Solving the Hidden Costs of Kubernetes
Who Am I?

Austin Parker
Principal Developer Advocate

@austinlparker

austin@lightstep.com
Kubernetes
Why Kubernetes?

Kubernetes provides a convenient abstraction for deploying, scaling, and running containerized services in a distributed system.

However, this convenience has come at a price.
What Kubernetes Doesn’t Do

Kubernetes doesn’t…

- Make infrastructure easier to manage by magic
- Make it easy to understand the changes in performance resulting from a deployment
- Help you aggregate telemetry data from your services without additional layers of tooling
- Solve traditional monitoring questions, like how to understand unknown–unknowns
Observe

Control

Observe
Stress (n): responsibility without control

What you can control

What you are responsible for
Hidden Costs
Monitoring Costs

Kubernetes adds complexity and cost to traditional time-series metric and log aggregation monitoring solutions.

As your application scales, tag count increases, which leads to increased bills.

Building it yourself? We’ll get to that...
Martin Fowler (in 2015) argues that the “microservices premium” incurs such a high price that you’ll want to build a monolith first, even if the application will be eventually migrated to a microservice architecture.

Kelsey Hightower is arguing for monoliths in 2020.

Time is a flat circle.
Bus Factor

Not only do we increase the surface area of things that can go wrong, but those failure states are less accessible on average without domain experts.

How resilient are you to…

- Managed service failure/deprecation
- Loss of institutional knowledge
- Unforeseen externalities
Observability
Observability is the process and practice of understanding your system.
Lightstep root-causes performance problems anywhere, from mobile all the way to the bottom of our distributed stack. This is the future of monitoring.

Observability is the ability to navigate from effect to cause.
What’s the difference?

Monitoring, as a discipline, requires you to pre-define *normal* and then freeze it, with ruthless efficiency, suborning agility and humanity and adaptiveness in the sake of producing a steady-state system.

There’s too many unknowns in modern software to just throw together some dashboards and call it a day.

Think about the things you might need or want to know...

- Application logs and metrics
- Kubelet statistics (CPU, Memory)
  - Container, Pod, Namespace, Cluster, Node...
- Resource Utilization
- Request tracing
How does observability help?

Observability provides a comprehensive and holistic approach to understanding your entire system, not just individual pieces. It accomplishes this by focusing on cultural and process changes, empowered by a different way of thinking about data.

It’s not just a new tool or dashboard to add to your existing mix.
Quantifying Hidden Costs
What it really costs to monitor a system

There’s two primary costs associated with traditional monitoring systems:

- Capital costs required for collection and storage of telemetry
- Operational costs required for engineering time to investigate telemetry
Calculating Capital Costs

Initial Factors

- Aggregating + Indexing Logs per Service:
  - Storage
  - Compute
  - Network
- Peak instance count
- Retention period
- Services involved in a request.

Initial Values

Assuming 50GB of log data a day, 14 day retention, high availability (no cold storage)

- 1 Master (Large Compute Optimized Instance) @ $89
- 2 Data (XL Memory Optimized Instance) @ $426
- 3 SSDs (General Purpose) @ $201
$716

Cloud Spend per Month @ 50GB/logs
~$3,386

Total after setup, maintenance (monthly)
Calculating Operational Costs

Initial factors

- How much does an engineer cost?
- How much time do they work in a year?
- How many disruptions are investigated per year?
- How long does an investigation take?
- How many developers are involved in investigating disruptions?
- How many deployments are being made, per developer?
- How many minutes are spent validating those deployments?

Initial values

We calculate an average annual cost of $202,500 for an engineer.

Each engineer works 2,268 hours a year, for an hourly cost of $89.

We calculate 10 disruptions per year, with 4 hours average per disruption in investigation time.

We calculate 50 deployments per developer per year (normalized), with 30 minutes of validation.
$3,571
Cost per disruption
$133,929

Annual cost of validating deployments
Efficiency is key!

Effective use of observability can reduce time spent on validating deployments and investigating disruptions by a factor of 5 or more!

What does that look like?

$3,571  $133,929
Efficiency is key!

Effective use of observability can reduce time spent on validating deployments and investigating disruptions by a factor of 5 or more!

What does that look like?

$3,571  $133,929
Efficiency is key!

Effective use of observability can reduce time spent on validating deployments and investigating disruptions by a factor of 5 or more!

What does that look like?

$712 $26,700
Holistic Observability
Tackling Hidden Costs

Holistic approaches to observability utilize context as a mechanism to control costs, both capital and operational.

This context is generated in two ways.
Communication Context

Observability gives us a *shared language* to discuss performance through *Service Level Objectives* and *Service Level Indicators*.

Observability helps avoid *alarm fatigue* by focusing on the data that’s important during a disruption.

This shared context helps your team save time and set goals more efficiently.
Data Context

Observability is built by combining multiple data sources (logs, metrics, traces) into *insights* rather than through separate dashboards and workflows.

Request traces provide context that reduces the search space for relevant metrics and logs.

These efficiencies not only help save time, but they reduce the storage and processing cost of sifting through telemetry data.
Implementing Observability
Keys to Implementing Observability

Make sure your services are observable.

Focus on key SLI’s, make sure SLO’s focus on business goals.

Put people at the center of your strategy.
Use open standards like OpenTelemetry for instrumenting your service code. Think about service meshes, like Istio, as a way to quickly bootstrap request tracing.

OpenTelemetry provides a single set of APIs, SDKs, and tools for generating distributed traces and metrics from your services.
Create Effective SLOs

Standardize around the ‘golden signals’ – latency, throughput, errors.

Prefer SLO’s that are aligned with business outcomes; Less thinking about “five nines” and more about “when people try to play a song, it plays”
Focus on People

Use observability insights and data as a component of your retrospectives and sprint planning.

Kubernetes increases complexity; Observability can make it more accessible to people that aren’t “experts”.

Tools (like Kubernetes!) are supposed to make hard things easier -- observability lets you see the impact of this, and not get surprised or burned out by changes.
Get Started Today

go.lightstep.com/trial
Q&A
Thank you.