Monitoring Is a Pain (matduggan.com)

345 points by kiyanwang 11 months ago | hide | past | favorite | 209 comments

jillesvangurp 11 months ago | next [–]

It's hard because it's being pushed and done by people with different agendas that also aren't necessarily the right agendas to be prioritizing.

- mistake #1, assuming it's a technical thingy that some techie person should do and they should just get on with it and do it. Whatever it is. The mistake here is that without guidance this person is going to be selecting some random tools mostly focused on operational things (logging, infrastructure). These have low value for other stakeholders. Yes you need that but these are also commodities.

- mistake #2, each of the stakeholders adds their preferred tools to the mix and the end result is a bunch of poorly integrated tools with a lot of complexity.

- mistake #3, assuming that selecting a tool means the job is done. It's not. Most of these tools assume that you have some notion of how to use the tool and what you want out of the tool. Getting the tool is a the beginning, not the end.

The combination is lethal. Lots of complexity, lots of data, lots of cost, and not a whole lot of value being delivered.

The fix is an observability mindset that exists in product, tech, and business departments. Select 1 tool, not 2, 5, or 10. The more tools, the more fragmented the effort and the more duplication of effort and mistakes. Have a plan, engineer for the system to be observable, have a notion of things that are important to track, etc. Don't dump the responsibility on some techie but actually reflect a bit on what you are looking to get out of this and what this is worth to you.

> tetha 11 months ago | parent | next [–]
>
> > - mistake #1, assuming it's a technical thingy that some techie person should do and they should just get on with it and do it. Whatever it is. The mistake here is that without guidance this person is going to be selecting some random tools mostly focused on operational things (logging, infrastructure). These have low value for other stakeholders. Yes you need that but these are also commodities.
>
> I'm honestly pretty much giving up on monitoring at work. I enjoy working on monitoring, and I think good monitoring and tracing is a strong edge you can have at a technical level. After a certain point, I'm pretty sure you will lose control without good monitoring, tracing and alerting.
>
> But practically, whenever there is a glimmer of hope, aka spare time, exists to do some work on the monitoring, the entire product stack realizes: There is spare time. Let's dump any combination of (i) introducing entirely new tech into the stack, (ii) Requesting weeks of analysis if postgres/any underutilized upstream component might be the cause of slowness for an individual database of a cluster on a group of postgres servers could be the cause of slowness, (iii) elevate something to "critical we cannot work and customers are crying just now" and (iv) some production outage by a shitty change due to an architectural weakness the product had since 3 years and we've been pointed it out, but we need a "task force" of "all qualified people on deck" except that the issue isn't related to e.g. the database, and as such, the postgres DBAs are twiddling thumbs. But "working together" means they must be part of "all hands on deck".
>
> Oh, and then everyone starts yelling why the we can't just pile new topics on infra every 3 month, and why we can't have an up-to-date modern monitoring stack at the same time from the same guys.

Monitoring and alerting just has zero visible value outside of infrastructure and operations imo and I'm pretty much done fighting for it. Sorry for being somewhat sour about this, I can be about things I really like.

javier2 11 months ago | root | parent | next [–]

I am the same, I love good monitoring. My place has been very focused on it, and we struggle to find time to work on it beyond basic «does it work». Yet We have so much monitoring we are drowning in metrics.

juancn 11 months ago | parent | prev | next [–]

That, and the tooling sucks.

The good tools are expensive, both Sumologic and Wavefront are amazing, but past a certain scale it's hard to justify the cost.

On the open source side Grafana and its stack (tempo, loki and prometheus) all just suck, they kinda scale and check all the boxes, but in a really stupid way (data nodes should probably be query nodes, rather than moving everything from storage and caching, everything is supposed to be a dashboard, don't get me started on the query languages).

The Grafana UI was built by the enemy and explorations are incredibly annoying. I'm more of an exploratory user, not a dashboard one (think level 3 on-call).

lazyasciiart 11 months ago | root | parent | next [–]

We just moved to grafana from New Relic and it makes me angry just to try and look at monitoring, we have lost *so much* utility. And the company spent a year on the transition and a shit ton of work, so it’s not (just) that we had good workflows set up that haven’t come across. (I am also a heavy exploratory user)

Jedd 11 months ago | root | parent | next [–]

If you were heavy users of APM, then that's going to be a step backwards, I'm sure - but Grafana's just a visualisation tool, so I'm not sure what *actual* stack you're comparing there.

I find NewRelic's UI to be more frustrating, but I've spent a lot more time with Grafana's (while very much not a fan of their relentless tinkering with UI layout).

ksaxena 11 months ago | root | parent | prev | next [–]

Have you tried Signoz (https://signoz.io/) yet?

mikeshi42 11 months ago | root | parent | prev | next [–]

Would love to learn what kind of exploration workflows you've missed in Grafana?

Context: Founder building an affordable alternative in the SaaS observability market (hyperdx.io) - I assume your company might have moved when NR introduced new seat-based pricing?

llama052 11 months ago | root | parent | prev | next [–]

What are you missing specifically that New Relic has?

We are a full grafana prometheus shop and the New Relic/Datadog price tag is insanity compared to the utilization we get out of the free tooling.

Once you get over the learning curve of promQL or whatever you're looking for it's not bad at all.

llama052 11 months ago | root | parent | prev | next [–]

> On the open source side Grafana and its stack (tempo, loki and prometheus) all just suck, they kinda scale and check all the boxes, but in a really stupid way (data nodes should probably be query nodes, rather than moving everything from storage and caching, everything is supposed to be a dashboard, don't get me started on the query languages).

hmm, you generally move prometheus data into a higher engine like Thanos. It does split layers up pretty well into separate components.

Honestly it does have a learning curve but the cost is generally free and compared to New Relic or something like DataDog they aren't all that different learning curve wise.

Either way it beats the days of ganglia or nagios static screenshots of some half baked python metric. Or even kibana where you'd have to run a gigantic ELK stack to get any data out of it.

> EdwardDiego 11 months ago | root | parent | next [–]
>
> > Either way it beats the days of ganglia or nagios static screenshots of some half baked python metric. Or even kibana where you'd have to run a gigantic ELK stack to get any data out of it.
>
> Holy hell, if you'd mentioned Munin also, I'd swear we worked for the same company. Spent ages making Munin play nice with Nagios.
>
> Ganglia was okay for monitoring Spark jobs running in a Yarn cluster, I guess, back in the early days it the GangliaSink shipped by default, and it was far better than trying to scrape every worker instance via JMX...
>
> And then yeah, we moved to ELK for metrics. Spent more time fixing up indexing and ingestion issues.
>
> Then I introduced Prometheus + Grafana, it was far simpler to run (at least until we got big enough to need Thanos...) and far simpler to create dashboards and write queries in.
>
> > llama052 11 months ago | root | parent | next [–]
> >
> > Oh man, Munin.. I had forgotten about that!
> >
> > We are currently running a databricks cluster and avoiding the static-ish ganglia screenshots and injecting prometheus exporters to get data into grafana. Honestly wish the entire data space was more open in that regard.
> >
> > Really glad things are evolving, it's not perfect but it's sure as hell better than it was!

Jedd 11 months ago | root | parent | prev | next [–]

Prometheus isn't Grafana's - so you can't hold Grafana responsible for PromQL either - but for scaling you want their Mimir product anyway.

What scaling issues are you suffering - limits, performance, costs?

