License)



Example CD pipeline from Spinnaker’s documentation: https://spinnaker.io/docs/concepts/#application-deployment

# Continuous Delivery Relies on Monitoring

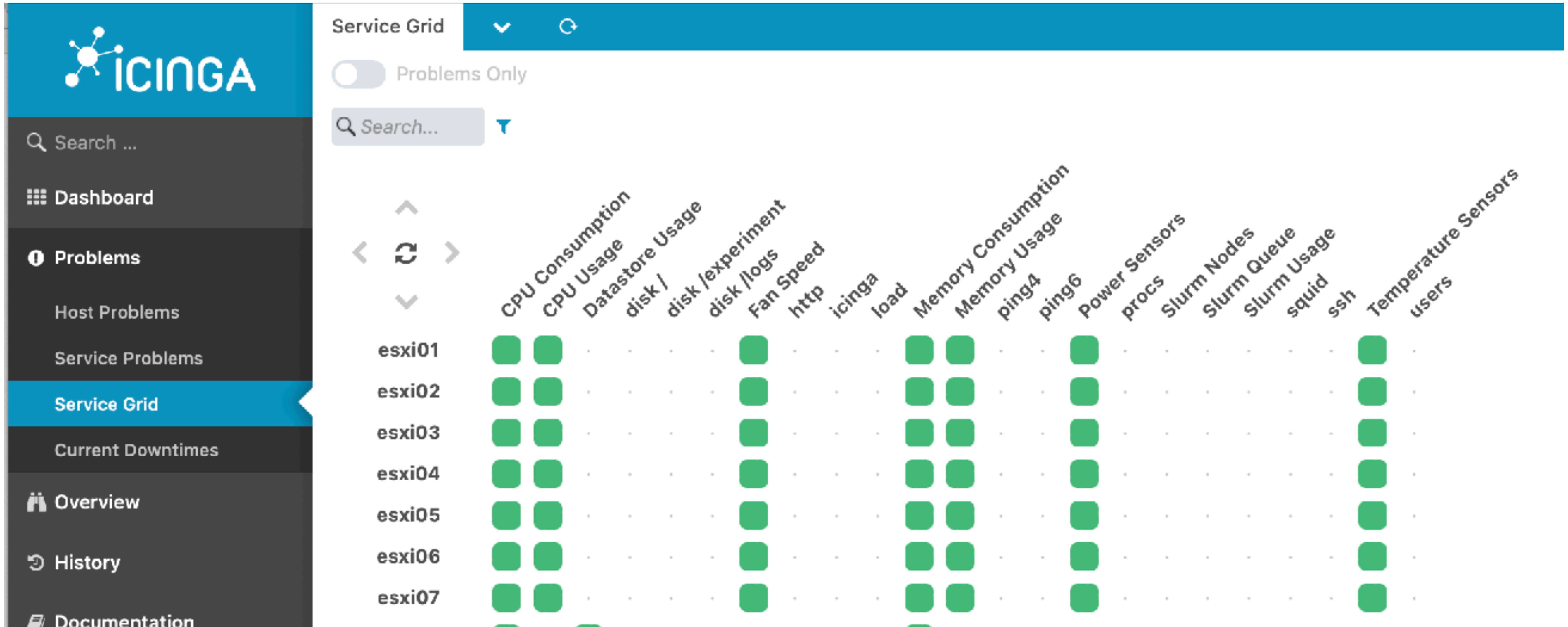
## Consider both direct (e.g. business) metrics, and indirect (e.g. system) metrics

- Hardware
  - Voltages, temperatures, fan speeds, component health
- OS
  - Memory usage, swap usage, disk space, CPU load
- Middleware
  - Memory, thread/db connection pools, connections, response time
- Applications
  - Business transactions, conversion rate, status of 3rd party components

# Tools for Monitoring Deployments

- Nagios (c 2002, GPL): Agent-based architecture (install agent on each monitored host), extensible plugins for executing “checks” on hosts

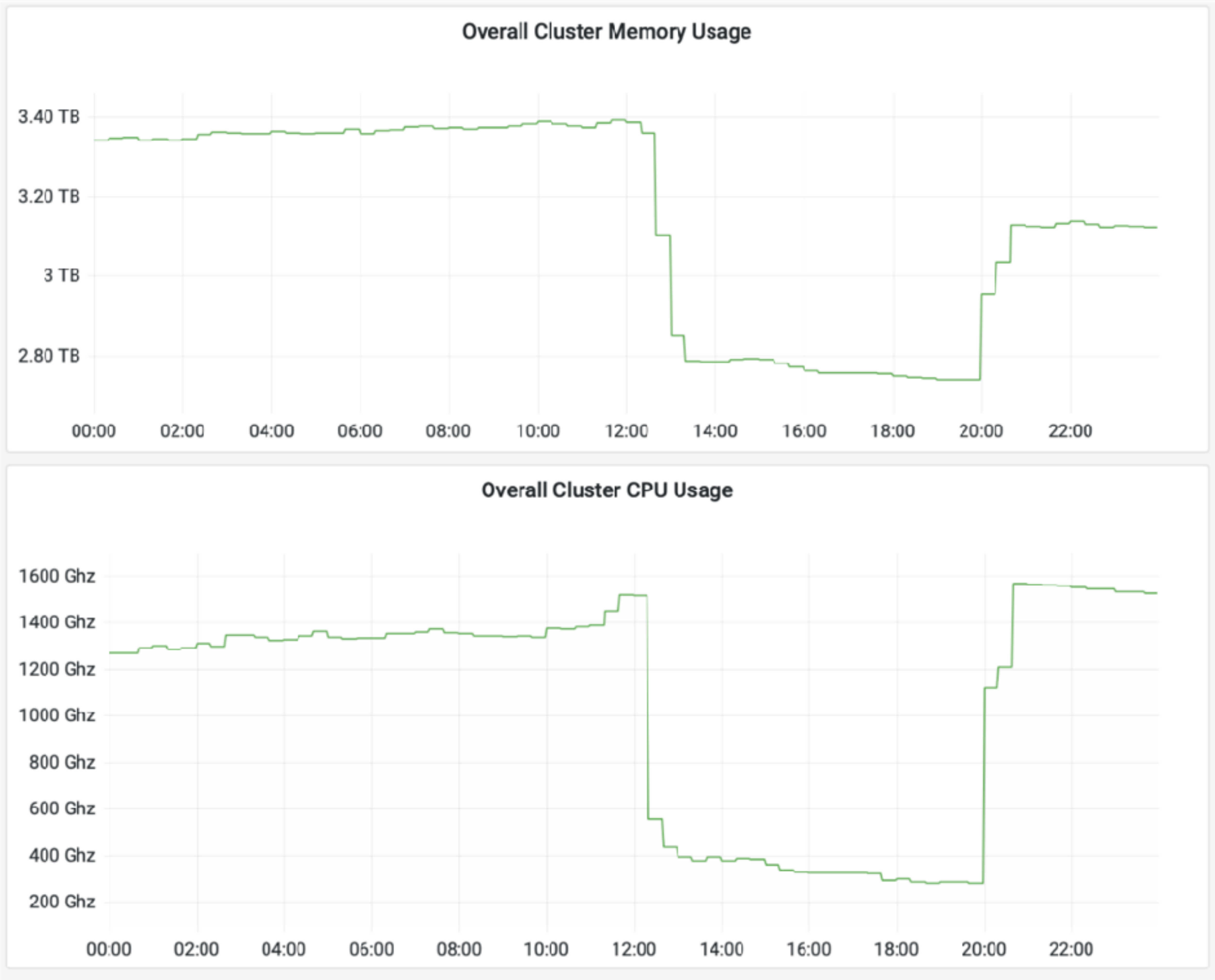
- Three significant forks: Icinga (c 2009), Shinken (c 2009), Naemon (c 2015)



