

What is Happening?

Attempting to Understand Our Systems

About Me

(obligatory sales pitch)

Beau Lyddon

Managing Partner at Real Kinetic

Currently live in beautiful Boulder, CO



Started a company:



Real Kinetic mentors clients to enable their technical teams to grow and build high-quality software

workiva



PSA:

I use a ton of slides so those offline can follow my narrative using just the slides.

So don't worry about reading every word as I will be verbalizing them out loud.

Also, I'm going to go real fast through the beginning. I'm cramming a lot in 30 min.

Let's get going

**Every company is becoming a
technology company**



Technology (especially software) is becoming a critical piece of every business

It's more and more difficult to
do jobs without understanding
technology

And it's not really about architecture diagrams

(They're needed but only part of the story)

It's about the “Why”

(From Andrew’s Presentation)

But it's not just us.
Or even those in R&D.

At this point it's
pretty much everyone

Our own team, peer teams, support & operations, management, R&D leadership, marketing, sales, executives, board members, customer service, investors, auditors and CUSTOMERS

And providing understanding
at their perspective is critical

All of these people have
slightly different perspectives
and needs

No single “diagram” or even
story will work

Much of engineering
leadership is becoming about
explaining our systems to
“the rest of the world”

(no more God syndrome)

And since we've generally sucked at this, the government and general population are starting to force our hands.

They are finally realizing that
"software is eating the world"
and that they don't really
understand it.

Which really freaks them out



Justifiably

We have not done a good job
helping others understand our
“stuff”

(MOM: What do you do again?
ME: Stuff)

Right, Mark?



So they're pushing back

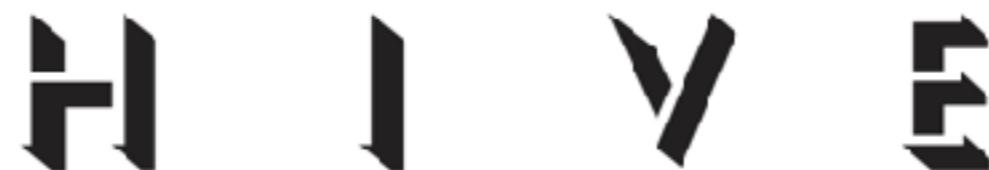
Mr. Mark Zuckerberg

FBI (encryption), Facebook
(data privacy), GDPR (data
privacy), Compliance

When we watch the congressional hearings and go “what morons” we should really be saying “we failed”

But now the cameras
are officially on us





BUSINESS

POLITICS

TECHNOLOGY

THE PLAYERS



Silicon Valley

SILICON VALLEY'S TECH GODS ARE HEADED FOR A RECKONING

How Facebook and Google became mercenaries—and now casualties—in the information war.



BY MAYA KOSOFF

OCTOBER 19, 2017 8:02 AM





But here's the real kicker



MANY OF US have
NO CLUE
what the hell
OUR SYSTEMS ARE DOING



So we need to start with
ourselves

We need to ensure that we can understand our systems and then work our way up

And provide the tools that
allow all to understand the
system from everybody's
perspective

This job is actually very difficult

(I believe it's more difficult to explain and fully understand than it is to actually build)

Why?

Our systems are more
complex than they've ever
been

(And only growing increasingly complex)



The “simpler” times

(It did not feel simpler at the time)

We had mainframes, Windows
apps, client server, etc

These were all very controlled
and constrained systems

(Or it at least if felt that way)

24/7 Uptime ... pfft

(We would take systems down for
nights and weekends. 5 9s. Ha!)



We hardly ever released

(Release cycles measured in years,
months if you were aggressive)

Realtime!?

...

What does that even mean?



You may not believe this
but we would run nightly
(or weekend) jobs to
create reports.

(On paper. PAPER!)

Devices?

We tell you what “device” you
will use.

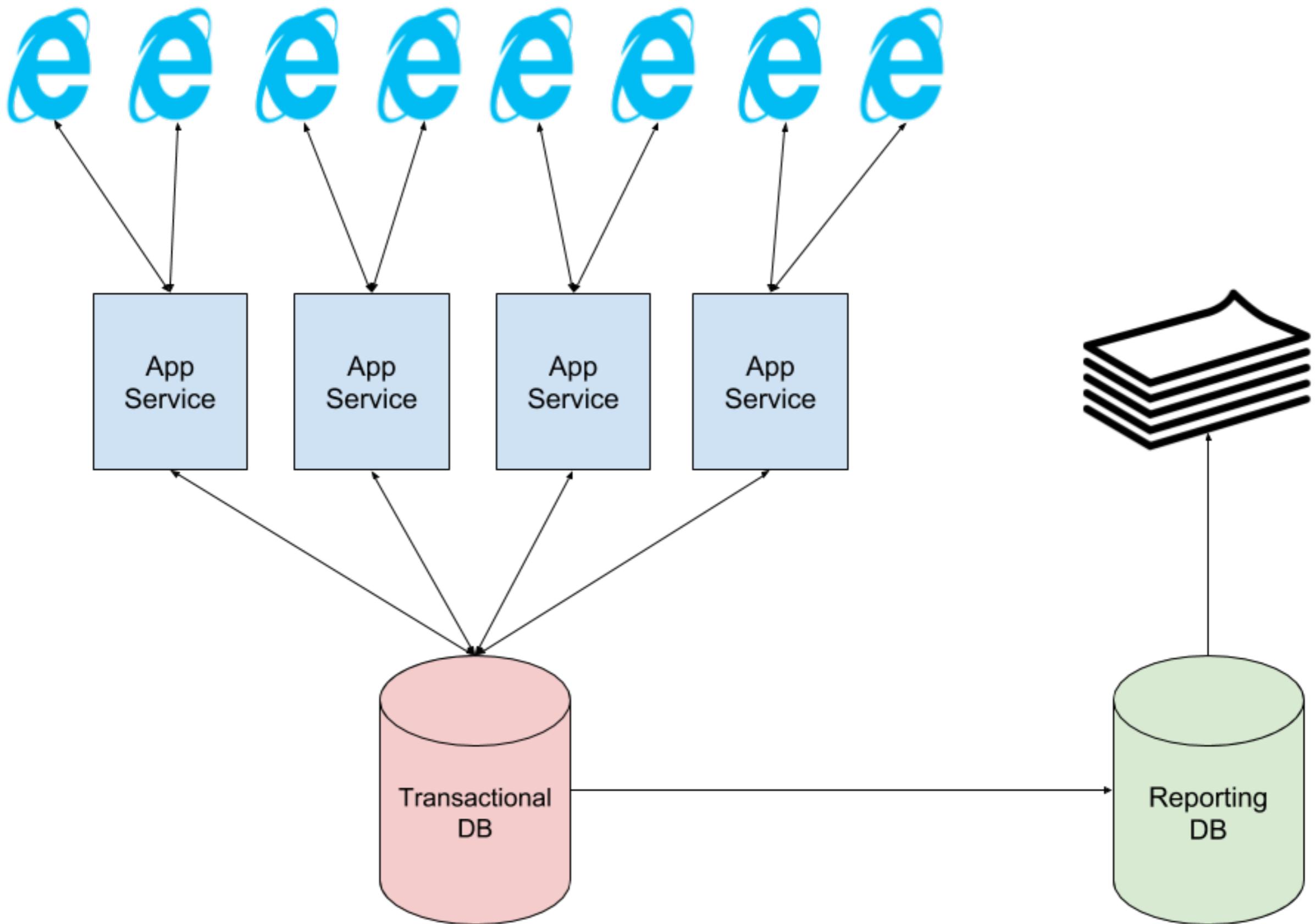
(Mainframe terminal, windows, IE, Blackberry)

Systems now?



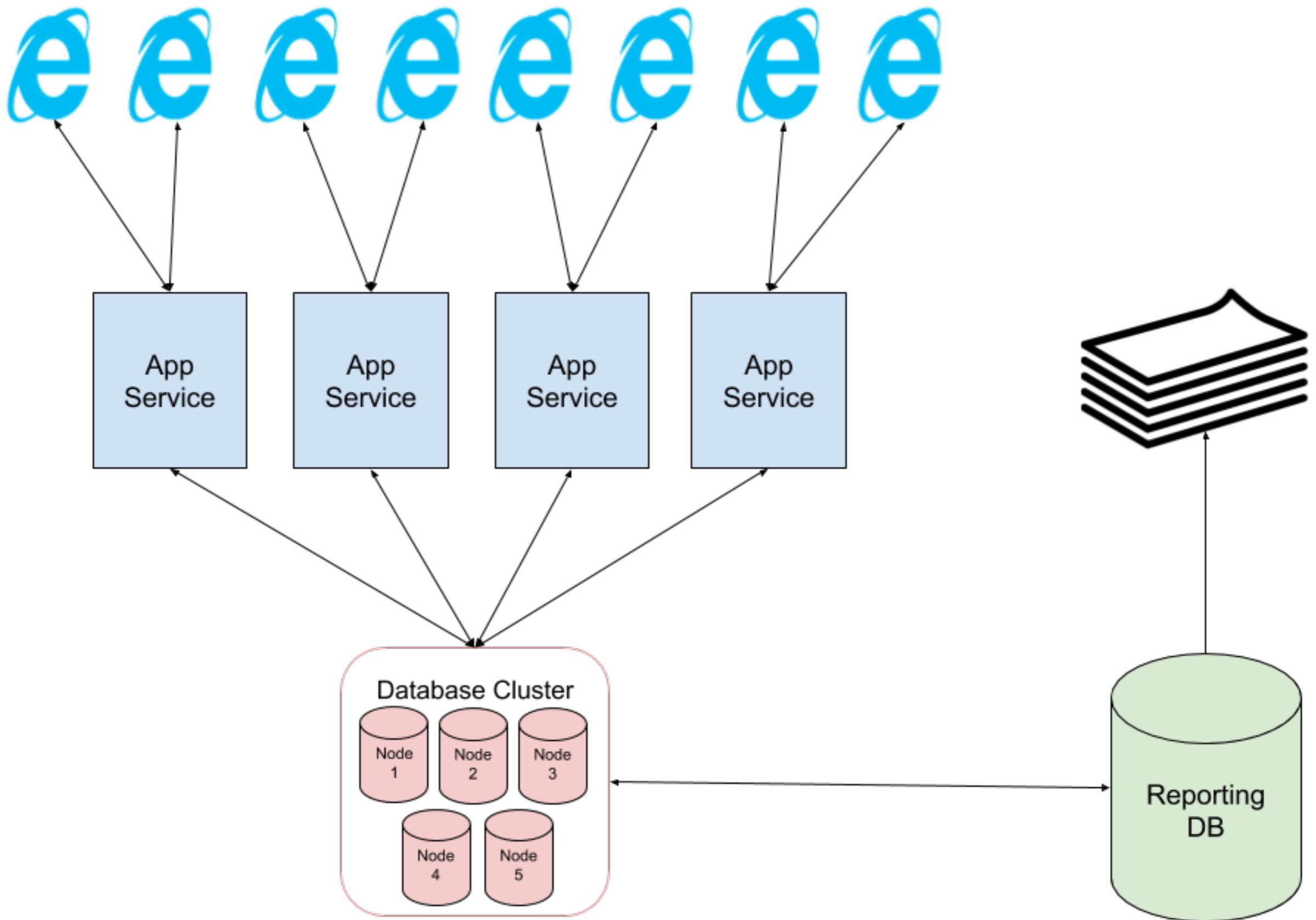
Let's start with a client
server architecture built
under the old constraints

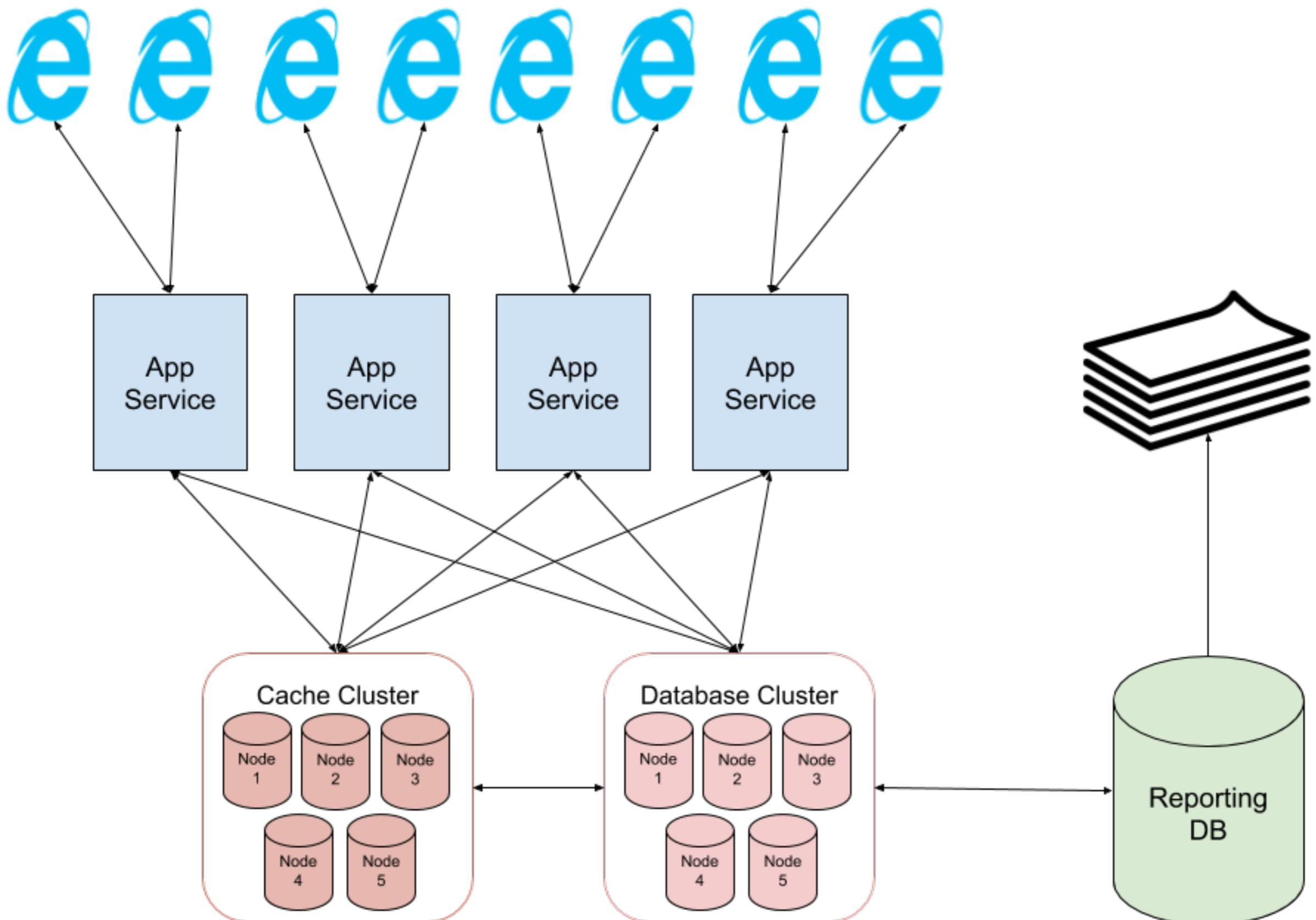
And then evolve it as our
constraints evolve



Downtime is unacceptable

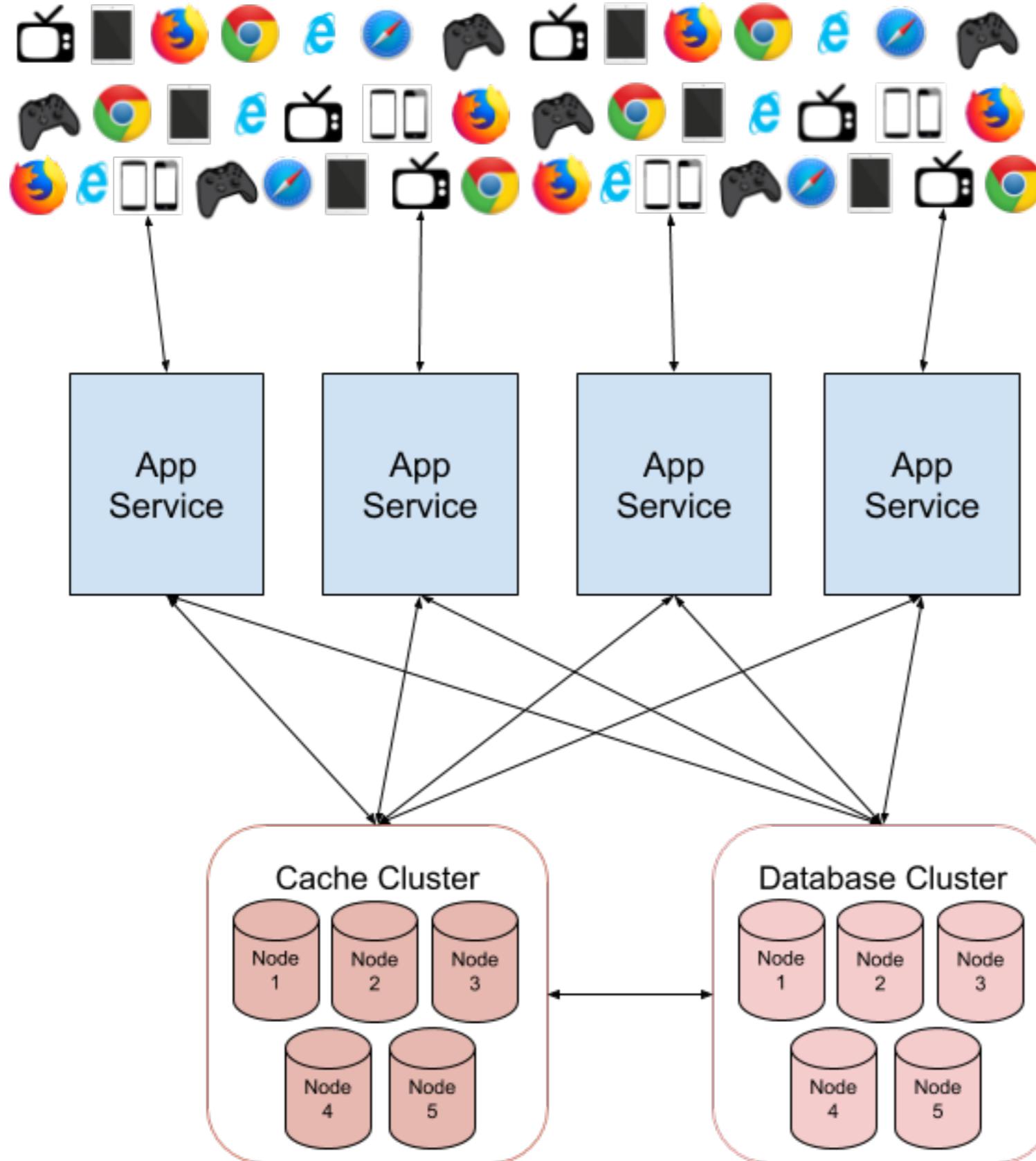
("x" 9s :/)





Devices you say?

Oh we've got devices.
All the damn devices.



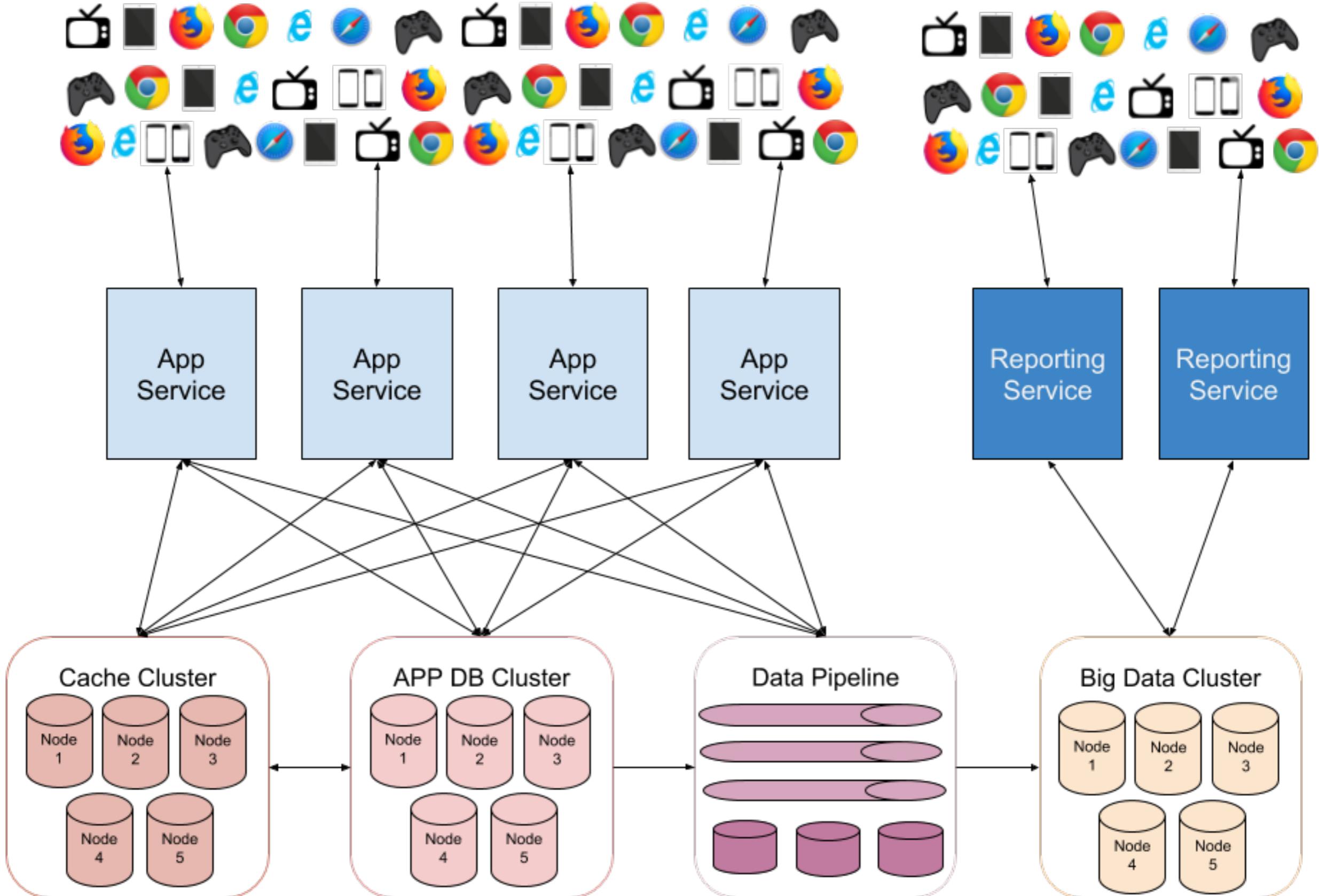
Realtime?

**“Uh yeah! I’m not waiting even
a second for what I want”**

No more “stale” reports

I want answers (data) now.