Exploration is pretty good in Grafana - sounds like you're using dashboards to do discovery / exploration rather than their explore tool, which got substantially better in recent versions with their metrics encyclopaedia feature.

applied_heat 11 months ago | root | parent | prev | next [–]

What do you use for exploring? KST-plot has been my go to for real time monitoring and after the fact analysis of time series data. It is super fast, I can pan, I can zoom, with keyboard shortcuts, and I can perform functions on

the data to create new time series. Airbus is/was a big user of KST for their air frame test data. I haven't found anything that can replace it

smcleod 11 months ago | root | parent | prev | next [–]

Sumologic is one of the worst tools I've ever had to use for monitoring, its "ok" for looking at a pile of logs, not great but "ok", but when companies try to use it for monitoring - gosh, it's like going back to the dark ages.

eschneider 11 months ago | parent | prev | next [–]

All excellent points. I'd just like to add that when tackling monitoring, know exactly what problem(s) you're trying to solve. Monitoring-for-monitoring's-sake is a fool's game. Are you monitoring to get usage patters? Debug production problems? Troubleshoot for customer service? Know what actionable information you want/need from your monitoring solution and let that guide the approach.

emmelaich 11 months ago | parent | prev | next [–]

The answer is to have clean lines of separation between collecting, reporting, alerting, analytics.

For collecting, choose a compact unambiguous format for the data and never change it. Then you can change the other stuff without boiling the ocean.

lelanthran 11 months ago | root | parent | next [–]

> The answer is to have clean lines of separation between collecting, reporting, alerting, analytics.

What's the difference between reporting and analytics? I think I know but I'm not sure it matches what you mean. [1]

[1] I assume that reporting is for detecting trends, capacity planning, etc and analytics is for diagnosing issues.

emmelaich 11 months ago | root | parent | next [–]

Nothing really I suppose. Maybe reporting is periodic and analysis is ad-hoc?

lelanthran 11 months ago | root | parent | next [–]

After thinking about it and read what the various tools do, I decided that reporting is for ops purposes (capacity planning , etc) and analytics is for product purposes (engagement, downloads, signups, etc).

KaiserPro 11 months ago | prev | next [–]

The Way(tm) that I was taught/experienced is as follows:

o Logs are there for ignoring. You need them precisely twice: once when you are developing, and once when the thing's gone to shit, but they are never verbose enough when you need them.

o Using logs to derive metrics is an expensive fools errand pushed by splunk and the cloud equivalents(ie cloudwatch and the like). Its slow, inaccurate and horrendously expensive.

o using logs for monitoring is a fools errand. Its always too slow, and really really fucking brittle.

o metrics are king.

o pull model metrics is an antipattern

o Graphite + grafana is still actually quite good, although time resolution isn't there.

o You need to raid your metrics stores

o We had a bunch of metrics servers in a raid 1, which were then in a raid 0 for performance, all behind loadbalancers and DNS Cnames with a really low TTL.

o Cloudwatch metrics are utterly shite

o Cloudwatch is actually entirely shit.

o tracing is great, and brilliant for performance monitoring.

o Xray from AWS is good, but only really for lambdas.

o tracing is fragile and doesn't really plug and play end to end, unless you have the engineering discipline to enforce "the one true" tracing system *everywhere*

but what do you monitor?

http://widgetsandshit.com/teddziuba/2011/03/monitoring-theor... this still is canonical.

In short, everything should have a minimum set of graphs, CPU, Memory, connections, upstream service response times hits per second and query time, at a minimum.

You can then aggregate those metrics into a "service health" gauge, where you set a minimum level of service (ie no response time greater than 600ms, and no 5xx/4xx errors or similar) red == the service isn't performing within spec, yellow == its close to being outside spec, green == its inside spec.

if you are running a monolith, then each subsection needs to have a "gauge". for microservice people, every microservice. You can aggregate all those gauges into "business services" to make a dashboard that even CEOs can understand.

perlgeek 11 months ago | parent | next [–]

I think it hasn't really been settled if pushing or pulling metrics is the anti-pattern, it seems to change every 5 to 10 years which one is currently hot.

preseinger 11 months ago | root | parent | next [–]

yep

while they have different sets of pros and cons, neither is generally preferable to the other, they both get the job done with basically the same cost

valyala 11 months ago | root | parent | next [–]

There is no need to choose between push and pull, since both methods have their pros and cons [1]. Just use monitoring system, which supports both methods [2].

[1] https://docs.victoriametrics.com/keyConcepts.html#write-data

[2] VictoriaMetrics + vmagent

preseinger 11 months ago | root | parent | next [–]

nobody would use victoria metrics in earnest, i hope!

vsz 11 months ago | root | parent | next [–]

I use VM both at home and at work for 10s of millions of active time series and it's great.

It just runs and the default config is sustainable unlike some of the other solution.

Pull/push mostly doesn't matter other than config, it's the number of metrics and series of the prometheus ecosystem that's the real problem and being able to handle them without OOMing to 0 availability that the problem.

valyala 11 months ago | root | parent | prev | next [–]

Why?

dengolius 11 months ago | root | parent | next [–]

+1

betaby 11 months ago | parent | prev | next [–]

> o pull model metrics is an antipattern

And sadly somehow whole Prometheus/Cloud is built on idea of pulling GET /metrics I personally also think it's an antipattern, yet such design is dominant. Streaming telemetry via GRPC is rarity.

pphysch 11 months ago | root | parent | next [–]

Pulling/polling isn't suitable for high throughput (e.g. network flows, application profiling, any sub-second sampling frequency), but it's totally fine for 99% of observability use cases. In fact, I would argue pushed metrics are an anti-pattern for most environments, where the performance upsides are not worth the added complexity & reduced flexibility.

There is real value to the observability system being observable by default. It is so nice to be able to GET /metrics endpoint from curl and see real-time metrics in human-readable format.

Pull by default, and consciously upgrade to push if you need more throughput.

KaiserPro 11 months ago | root | parent | next [–]

I think the issue for me is that "pull" requires me to open up lots of services/hosts/sidecars to allow inwards connections. Thats a lot more things to monitor and test to see if its broken.

having a single dns record that I can route based on location/traffic/load/uptime autonomously is, I think, super convenient.

For example, if I want to have a single metrics config for a global service I can say: "server: metrics-host" and depending on the dns search path it'll either get the test, regional or global metrics server. (ie *.local,* .us-east-1 or *.company.tld)

However for most people its a single DNS record with a load balancer. When a host stops pushing metrics, you check the host aliveness score and alert.

llama052 11 months ago | root | parent | prev | next [–]

I'd still argue that it's easier to scale Pulling than it is a distributed push. It's kind of why prometheus went that route in the first place.

Back in the days of puppet or nagios, which would take requests and not pull. It was very common to hear about them cascading and causing huge denial of service issues and even causing massive outages because of it. For the simple fact that it's way harder to control thousands of servers sending out data on a timeframe versus a set of infrastructure designed to query them.

If I recall correctly facebook in the early days had a full on datacenter meltdown due to their puppet cluster pushing a bad release causing every host to check in, they were offline for a full day I think, they couldn't update the thousands of hosts because things were so saturated.

However in the case of polling, you'd dictate that from the monitoring servers themselves, you can control and dictate that without causing sprawls of outages and calls from everything.

Pull model can obviously scale(a lot of the shortcomings are now addressed in Thanos/Cortex): https://promcon.io/2016-berlin/talks/scaling-to-a-million-ma...

KaiserPro 11 months ago | root | parent | next [–]

> puppet cluster pushing a bad release causing every host to check in,

It was probably chef, but yeah I totally can see that happening.

in terms of scaling, nowadays everything either shards or can sit behind a load balancer, so partitioning is much more simple nowadays.

for network layout though, having hosts that can get access to a large number of machines is something I really don't like. Traditional monitoring, where you have an agent running as root and can execute a bunch of functions is also a massive security risk, and has largely moved to other forms of monitoring.

marcosdumay 11 months ago | root | parent | prev | next [–]

The most environments for what pushing is an anti-pattern (I do agree those are most) also should avoid complex monitoring tools, complex cloud architectures, and most of the troublemakers on the entire discussion here.