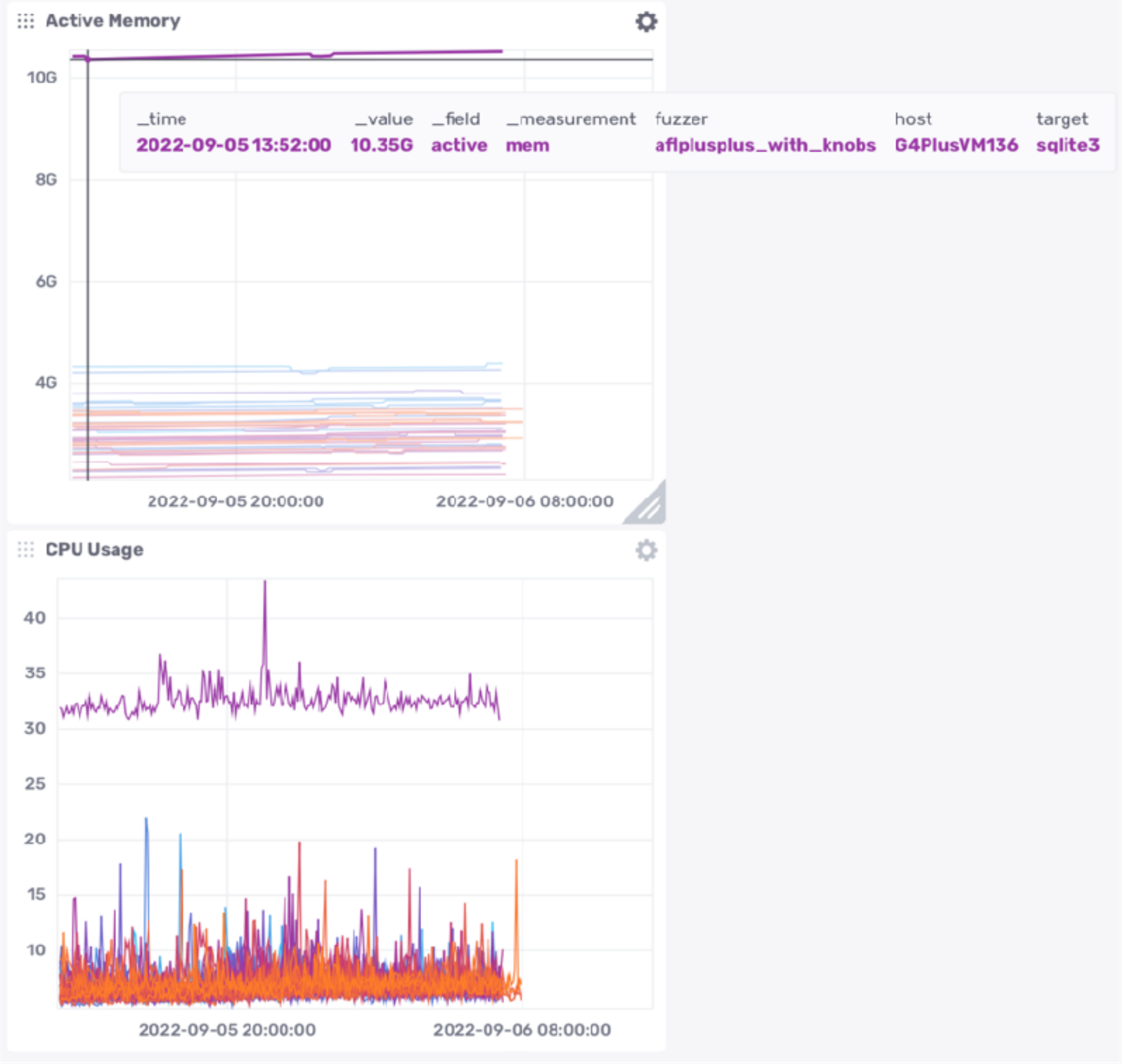
| | CPU Consumption | CPU Usage | Datastore Usage | disk / | disk /experiment | disk /logs | Fan Speed | http | icinga | load | Memory Consumption | Memory Usage | ping4 | ping6 | Power Sensors | procs | Slurm Nodes | Slurm Queue | Slurm Usage | squid | ssh | Temperature Sensors | users |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| esxi01 | 🟩 | 🟩 | · | · | · | · | 🟩 | · | · | · | 🟩 | 🟩 | · | · | 🟩 | · | · | · | · | · | · | 🟩 | · |
| esxi02 | 🟩 | 🟩 | · | · | · | · | 🟩 | · | · | · | 🟩 | 🟩 | · | · | 🟩 | · | · | · | · | · | · | 🟩 | · |
| esxi03 | 🟩 | 🟩 | · | · | · | · | 🟩 | · | · | · | 🟩 | 🟩 | · | · | 🟩 | · | · | · | · | · | · | 🟩 | · |
| esxi04 | 🟩 | 🟩 | · | · | · | · | 🟩 | · | · | · | 🟩 | 🟩 | · | · | 🟩 | · | · | · | · | · | · | 🟩 | · |
| esxi05 | 🟩 | 🟩 | · | · | · | · | 🟩 | · | · | · | 🟩 | 🟩 | · | · | 🟩 | · | · | · | · | · | · | 🟩 | · |
| esxi06 | 🟩 | 🟩 | · | · | · | · | 🟩 | · | · | · | 🟩 | 🟩 | · | · | 🟩 | · | · | · | · | · | · | 🟩 | · |
| esxi07 | 🟩 | 🟩 | · | · | · | · | 🟩 | · | · | · | 🟩 | 🟩 | · | · | 🟩 | · | · | · | · | · | · | 🟩 | · |