(Oh, and it better be visual and interactive)

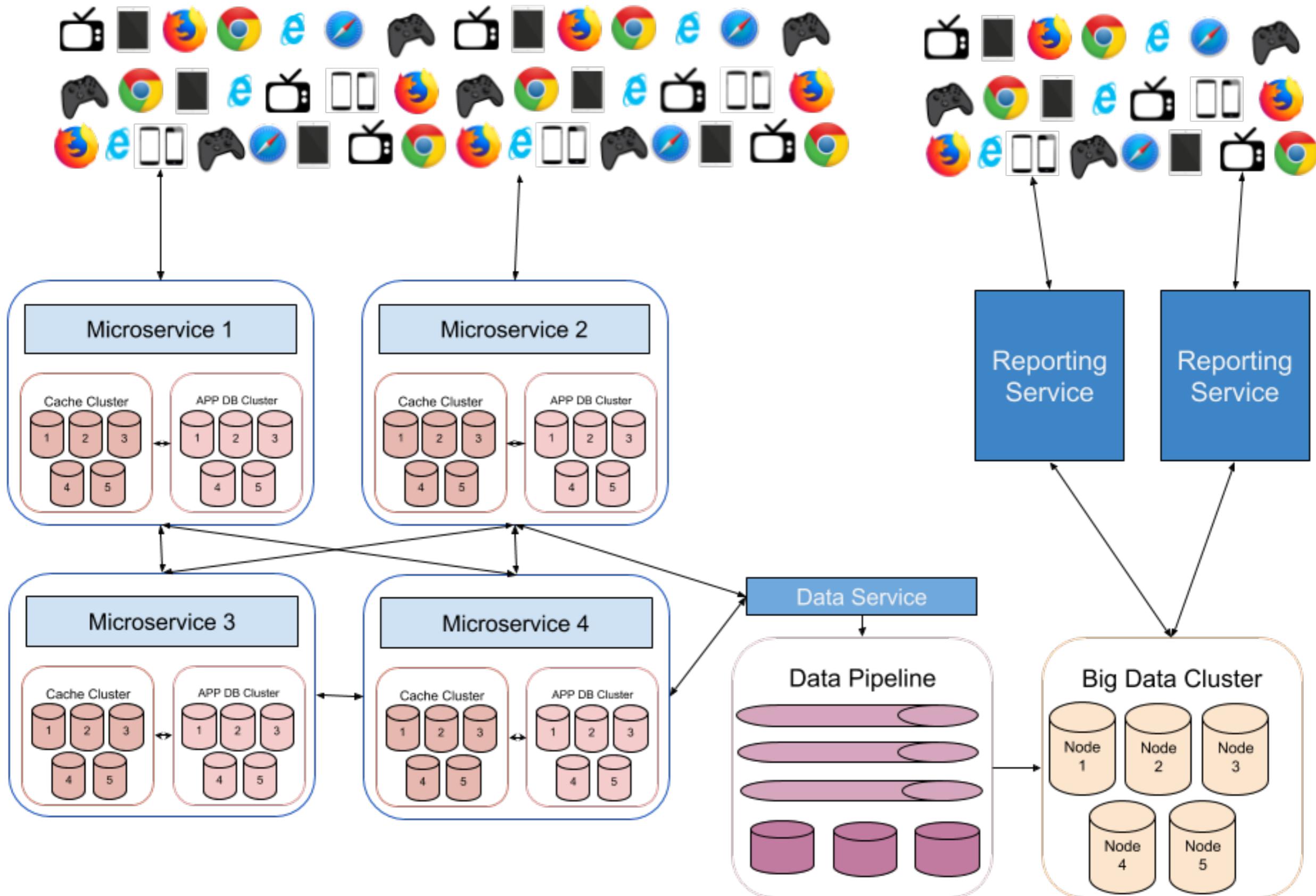


As fast as possible releases

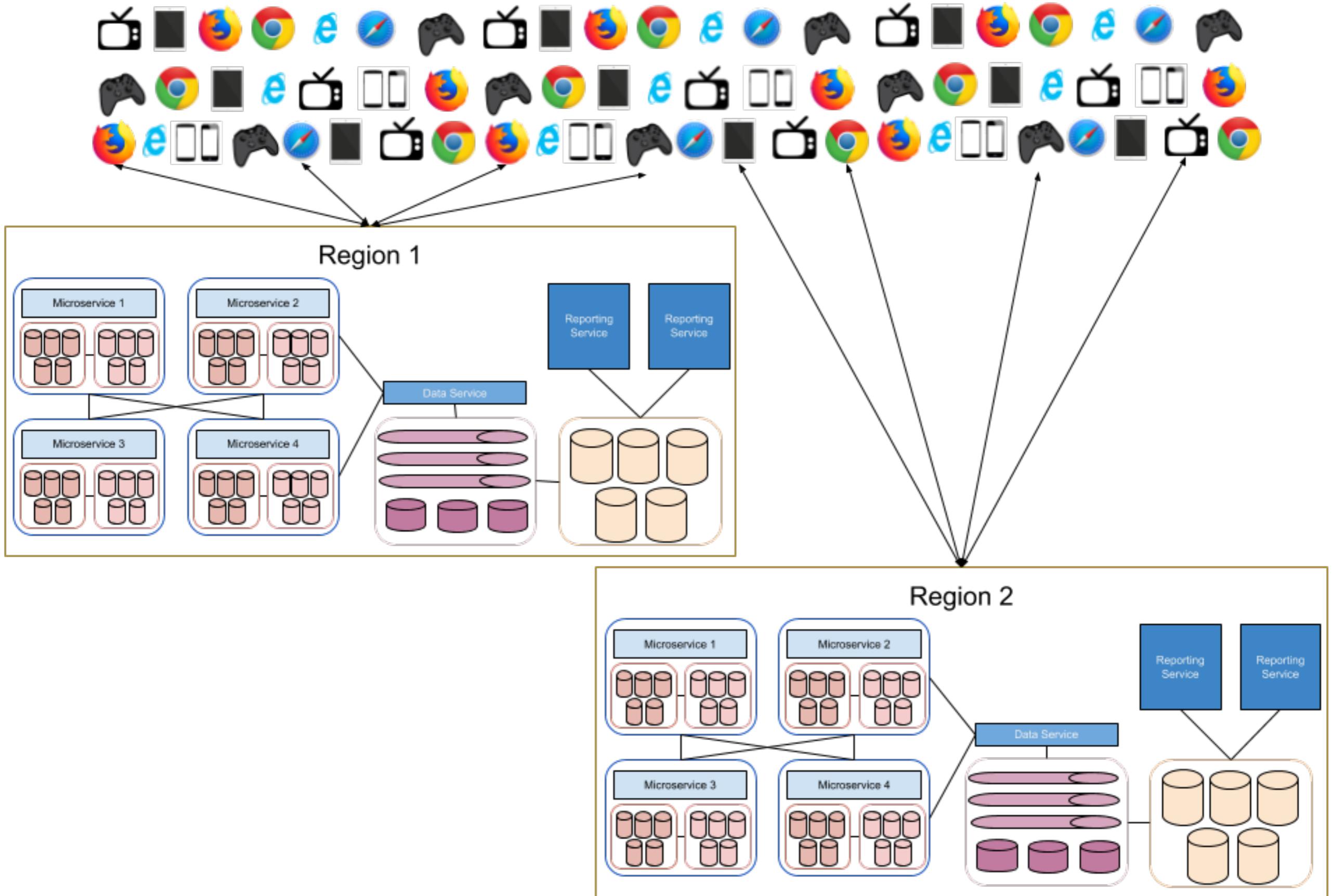
(From years to months to days to multiple
an hour and maybe even faster at scale)

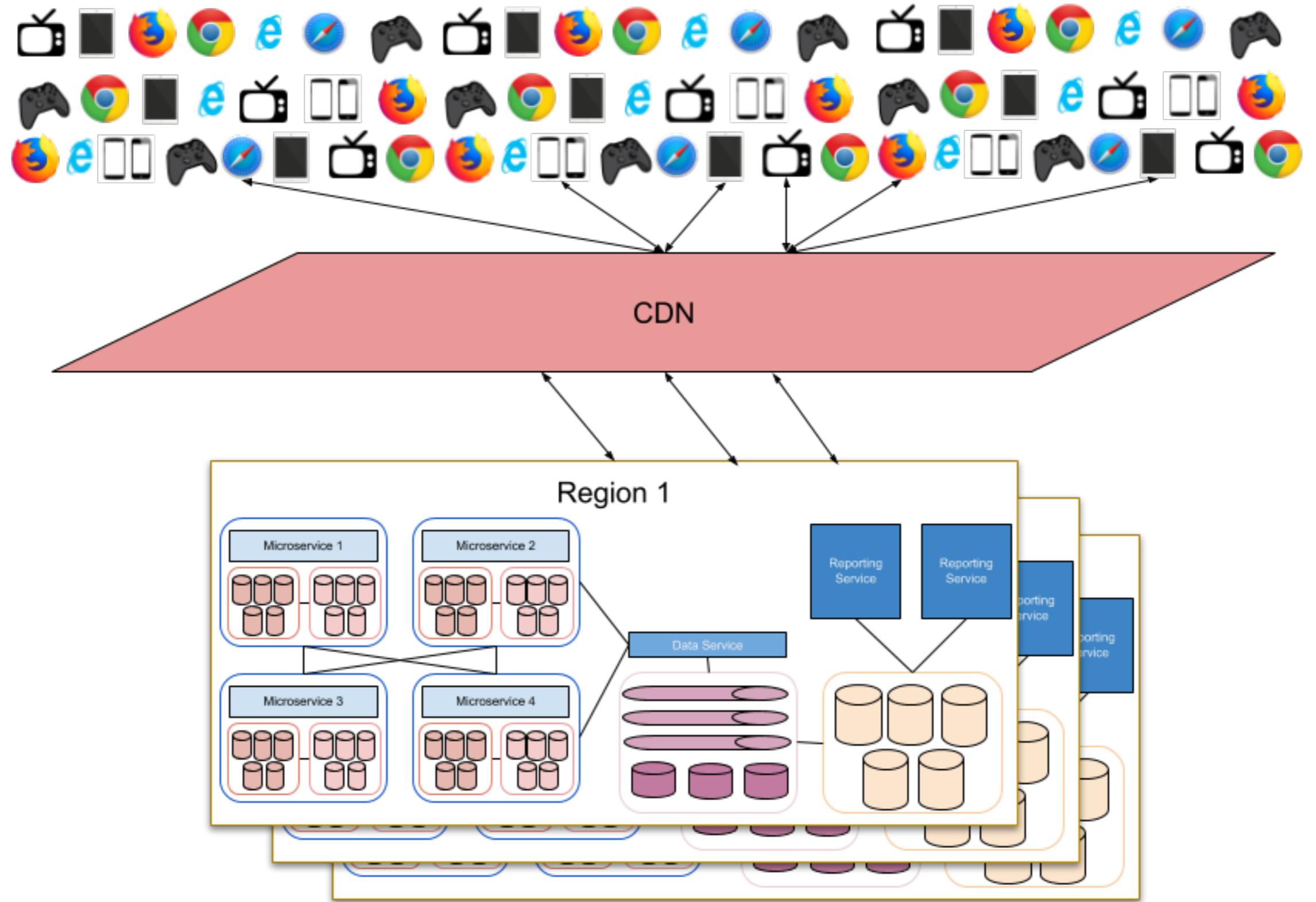
And anybody can release.
For any reason.

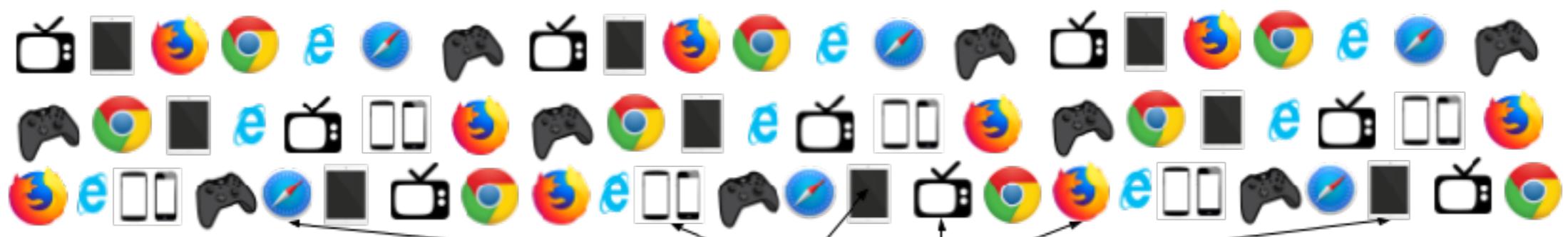
(You must release to keep up with demand and to
quickly fix issues)



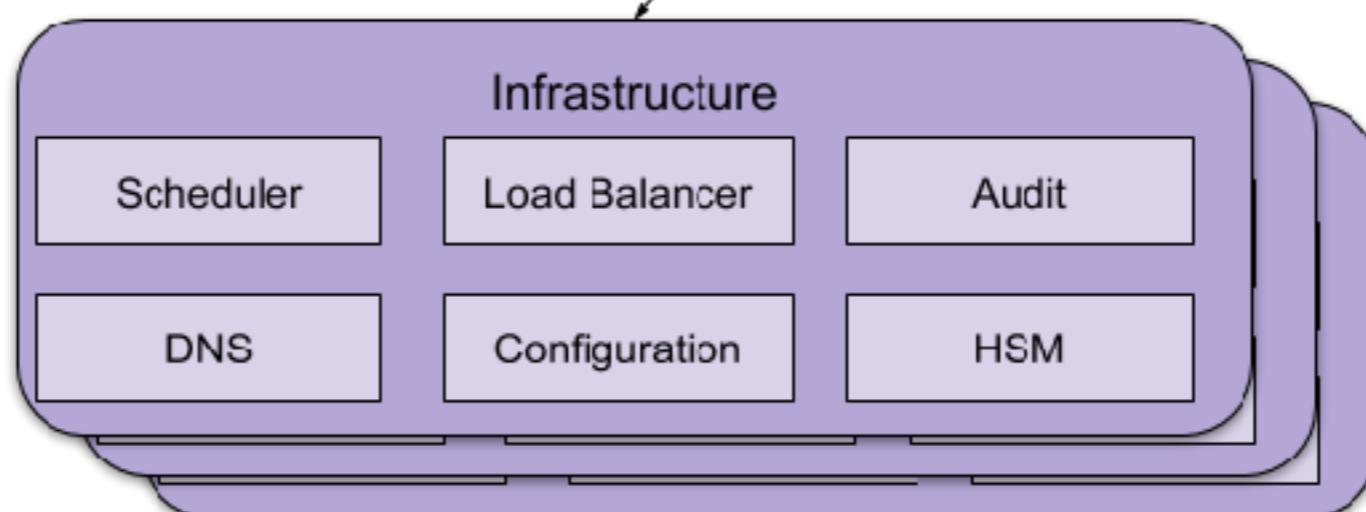
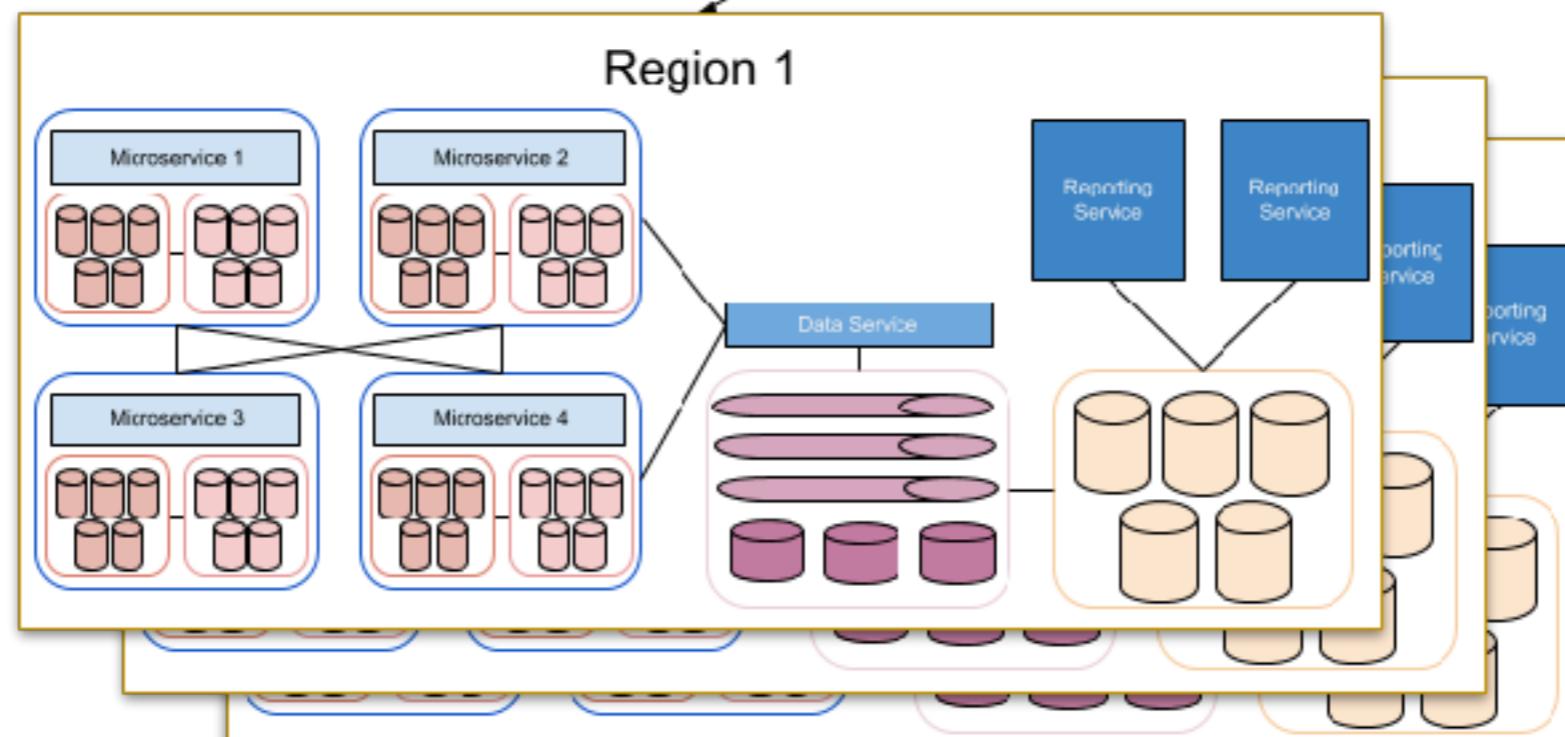
We expect access from
anywhere at anytime

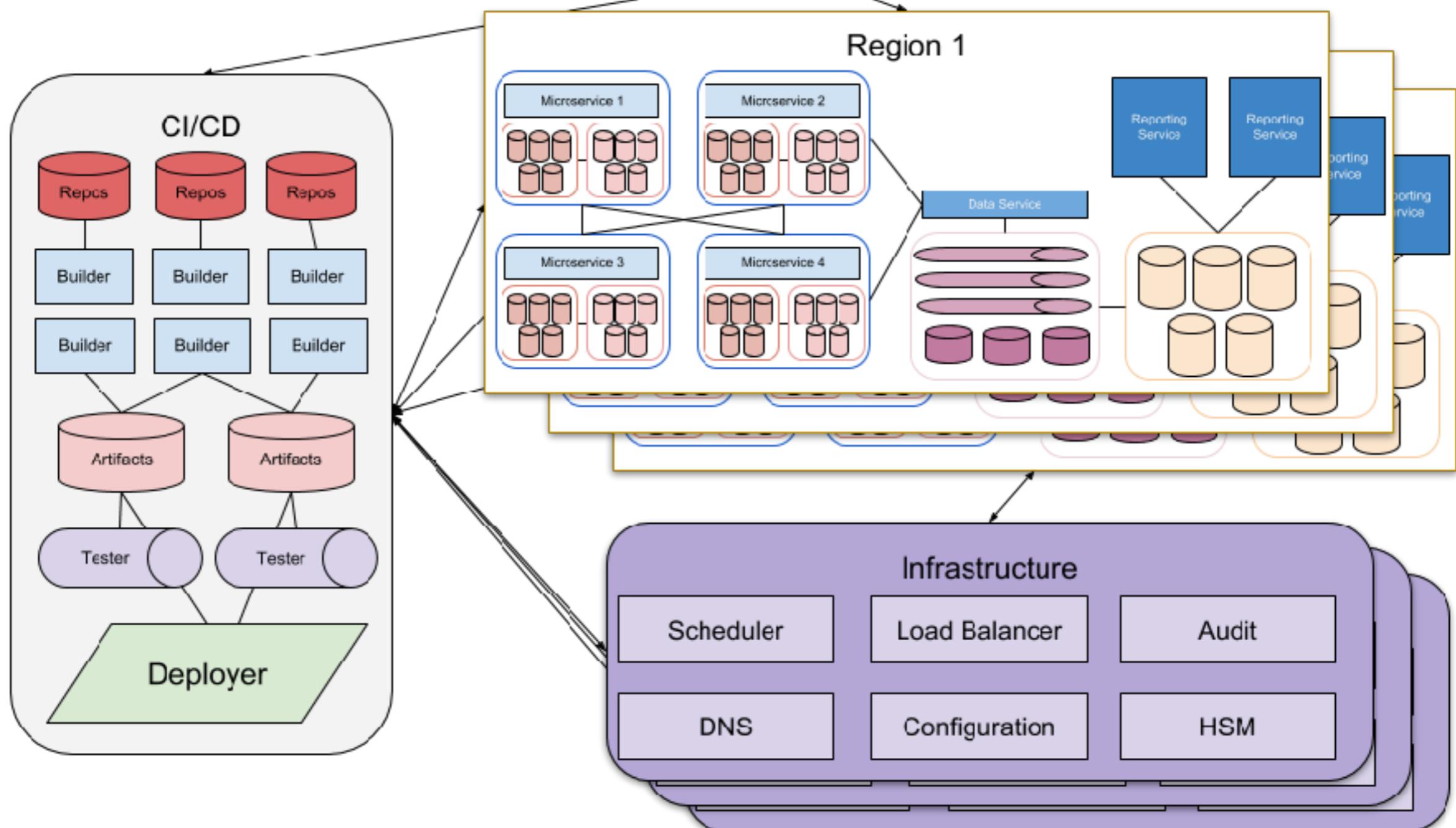
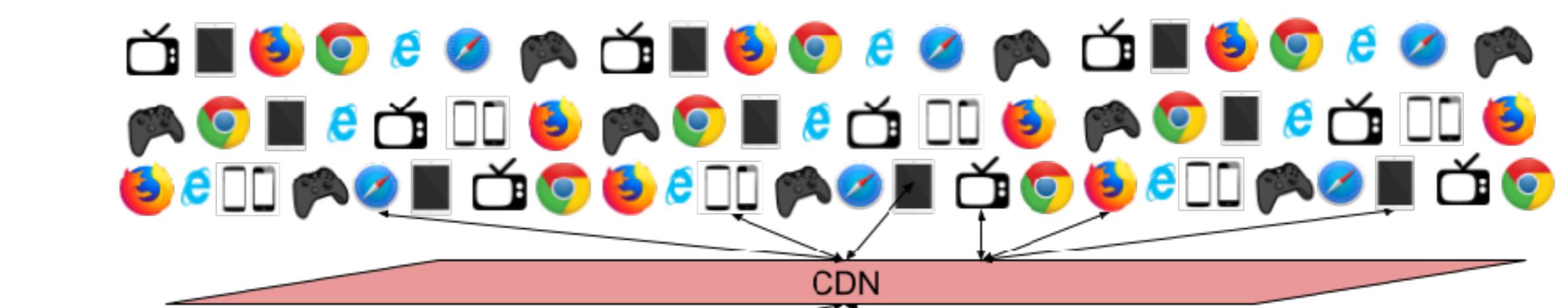






CDN





The Modern Technology Cluster #*@!