So, if you need to architect your metrics, the odds are much higher that you are one of the exceptions that also need to think about pulling or pushing them. (Or you are doing resume-driven architecture, and will ignore every advice here anyway.)

pnt12 11 months ago | parent | prev | next [–]

I don't know a lot about monitoring, why do you say pulling is an anti pattern?

My understanding is that with the push pattern you submit a metric when it's available, with the pull pattern you make them available via an interface.

I've read the first is not as performant, as it leads to submitting lots of metrics. Although I can think of an alternative, which is storing them and pushing batches periodically?

briffle 11 months ago | root | parent | next [–]

Pulling metrics from your service is easy and simple. Pulling metrics from thousands and thousands of services, spread out over many clouds/regions/environments, puts a HUGE strain on the single pulling server. Especially if it then evaluates its list of rules for each pull, etc.

But for pushing, if you don't get a message from service X in the last hour, is it down, or is it just not being used. So things like heartbeats intervals need to be configured, etc.

llama052 11 months ago | root | parent | next [–]

> Pulling metrics from your service is easy and simple. Pulling metrics from thousands and thousands of services, spread out over many clouds/regions/environments, puts a HUGE strain on the single pulling server. Especially if it then evaluates its list of rules for each pull, etc.

Oh god if you're trying to do that with a single pulling server you're doing something terribly wrong.

> But for pushing, if you don't get a message from service X in the last hour, is it down, or is it just not being used. So things like heartbeats intervals need to be configured, etc.

The inverse is true on this situation, if a server is attempting to accept traffic from thousands of services it's got the potential to get DOS'd pretty damn quick, and you can't even control that flow easily. At least from a pulling server you can dictate that on one host, or scale that out and aggregate in another layer. (Thanos/Cortex).

I'll always prefer pulling versus pushing because you gain more control and prevent yourself from blasting a server into space with a thousand calls it can't refuse.

marcosdumay 11 months ago | root | parent | prev | next [–]

Pulling implies you go storing the bare data, and run through the data when it's needed.

Pushing implies that you run through the data as it appears, and sends the results to the log aggregator for storage as soon as they exist.

Pulling has a lot of problems with storage and storage bandwidth.

baq 11 months ago | root | parent | next [–]

Pushing has the Byzantine generals problem where you have no idea if your data actually ended up anywhere and still need to pull to backfill.

marcosdumay 11 months ago | root | parent | next [–]

I'm not sure we are talking about the same thing, because I see no need at all to backfill monitoring data. In fact, it's one of my guidelines to decide if something is monitoring or logging; logs can not have holes.

You push as a best effort. It's up to the receiving party to react to a lack of data.

KaiserPro 11 months ago | root | parent | prev | next [–]

> Pushing has the Byzantine generals

not really.

the general's problem is about trust and authentication.

however, to your point about not being sure if the metrics ever get there, there is another way.

If you just have metrics, then its a single point of failure. so you need another basic "alive" check. for web services most people already have some sort of /__health or other check to allow loadbalancers to

work. marrying up the two sources of data allows you to work out if its the service, metrics or loadbalancer thats not working.

guhidalg 11 months ago | parent | prev | next [–]

This matches my experience in Azure. I'd add a few more:

o Log program inputs, outputs, and state changes. If you log something like "We are in function X", that's useless without a stack trace showing how we got to that state.

o Assume your developers are going to forgo logging "the right thing" and instrument the ** out of your application infrastructure (request/response processing, DB accesses, 3rd-party API calls, process lifecycle events, etc...).

o Make logging easy. I may get flamed but I think dependency injecting loggers is an anti-pattern. Logging is such a cross-cutting concern that its easier to have a static Logger or Metric object that is configured at startup. You do need some magic to prevent context from spilling from one request to another, but at least C# has these language features (AsyncLocal<T>) and I assume others do too.

o Your monitoring alert configuration should be in source control.

klysm 11 months ago | root | parent | next [–]

DI logging is continent to just add extra context to the logs but I agree on static Log being always accessible in the same way everywhere

mike22 11 months ago | root | parent | next [–]

In Java, Lombok @Slf4j on a class is very convenient. Adds a static field for the logger.

citrin_ru 11 months ago | parent | prev | next [–]

> Logs are there for ignoring. You need them precisely twice: once when you are developing, and once when the thing's gone to shit

good logs are very valuable both when things are still working and when there is an outage.

E. g. I with nginx even on a loaded servers error.log is small and it is possible to skim it from time to time to ensure there are no unexpected problems. access log is useful for ops tasks to - when possible I use tab separated access log which can be queried using clickhouse-local.

Sometimes software write bad (useless) logs, but this is not a problem with logs in general.

Also it may be useful to collect metrics for number of messages on different levels (info/warn/error/crit) - sudden changes in rate of errors is something which should be investigated.

kccqzy 11 months ago | parent | prev | next [–]

While I agree generally, I think it's useful to distinguish between freeform logs and structured logs. Freeform logs are those typical logs developers form by string concatenation. Structured logs have a schema, and are generally stored in a proper database (maybe even a SQL database to allow easy processing). Imagine that each request in a RPC system results in a row in Postgres. Those are very useful and you can derive metrics from them reasonably well.

kayodelycaon 11 months ago | root | parent | next [–]

Structured logging has saved my ass so many times. Make a call to an API? Log it. Receive a call to an API? Log it. Someone starts pointing fingers, pull out the logs.

I love to attach logs to other rows in the database: (tableName, rowID, logID, actionID?). Something goes wrong in the middle of the night? Here's all the rows affected.

Log retention is easy. Use foreign keys with cascade delete, removing a row from the Log table removes all the log data. If you need to keep a log, add a boolean flag. Need to be sure? Use a rule to block deletion if the flag is set. (You're using Postgres, right?)

dalyons 11 months ago | root | parent | next [–]

That’s all fine until you are running at reasonably large volume. Logging using an rdbms breaks down very quickly at scale, it’s very expensive tool for the job.

valyala 11 months ago | root | parent | next [–]

Agreed that storing structured logs into a relational database can be very expensive. But you can store structured logs into analytical databases such as ClickHouse. When properly configured, it may efficiently store and query trillions of log entries per node. Google for Cloudflare and Uber cases for storing logs in ClickHouse.

There are also specialized databases for structured logs, which efficiently index all the fields for all the ingested logs, and then allow fast full-text search over all the ingested data. For example, VictoriaLogs [1] is built on top of architecture ideas from ClickHouse for achieving high performance and high compression rate.

[1] https://docs.victoriametrics.com/VictoriaLogs/

dalyons 11 months ago | root | parent | next [–]

sure. but thats not what the OP was saying. A pipeline of evented/ETLd/UDPd logs to an analytical db like clickhouse is a fairly standard and reasonable thing to do at scale. Not putting it in postgres.

lelanthran 11 months ago | root | parent | prev | next [–]

Why does it break down?

Too many row insertions? Too many dB clients at the same time?

I'm genuinely curious here.

klysm 11 months ago | parent | prev | next [–]

I strongly disagree with the link you’ve provided saying that queue bases systems are hard to measure the health of. I think the queue size is a good way to monitor the health of things, assuming you have some not super pathological workload. If your queues have a bounded size then alerting at various points towards that upper bound makes perfect sense

jon-wood 11 months ago | root | parent | next [–]

I’ve found alerting on queue length to be a path to false alarms, what most people care about is queue latency. How long does a message sit in the queue before being processed?