# Monitoring can help identify operational issues

## Specialized tooling for building dashboards



Grafana (AGPL, c 2014)

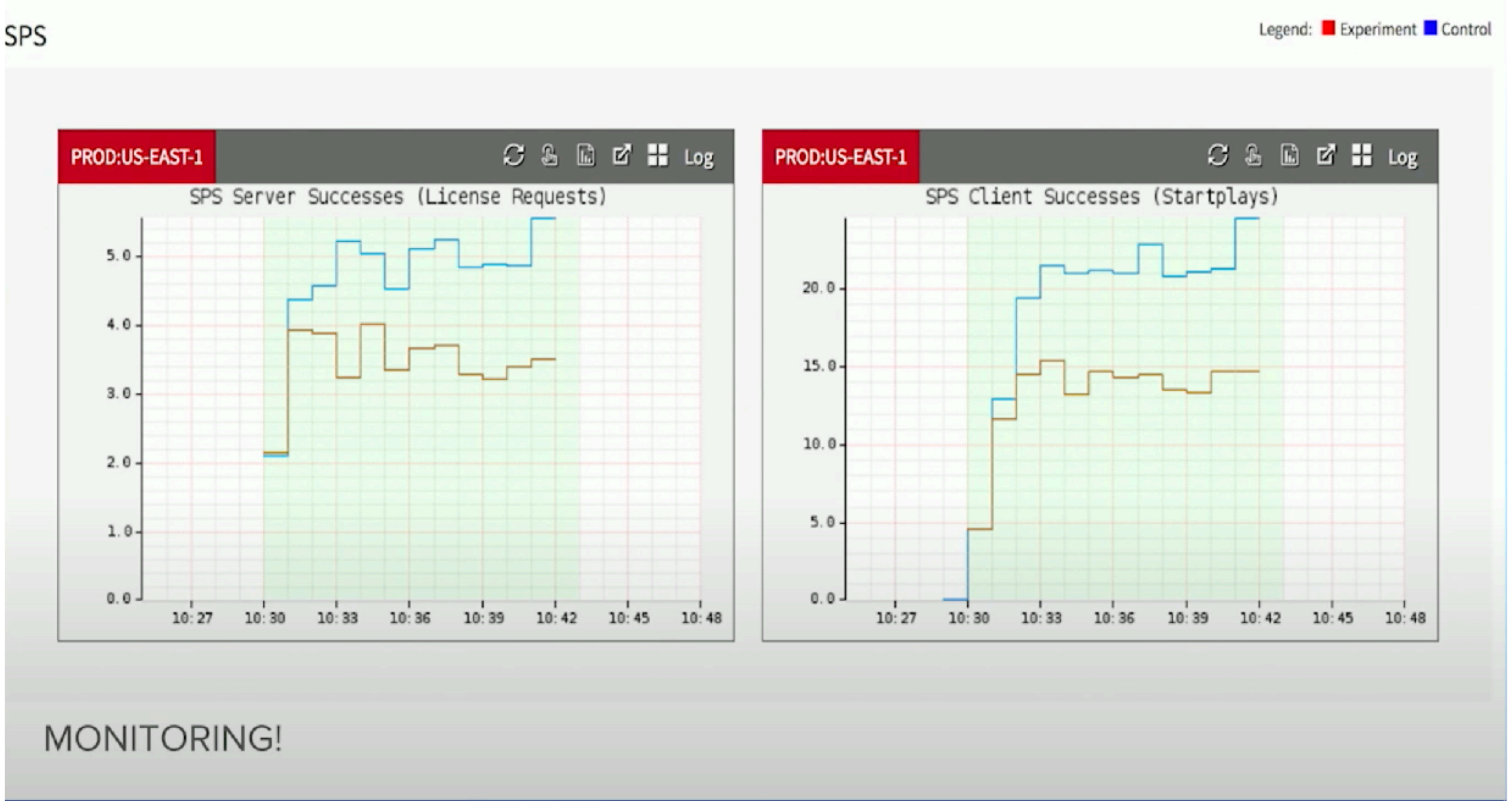


| _time | _value | _field | _measurement | fuzzer | host | target |
|---|---|---|---|---|---|---|
| 2022-09-05 13:52:00 | 10.35G | active | mem | aflplusplus_with_knobs | G4PlusVM136 | sqlite3 |

InfluxDB (MIT license, c 2013)