The Modern Technology ~~Cluster #*@!~~ Stack

The complexity has
risen significantly

But don't worry OSS is here to
save you

(SPOILER: Only, kinda)

Cloud Native Landscape

v0.9.0



Those are the tools that Tyler mentioned that you can use but you need to wrap with your "glue code" (your culture, your processes)

Oh ... and I'm not done

There are significantly more nodes in the system

And many connections
between these nodes
to handle scale

These connections create dependency trees

And even more the nodes
and connections are
constantly changing

All while we must
maintain usage rates

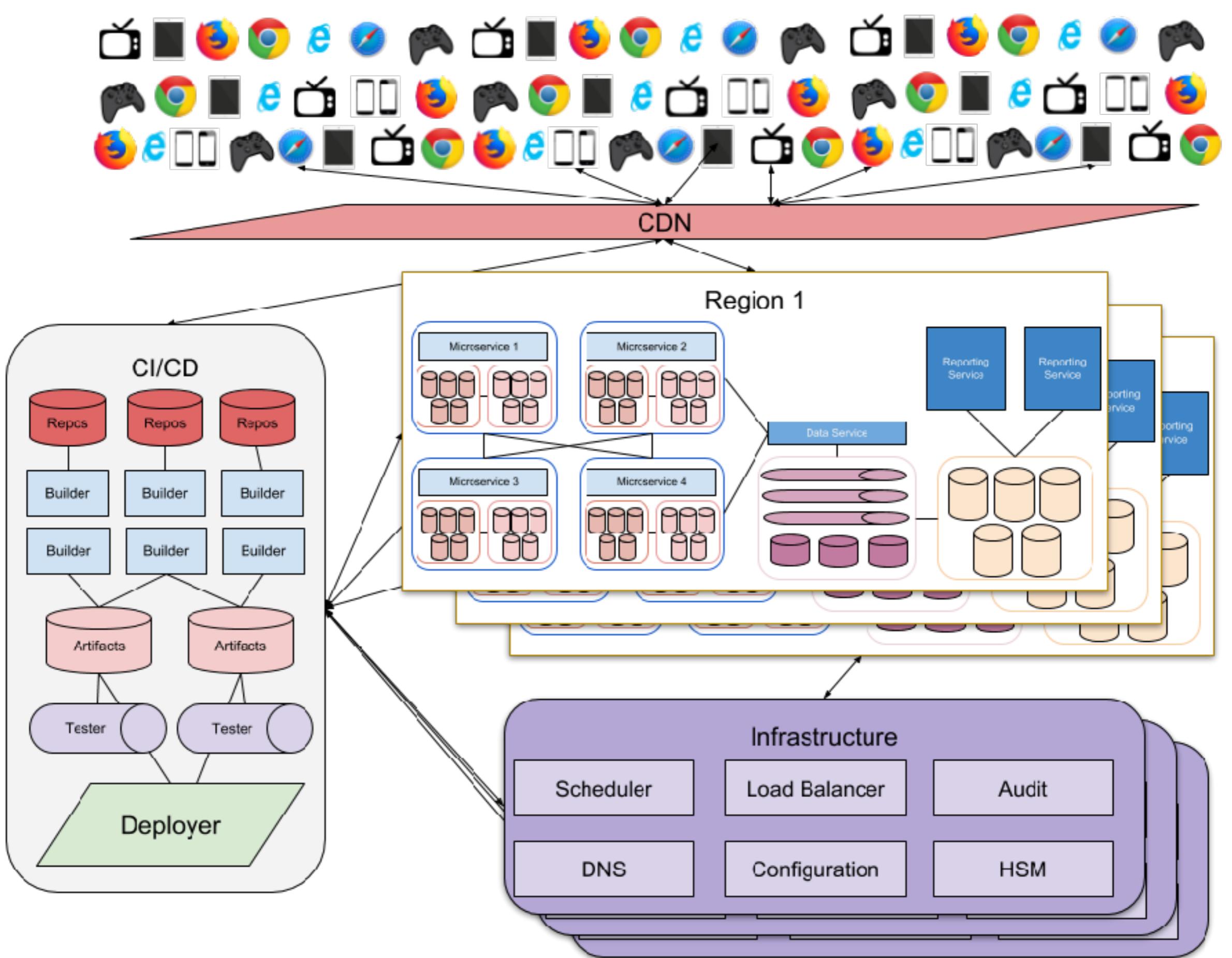
Thus we end up with
different versions of the
same type of node
potentially within a single request

All of this (and more) leads to our systems producing emergent behaviors that can't be predicted.

In other words our systems
are becoming much more
similar to “living” systems

(Cities, governments, ecological, biological, etc)

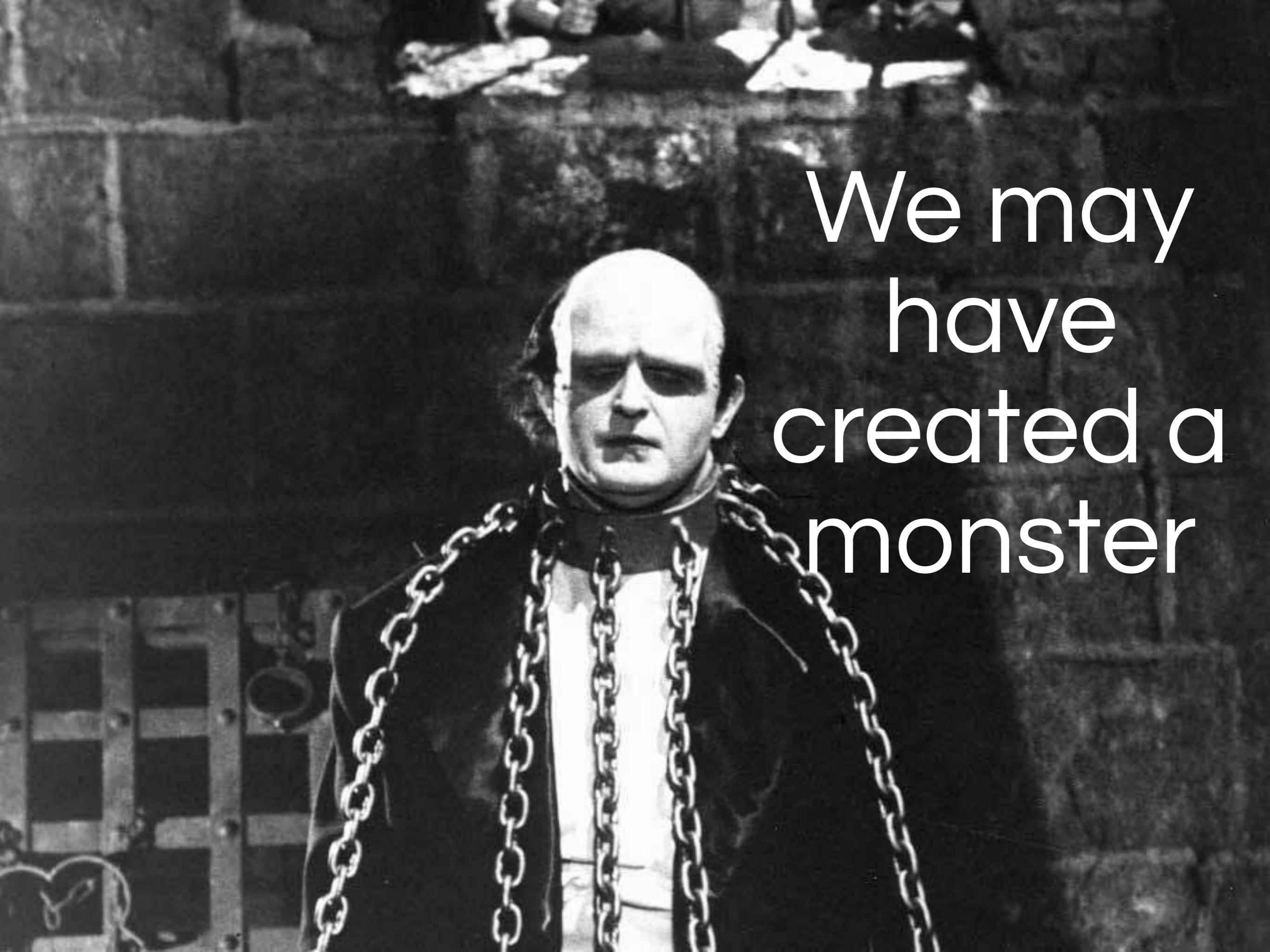
So this ...



is kind of ... like ... alive?

IT'S ALIVE!





We may
have
created a
monster



And it
might kill
us!

F*\$@!

Let's go back to the old way

Except it's too late.

This actually works.



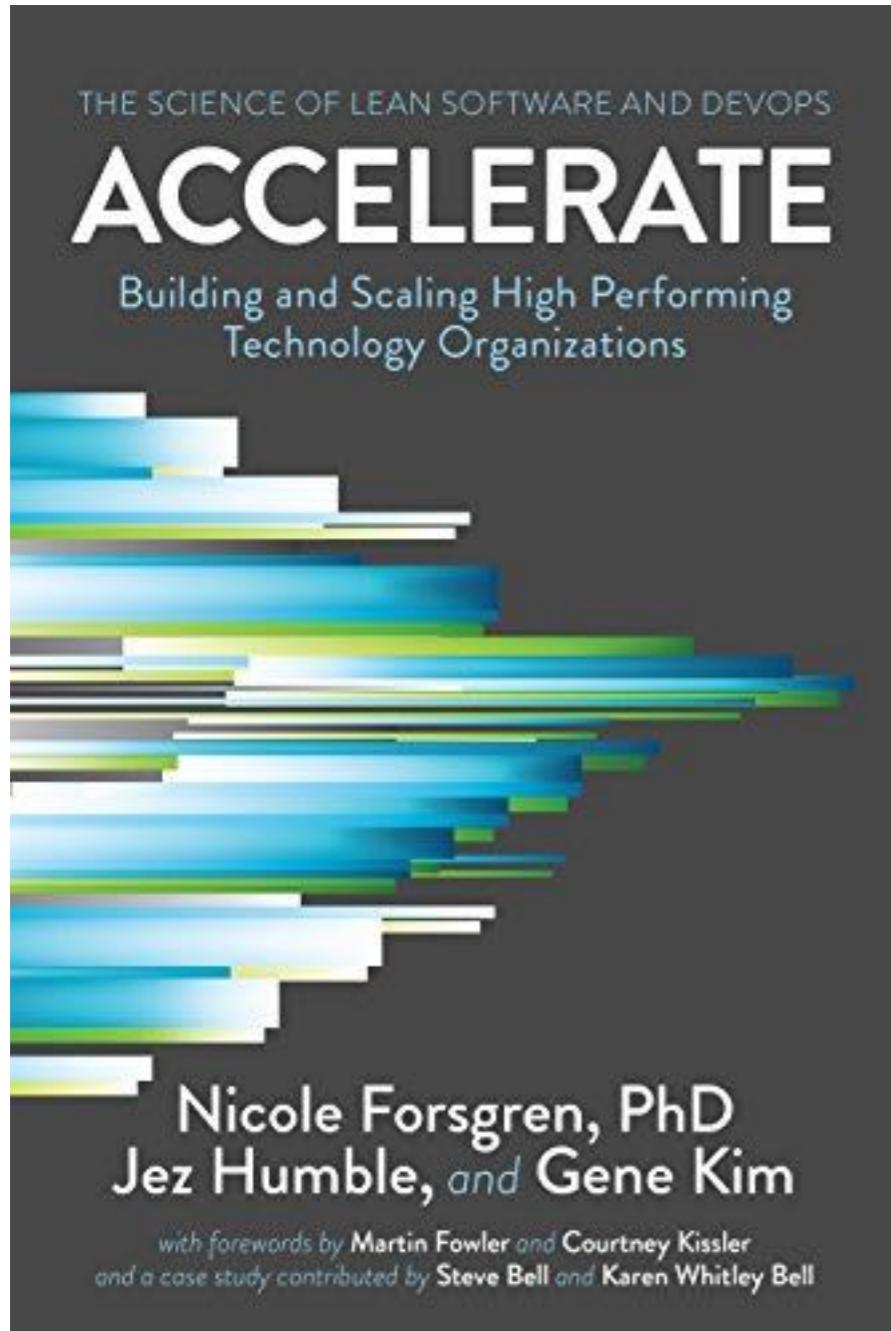
Beyond the obvious successful companies (Google, Amazon, Facebook), the research backs up that these systems help all types of companies that embrace them across all industries.

Dynamic systems that support
rapid development and
experimentation directly
increase quality and velocity

Thus IT becomes a differentiator and is no longer a cost center

DevOps is a critical piece of
this transformation

If you don't have a dynamic system
that supports experimentation and
rapid release and embrace DevOps
you will be beat by those that do



Accelerate:

The Science of Lean Software and DevOps:

Building and Scaling High Performing Technology Organizations

a16z Podcast: Feedback Loops — Company Culture, Change, and DevOps
<https://a16z.com/2018/03/28/devops-org-change-software-performance/>

So if this isn't your world,
it likely will be in the future

So what can we do to attempt
to understand the chaos?

An example from our
past experience at Workiva

“Calc”

A system and method that efficiently, robustly, and flexibly permits large scale distributed asynchronous calculations in a networked environment, where the number of users entering data is large, the number of variables and equations are large and can comprise long and/or wide dependency chains, and data integrity is important

Or ... a distributed
calculation engine

**Built on stateless runtimes
with no SSH or live debugging**

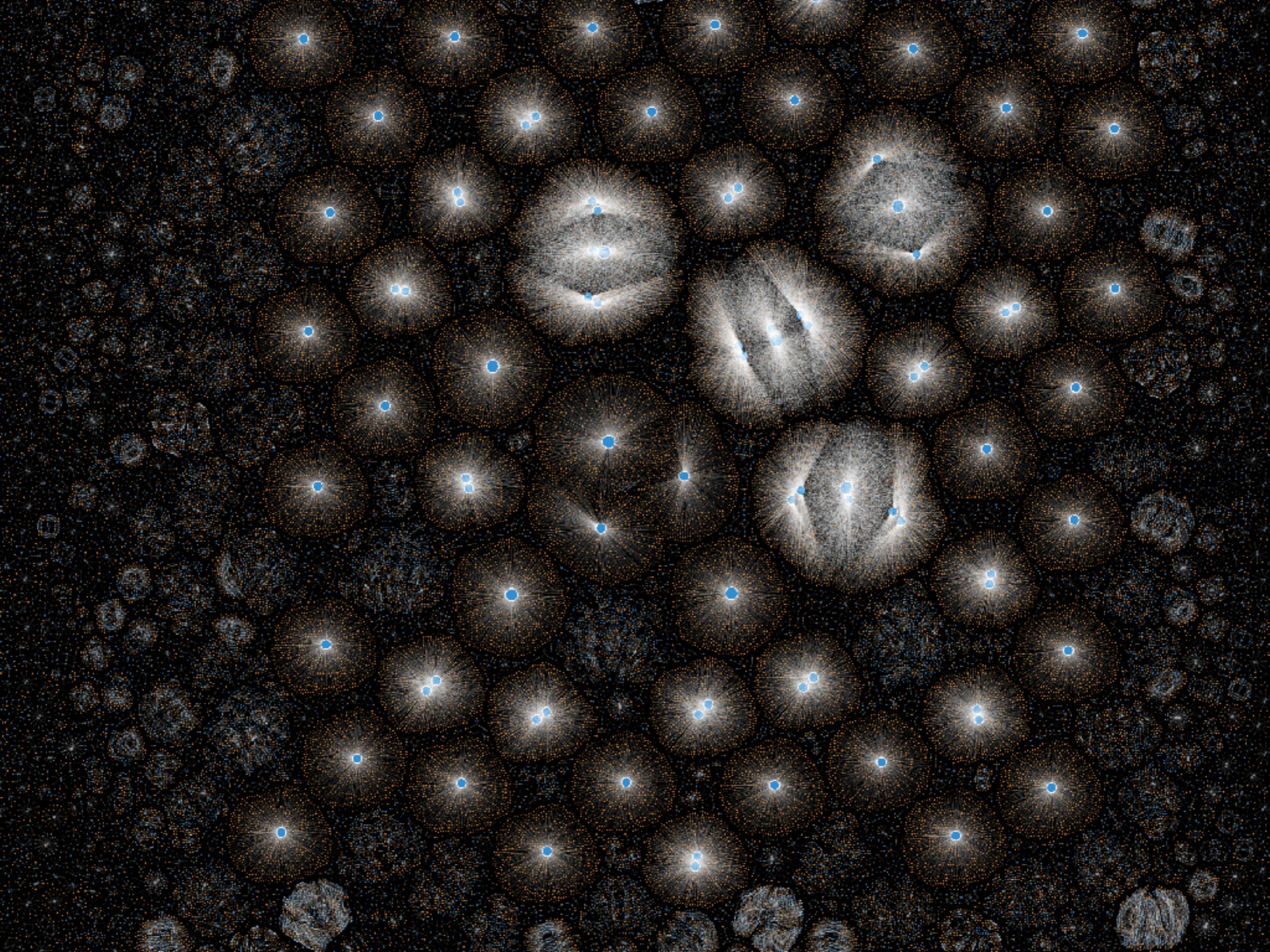
(Serverless in 2011, yep it was a thing)

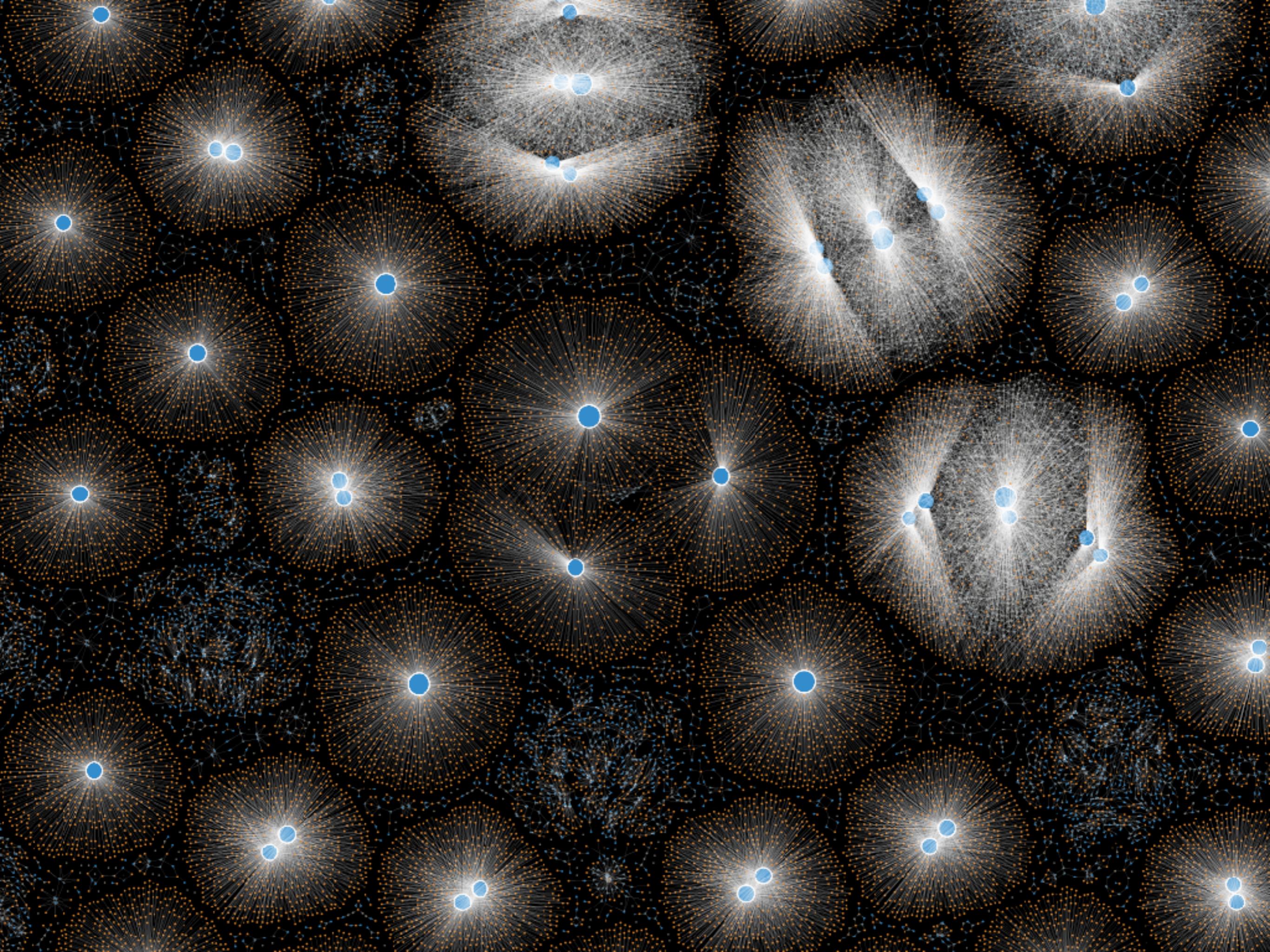
Not that SSH or Debuggers
would have mattered

Massive Scale

(Millions of nodes)

A tease ...





Structure, and thus behavior,
changed when the data
changed

(Very dynamic)

What is the state of the system?

Is it done?

What is done?

Is it broken?

What is broken?

What is fast/slow?

A single actor in the system
does not know the status of
the overall system.

There is no obvious way to track the status of the system unless the nodes within the system help us

To have any chance of keeping up with the understanding of systems we need the systems to self describe

And of course we need
automation and self healing

And to have self description, automation, and self healing we need data. We need the systems to give us data to provide necessary context.

So what are the specifics?

We'll start by working our way
up from the code

1. Pass a context object to
basically everything

```
type Context =  
{ user_id :: String  
, account_id :: String  
, trace_id :: String  
, request_id :: String  
, parent_id :: Maybe String  
}
```

What goes on the context?

Think about the data you wish
you had when debugging an
issue

(This is why your devs should
support their own systems)

What is the data that would change the behavior of the system?