For some jobs the upper bound for that can be measured in days, for others it’s milliseconds, either way no one on the business side cares whether you have zero or a million jobs on the queue so long as they leave the queue

quickly enough.

klysm 11 months ago | root | parent | next [–]

Good point, it depends on the dynamics of the queue but that’s also a good thing to measure.

kbar13 11 months ago | parent | prev | next [–]

i agree with your statement on logs and i see where you're coming from with cloudwatch. i think if you're coming from graphite and grafana then it makes total sense why you would dislike cloudwatch. cloudwatch requires a bit more planning than the intuitive metrics you can typically throw at and query from graphite. it also definitely does feel weird to have to ship multiple dimensions of seemingly the same metric to do the queries that you really want. however, once you design the rollups of metrics, you get everything you need, plus you don't have to worry about operating a mission-critical monitoring stack. and you can still build your dashboards in grafana and alert as you like.

i've never really found tracing to be useful - i've used pprof and an internally developed version which uploads profiles from production instances and shows you a heatmap for stuff that require deeper digging than metrics

kunley 11 months ago | parent | prev | next [–]

Two big minuses of Graphite:

- not scalable, even at a moderately sized level; at one work we had graphite workers trying to flush-write the data for dozens of hours on a regular basis, and numerous engineers spent a lot of time trying to optimize that with no good outcome

- no labels, so you must flatten them into a metric name

KaiserPro 11 months ago | root | parent | next [–]

> not scalable

Yeah sharding it is a pain. We got its to a million active metrics (ie a metric that was updated in the last 5 mins) on a single instance, with the wisper backend.

I suspect the newer DB based backend are better scalability wise.

> no labels, so you must flatten them into a metric name

I _personally_ don't like labels, I believe that metrics should be one dimensional. I know that puts me at odds with others. I can see the appeal as it allows you too easier group instance metrics into service level ones.

deepsun 11 months ago | root | parent | next [–]

Check victoriametrics, they got Graphite query language and ingestion protocol, but on top of their storage.

That thing can scale to a billion of active metrics, not just million.

mdaniel 11 months ago | root | parent | next [–]

And a new(?) logging ingestion thing: https://news.ycombinator.com/item?id=36429526
https://docs.victoriametrics.com/VictoriaLogs/
https://github.com/VictoriaMetrics/VictoriaMetrics/releases/... (Apache 2)

morelisp 11 months ago | root | parent | prev | next [–]

For some reason junior devs / SREs always go nuts putting things into labels, I guess it's the same insecurity and/or brain rot of only-one-class-per-file or three-package deep source hierarchies for a total 1k line project.

The rule of thumb I give them is that if it doesn't make dimensional sense to average across across them, never use a label.

IsopropylMalbec 11 months ago | root | parent | next [–]

If a service which provides metrics supports multiple tenants wouldn't those metrics at the very least need a label to indicate which tenant the metrics are for?

morelisp 11 months ago | root | parent | next [–]

Why doesn’t it make dimensional sense to average across tenants?

VectorLock 11 months ago | parent | prev | next [–]

>pull model metrics is an antipattern

While I agree there are a lot of Prometheus people who would disagree.

brad0 11 months ago | parent | prev | next [–]

Could you elaborate on Cloudwatch metrics?

KaiserPro 11 months ago | root | parent | next [–]

A few things that I dislike:

The temporal resolution is unpredictable (as in the reduction of resolution with age is separate from the view resolution.)

Its slow as shit as in it (used) to take a good while to update with this time bucket's value, and that value seems to be able to change wildly (more than just an average)

The min/max/average/count/$other parameters are poorly documented and inconsistently applied across services. therefore what chance do you have for implementing them properly.

It's expensive(well can be)

deletion is different from hiding. its less of a problem with grafana/custom renderers, but its nasty.

the lack of functions for slicing/dicing and combining annoy me. (again that might have changed in the last year or so)

trabant00 11 months ago | prev | next [–]

Good luck trying to monitor using debugging tools like logs and metrics. Devs thinking they can do Ops because reasons. What you really need is state monitoring. You know, the old Nagios style that everybody thinks is out of date because, again, reasons. Just start by having e2e tests for everything and add intermediate tests when you discover points of failure and bottlenecks. And don't give me that "Nagios does not scale" bullshit. You don't need to and can't monitor 9 trillion things. You also have Icinga and other options that scale for state monitoring.

In place of argumentation which would be too long, let me give you an analogy. Monitoring is telling you the fire alarm went off. The exact reason is left for a investigation and that's where saving data like logs and metrics helps. But first you need to evacuate the building and call the fire department. If instead of detecting the smoke (e2e) you try to monitor all the data like

pressure in gas pipes, all the pipes fittings, how many lighters are in the building and where, etc, you will wake up every night 10 times for nothing and when a fire starts you will burn sound asleep.

Oh, and don't try to scale Prometheus, that's what things like victoriametrics are for.

wspeirs 11 months ago | parent | next [–]

>If instead of detecting the smoke (e2e) you try to monitor all the data like pressure in gas pipes, all the pipes fittings, how many lighters are in the building and where, etc, you will wake up every night 10 times for nothing

Without this you'll always be awoken by smoke, and at that point it's too late... there's already a fire. However, if you can monitor other things (gas pipes, lighters, etc), you might be able to remediate a problem *before* it starts smoking or burning.

There's no one-size-fits-all. Start with some obvious failure (e2e tests/checks; aka smoke), and when you root-cause, add in additional checks to hopefully catch things *before* the smoke. However, this also requires you update (or remove) these additional checks as your software and infra change... that's what most people forget, and then alert fatigue sets in, and everything becomes a non-alert.

trabant00 11 months ago | root | parent | next [–]

> this also requires you update (or remove) these additional checks as your software and infra change... that's what most people forget, and then alert fatigue sets in, and everything becomes a non-alert

This is what I forgot to mention, monitoring is a process, not a tool you put in place and forget about. And I agree that this is the main cause monitoring goes to shit. But if you think about it a little deeper you arrive at the requirement to have monitoring done by somebody who understands the business and the infra/software stack and has been at that company a few years at least. While in reality monitoring is mostly an afterthought cost center and employees are rotated far too often. It's not a glamorous position either, I don't think I once met somebody who wants to work there.

teddyh 11 months ago | root | parent | prev | next [–]

> *when you root-cause, add in additional checks to hopefully catch things before the smoke.*

No, what you should do is *update the software* to *handle* or *avoid* the problem gracefully. Keep the smoke detectors, but *don't* keep adding tiny things to monitor and keep in sync with your ever-changing software.

trabant00 11 months ago | root | parent | next [–]

That can't always be done. CAP theorem is still true for example.

pphysch 11 months ago | parent | prev | next [–]

> What you really need is state monitoring. You know, the old Nagios style that everybody thinks is out of date because, again, reasons. Just start by having e2e tests for everything and add intermediate tests when you discover points of failure and bottlenecks.

"State monitoring" are just different metrics. State of X at time Y was Z. Blackbox Exporter covers a lot of ground here. If you need more advanced application instrumentation, you can build that, but there is real value in having all of your metrics in one place instead of siloing different bits out to Nagios.

trabant00 11 months ago | root | parent | next [–]

> "State monitoring" are just different metrics

True but also misleading. In the end everything is really math but that is unhelpful. What matters is the model you apply to your numbers. And state monitoring as thought by Nagios is that you have 4 states: ok, warning, critical, unknown. And you have hard and soft states. This can be replicated with prometheus queries but it's incredibly complex and error prone, not to mention you will destroy your metrics storage IO.

Theory crafting aside I've created in a few months a monitoring system using Icinga that had virtually no false positives or negatives. That you could trust to phone you straight though without fear it will miss something important for business or wake you up for nothing. I never saw anything like this done with metrics or logs done by anybody.

pphysch 11 months ago | root | parent | next [–]

> I never saw anything like this done with metrics or logs done by anybody.

Well, I assure you it can be done and it is not nearly as scary as you presume. I migrated an old Nagios+Cacti system to Prometheus (well, VictoriaMetrics) for a DC with thousands of servers.