# Continuous Delivery Tools Take Automated Actions

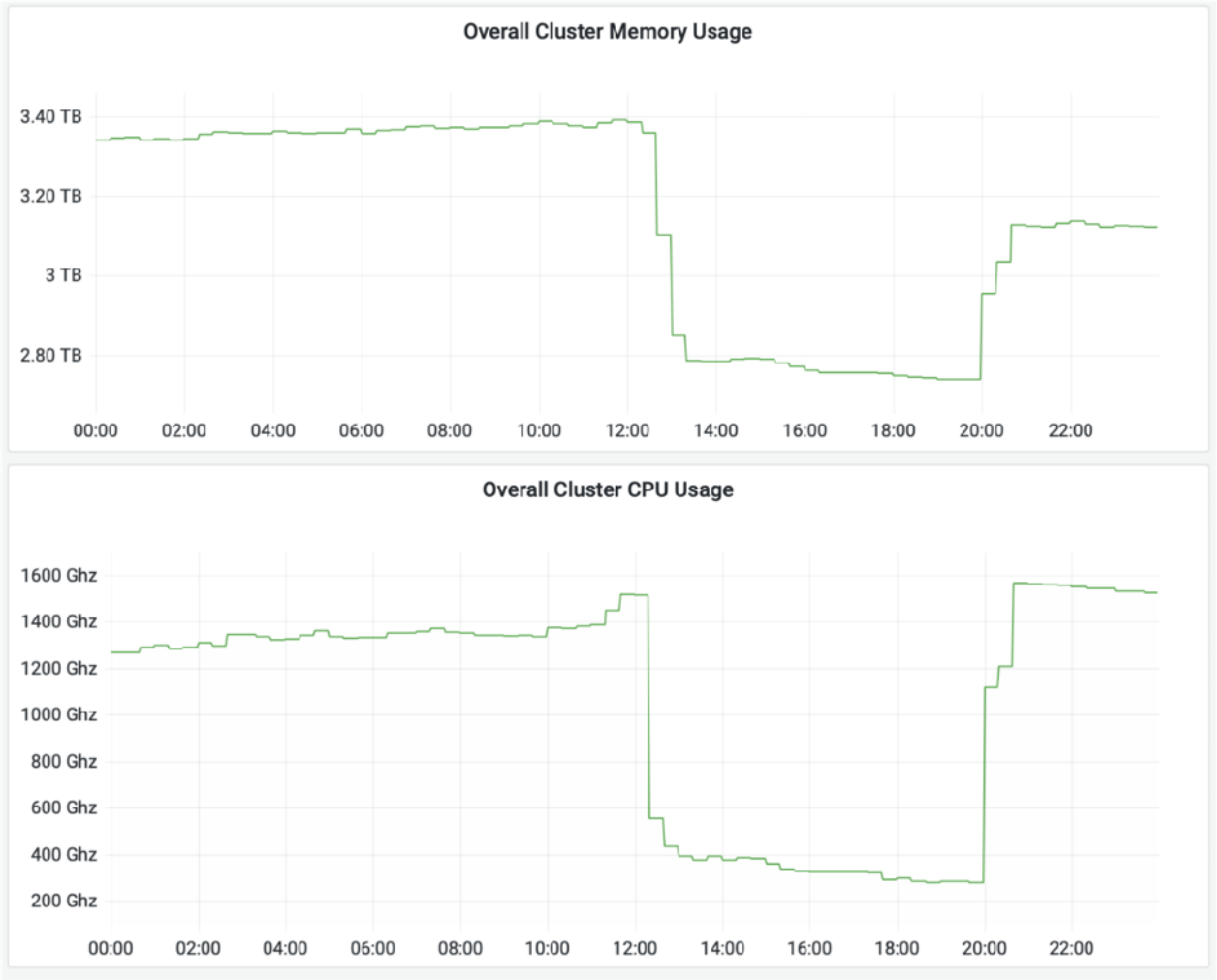
## Automated roll-back of updates at Netflix based on SPS



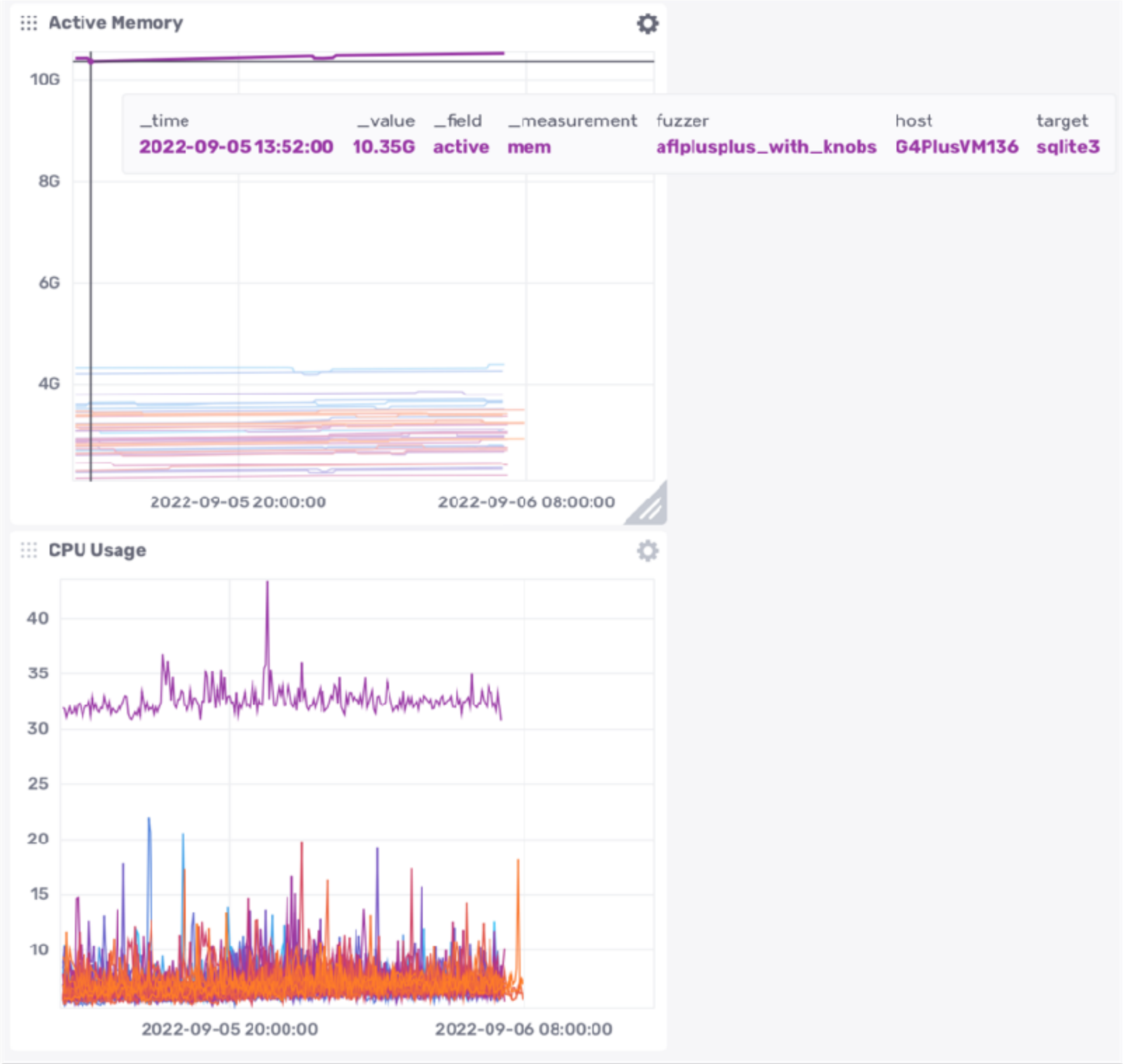
https://www.youtube.com/watch?v=qyzymLlj9ag