The user (and/or company), time, machine stats (CPU, Memory, etc), software version, configuration data, the calling request, any dependent requests

What of that can we get for
“free” and what do we need to
pass along

(Free == Machine Provided
Memory, CPU, etc)

The data we can't get for
“free” should go on the context

(Data that is “request” specific
User, Company, Calling Request Id)

There are side-benefits as well

If you're a SaaS company you
should probably pass
licensing data as part of the
context

This will allow you to move
processes around based on
their license

Imagine routing traffic to specific queues based off user, account, license and environment (usage, resources available)

(The ability to isolate processes at runtime
Amazon is the king of this)

Also, think about GDPR and needing to track user actions, data and what they have approved the system to do

Please, use some data structure to pass contextual data to all dependent functions

This is the easiest thing you
can start doing today

Oh, and then make sure to log
that context on every request

And speaking of logging

2. Structure your logs

JSON is fine

I'm tired of writing regex's to
scrape logs because we're too
lazy to add structure at the time
it actually makes the most sense

```
{  
  "env": "Dev",  
  "server_name": "AWS1",  
  "app_name": "MyService",  
  "app_loc": "/home/app",  
  "user_id": "u1",  
  "account_id": "a1",  
  "logger": "mylogger",  
  "platform": "py",  
  "trace_id": "t1",  
  "parent_id": "p1",  
  "messages": [  
    {"tag": "Incoming metrics data",  
     "data": "{\"clientid\":54732}",  
     "thread": "10",  
     "time": 1485555302470,  
     "level": "DEBUG",  
     "id": "0c28701b-e4de-11e6-8936-8975598968a4"}]  
}
```

You can take this as far as
you'd like



Very structured with a type system, code reviews, etc

There are many existing libraries

(Too many to list. Just Google “Structured logs” and
your language of choice)

But at minimum get your logs
into a standard format with
property tags

3. Create a data pipeline

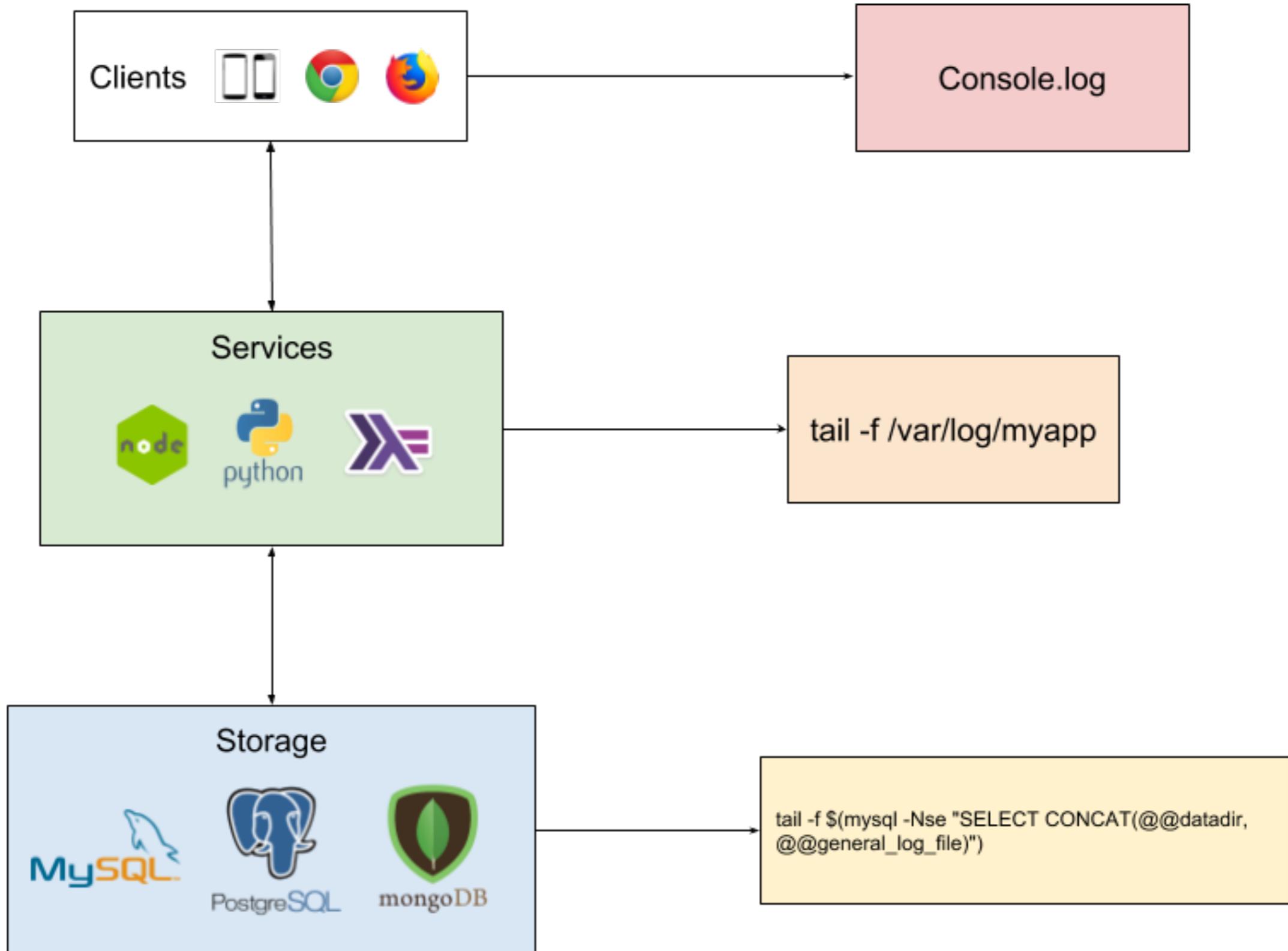
There is a ton of data that you want and need to collect

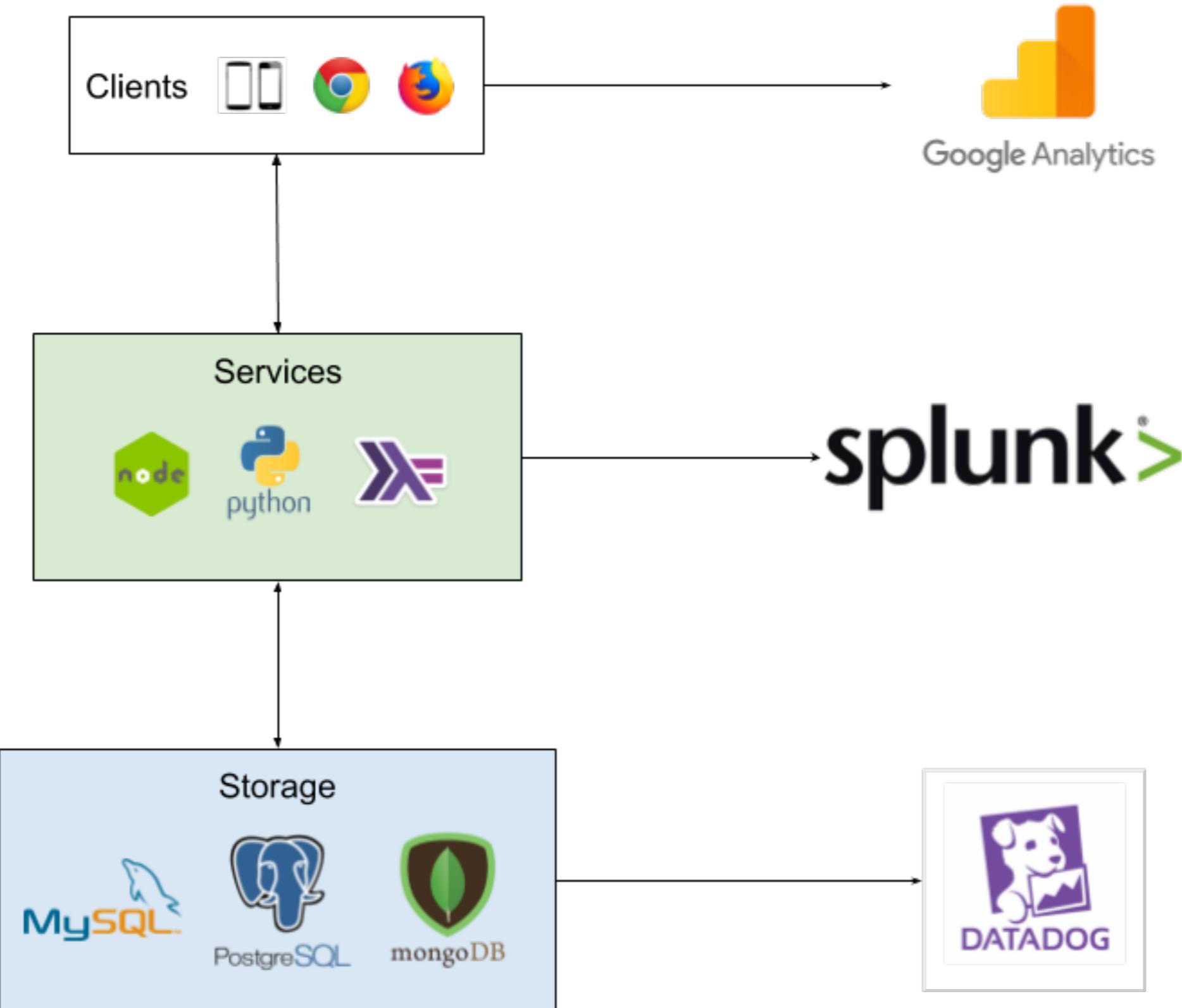
Logs, metrics, analytics,
audits, etc

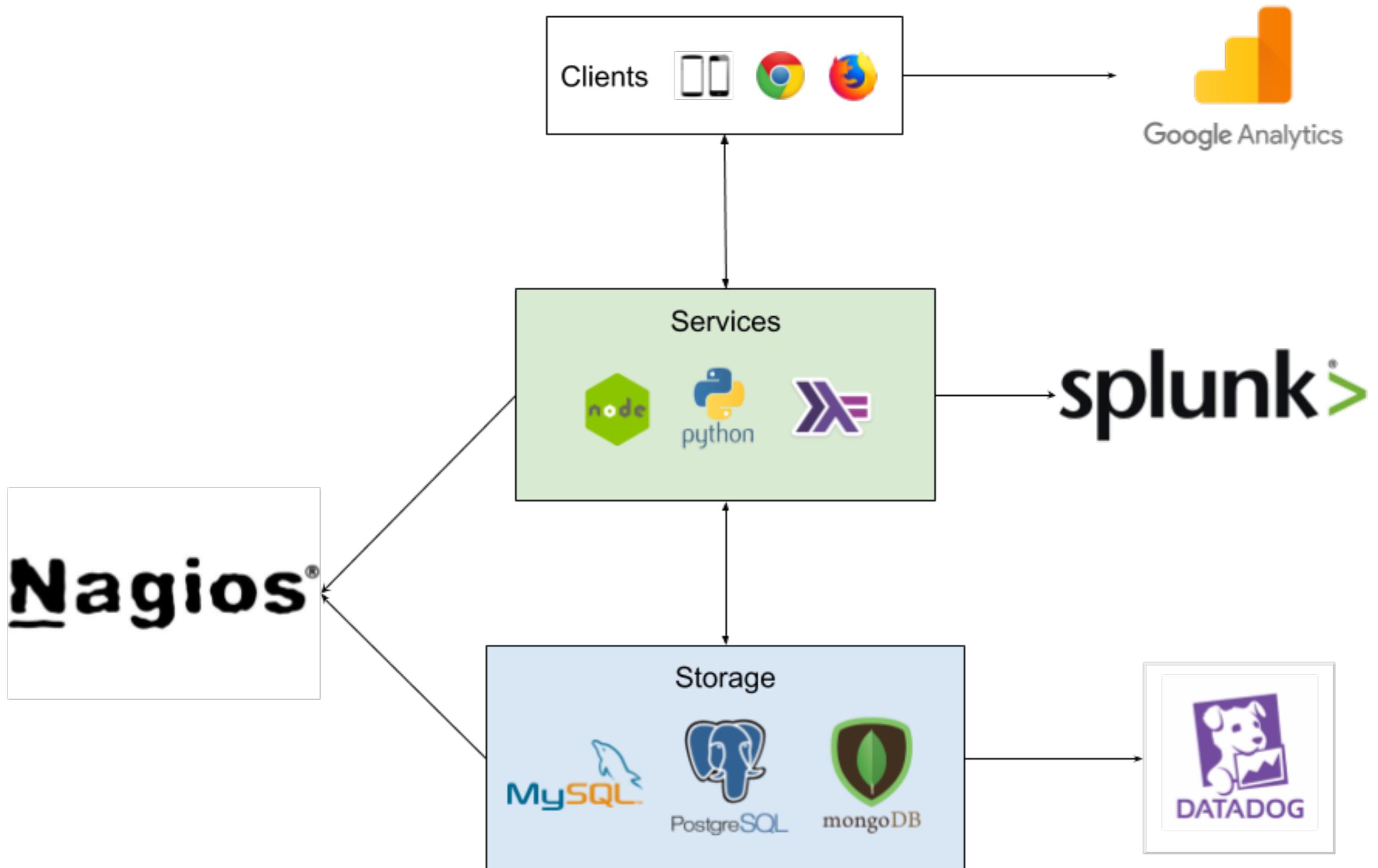
We want to make it as simple,
yet robust as possible

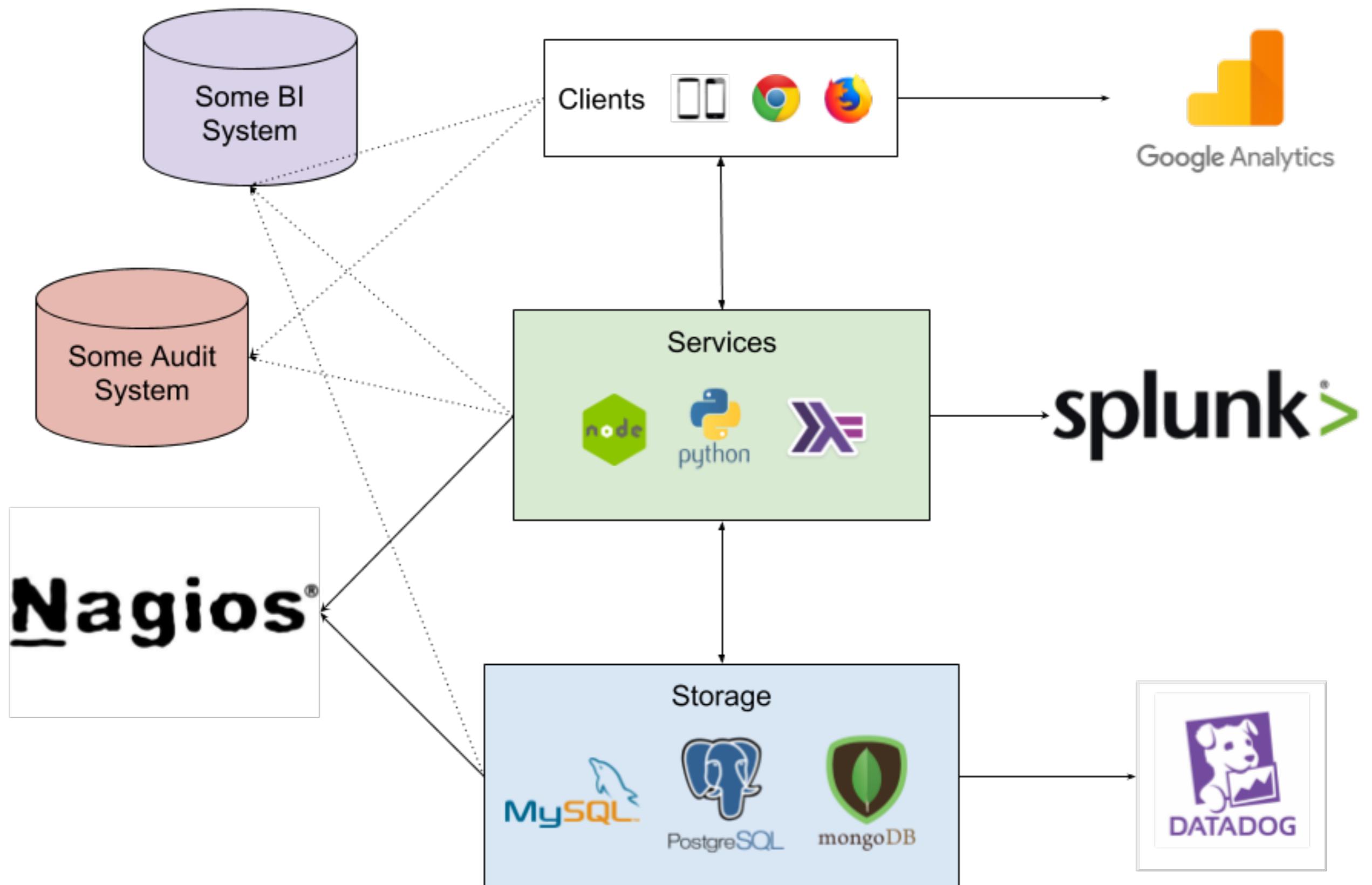
But most importantly we want
some system that has all of
the data

What we often see
at the beginning:







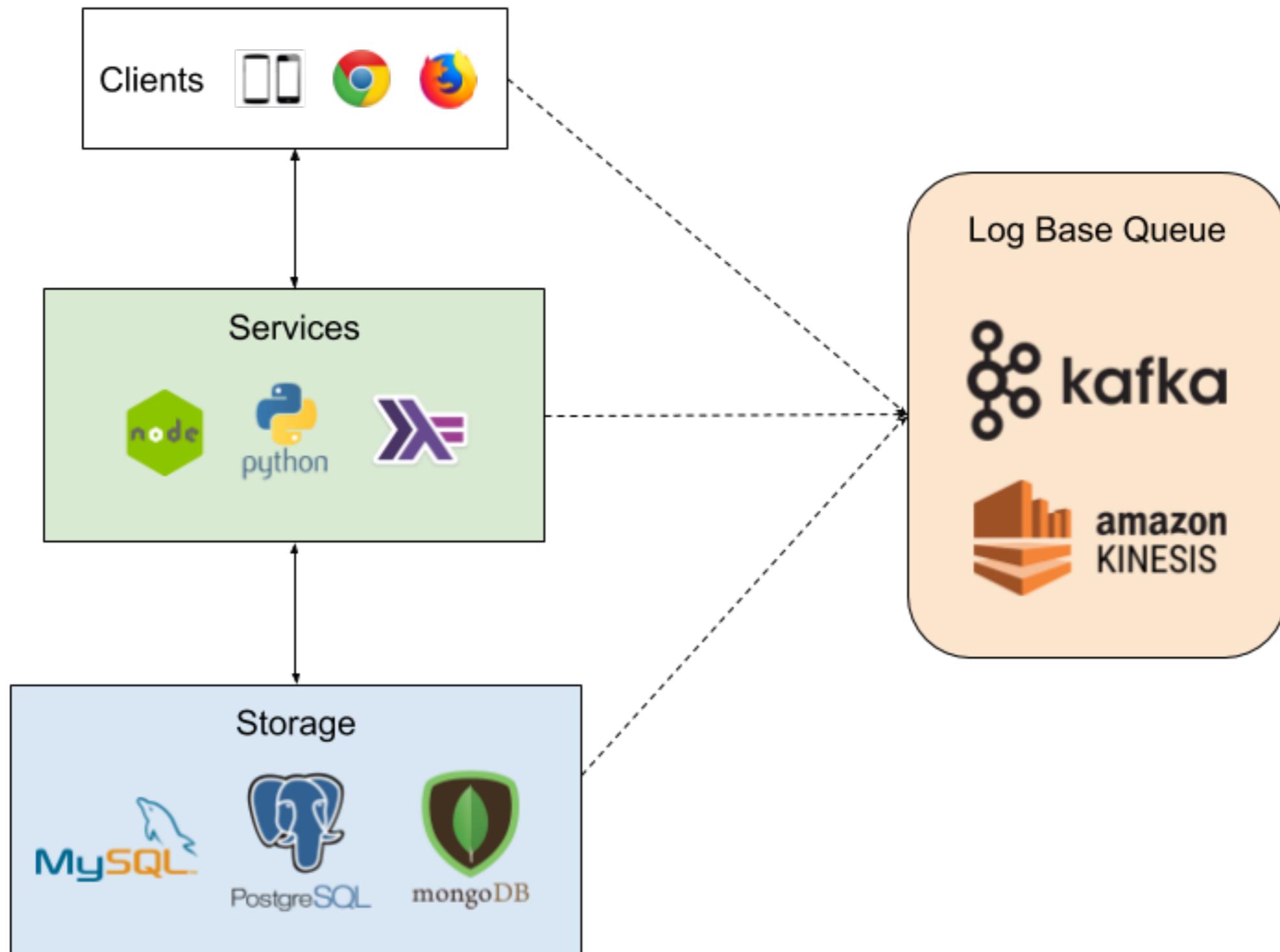


And now your services are
spending more time with non-
critical path dependencies
than those on critical path

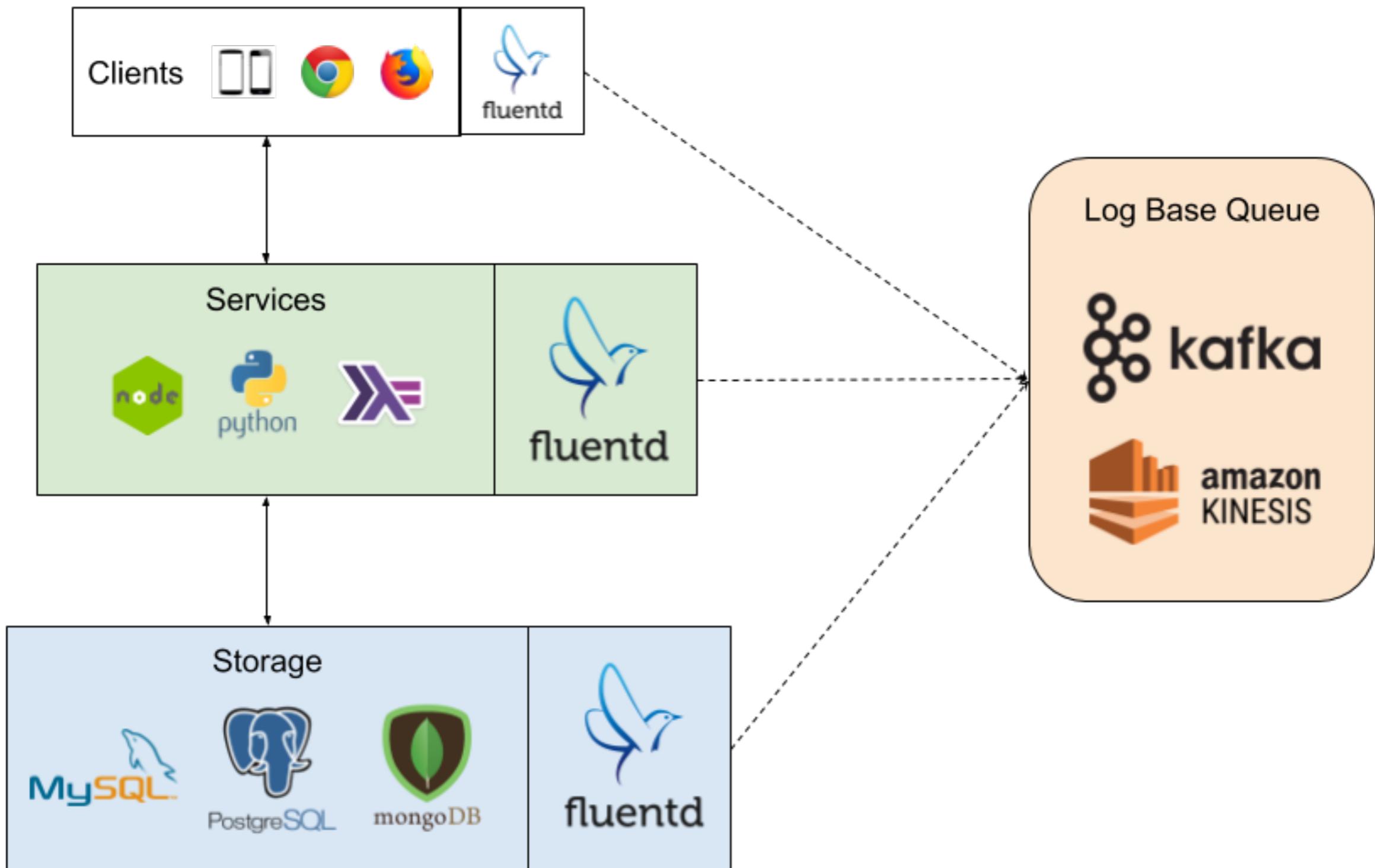
Standardize & simplify

A single data pipeline (queue)

(Or use a pull process. Just get your logs into a central location)

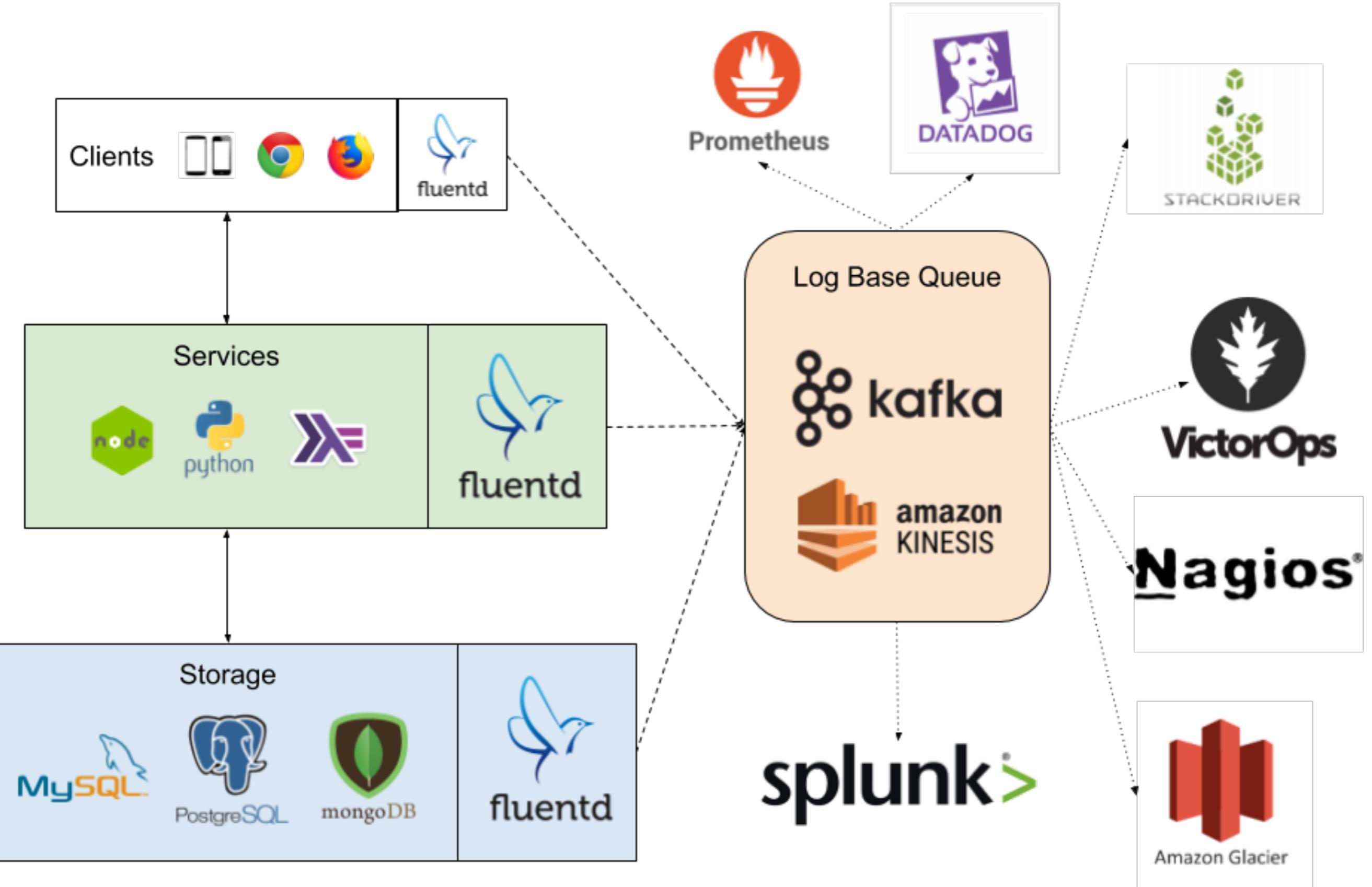


Look into “sidecar” style
collection



This allows you to write to
stdout and the sidecar will
collect and push to your
queue

The data pipeline provides a layer of abstraction that allows you to get the data everywhere it needs to be without impacting developers and the “core” system



Where should all of the data go?