Nagios using 4-valued datatypes for virtually EVERYTHING is frankly just poor design. As soon as you move away from binary state, the semantics becomes ambiguous.

Consider these two monitoring systems:

- Nagios-style: 1+ state metric for your HTTP service. "Warning" indicates something wrong but not critical. What exactly? Usually you will see the warning and then look at metrics in sysstat or Cacti as a follow up.

- Prometheus-style: 1+ state metric for HTTP probe, and a few relevant performance metrics. RPS, latency, etc.

The merit of the latter is you use the exact same alerting + visualization infrastructure for both types of data. You create a CRITICAL alert if the service is down, and WARNING alerts if RPS/latency crosses thresholds.

Importantly, this data is not siloed into Nagios and can be used for other purposes such as usage analytics and cross-domain incident analysis. The alerting service is just one possible consumer.

trabant00 11 months ago | root | parent | next [–]

And had the final result of the migration completely automated alerts? Or did you have to employ round the clock L1s to weed through the alerts?

pphysch 11 months ago | root | parent | next [–]

Of course the alerts work. Alertmanager can do it all.

Meanwhile, a lot of the admins had routed Nagios noise to junk inboxes.

trabant00 11 months ago | root | parent | next [–]

That's not what I'm asking. Of course Alertmanager works. Did you get the system to the point where all the false positives are eliminated and you don't miss any failure either, such that you don't need L1 to stay in 3 shifts in front of a couple of monitors and decide when to call Ops? Meaning you can have your alerting system automatically and directly call L3.

pphysch 11 months ago | root | parent | next [–]

This discussion is getting pretty far beyond the scope of Nagios vs. Prometheus.

Yes, alertmanager gives you full control over the routing and deduplication of alerts.

trabant00 11 months ago | root | parent | next [–]

I am not asking about the tech. I know it. I did VictoriaMetrics exclusively for 2 years. I submitted bug fixes to them. I wrote a metrics duplicator loadbalancer with local buffer which stood before vminsert to get HA without paying VM for license. I was ingesting 400Mbits per second of metrics from 20 teams, hundreds of apps.

I am asking if you got the queries good enough to alert L3 directly.

But I'm not really asking, and you did not really miss understood my questions either. We both know the answer.

dengolius 11 months ago | root | parent | next [–]

Could you please elaborate on "I wrote a metrics duplicator loadbalancer with local buffer which stood before vminsert to get HA without paying VM for license. I was ingesting 400Mbits per second of metrics from 20 teams, hundreds of apps." ? Why you need to write own duplicator while vmagent can achive such case?

trabant00 11 months ago | root | parent | next [–]

That was added in https://github.com/VictoriaMetrics/VictoriaMetrics/issues/14... I did it before this, when replication between clusters was not available (it was in Enterprise if I remember correctly).

dengolius 11 months ago | root | parent | next [–]

Ok, I see.

preseinger 11 months ago | root | parent | prev | next [–]

icinga? nagios?

you're stuck in 1999 friend

these tools, and the patterns they enable, have been outmoded for over a decade

trabant00 11 months ago | root | parent | next [–]

Your arguments won me over

buro9 11 months ago | prev | next [–]

Monitoring all the things can be hard.

There are a lot of different solutions, that have different trade-offs for different volumes of data, cardinality of data, how out of order / async the data may arrive, whether you need push or pull delivery, etc, etc.

These apply to you and only you know the parameters and considerations of your application, and the environment it runs in (perhaps you come with the baggage of "this is instrumented in x and I can't change that" in addition to volume and cardinality considerations, perhaps it's a point-of-sale in a shop that only connects to the internet once a day for upload of the data).

But for the simplest solution, you're probably talking Prometheus + Loki/Elastic + Tempo + Pyroscope/Polar Signals + Grafana. This is simple as in "all pieces are proven and work in the small and get you fairly far"... but you still have to compose the solution out of those parts. And later you may wish to graduate to Grafana Cloud when the pain of running it all yourself is taking more time and effort than running the systems you're supposed to be working on. Oh, and if you really want to lean in on traces for everything, take look at Honeycomb too.

As to "the monitoring costs more than the app"... yup, at Cloudflare for a very very long time there was little monitoring of DNS requests or billing of it... because the act of counting it and recording it cost more than just serving the request. This principle scales well (or badly if you're looking at your credit card bill)... and is also why those who make these systems really have to keep in mind how to make the instrumentation cost less (eBPF?) as well as the ingestion and retention cost less (various methods, including aggregation, sampling, etc).

> hinkley 11 months ago | parent | next [–]
>
> My company does a lot of things the dumbest way possible but they’re always been good at tracing service requests and telemetry. Any time correlation IDs were missing was considered a regression. And we are quick to agree to action items that increase visibility from a postmortem. The one complaint I have is we don’t move things out of Splunk into graphs as fast as I’m comfortable with.
>
> But the thing I see we are all missing for telemetry is a sense of coverage. We changed backends last year and as we were wrapping that up I found big chunks of code that were missing coverage. Meanwhile we found a bunch of stats that nobody looks at, meaning the SNR goes down over time. There are only so many dashboards I can watch.
>
> So some sense of code blocks missing telemetry, and data that is collected but has no queries, queries that exist but never run (that dashboard nobody remembers making and isn’t in the runbooks anywhere) would be a great goal for a next level of maturity.

awinter-py 11 months ago | prev | next [–]

Strong yes, particularly this line:

> the cost of monitoring an application can easily exceed the cost of hosting the application even for simple applications

(especially if you are looking at saas monitoring for a single-box or serverless thing)

I suspect SIEM is the most established use case for ingesting logs to something actionable; wonder if there are concepts there which port generally.

Datomic's approach to change capture seemed very good when it came out -- I wish this were a thing I could cheaply turn on and off on SQL columns. As things are, every ORM seems to resist having a table with no primary key so 'log important things to the DB' is hard in practice.

In postgres, mvcc makes it expensive to edit a scalar, so it is impractical to store counts. (There is an old uber blogpost I think about why they use mysql for this use case).

Specifically for alerting, I wish I could define application logic statistics in my application rather than PromQL / datadog json. Would be much easier to express 'warn me if this ever fails', 'expect 50% failure', 'warn me if too many retries' type stuff.

On the 'looking at logs' side, the hosted monitoring tools I've used are often uncomfortably slow, even when my data is small. I understand it's a hard problem but I miss graphite / whisper.

(not even getting into mixpanel / posthog type product-level observability)

mst 11 months ago | parent | next [–]

> every ORM seems to resist having a table with no primary key

perl's DBIx::Class doesn't honestly give a shit.

I understand there are many perfectly understandable reasons why people dislike perl itself, but it's really annoying to use ORMs in more popular languages and keep finding things perl developers have been able to take for granted for 15+ years that Just. Aren't. There.

I'm not asking anybody to like perl who doesn't already, it's definitely an acquired taste, but I think it's understandable for me to wish people developing libraries for other languages would get around to stealing some of DBIx::Class' less commonly available features (I may eventually get around to doing so myself but I strongly suspect that for any given language there are quite a few people out there who could do a better job of it, faster, were they to decide it sounded like sufficient fun to spend time on it, so unless/until -I- decide it sounds like sufficient fun I'm going to keep mentioning it in the hopes somebody else does ;).

rjbwork 11 months ago | parent | prev | next [–]

>I wish this were a thing I could cheaply turn on and off on SQL columns.

You can turn it on and off for whole tables in systems that have decided to implement SQL:2011 such as MSSQL, Oracle, MariaDB, etc. and then issue time travel queries to see what the state of the database was at any given point in time.