# From Monitoring to Observability

## Understanding what is going on inside of our deployed systems



Grafana (AGPL, c 2014)

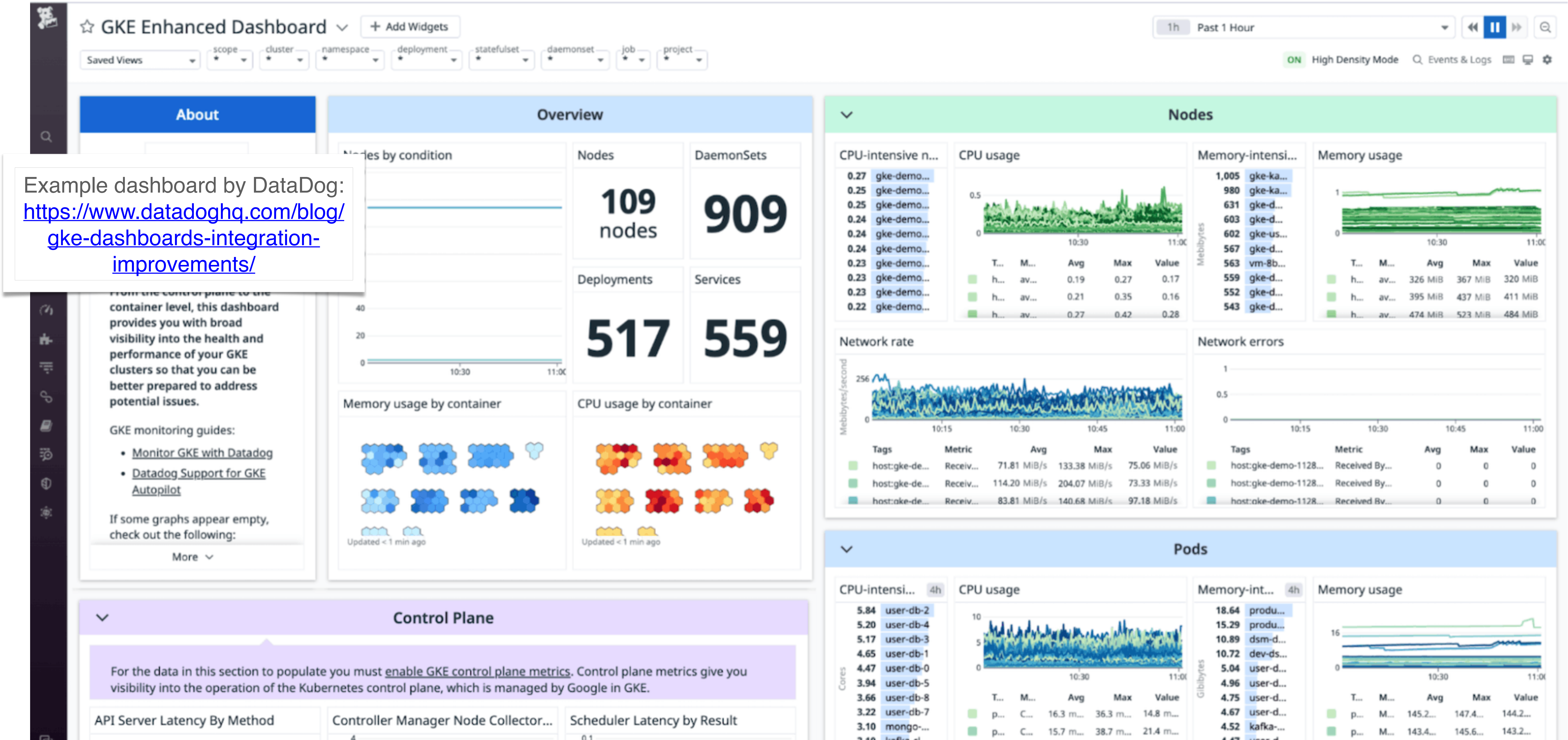


| _time | _value | _field | _measurement | fuzzer | host | target |
|---|---|---|---|---|---|---|
| 2022-09-05 13:52:00 | 10.35G | active | mem | aflplusplus_with_knobs | G4PlusVM136 | sqlite3 |

InfluxDB (MIT license, c 2013)

# From Monitoring to Observability

## Understanding what is going on inside of our deployed systems



Nodes: 109 nodes

DaemonSets: 909

Deployments: 517

Services: 559

From the control plane to the container level, this dashboard provides you with broad visibility into the health and performance of your GKE clusters so that you can be better prepared to address potential issues.

GKE monitoring guides:

- Monitor GKE with Datadog
- Datadog Support for GKE Autopilot

If some graphs appear empty, check out the following:

For the data in this section to populate you must enable GKE control plane metrics. Control plane metrics give you visibility into the operation of the Kubernetes control plane, which is managed by Google in GKE.

API Server Latency By Method

Controller Manager Node Collector...