At minimum all data should go
into a cheap, long term
storage solution

(AWS Glacier, etc)

You'll want this data for historical system behavior to help "machine learn" your system into automation

Ideally, all data should go into a queryable, large scale data storage solution. (solid time based query capabilities a plus)

(Google BigQuery, AWS Redshift)

This is why we
structure our logs

There are more targeted or
customized solutions starting
to fill the space



“Start solving high-cardinality
problems in minutes”

(honeycomb.io)

From their marketing ...

High-cardinality refers to columns with values that are very uncommon or unique. **High-cardinality** column values are typically identification numbers, email addresses, or user names. An example of a data table column with **high-cardinality** would be a USERS table with a column named USER_ID.

Query anything. Break down,
filter, and pivot on high-
cardinality fields like `user_id`.

Once again, this is why
we structure our logs

See the raw data behind every result.

See the exact events leading to an issue, who was affected, and how.

Share queries, results, and history. Collaborate.



Many other options ...

(Still a bit too dashboard based but trending in the right direction)



splunk®>

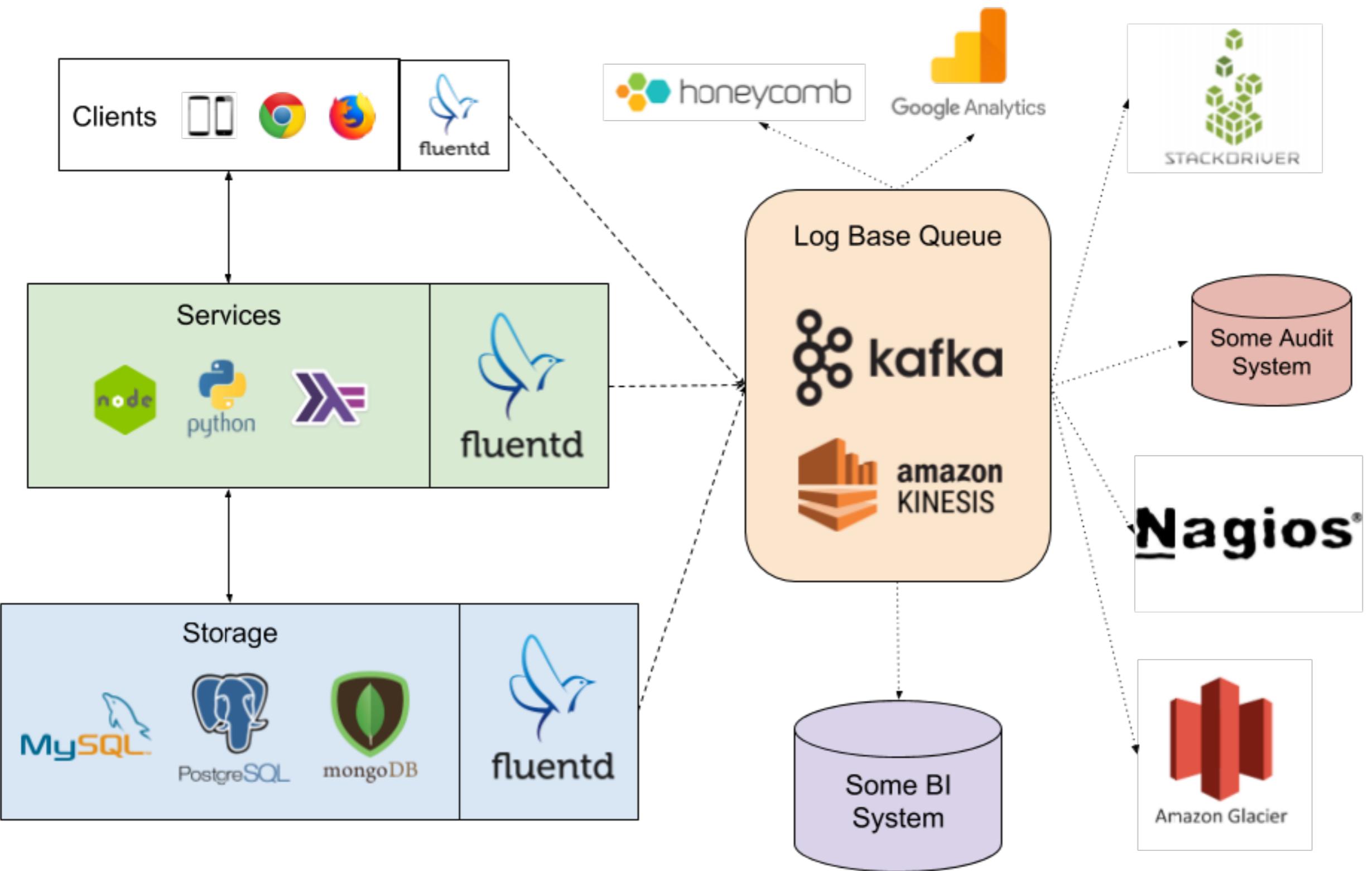
Prometheus



The beauty of the data pipeline is you can use 1 or many. And test multiple in parallel if you'd like without interrupting development.

(Just don't forget to have Devs user test the solutions as well)

You're still going to end up
with multiple consumers



Back to the structured logs thing

4. Structure and standardize all data leaving a system

Provide libraries to add
structure to not just logs but
also metrics, audits, etc

We like having one standard
across the board

But you can also break them
apart by “type” ...

Metrics, audits, tracing, etc

As long as you get it
standardized across systems

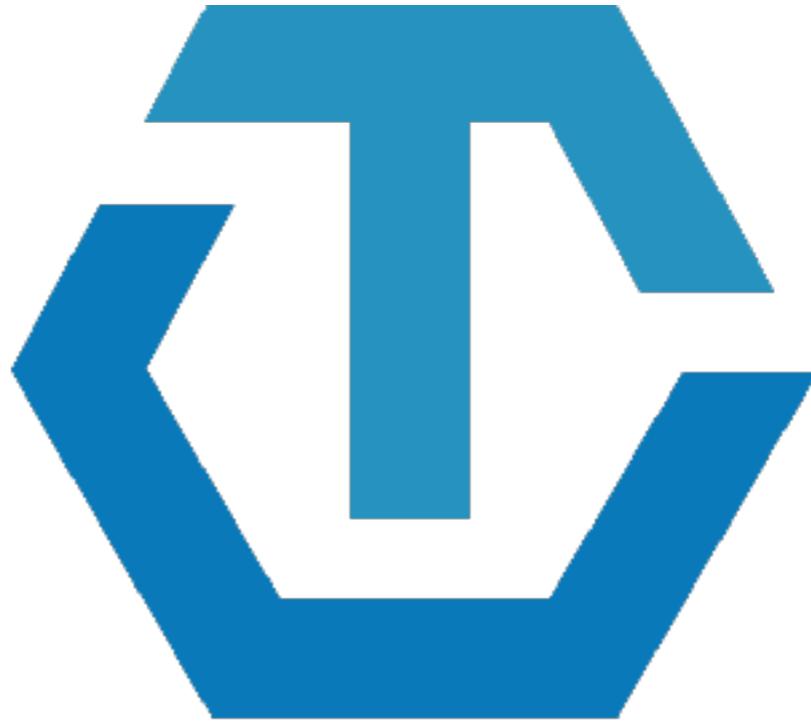
But people are quickly
realizing that this data is all
related and the separation is
arbitrary



OpenCensus

A single distribution of libraries for metrics and distributed tracing with minimal overhead that allows you to export data to multiple backends.

<https://opencensus.io>

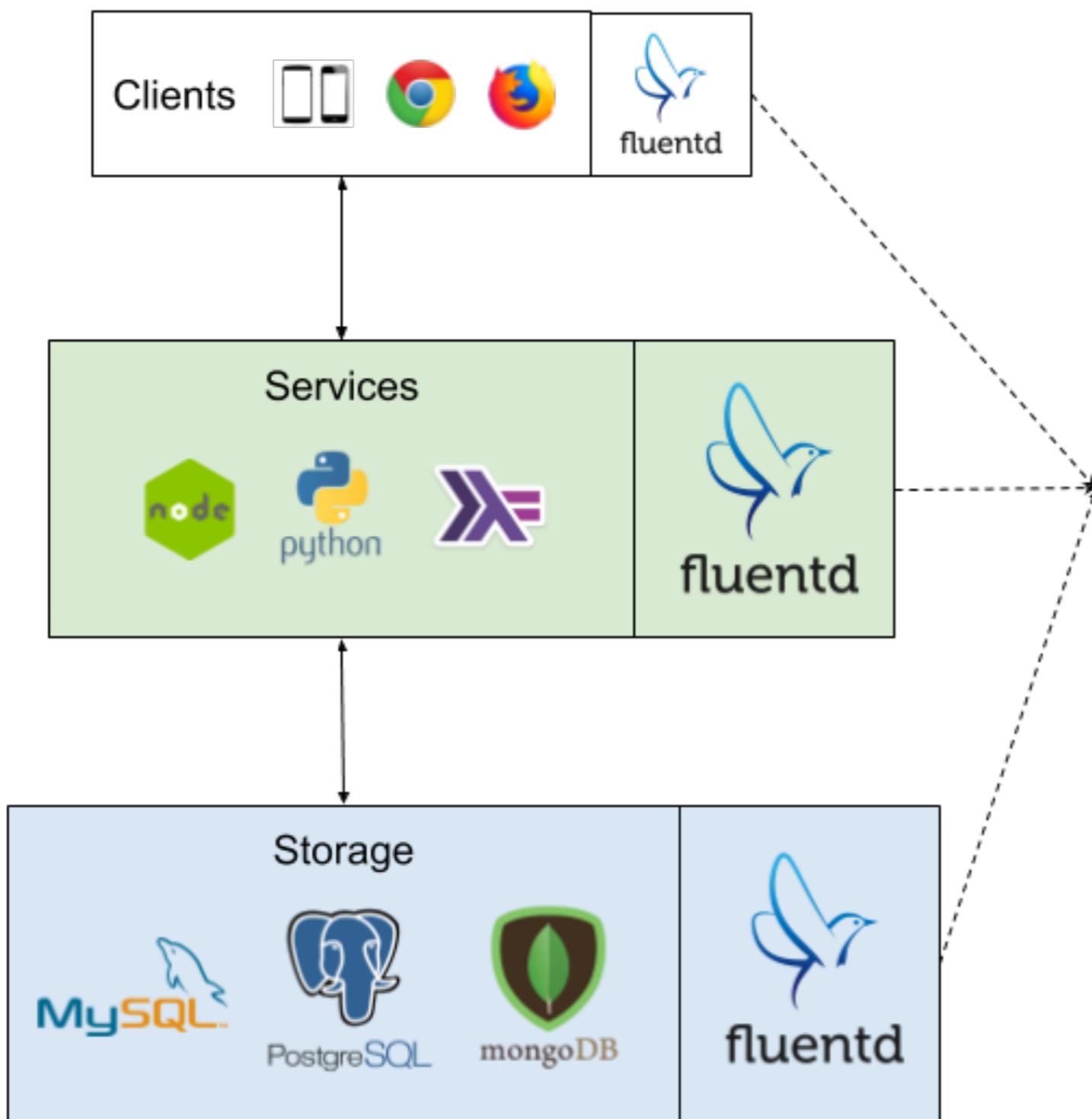


OPENTRACING

Vendor-neutral APIs and instrumentation for distributed tracing.

<https://opentracing.io>

Most of the “infrastructure data” players are converting support for all styles of system data collection



Super Data Collector



splunk®

With a data pipeline you'll be
setup to handle whatever
tool(s) come next

(Leverage abstractions at the integration layers to allow
easier adaptation to change)



5. Minimize, isolate and track dependencies

**Unmanaged dependencies
are where throughput goes to
die**

(And what creates and increases complexity
faster than anything else)

Golang got a few things
correct.

One of them is promoting
code duplication over
introducing unnecessary
dependencies

Personally, I promote the
Golang + Haskell approach

A dependency can be introduced when it is well formalized and worth the cost

(In the Haskell world you'll see laws for APIs. These are pretty stable APIs.)

Quick Note:

Avoiding dependencies does
not mean “build everything”

Javascript Padding Library

!=

AWS Dynamo

Using Dynamo + client library
is less code and likely no
additional dependency vs
building from scratch

And way better than building your own database

(Even though these days people seem to think
building a database is easy and necessary)

Back to regular schedule
programming

If you're going to introduce
dependencies then clearly
track and pin them

Ideally a single file in the
project/repo.

(Or in an aggregate repo)

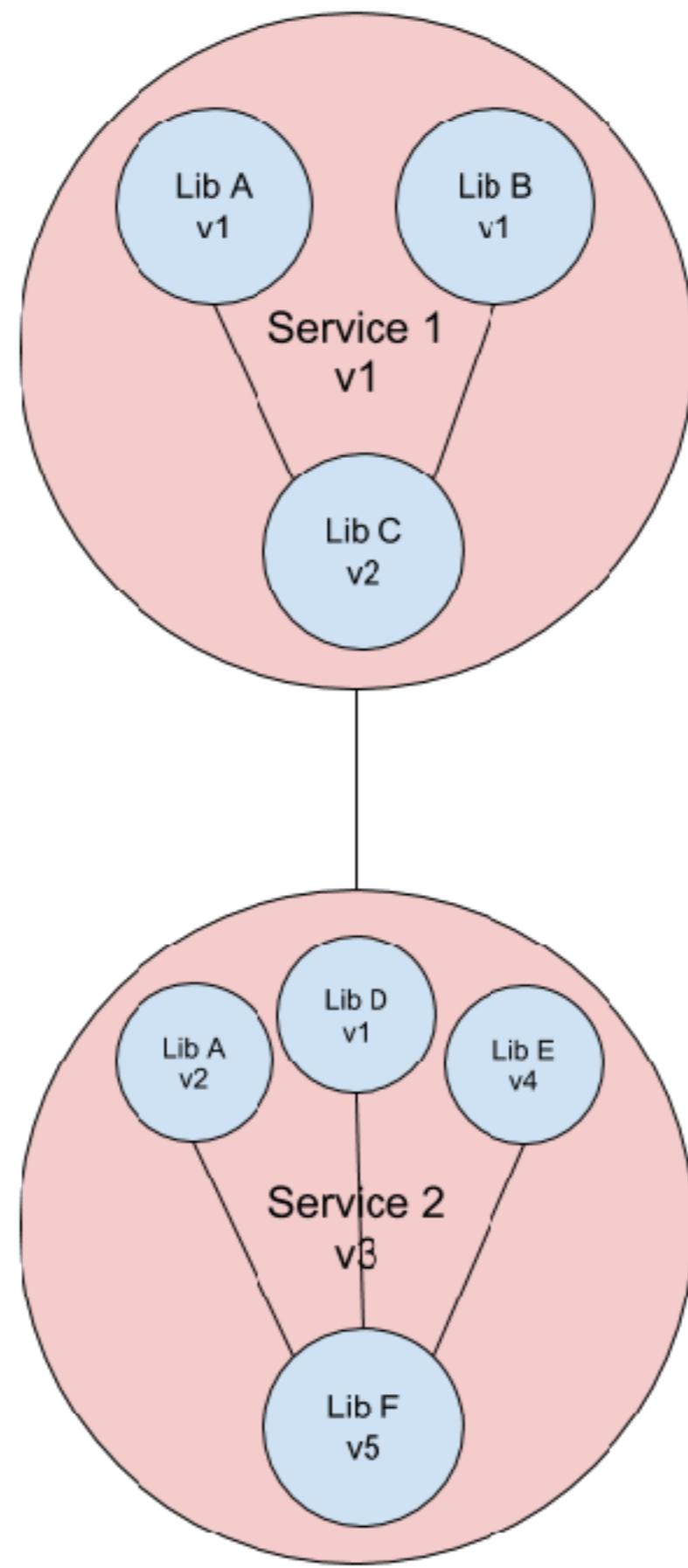
If possible standardize the
spec for these files

Then create a process that aggregates the dependencies into an overall mapping to give a picture of the system

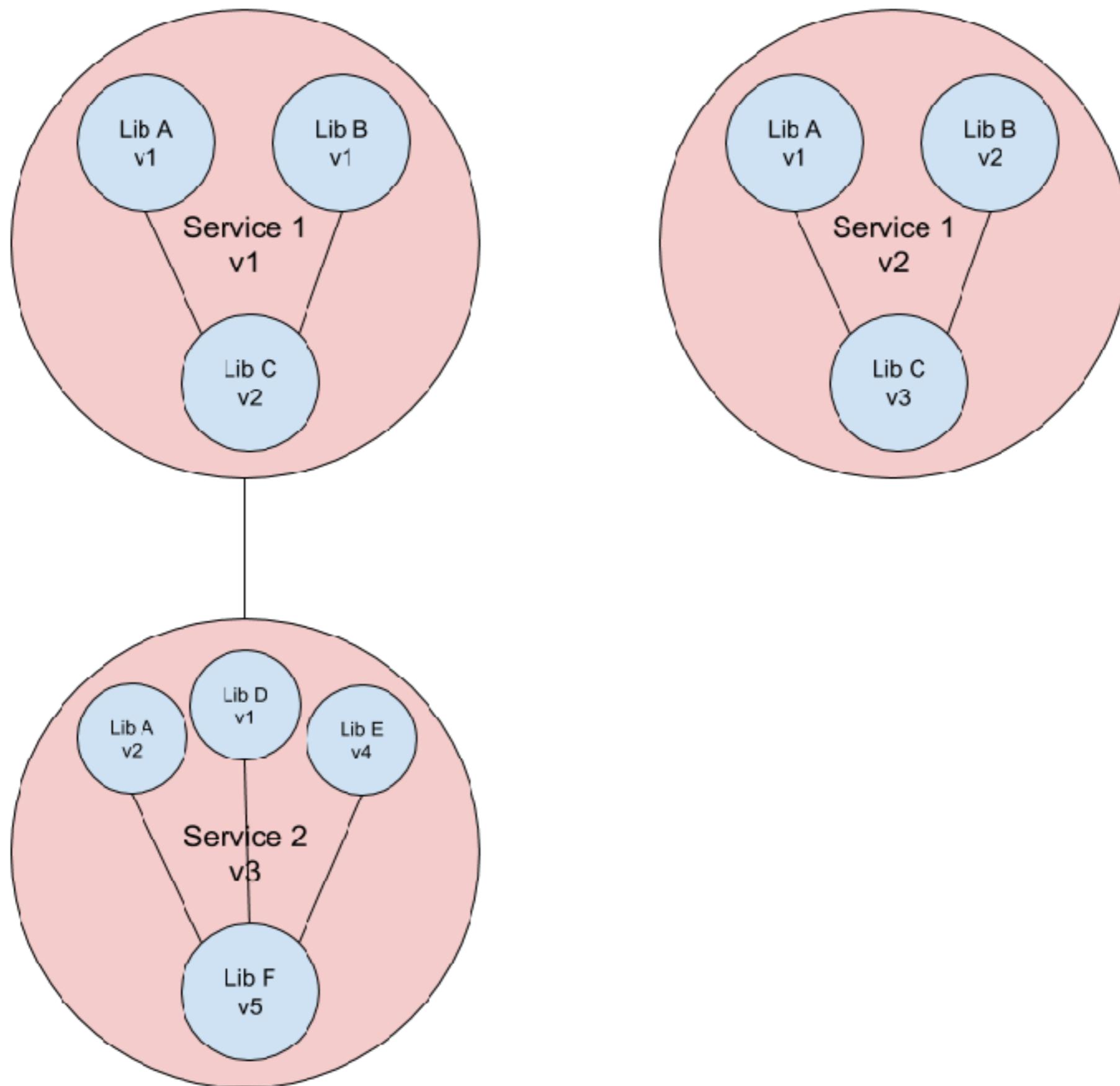
This goes for services as well
as libraries