Keyword here is "Temporal Table".

awinter-py 11 months ago | root | parent | next [–]

did not know about this -- thank you for the rec

lukeasrodgers 11 months ago | parent | prev | next [–]

What do you mean "expensive to edit a scalar"? If we're thinking of the same Uber article on mysql -> postgres, IIRC it had to do with write amplification related to updates to indexed columns. I don't think postgres has any problems storing counts, but would be interested to know if I'm wrong.

awinter-py 11 months ago | root | parent | next [–]

this article https://www.uber.com/blog/postgres-to-mysql-migration/

you're right -- I misunderstood. I thought mysql could do in-place writes to rows, saving mvcc / vacuum work

let me amend my point to: a system designed for storing counters can optimize for that case, while sql DBs will do relatively more IO to incremental a counter

j3s 11 months ago | parent | prev | next [–]

> I miss graphite / whisper.

as someone who had to manage a system like this, i certainly don't. it doesn't scale well, it requires way too much upfront design thought (since metrics are stored in arbitrary dirs), and it's a massive pain to install and manage. not to mention,

it's barely developed anymore so much needed features (like carbon cache restarts not taking a million years bc of the slow sequential flush to disk) will probably never see the light of day.

for better or worse, the industry moved on. speaking personally, prometheus is so much better than graphite ever was, in every conceivable way.

valyala 11 months ago | root | parent | next [–]

There are Graphite-compatible monitoring systems, which use much more efficient storage format than Whisper. They resolve the most annoying issues associated with Graphite for large-scale setups - high disk IO and high disk space usage. See [1] and [2].

[1] https://github.com/go-graphite/carbon-clickhouse

[2] https://docs.victoriametrics.com/#graphite-api-usage

roflyear 11 months ago | parent | prev | next [–]

Well, "easily" - I am not totally sure, but it can happen without much trouble. I guess depends on your definition of easily.

jeffbee 11 months ago | prev | next [–]

One thing I like to evangelize is the similarity between logging and tracing. A trace could be assembled from properly-formatted distributed log messages, and some tracing systems are implemented this way. If you have a working tracing setup, and you trace every request as some people do, then using a span event [1] can be an easier way to get information about what happened during the request than it would be to find all the associated debug logs.

1: https://opentelemetry.io/docs/concepts/signals/traces/#span-... A span event is data that is associated with a point in time rather than an interval.

phillipcarter 11 months ago | prev | next [–]

> The problem with me and tracing is nobody uses it. When I monitor the teams usage of traces, it is always a small fraction of the development team that ever logs in to use them. I don't know why the tool hasn't gotten more popularity among developers.

I think there's two primary reasons:

1. Historically it was too hard to get easy value early. That's probably changed with OpenTelemetry since it has so much automatic instrumentation you can install. Even 3 years ago it just wasn't feasible to install an agent or write 5 lines of code and then get most of the libraries you use instrumented for you. Now it is, and it's fully portable across tools.

2. Cost, which still isn't solved yet. Tail-based sampling is still too hard, but you need it if you want to do stuff like always capture 100% of error traces but get only a statistically representative sample of 200 OK traces. There's some solutions for that today (e.g., my employer Honeycomb has such a tool), but it's still locked away with vendors and too hard to use.

I do expect the narrative on tracing to change in the coming years, though. OpenTelemetry is growing in popularity and it has enough features and stability for a wide range of people to adopt it. I think the cost issue will also be dealt with over time.

wspeirs 11 months ago | parent | next [–]

>Cost, which still isn't solved yet.

I'd argue it _is_ solved... store your logs in S3. At ~$0.02/GB you can store a _lot_ of logs for like $20. The problem is that _most_ solutions (Honeycomb included) are SaaS-based solutions, and so they have to charge a margin on top of whatever provider is charging them.

You just need a tool (like log-store.com) that can search through logs when they're stored in S3!

chimeracoder 11 months ago | root | parent | next [–]

> You just need a tool (like log-store.com) that can search through logs when they're stored in S3!

As someone who has run logging systems professionally, "just" is doing a *lot* of work in that sentence.

phillipcarter 11 months ago | root | parent | prev | next [–]

At the small volume of data where you could "just put stuff in S3 and search it later", the cost of tracing with a SaaS vendor is also very small. Far smaller than the developer time needed to set up something else.

OldManRyan 11 months ago | root | parent | next [–]

Depends on how much you value not being vendor locked in. I'll spend a week of dev time in order to save a month or two in migration pain later down the line when my vendor jacks up cost. See Datadog as an example.

phillipcarter 11 months ago | root | parent | next [–]

You can have your cake and eat it too with OpenTelemetry (see my original post). I don't see how that has to do with the cost of tracing being a factor in inhibiting its adoption though? I agree that vendor neutral instrumentation is worth it.

OldManRyan 11 months ago | root | parent | next [–]

> ...the cost of tracing with a SaaS vendor is also very small. Far smaller than the developer time needed to set up something else.

I'm mainly responding to this point regarding developer time outweighing the cost of just going with SaaS vendor that provides that capability. I don't think we disagree with each other though

wspeirs 11 months ago | root | parent | prev | next [–]

>At the small volume of data where you could "just put stuff in S3 and search it later"

I was arguing exactly the opposite... leverage S3's near-infinite storage so you can store whatever you'd like. Searching through it _can_ be fast with the right tool.

poulsbohemian 11 months ago | root | parent | prev | next [–]

I spent a lot of years traveling around the world fixing problem for major companies... logs aren't enough. Not only do you have the issue of auditors who are scared you might put things in logs that shouldn't be there, but there are security / access problems and the very basic problem of "did we log the thing we now need?" Not to say that logs can't be helpful, but they generally aren't enough on their own.

roflyear 11 months ago | root | parent | prev | next [–]

Depends on what you're logging. If you're logging *everything* it can add up.

doctorpangloss 11 months ago | parent | prev | next [–]

If I'm going to pay for a vendor APM, who's going to outpace NewRelic? They already do all this stuff - automatic instrumenting, "tail based sampling," etc.

phillipcarter 11 months ago | root | parent | next [–]

In some cased OpenTelemetry already has just as much or more support as the vendor-specific agents.

But really the play here is future-proofing. A lot of observability vendors are in the business of offering steep discounts in year one, locking you in with even more proprietary tech, and increasing your bill. When your instrumentation is done with OpenTelemetry, it takes 15 minutes to start sending data to a competitor.

As for sampling, it's still not great with New Relic. Their proprietary system lets you sample based on span attributes and combine that with a random sampling. But they don't dynamically upsample or downsample based on arbitrary tags in the data, density of traffic, and a target sampling rate or throughput. They don't let you define rules and combine them with a dynamic system or something else. All of this is necessary once you have a large volume of trace data and don't want to break the bank to observe it. They don't track the sampling rate and re-weight counts on the backend. At lower or medium-volume traffic what they have suffices (as does the OSS solutions), though.

This isn't a knock on New Relic -- it's a similar story pretty much anymore. Cost management at any level just isn't there yet for tracing. The building blocks are there and I expect it to be fixed, but it does prevent widespread adoption.

> doctorpangloss 11 months ago | root | parent | next [–]
>
> > All of this is necessary once you have a large volume of trace data and don't want to break the bank to observe it
>
> In my opinion, there are two approaches:
>
> - I try to do this monitoring as cheaply as possible, which means 100% free software and on premises hardware / minimally used cloud resources. No vendors.
>
> - I pay a vendor.
>
> And nothing else.
>
> From my experience as a SaaS vendor, evaluating paid monitoring options (including NewRelic and Honeycomb) and having used them in the past: Anything more than $0 that doesn't increase the bottom line by a greater asymptotic value is "expensive."
>
> When I worked in an organization that is trying to find (1) a marginally cheaper way to do monitoring between two vendors, and therefore (2) predict the future, like how much their usage will be or how much they will grow... the most expensive part of the whole process is all the people working at the organization. That number never seems to go down in response to the choice you make between vendors, it only seems to go down in response to interest rates or catastrophic business externalities. So in my opinion, it's sort of moot which vendor you choose, if you have already crossed the Rubicon to paying vendors.
>
> So in an intellectually honest way, no matter how you slice it, NewRelic is "just as expensive" as everything else.
>
> > A lot of observability vendors are in the business of offering steep discounts in year one, locking you in with even more proprietary tech, and increasing your bill.
>
> The hard, intellectually honest, no-bullshit truth of the matter is that the SaaS vendor's job with pricing is turning whatever the customer likes to hear (upfront invoices and discounts, royalties, per use pricing,

whatever) into whatever the investors / owners like to hear (90% margins, hockey stick revenue growth, hockey stick user growth, you can take a long vacation, etc.) via deal term nonsense.