Scheduler Latency by Result

| CPU-intensive n... | |
|---|---|
| 0.27 | gke-demo... |
| 0.25 | gke-demo... |
| 0.25 | gke-demo... |
| 0.24 | gke-demo... |
| 0.24 | gke-demo... |
| 0.24 | gke-demo... |
| 0.23 | gke-demo... |
| 0.23 | gke-demo... |
| 0.23 | gke-demo... |
| 0.22 | gke-demo... |

| Memory-intensi... | |
|---|---|
| 1,005 | gke-ka... |
| 980 | gke-ka... |
| 631 | gke-d... |
| 603 | gke-d... |
| 602 | gke-us... |
| 567 | gke-d... |
| 563 | vm-8b... |
| 559 | gke-d... |
| 552 | gke-d... |
| 543 | gke-d... |

| Tags | Metric | Avg | Max | Value |
|---|---|---|---|---|
| host:gke-de... | Receiv... | 71.81 MiB/s | 133.38 MiB/s | 75.06 MiB/s |
| host:gke-de... | Receiv... | 114.20 MiB/s | 204.07 MiB/s | 73.33 MiB/s |
| host:gke-de... | Receiv... | 83.81 MiB/s | 140.68 MiB/s | 97.18 MiB/s |

| Tags | Metric | Avg | Max | Value |
|---|---|---|---|---|
| host:gke-demo-1128... | Received By... | 0 | 0 | 0 |
| host:gke-demo-1128... | Received By... | 0 | 0 | 0 |
| host:gke-demo-1128... | Received By... | 0 | 0 | 0 |

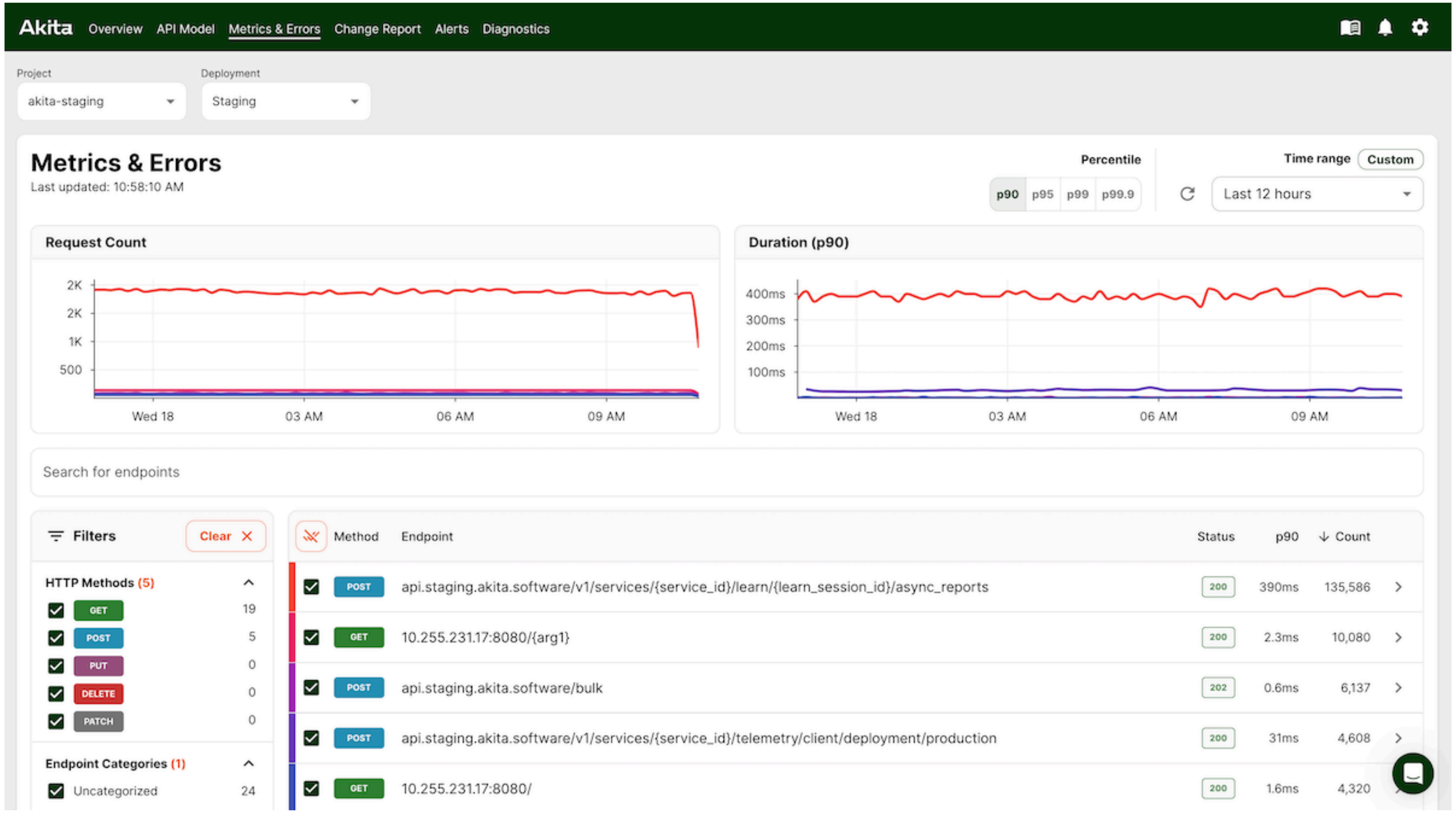
| CPU-intensi... 4h | |
|---|---|
| 5.84 | user-db-2 |
| 5.20 | user-db-4 |
| 5.17 | user-db-3 |
| 4.65 | user-db-1 |
| 4.47 | user-db-0 |
| 3.94 | user-db-5 |
| 3.66 | user-db-8 |
| 3.22 | user-db-7 |
| 3.10 | mongo-... |

| Memory-int... 4h | |
|---|---|
| 18.64 | produ... |
| 15.29 | produ... |
| 10.89 | dsm-d... |
| 10.72 | dev-ds... |
| 5.04 | user-d... |
| 4.96 | user-d... |
| 4.75 | user-d... |
| 4.67 | user-d... |
| 4.52 | kafka-... |

Example dashboard by DataDog: https://www.datadoghq.com/blog/gke-dashboards-integration-improvements/