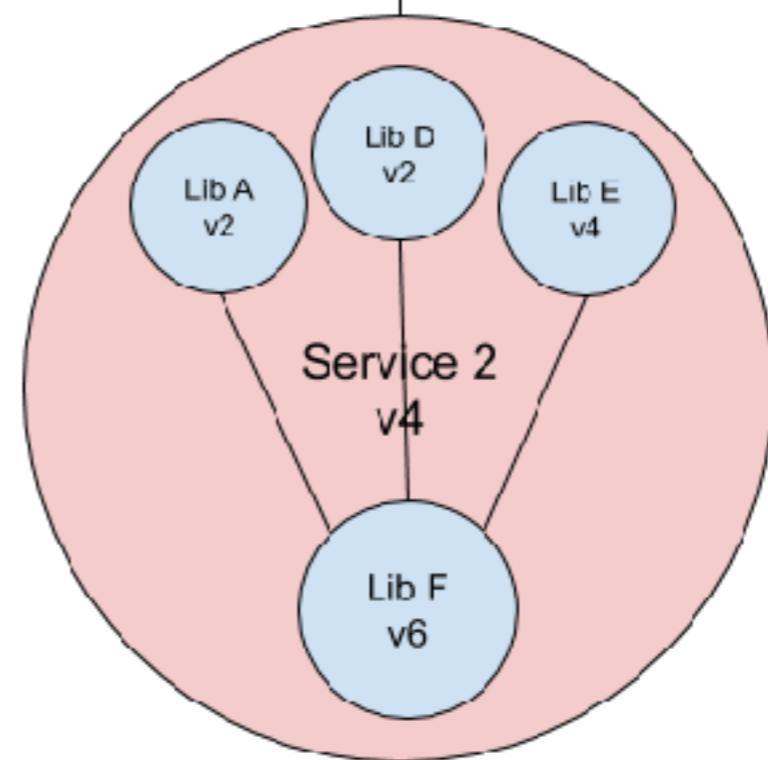
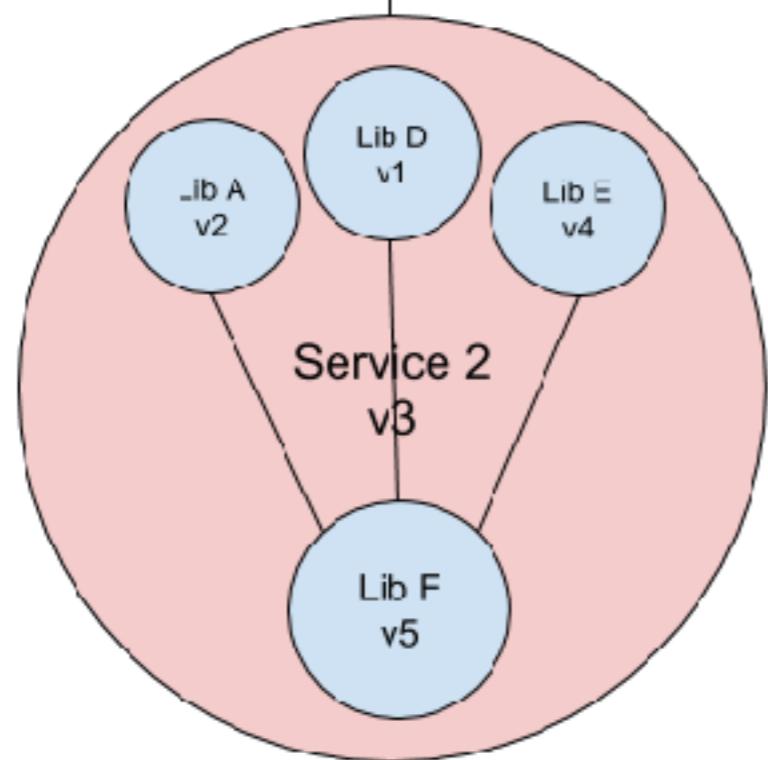
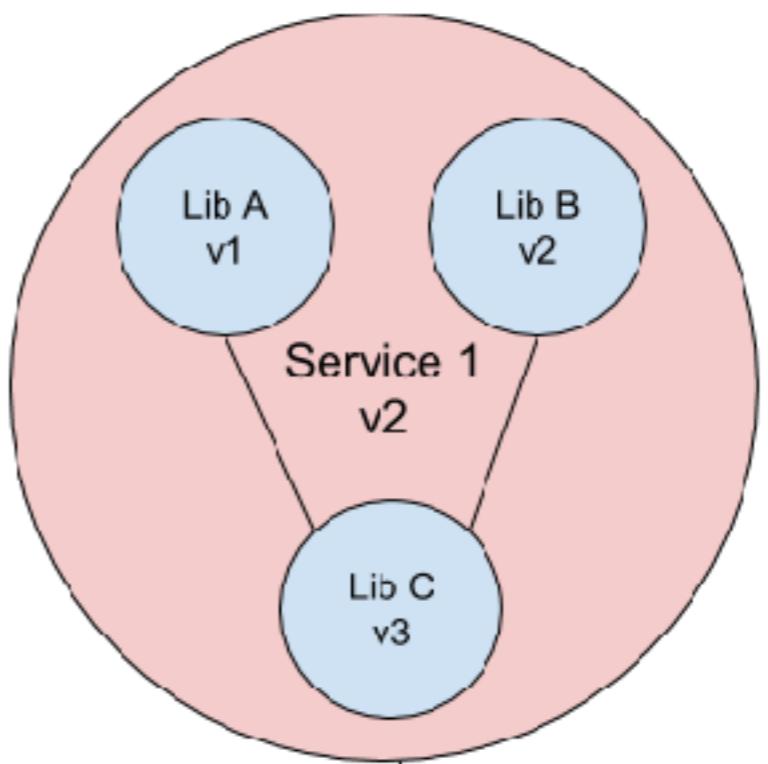
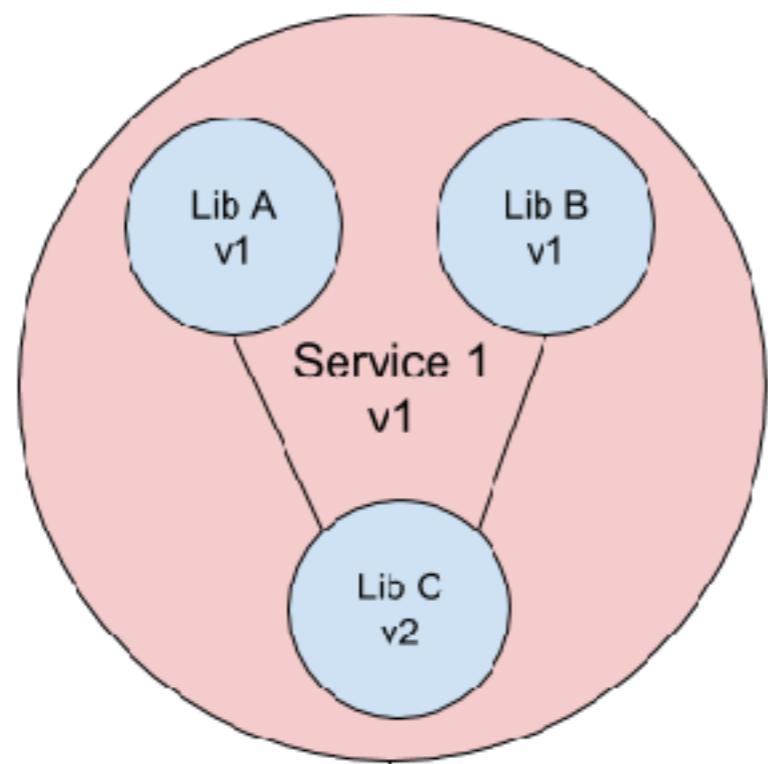
Then you can generate
diagrams

(Free architecture diagrams!)



Which you can then visualize
over time





Netflix has some great examples and tools

(Those #%*@\$!# are always leading the charge)

Out of necessity

Spigo and Simianviz

spigo and simianviz

[glitter](#) [join chat](#)

[godoc](#) [reference](#)

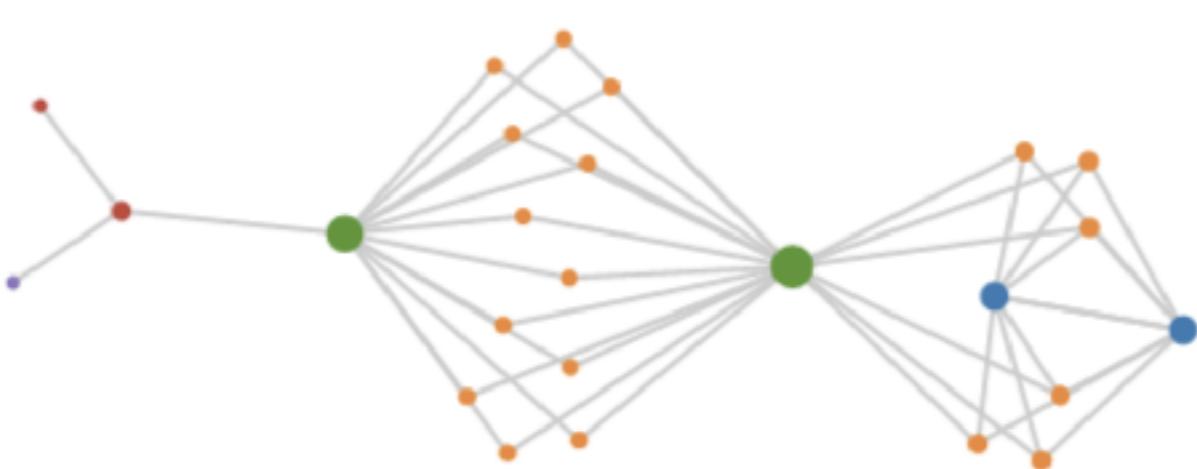
Wiki Instructions

SIMulate Interactive Actor Network VIsualization - simianviz - also visualize the simian army in action. Follow [@simianviz](#) on twitter to get update notifications.

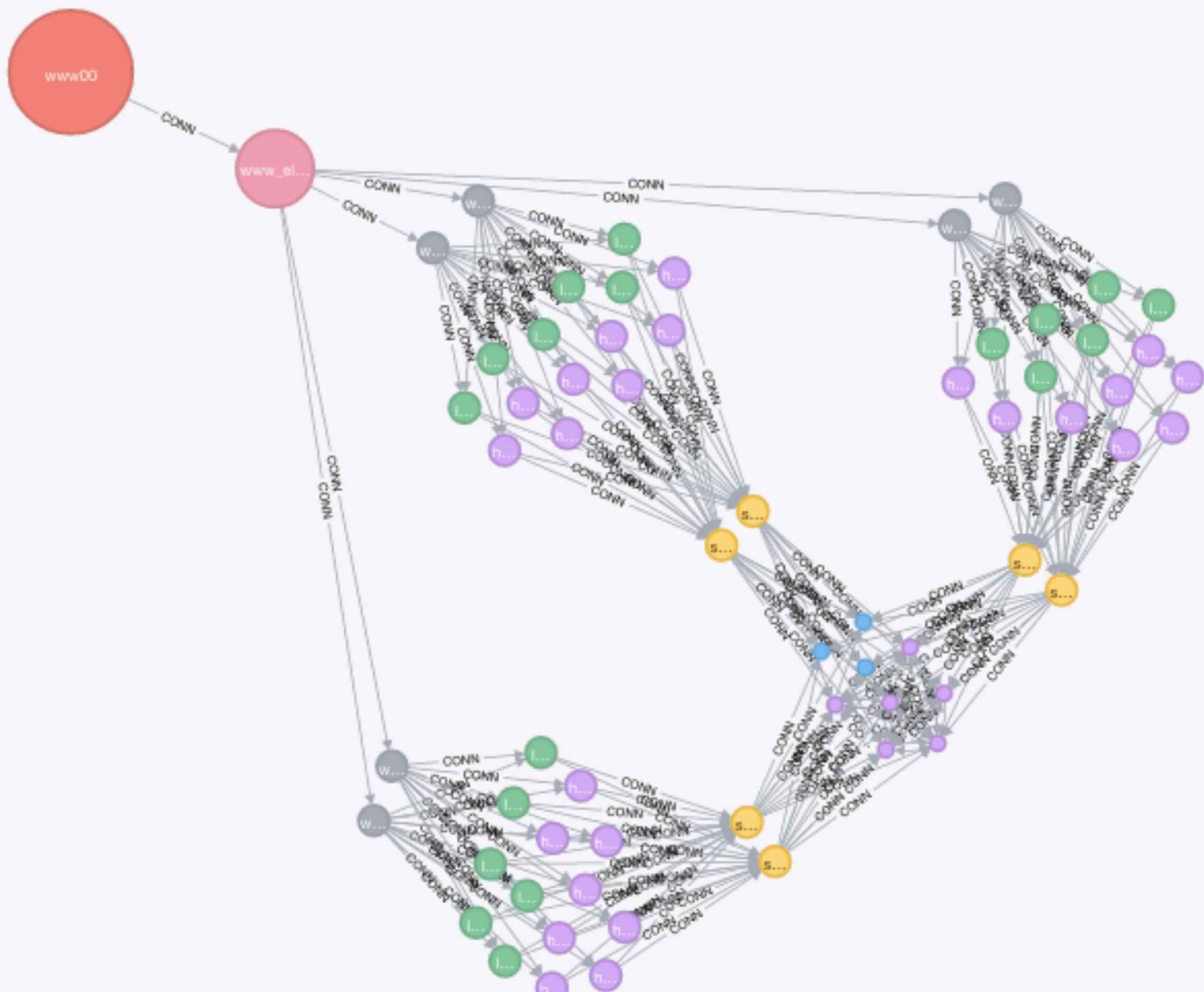
Originally called Simulate Protocol Interactions in Go - spigo - the name spigo is taken, however simianviz wasn't, so domains have been registered etc. and the name will transition over the coming months.

Using simianviz to view architecture diagrams

Launch the dependency graph visualization in your browser



For a local installation of the UI, with no network dependencies, you can start the service and browse localhost:8000

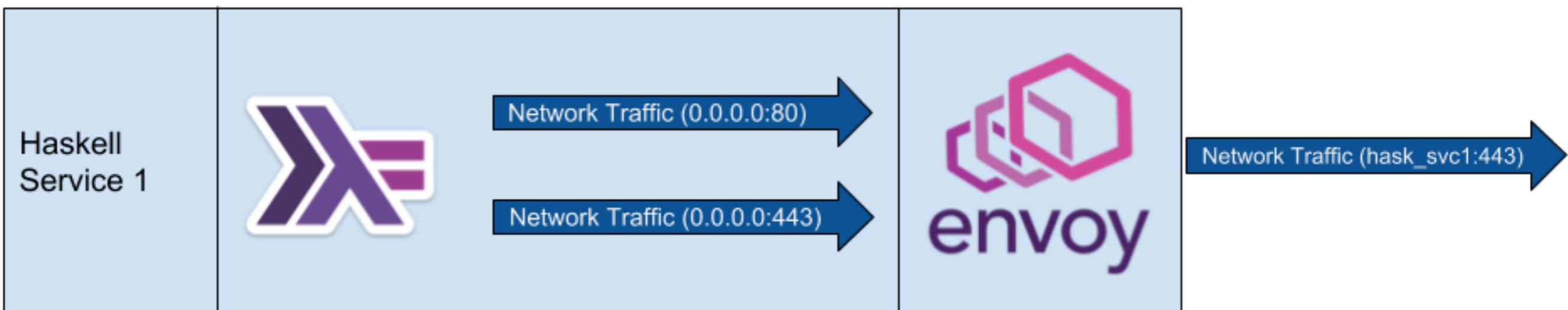
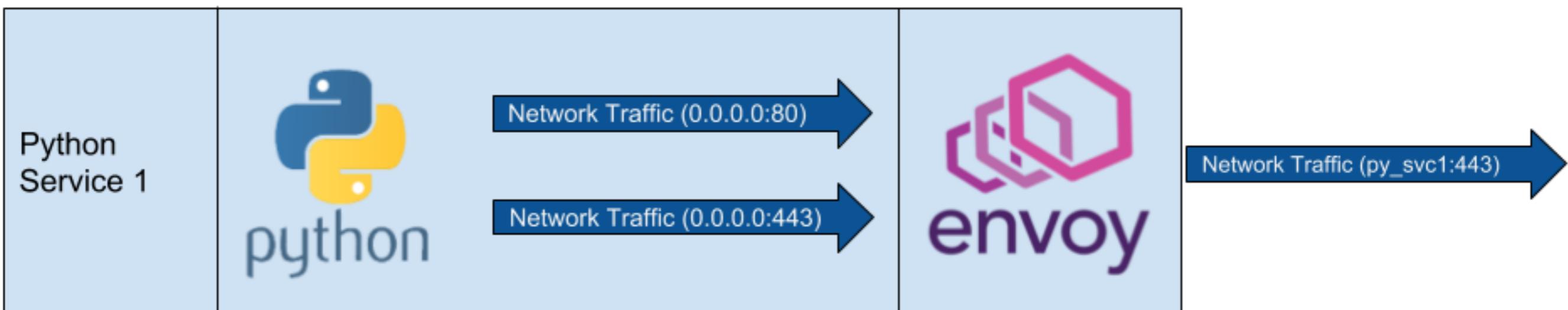
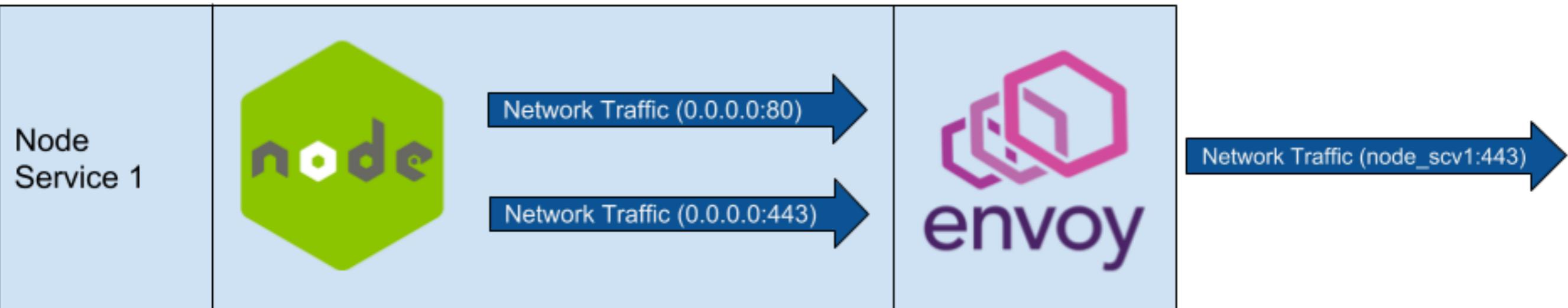


That said ...

Service/network
dependencies are still a
nightmare

6. Use network sidecars
(service mesh, proxies) to
better isolate and handle
these dependencies

Similar concept to the data pipeline except with even more “production” benefits



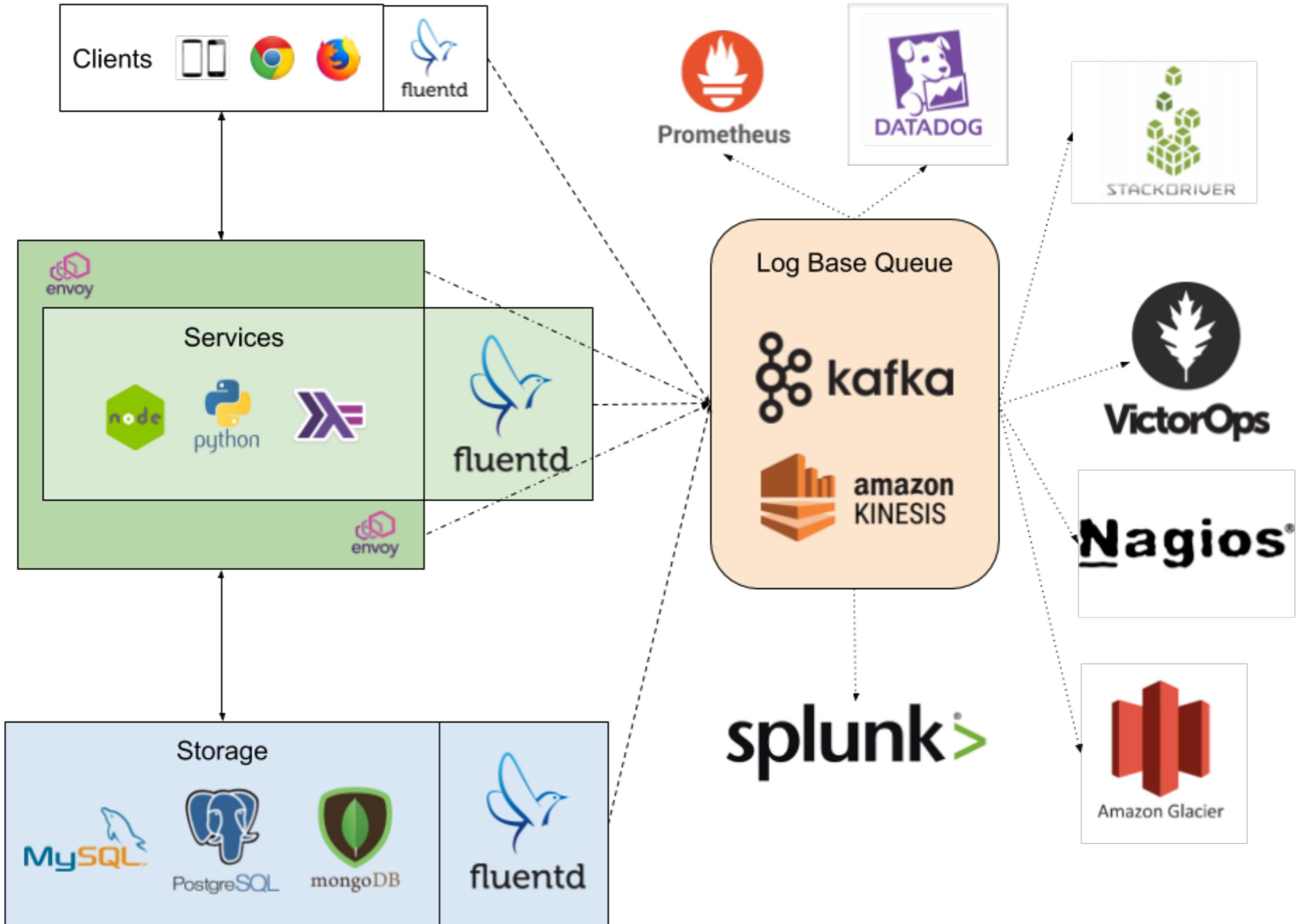
A combination of many of the
API Gateway, proxy, router, etc
solutions that exist today

Having a standard network proxy gives you:

Load balancing, service discovery, health checking, circuit breakers, standard observability (+tracing)

Using the sidecar allows you to
easily standardize without
introducing new dependencies
at the code and team level

And of course the meta-data
can be pumped to your same
data pipeline



Many new tools

Especially around Kubernetes

Vary from service mesh
focused to full bore micro-
service framework



Istio



linkerd



Kong

All of these come with “free” monitoring tools

And ...