For me, for example, I decide I want to make a certain amount of money, and then I turn it into one set of nonsense for customers (a premium fee or up front payment, set up to be economically equivalent) and another set of nonsense for investors (a 90% margin, a hockey stick growth).

You might be thinking: this is totally fringe and unorthodox. Listen, go read the Sony emails hack, and see how Michael Lynton deals with this obvious pricing reality. They are a huge, expertly run organization, and all their deals with people were constantly about making up some deal term bullshit the counterparty likes, and then Sony's expert staff of financial modelers gaming out the deal terms to determine the real economic value important to Sony. You don't have to operate in this vacuum as an IC anymore, you can peer *directly* into how executives *actually* work at giant corporations, if you dare, and see the realities for yourself.

Call me jaded, but having done this long enough, there is no story simpler than "the price is whatever the market will bear, and we'll tell you whatever you want to hear that maximizes it." So the most interesting price point is zero, which is not at all an unorthodox opinion, and indeed, many of the very most disruptive entrepreneurs got to where they were going by valuing their time at zero and rolling out metrics themselves.

lmm 11 months ago | root | parent | prev | next [–]

I found AppDynamics actually did that stuff better than NewRelic, but agree with the general point.

prpl 11 months ago | parent | prev | next [–]

Another thing - Traces (and often metrics for that matter) are often treated as something that is investigated one at a time. There's a lot of tools out there to help visualize a trace. They are not, usually, treated as something that should be analyzed at scale, and the tooling there is not great.

I think PromScale looked promising for some of those problems but it's also a deprecated product.

I think the state of the art is/will be ingesting traces into Iceberg for some form of analysis with whatever execution engine you want (Trino/Spark/etc...). A lot of places are doing this.

OpenTelemetry is clearly the future of monitoring, probably with a lot of lifting from Prometheus and other things.

zug_zug 11 months ago | root | parent | next [–]

So what I want is something like Splunk search for traces. I want to be able to write 12-shareable lines of custom-query and immediately get a pretty graph of multidimensional data "What % of users who just registered this month got a 500 more than half the time on this specific path?" and I want it as a pretty graph over time

phillipcarter 11 months ago | root | parent | next [–]

That's precisely what tools like Honeycomb and Lightstep offer. But most vendors don't have a traces-first approach to querying data.

zug_zug 11 months ago | root | parent | next [–]

Really? I haven't used honeycomb specifically, but you're saying it supports an arbitrarily complex query depth, with any number of joins essentially?

8note 11 months ago | root | parent | prev | next [–]

I kinda hacked x-ray onto things, but it does give you an overview aggregate for each node

I'm not sure if I've *really* found it useful though. Im sure if I didn't know the service, itd be great, but I already know where to look for whatever problem

fbdab103 11 months ago | parent | prev | next [–]

>When I monitor the teams usage of traces, it is always a small fraction of the development team that ever logs in to use them...

Like fire extinguishers, they are nice to have when you need it.

rsync 11 months ago | prev | next [–]

Higher value needs to be placed on "feel" and the ineffable heuristics that human brains automatically build up as they operate a system.

This is one of the reasons I strongly dislike the idea of remote or telepresence inspections of factories. You need to just walk the factory floor and use all of your senses (including smell). If you do this enough, *you'll know* when something is wrong. The pitch of the machines, the vibrations, the hum ... and you'll catch it before your rules do.

A personal example:

rsync.net has almost zero fraud detection for payments and signups. However, *I, personally*, have been watching the real-time order flow[1] for decades *and I just know*. A fraudulent signup just sticks out like a flashing red light. I could not teach this to you and I could not codify my heuristics.

I have spent decades thinking about how to augment - or replace - these real-time human observations with warnings and alerts and thresholds and I am *deeply skeptical* of their value relative to human interaction with the functioning system.

[1] ... with tail ...

verve_rat 11 months ago | parent | next [–]

How do you detect fraud when you are asleep?

rsync 11 months ago | root | parent | next [–]

We don't auto-create accounts - we do them in batches throughout the day.

I am not typically involved in that but I *do* see the order flow - usually early in the morning.

I can usually spot them before they are created *or* if they have been created already we just disable it and make some effort to notify the cardholder that something fishy is going on (and ask them to accept our refund rather than issue an annoying chargeback which only complicates the refund process).

This process doesn't scale. That's fine with me and everyone here.

giovannibonetti 11 months ago | prev | next [–]

When it comes to monitoring, the first thing I look for is a good data store. In the open-source world, probably the best one is Clickhouse, and there are multiple solutions built on top of it like SigNoz [1] and Uptrace [2].

[1] https://github.com/SigNoz/signoz [2] https://github.com/uptrace/uptrace

serverlessmom 11 months ago | parent | next [–]

Definitely +1 for SigNoz, much more cohesive view of metrics, logs, and traces

lrem 11 months ago | prev | next [–]

I share the love of tracing. Internally at Google I sometimes stare at traces traversing hundreds of RPCs. It's about the only tool where I can very quickly go "wait, this shouldn't be the costly part" and "ugh, why are these sequential", then look exactly what the particular requests were and where the QoS was set wrong.

XorNot 11 months ago | prev | next [–]

The rule I follow personally for log levels is:

DEBUG - anything which would otherwise be a 1 line code comment

INFO - all business process relevant actions. Anytime something is committed to a database is a fairly good place to start.

WARN - abnormal outcomes which collectively might indicate a problem. i.e. a warning log should be printed when something succeeds but takes much longer then it should, or if I'm going to fail an action which will be retried later. Anything which logs at this level should be suppressible by reconfiguring the application to define the behavior as normal.

ERROR - Anything which should work, didn't, and isn't recoverable. Actions requested by the user which don't succeed should log at this level, since we're failing to meet a user request.

twic 11 months ago | parent | next [–]

Our logging is tied to our monitoring system. So for us, it is:

DEBUG - throw it away

INFO - put it in a log file

WARNING - put it on a dashboard

ERROR - put it in PagerDuty

This does make it very easy to decide on the level. Do i want to be woken up at five in the morning for this? No? Then it's definitely not ERROR!

marcosdumay 11 months ago | root | parent | next [–]

Well, then I'd say ERROR is a very bad name. Because it's a well defined word with a precise meaning. If you tell somebody "an error happened here" they will have no trouble understanding it, and what they understand won't be "we need to wake up somebody to deal with this NOW!".

Those names are proof that your logging structure was created for debugging, not for production.

twic 11 months ago | root | parent | next [–]

I'm afraid this is an entirely fantastical objection. I don't believe we have ever had the confusion you describe, and this structure was created entirely for production use.

Supermancho 11 months ago | root | parent | prev | next [–]

> Well, then I'd say ERROR is a very bad name.

The name is irrelevant. The issue being addressed is a poor assertion. Log levels are not meaningless in a standardized system. It's trivial to map a set of levels to a useful set of conditions. Like any monitoring (or

logging), the system has to be well-regulated and maintained to be useful. In good faith, log levels are useful.

fatnoah 11 months ago | parent | prev | next [–]

That's pretty close to my definition as well, though I add FATAL as well, but mostly for extreme cases of "there's no way the system can function" type of errors, such as missing/invalid configuration or similar.