# Consider Observability of Apps, Too

## Track latency, error rates, etc. to discover problems before



Project: akita-staging

Deployment: Staging

### Metrics & Errors

Last updated: 10:58:10 AM

Percentile: p90

Time range: Last 12 hours

Filters

HTTP Methods (5)

| | | |
|---|---|---|
| ☑ | GET | 19 |
| ☑ | POST | 5 |
| ☑ | PUT | 0 |
| ☑ | DELETE | 0 |
| ☑ | PATCH | 0 |

Endpoint Categories (1)

| | | |
|---|---|---|
| ☑ | Uncategorized | 24 |

| | Method | Endpoint | Status | p90 | ↓ Count |
|---|---|---|---|---|---|
| ☑ | POST | api.staging.akita.software/v1/services/{service_id}/learn/{learn_session_id}/async_reports | 200 | 390ms | 135,586 |
| ☑ | GET | 10.255.231.17:8080/{arg1} | 200 | 2.3ms | 10,080 |
| ☑ | POST | api.staging.akita.software/bulk | 202 | 0.6ms | 6,137 |
| ☑ | POST | api.staging.akita.software/v1/services/{service_id}/telemetry/client/deployment/production | 200 | 31ms | 4,608 |
| ☑ | GET | 10.255.231.17:8080/ | 200 | 1.6ms | 4,320 |

Screenshot: https://www.akitasoftware.com/blog-posts/plug-and-play-endpoint-views-for-metrics-errors

# Monitoring Services Take Automated Actions

iCINGA
Search ...
Dashboard
Problems
Overview
History
Event Grid
Event Overview
Notifications
Timeline
Documentation
System
Configuration
jon
Notifications
« 1 2 3 4 5 6 7 ... 24 25 » # 25 Sort by Notification Start
Search...
Notification

| | | |
|---|---|---|
| OK<br>2022-02-18<br>08:49:05 | **Slurm Nodes** on **nagios**<br>`OK - 0 nodes unreachable, 332 reachable` | Sent to jon |
| OK<br>2022-02-18<br>08:49:05 | **Slurm Nodes** on **nagios**<br>`OK - 0 nodes unreachable, 332 reachable` | Sent to icingaadmin |
| WARNING<br>2022-02-18<br>08:45:05 | **Slurm Nodes** on **nagios**<br>`WARNING - 7 nodes unreachable, 326 reachable` | Sent to jon |
| WARNING<br>2022-02-18<br>08:45:05 | **Slurm Nodes** on **nagios**<br>`WARNING - 7 nodes unreachable, 326 reachable` | Sent to icingaadmin |
| CRITICAL<br>2022-02-18<br>08:42:05 | **Slurm Nodes** on **nagios**<br>`CRITICAL - 65 nodes unreachable, 161 reachable` | Sent to icingaadmin |
| CRITICAL<br>2022-02-18<br>08:42:05 | **Slurm Nodes** on **nagios**<br>`CRITICAL - 65 nodes unreachable, 161 reachable` | Sent to jon |
| WARNING<br>2022-02-18<br>08:40:05 | **Slurm Nodes** on **nagios**<br>`WARNING - 12 nodes unreachable, 205 reachable` | Sent to icingaadmin |
| WARNING<br>2022-02-18<br>08:40:05 | **Slurm Nodes** on **nagios**<br>`WARNING - 12 nodes unreachable, 205 reachable` | Sent to jon |
| CRITICAL<br>2022-02-18<br>08:34:07 | **Slurm Nodes** on **nagios**<br>`CRITICAL - 204 nodes unreachable, 145 reachable` | Sent to icingaadmin |

## Current Service State

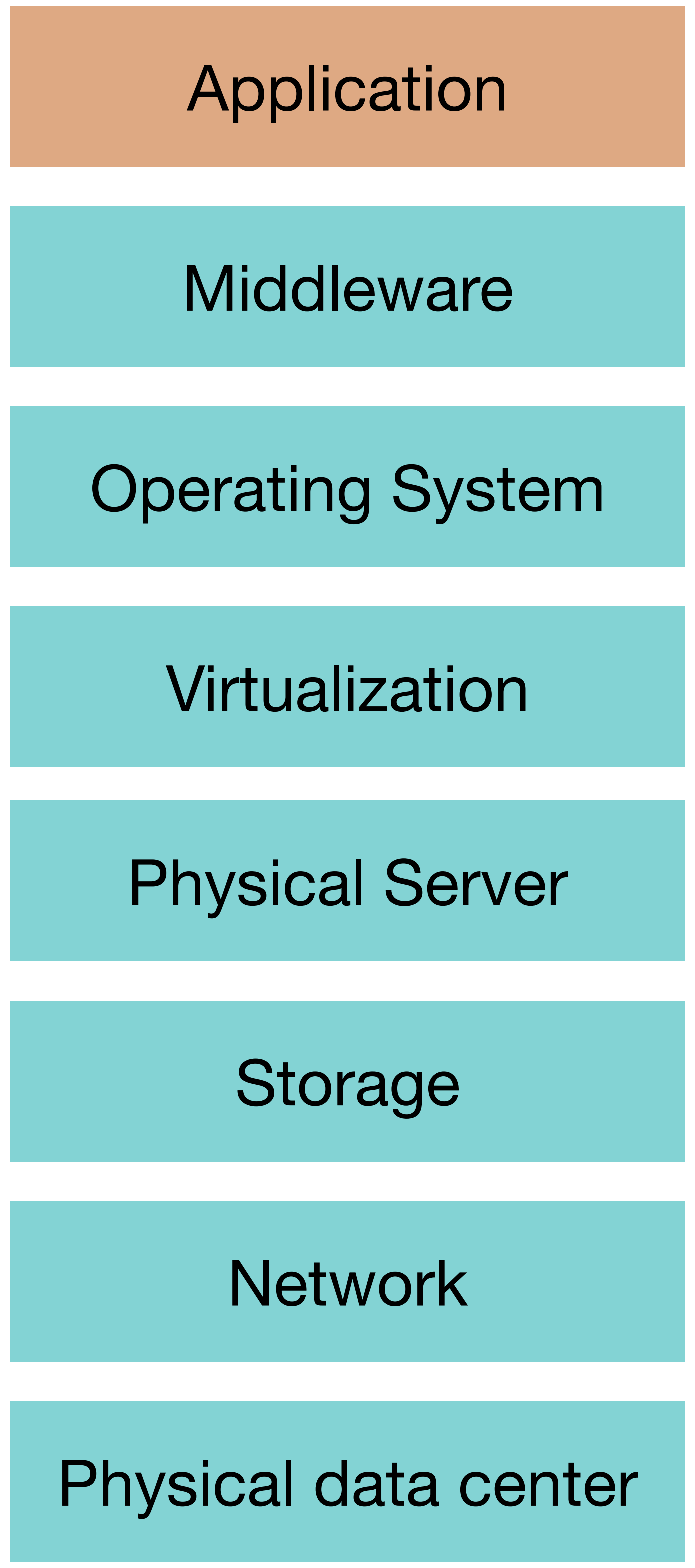
| | |
|---|---|
| UP<br>since 2021-11 | **nagios**<br>::1<br>127.0.0.1 |
| OK<br>for 1m 52s | Service: **Slurm Nodes** |