7. Distributed Tracing

We need better ways to
visualize our systems

Charts, dashboards are nice
for looking at system
behaviors in a generic, data
driven perspective

But that layer of abstraction
(while helping isolate
variables) removes a layer of
intuition

We need the ability to also
visualize specific and
aggregate behavior

Tracing is one example

```
def my_func(*args, **kwargs):
    logging.info("start")
    analytics.store("my_func", "start")
    do_something()
    do_something_else()
    do_another_thing()
    logging.info("end")
    analytics.store("my_func", "stop")
```

This is really slow and we don't
know why so we start doing
naive timing crap

```
def my_func(*args, **kwargs):
    logging.info("start {}".format(time.now()))
    analytics.store("my_func", "start")
    do_something()
    do_something_else()
    do_another_thing()
    logging.info("end {}".format(time.now()))
    analytics.store("my_func", "stop")
```

Let's take advantage of our context and structured logging to enable tracing

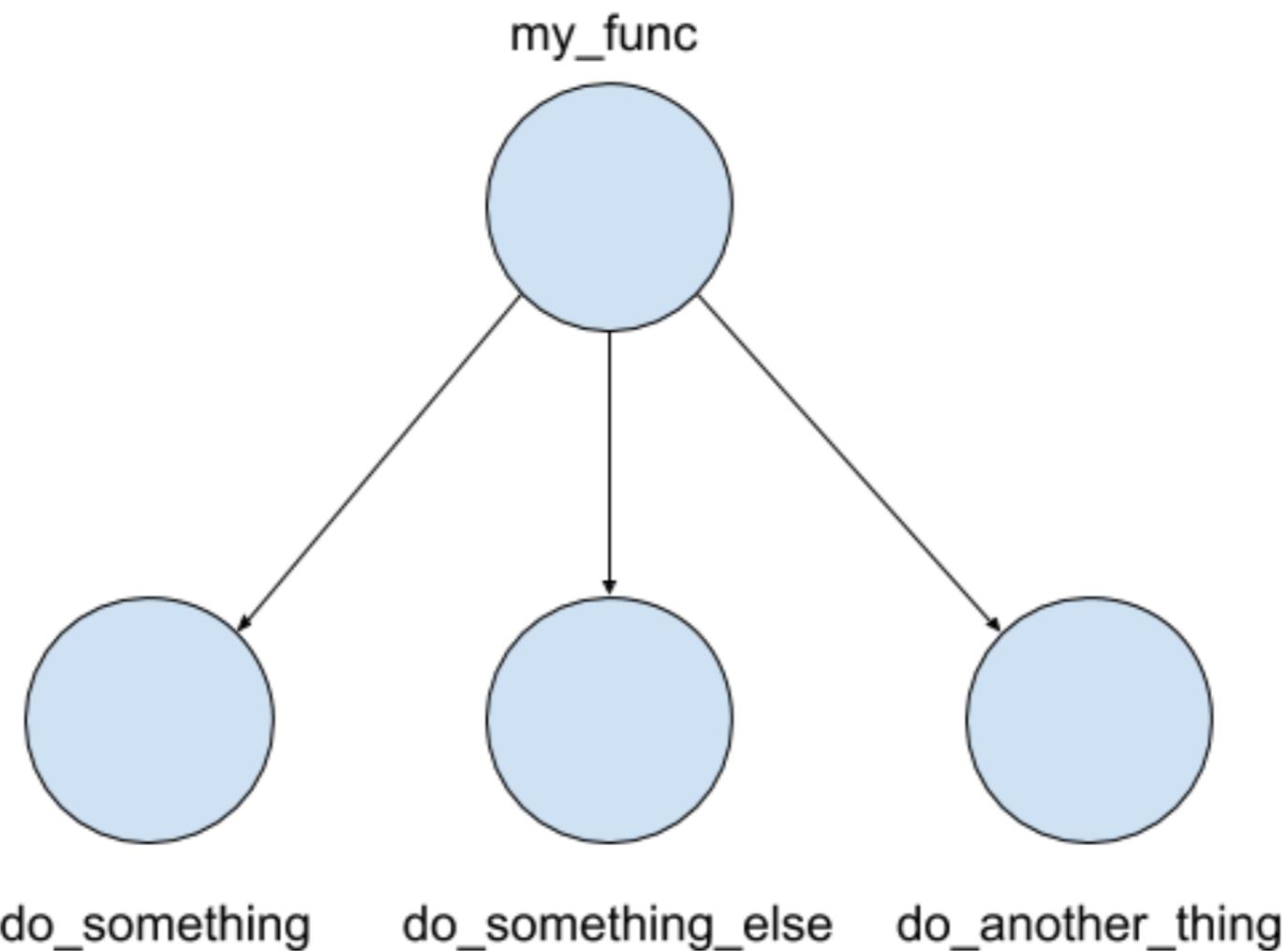
```
ctx = { "trace_id": "t1", "parent_id": None, "id": "newgenid" | more}
```

```
@trace()  
def my_func(ctx, *args, **kwargs):  
    do_something(ctx)  
    do_something_else(ctx)  
    do_another_thing(ctx)
```

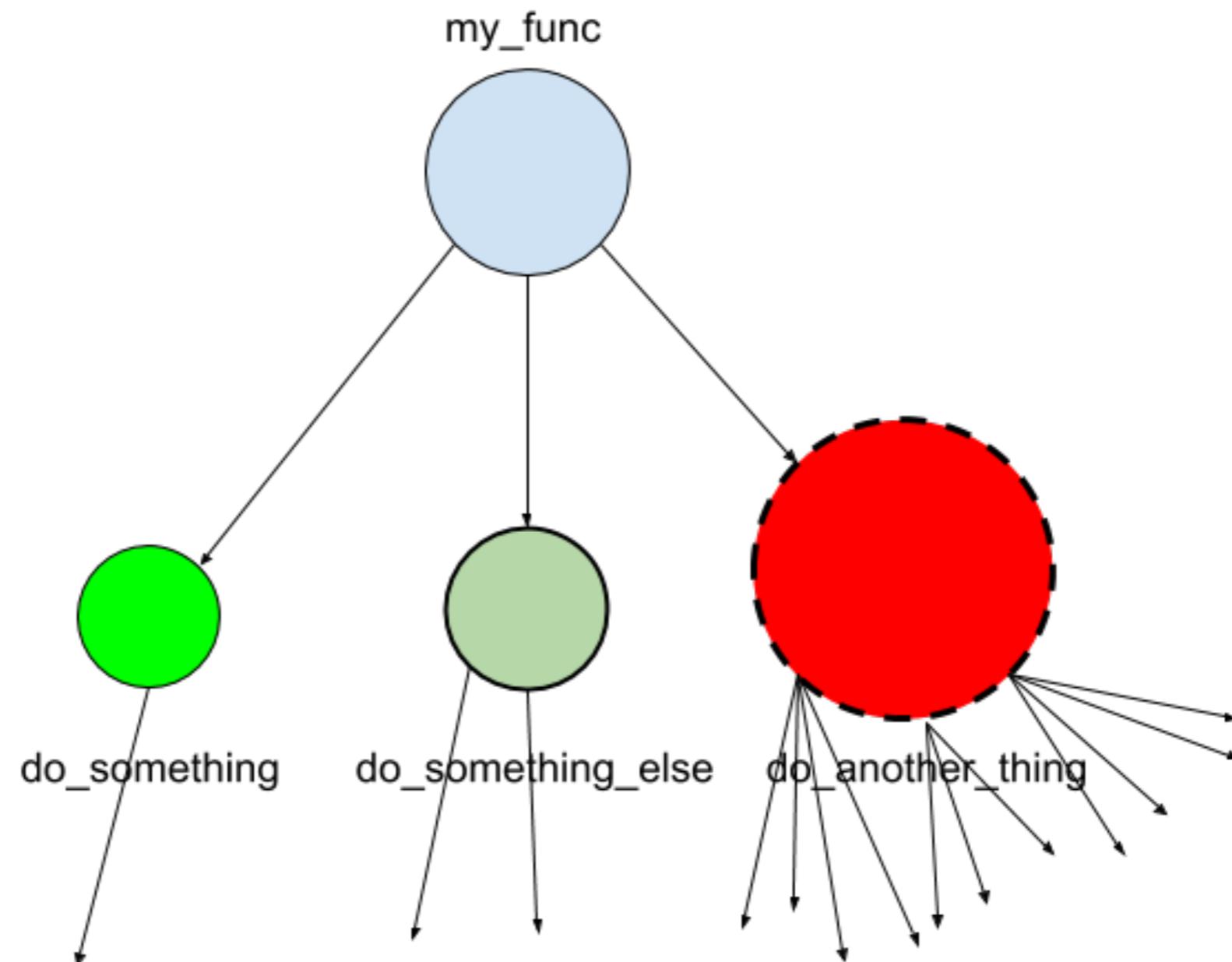
```
ctx = { "trace_id": "t1", "parent_id": "newgenid", "id": uuid.new | more}
```

```
@trace()  
def do_something(ctx, *args, **kwargs):  
    some_other_crap ...
```

This will give us the ability to
get a call graph



And since we're collecting all of the metadata that we can we know the characteristics of these nodes

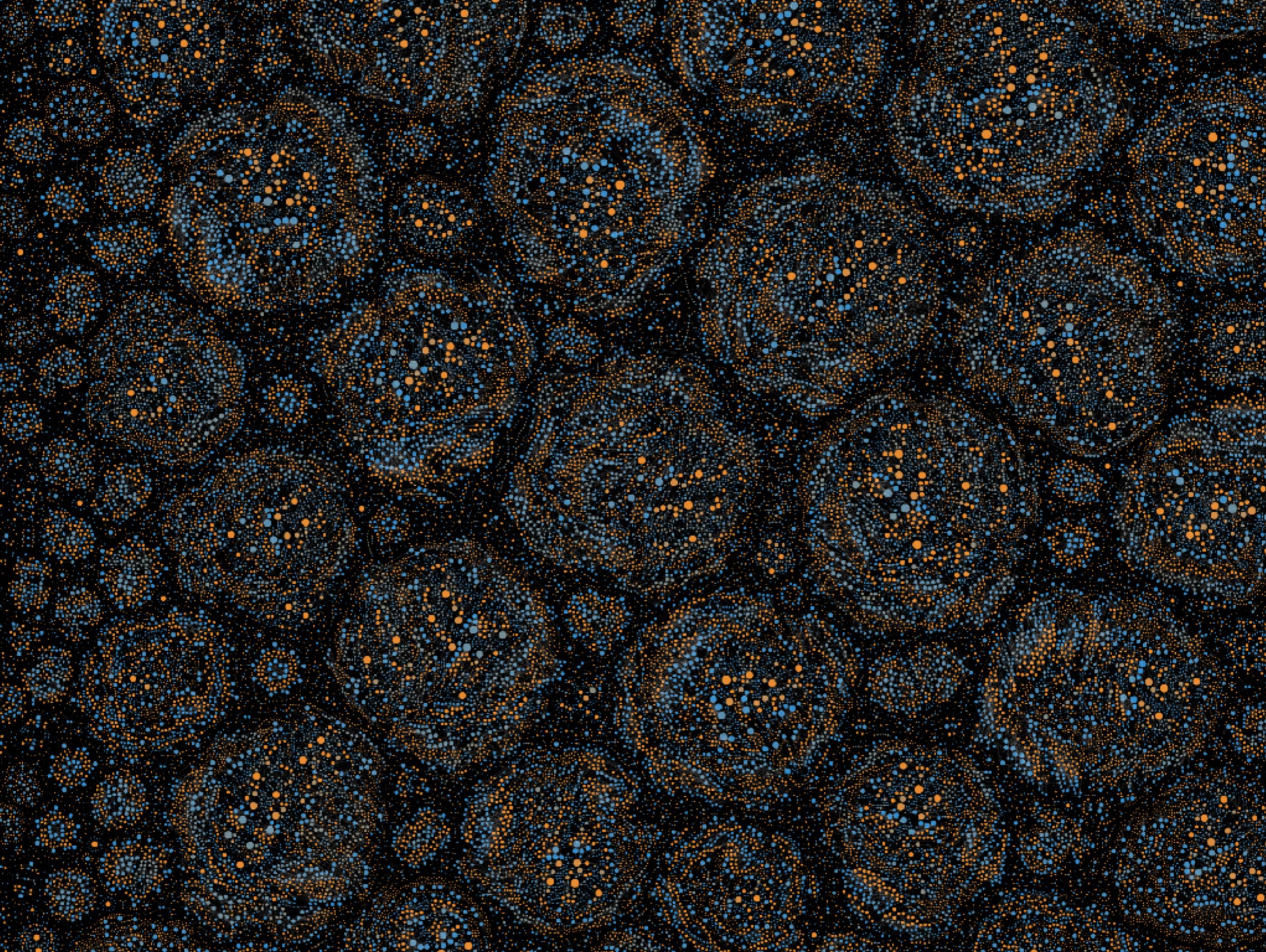


Oh crap, those aren't "pure" functions. They're all doing IO.

(Stupid ORMs and their poor abstractions. A good abstraction would make it clear there is IO happening)

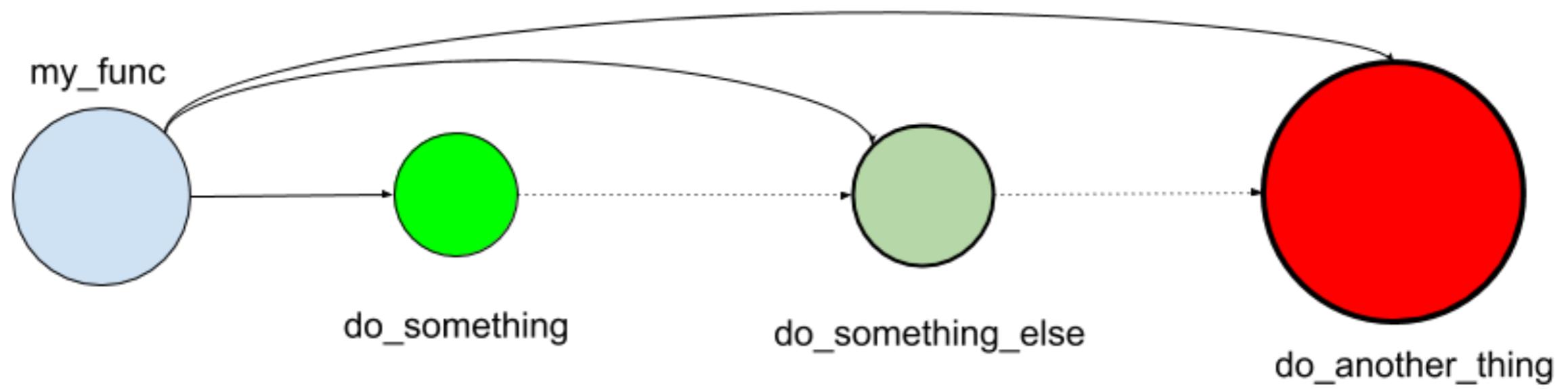
This visualization does a good job showing dependencies

(And is very good at representing larger, distributed, asynchronous processes)



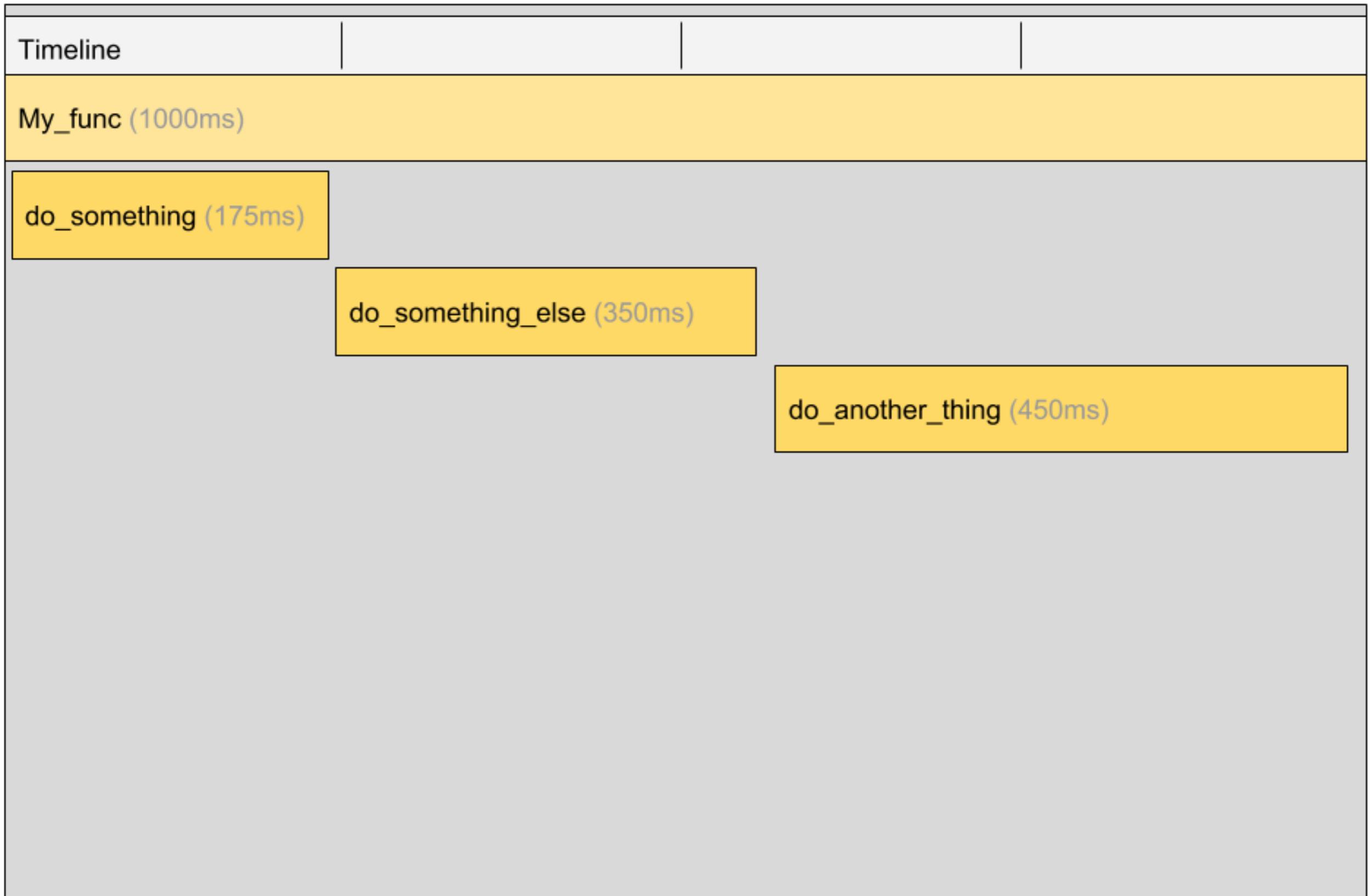
But it's not great for all needs

In our example these are
synchronous processes



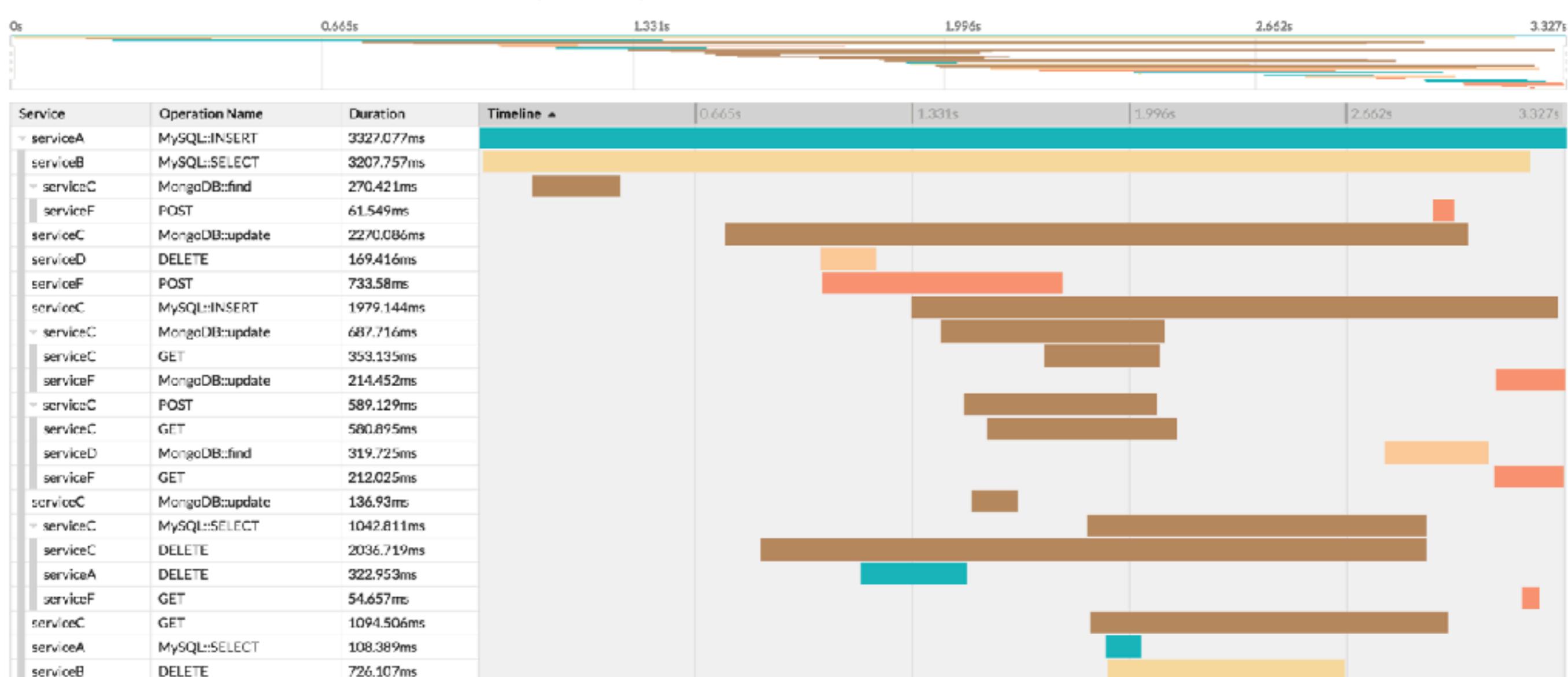
This style isn't intuitive for the
actual stack + performance
within the process

Standard Tracing View



serviceA: MySQL::INSERT

Trace Start: November 8, 2016 6:53 PM | Duration: 3.327s | Services: 5 | Depth: 3 | Total Spans: 35

[View Options](#)[Search...](#)

Come with the ability to
search, discover traces



Find Traces

Service

serviceA

all

Tags

http.status_code:400|http.status_code:200

Start Time

11/09/2016

12:00 AM

End time

11/09/2016

06:50 PM

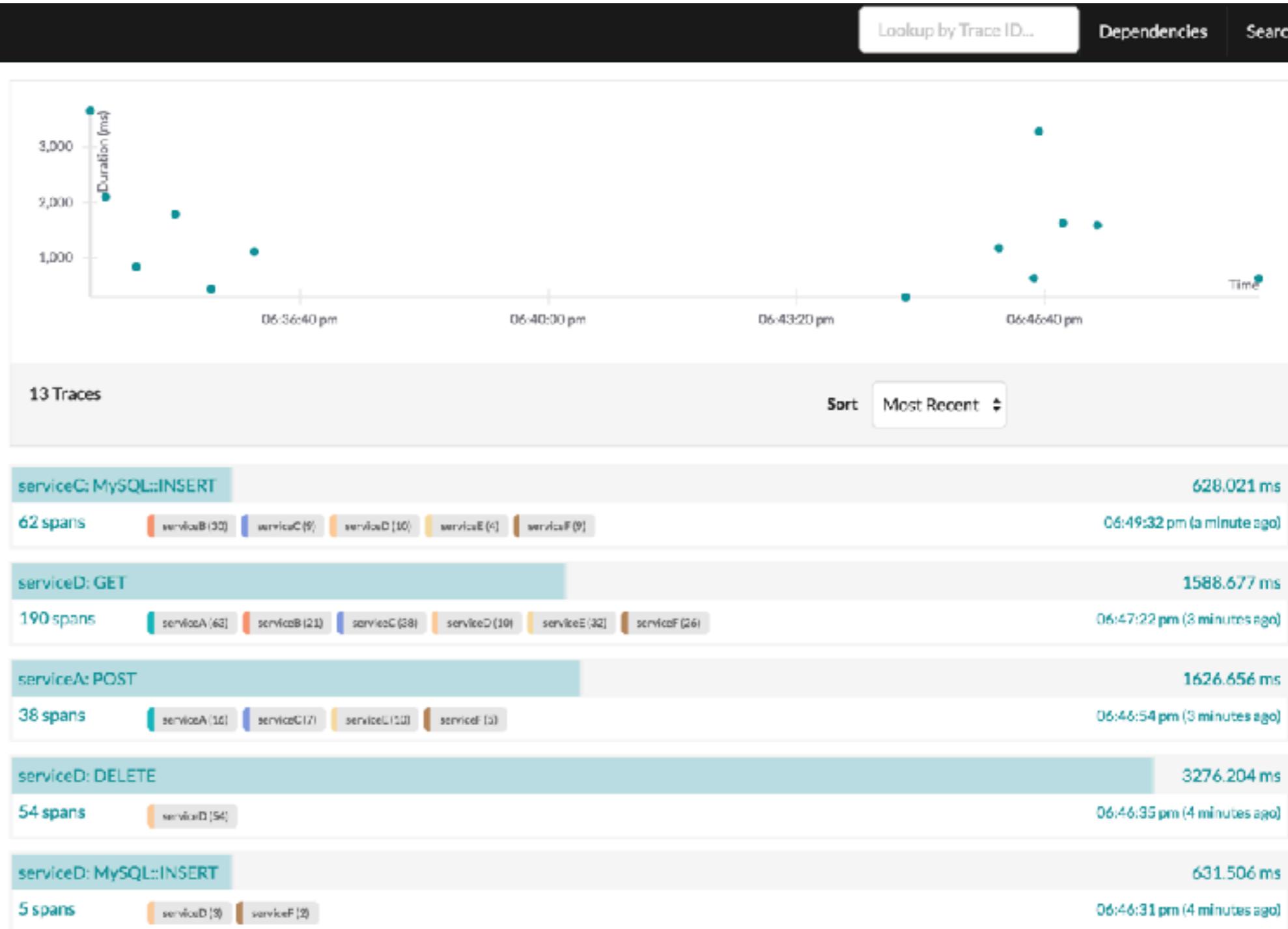
Min Duration Max Duration

e.g. 1.2s, 100ms, 5s e.g. 1.1s

Limit Results

50

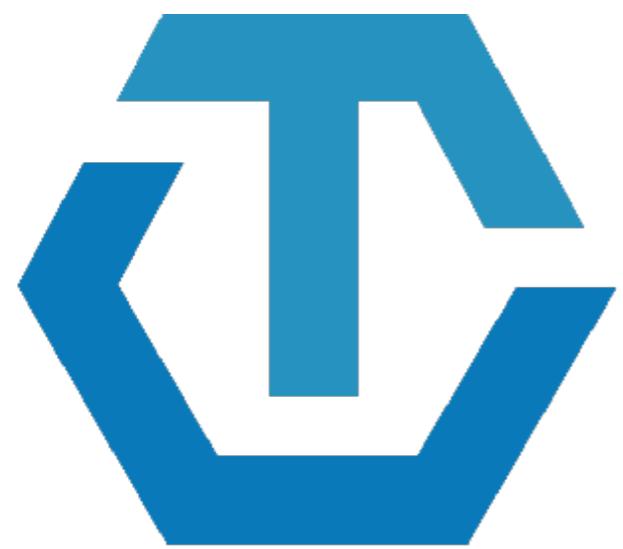
Find Traces



Tracing standards and systems are quite immature but growing (and hopefully stabilizing) quickly

2 Parts

The spec



OPENTRACING



OpenCensus



Distributed Trace Context Community Group

This specification defines formats to pass trace context information across systems. Our goal is to share this with the community so that various tracing and diagnostics products can operate together.

<https://www.w3.org/community/trace-context/>

<https://github.com/w3c/distributed-tracing>

Pick something. Use structured
logging + data pipeline to pass
off (and transform if necessary)
to tracing aggregator

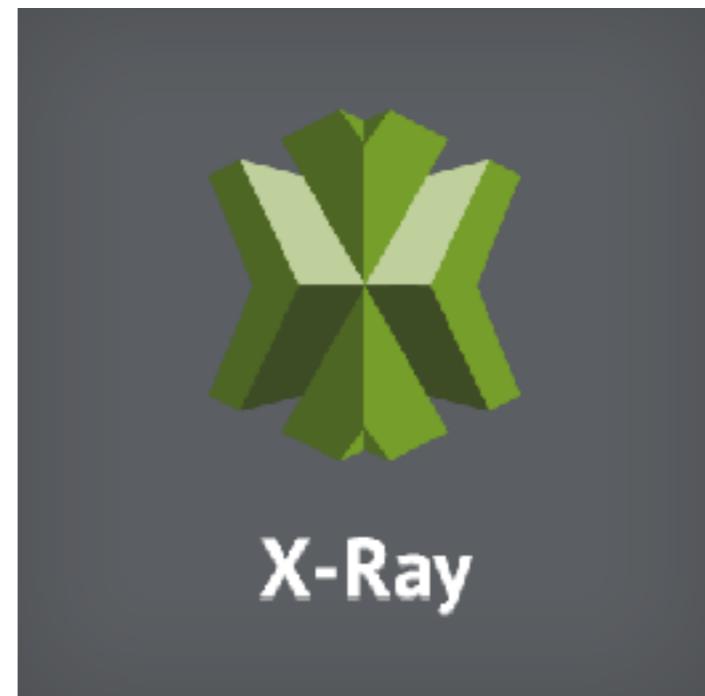
The aggregators



JAEGER



Z I P K I N



And as mentioned many of the collectors are including (or in the process of adding) tracing as part of their offerings

Super Data Collector



Prometheus



splunk®

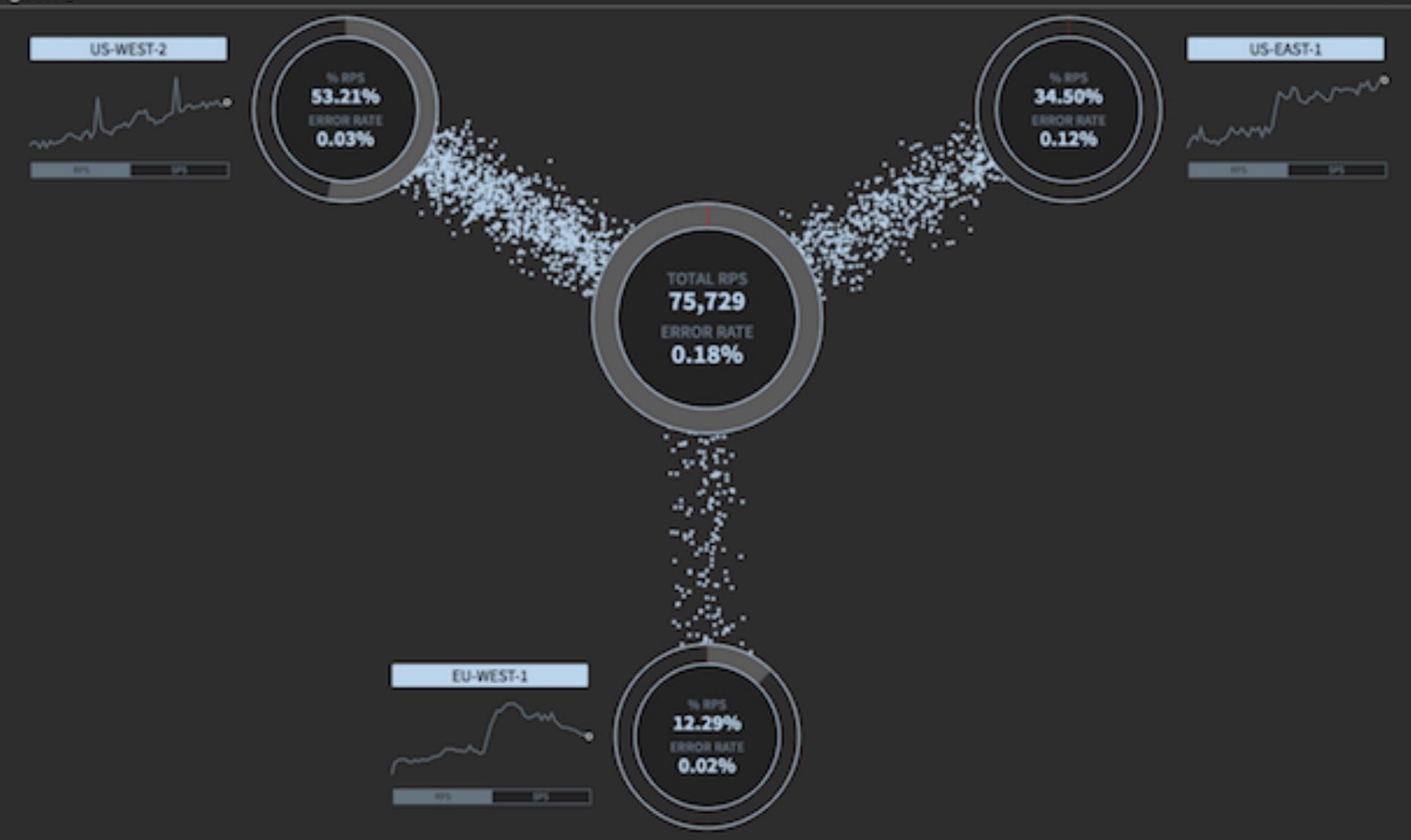
But any system that lets you
query and aggregate
relationships will give you the
base system necessary

Give your users the ability to
create the visualizations and
“traces” that map to their use
case

Those Netflix folks again

vizceral

<https://github.com/Netflix/vizceral>





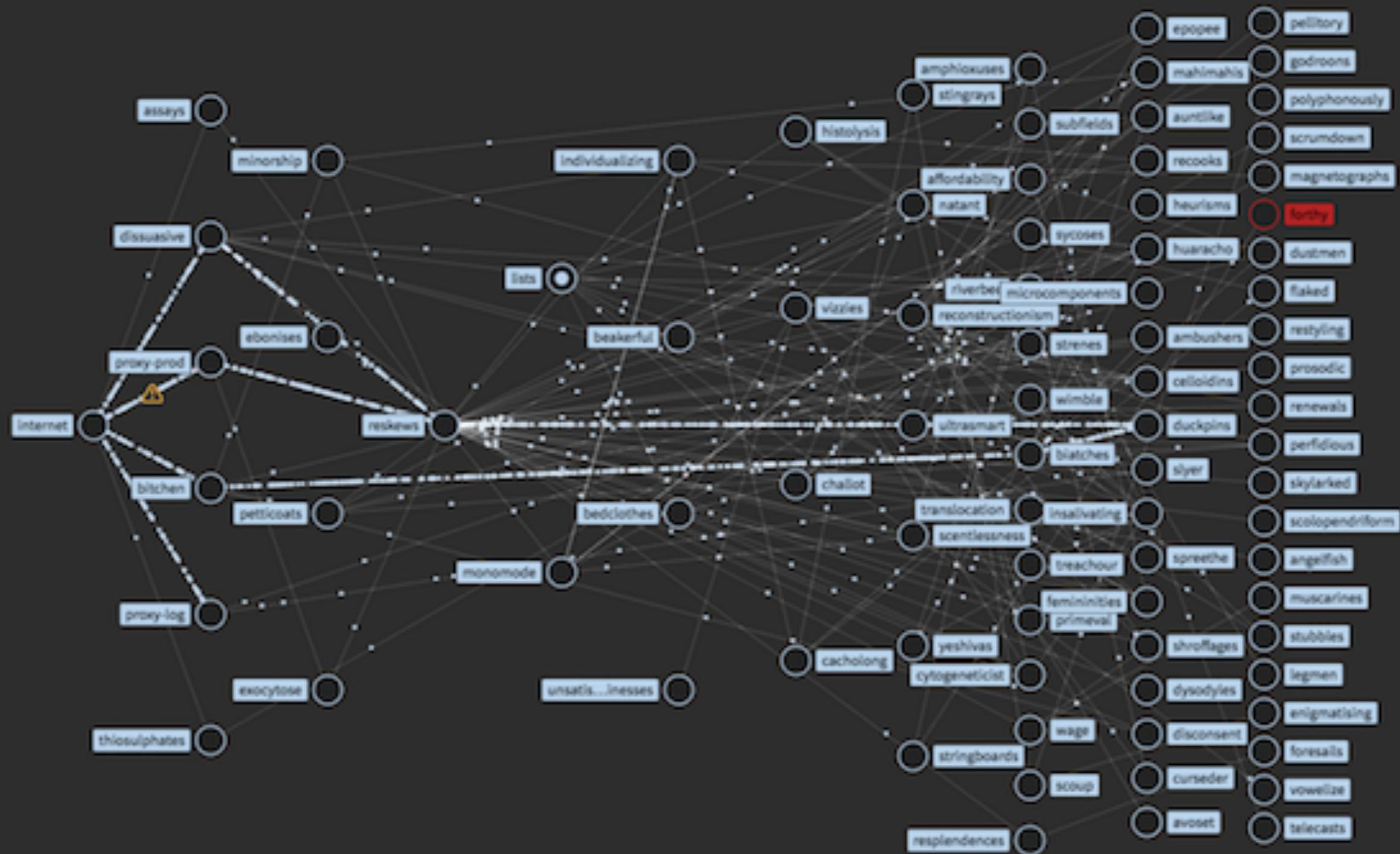
► LiveSlides web content

To view

Download the add-in.

liveslides.com/download

Start the presentation.





► LiveSlides web content

To view

Download the add-in.

liveslides.com/download

Start the presentation.

8. Provide the ability to “trace” through the system without impact

Some folks call this the
“Tracer Bullet”

It is a way to simulate a request through the system that makes no “destructive” change

In other words:
Send request that NoOP
writes to storage, writes to 3rd
Party apps

(Be careful to impact 3rd party quotas, licenses.)

FYI, this is how companies like
Amazon test their AWS APIs

Leverage the context

```
type Context =  
{ user_id :: String  
, account_id :: String  
, trace_id :: String  
, request_id :: String  
, parent_id :: Maybe String  
, request_type :: (STANDARD, TRACE)  
}
```

```
def my_func(ctx, id, data):
    my_thing = db.get(id)
    my_thing.data = data
    if ctx.request_type != REQUEST_TYPE	TRACE:
        # Write to storage
        my_thing.put()

# More ideally we wrap our storage layer to use the flag
```

This looks like a feature flag

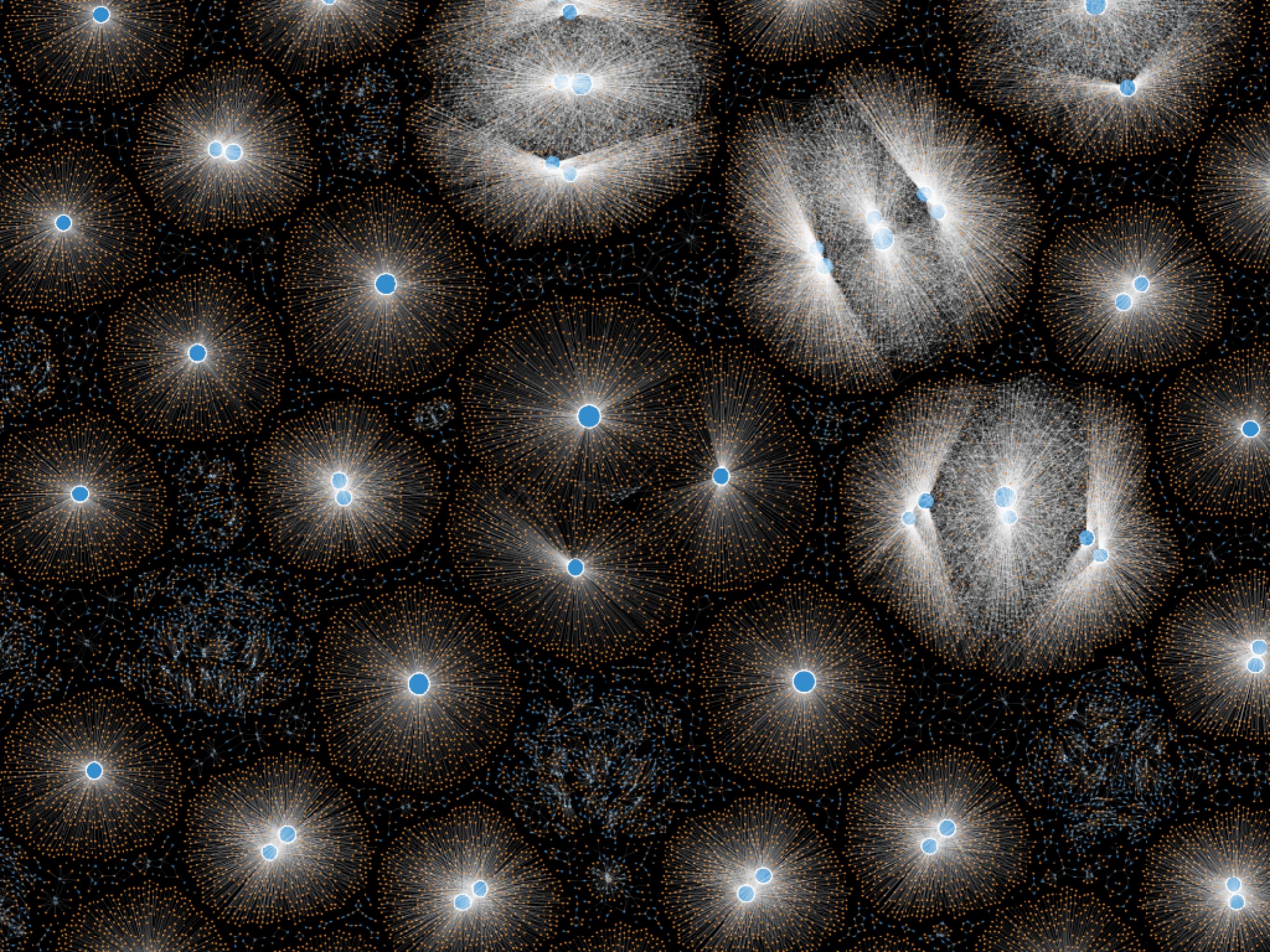
Yes!

Use feature flags!

And you can use them for
more than just features

Just make sure you log those flags as part of your context so your tools can properly tag the data

Tracer bullets is how we generated our graphs



And now I'm going to get
“rant-y”

9. Provide the ability to experiment and test in production

Tracer bullets, feature flags
allow us to use our production
system for gathering
information

We should also support
“tester” accounts so you can
fully mimic all user actions in a
production system

All of the work you need to do
to support this is work that you
should do anyway to fully
support multi-tenant apps

The ability to isolate services,
accounts, actions on demand

The ability to stop, interrupt,
move bad acting services,
users, etc

Ideally, support chaos tools in production

(Also, use chaos tooling! :))

Allowing folks to experiment and learn within the production system helps them build an intuition for the system, its behavior, and their impact on that behavior

10. Use tools (custom if necessary) to simulate usage

Load testing, chaos, general traffic simulation

Using network proxies and a data pipeline will allow you to capture actual traffic ...

Of which you can then replay
to simulate certain traffic
patterns, etc

11. Kill environments

Less environments means ...
Less environments

Less things to maintain and understand means we can put more time in understanding other systems

**“Production” (any environment
of which customers have
access) is the only
environment that matters**

So why do we spend so much
time not in production?

We know replicas and models
are not as good as the real
thing

Yet we continue to
build that way.

And worse we allow shortcuts
in other environments that
won't work in production

(SSH in Dev, No SSH in Prod)

Wouldn't you also want those
tools and abilities in
production?

We don't invest in building
production capable tools for
dev because ... time?

So instead you're going to wait
until you have a production
issue?

Scenario: Massive Outage

Boss: What are we doing to resolve the issue?

You: Well, not much. Normally I would do “x” but I can’t because those only work in dev environments. So I’m going to attempt to hack together some duct tape solution that I’ll never use again. And I’m going to run it now in production without going through the code review process.

If you've done everything mentioned then why would you need other environments?

(Quick answer: If you need to change/test core infrastructure that impacts all users at all times)

Do your best to force as much development and testing in production as possible

Quick answer: If you need to change/test core infrastructure that impacts all users at all times

In closing ...

There's so much more we can
do that I didn't get to

And it all starts with empathy
for our peers and users

Please come talk to me I
would love to discuss further

@lyddonb

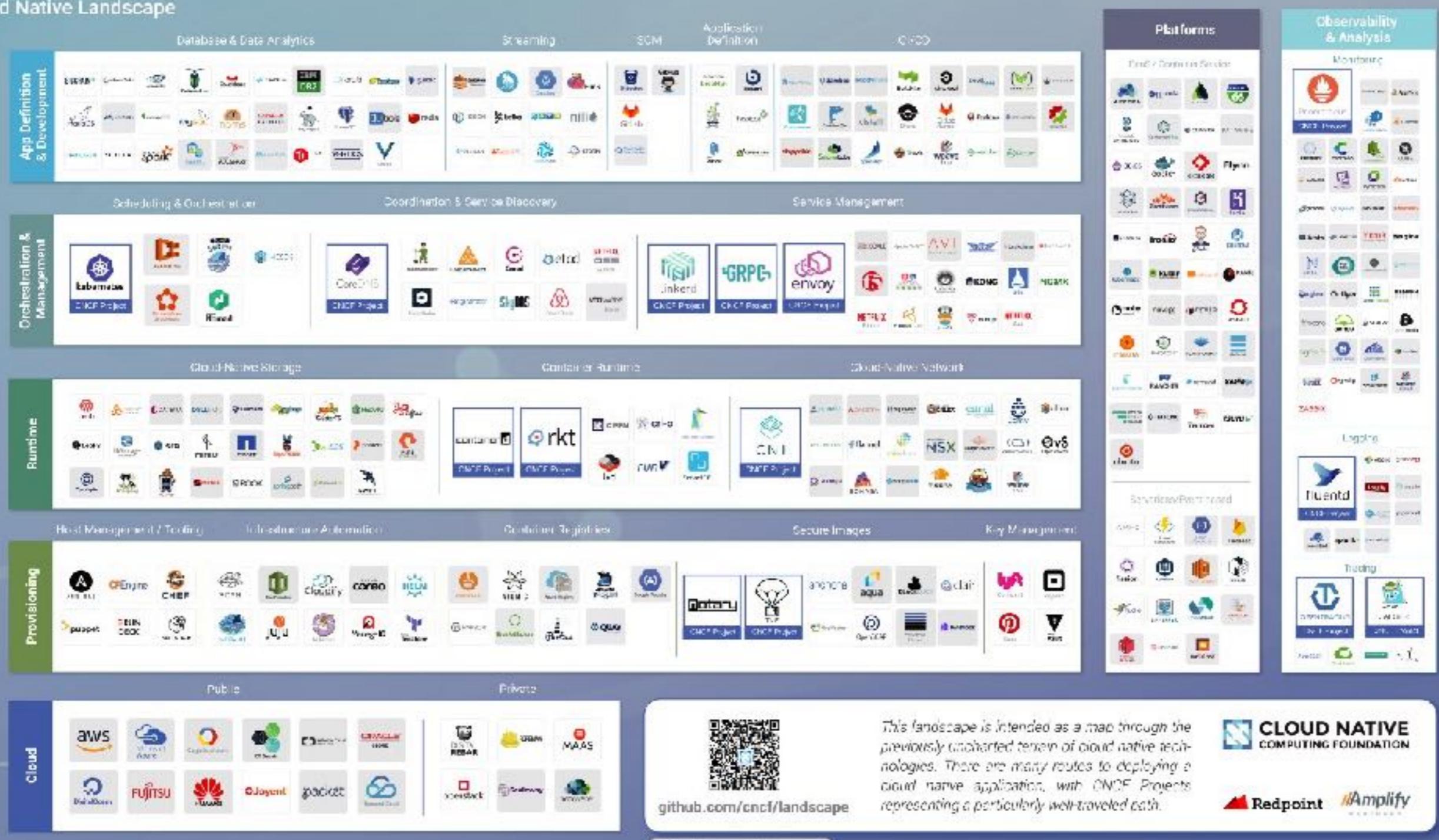
Quick Recap:

- Pass a context
- Structure your logs
- Create a data pipeline
- Structure all system data and pass to pipeline
- Minimize, track and build visualizations for dependencies
- Leverage service meshes
- Distributed Tracing
- Support NoOp, experimentation, simulation in production
- Then kill as many non-production environments as possible

And here are
all those tools again:

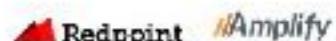
Cloud Native Landscape

2050



© 2010 Kuta Software LLC

This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path.



Thank You

@lyddonb

@real_kinetic



Real Kinetic mentors clients to enable their technical teams to grow and build high-quality software

Resources & References

- [Cloud Native Landscape](#)
- [Incidents Are Unplanned Investments](#)
- [stella.report](#)
- [How to Keep Your Systems Running Day After Day - Allspaw](#)
- [Honeycomb](#)
- [More Environments Will Not Make Things Easier](#)
- [Silicon Valley's Tech Gods Are Headed For A Reckoning](#)
- [On purpose and by necessity: compliance under the GDPR](#)
- [ACCELERATE: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations](#)
- [a16z Podcast: Feedback Loops — Company Culture, Change, and DevOps](#)
- [System and method for performing distributed asynchronous calculations in a networked environment](#)
- [You Could Have Invented Structured Logging](#)
- [What is structured logging and why developers need it](#)
- [How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript](#)
- [W3C Distributed Trace Context Community Group](#)
- [Load Testing with Locust](#)

Products, Libs, Etc

- [Splunk](#)
- [Datadog](#)
- [Nagios](#)
- [Apache Kafka](#)
- [Amazon Kinesis](#)
- [FluentD](#)
- [Prometheus](#)
- [Google Stackdriver](#)
- [VictorOps](#)
- [Amazon Glacier](#)
- [Google BigQuery](#)
- [Amazon Redshift](#)
- [OpenCensus](#)
- [OpenTracing](#)
- [Haskell](#)
- [Go](#)
- [AWS DynamoDB](#)
- [Spigo and Simianviz](#)
- [Envoy](#)
- [Kubernetes](#)
- [Istio](#)
- [Linkerd](#)
- [Kong](#)
- [Jaeger](#)
- [Zipkin](#)
- [AWS X-Ray](#)
- [Stackdriver Trace](#)
- [Vizceral](#)