## Event Details

| | |
|---|---|
| Type | Notification |
| Start time | 2022-02-18 08:42:05 |
| End time | 2022-02-18 08:42:05 |
| Reason | Normal notification |
| State | 🟥 CRITICAL |
| Escalated | No |
| Contacts notified | 2 |
| Output | `CRITICAL - 65 nodes unreachable, 161 reachable` |

# PaaS is the Simplest Choice for App Deployment

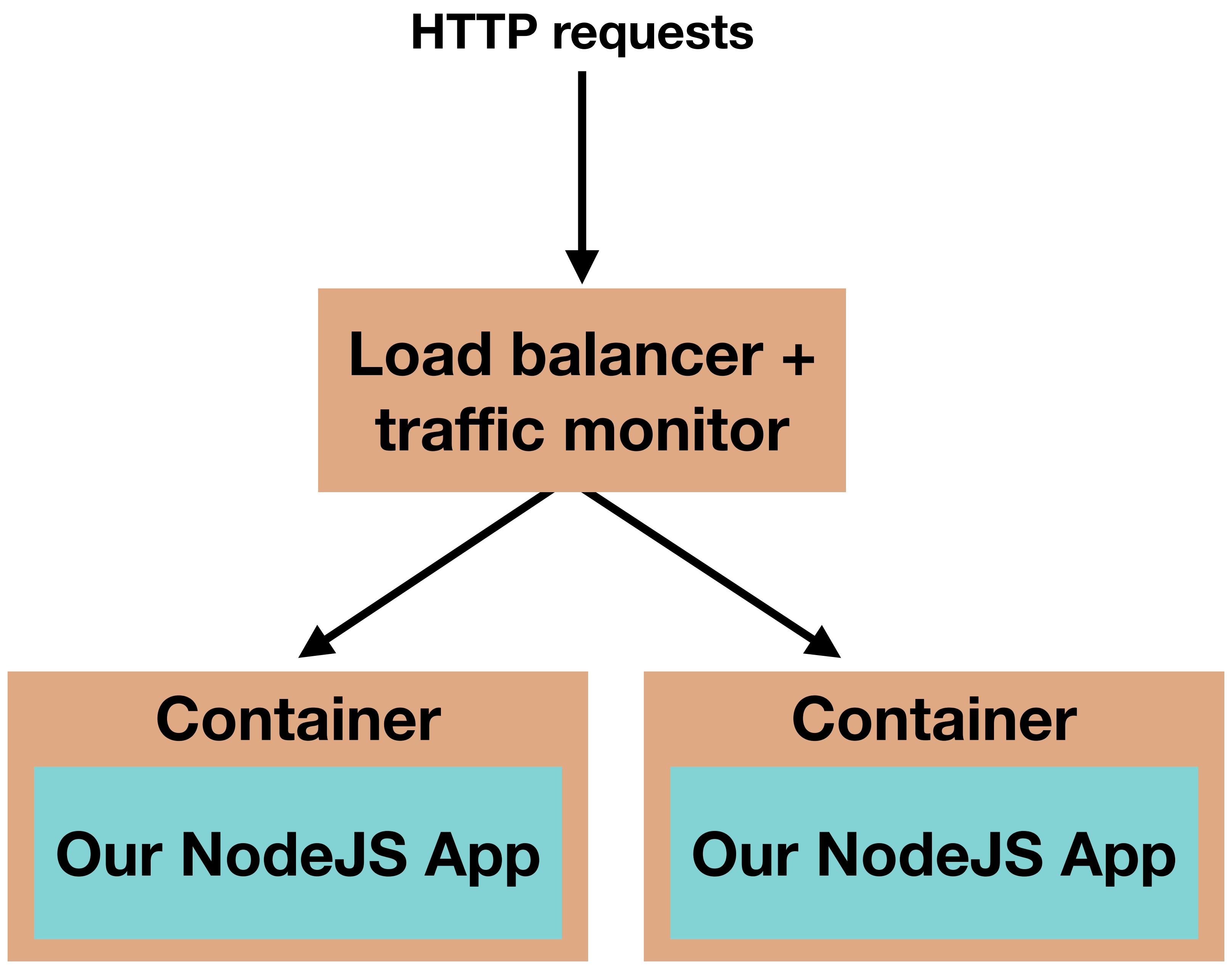
- Platform-as-a-Service (PaaS) products provide common components that most apps need, fully managed by the vendor: load balancer, monitoring, application server
  - Examples: Heroku, AWS Elastic Beanstalk, Google App Engine
- Some PaaS products are designed to deploy apps as *single functions* that are invoked when a web request is made, and don’t run otherwise (“function-as-a-service”)
  - Examples: AWS Lambda, Google Cloud Functions, Azure Functions
- Some PaaS products also provide databases and authentication
  - Examples: Google Firebase, Back4App



PaaS

# Heroku is a Platform as a Service

- Takes as input: a web app (e.g. NodeJS app)
    - No need to provide a container, entry point to our code is enough, e.g. “npm start”
- Provides: hosted web app at our choice of URL, with ability to scale resources up/down on-demand
    - Load balancer is fully managed by Heroku, makes scaling transparent
    - Can auto-scale down to use no resources, then only launch a container once a request has been received
    - Dashboard provides monitoring/reporting



# Next Steps

- Thursday’s discussion:
  - Canopy (end-to-end performance tracing for large systems at Facebook)
    - Pay more attention to the problem that they are solving and what the evaluation shows as opposed to how they implemented this
  - Study of configuration evolution in cloud systems
- Next week: Collaboration in SE, project status update Tues by 11am