My philosophy is that INFO is the default runtime configuration. At smallish scales (< 1B requests/month) I've seen this approach work very well, especially when the Eng team is on board with high-quality log messaging. It was an investment to get to that state at my last startup, but the massive improvement to QoL for Eng and Ops teams was totally worth it.

kerblang 11 months ago | parent | prev | next [–]

Not sure the descriptions are ideal here, but the 4 levels selected are completely adequate for most purposes in business information systems. If someone really wanted, you could cut it down to INFO & ERROR - I secretly think of DEBUG & WARN as tolerable fluff that keeps fussy people satisfied that things are complicated enough without making a mess.

Per the article's concerns, no, I would not try to use this for auditing, metrics, or anything like that - just as a tool for devs to find out "OK what the heck is going on here?"

I just collect all my logs on a big, fast file server where I can grep the crap out of them, with way more success than I've ever had with Fancy Stack Of Whiz Bang Thingies W/ Web Interface, although I'm always willing to watch someone attempt another Fancy Stack Fail...

I recommend pressuring devs to make log messages coherent with good spelling and grammar - like anything else, it's "Write to be read," i.e. treat your reader with some respect, it's not that hard. Logs often end up thrown in the trash by sysadmins because they look like trash. It's especially hard to convince them that the logs are good enough that they can diagnose problems themselves instead of covering their eyes and asking me to look at the logs for them, and I don't entirely blame them.

doctorpangloss 11 months ago | prev | next [–]

> You can now track all sorts of things across the stack and compare things like "how successful was a marketing campaign". "Hey we need to know if Big Customer suddenly gets 5xxs on their API integration so we can tell their account manager." "Can you tell us if a customer stops using the platform so we know to reach out to them with a discount code?" These are all requests I've gotten and so many more, at multiple jobs.

But now you can ask ChatGPT GPT4 to author PromQL and Jaeger queries for you, which it is incredibly good at.

The real, actual pain point - not cost, not maintenance, etc. - between the bottom line affecting stuff like customer success management and metrics was always turning the English language problem into the thing that goes into the computer. And that is seriously going away, the only people who think it's not are the experienced ones who, for some reason, months into the distribution of this insane technology, haven't tried asking for a PromQL query yet!

gofreddygo 11 months ago | parent | next [–]

> the English language problem

This is so true, and IMO underlooked for other flashy capabilities. GPT transformers (even bad ones) have nailed english grammar. This opens up a lot of doors for a lot of small businesses.

SoftTalker 11 months ago | prev | next [–]

I've outsourced monitoring to my users. They let me know when something isn't working.

awestroke 11 months ago | parent | next [–]

Your increased churn rate is probably cheaper than a Datadog plan

politelemon 11 months ago | root | parent | next [–]

I'm hearing way too many stories very recently about Datadog's extortionate pricing. I wonder if there's something going on and if they're on an aggressive campaign.

mikeshi42 11 months ago | root | parent | next [–]

It's been happening for a while now - I assume it's more in the news now as companies are starting to tighten budgets for the first time in a long time.

We've [1] been taking advantage of it - counter-positioning against DD by offering low cost and really straightforward pricing while NR and DD both have chosen non-sensical (to me and many others) pricing models.

[1] https://www.hyperdx.io

knorker 11 months ago | parent | prev | next [–]

Crowd sourced consensus driven monitoring based on multiple instances of general AI.

spentu 11 months ago | parent | prev | next [–]

I also have done this for some projects. Sadly they seem to be unreliable to degree of using alerts with ms Teams..

hagen1778 11 months ago | parent | prev | next [–]

How do you silence noisy alerts?

ckozlowski 11 months ago | prev | next [–]

This was a good read. But the question that I kept waiting to be asked was "What do I need to monitor?"

In my opinion, the way to avoid many of the problems and complexities that author lists is to start with the goals. What are the business/mission goals? What is the workload (not the monitoring tool) trying to accomplish? What does the thing *do*?

Once you have that, you can go about selecting the right tool for the job. The author notes that there's no consensus on what logs are for. Or is buried under signal-to-noise ratio for metrics. But when you start with a goal, and then determine a KPI you want to obtain to measure that, you'll then be able to ascertain where a log, metric, trace, or combination thereof can reflect that. And you only need create the ones needed to measure that. The scope of the problem becomes far more manageable.

It make take a couple tries to get right, and a user may find that they need to add more over time. But the monitoring doesn't need to change often unless the deliverables or architecture of the workload does. And creating those should be part of the development process.

By starting with goals and working from there, monitoring becomes a manageable task. The myriad of other logs and metrics created can be safely stored and kept quiet until they're needed to troubleshoot or review an incident. That falls into observability.

Observability is something you have. Monitoring is something you do.

I haven't addressed the difficulties the author states in creating various traceability mechanisms, how to parse all of these logs from various platforms, or some of the other technical challenges he states. There are usually solutions for these (usually...), some easier than others. But by narrowing the scope down to what's needed to achieve your given insights, the problem set becomes only what's required to achieve that aim, and not trying to build for every conceivable scenario.

mst 11 months ago | parent | next [–]

> What do I need to monitor?

I think it's worth looking at those decisions as wanting to be made both top-down -and- bottom-up.

Top-down as in "start from what makes the business a viable business and then analyze downwards from there" which gets you things like user-facing site/API availability, performance, error rates etc. (and maybe things like per-user usage rates depending on what you're doing and what your business model is).

Bottom-up as in "start from what makes the infrastructure exist at all and work up from there" which gets you things like disk usage, RAM, CPU, network link saturation - all the low level stuff that won't affect your top-down metrics until it does, at which point everything will catch fire at once.

They'll hopefully meet somewhere in the middle in a way that makes sense, and you could perhaps argue with per-internal-service monitoring as being a sort of middle-outwards, but I suspect the highest and lowest level checks are probably the most useful ones to start with and then you extend from there as you get a feel for what situations cause those to fire off and start monitoring the mid-range of the '5 whys' rather than just 1 and 5.

(I'm not sure I've made this as clear as I wanted but such is the peril of waxing philosophical about things)

poulsbohemian 11 months ago | parent | prev | next [–]

>"What do I need to monitor?"

I did this stuff professionally for about a decade... the short version is, consider the nodes of the graph. Stuff breaks because the front end can't reach the back, the back can't take a message off the queue, the database stopped taking connections, etc. IE: most apps of any size or complexity have reliance on other systems or infrastructure components, and these becomes things that break, timeout, don't scale, etc. That's a good starting point.

llama052 11 months ago | prev | next [–]

Honestly we've had great luck with HA prometheus instances shipping data to Thanos. All in kubernetes so we get to use pre-built operators. We don't really have to touch it other than updates.

We are a 3 man team and we are holding around ~4m active series with around a year of downsampled data in a storage bucket. Doesn't really cost us anything other than some servers on a cluster. We also use Loki for logs and Tempo for tracing in the same pattern.

One of the reasons I like Kubernetes, it's rigid enough compared to the wild west of VM land, and utilizing operators and tooling can get you solutions that just work provided you have enough core knowledge to fill in blanks.

perpil 11 months ago | prev | next [–]

If you want to do monitoring and logging at scale, the majority of the best practices are captured in this article: https://aws.amazon.com/builders-library/instrumenting-distri...

These are the key insights we learned running hundreds of services at Amazon scale and you'll learn something whether you're just starting, you think you know it all or you need some actionable ways of improving your operational excellence.

jeffbee 11 months ago | parent | next [–]

Notable they are not discussing centralized collection and indexing of debug logs, they implicitly leave the logs on the disk of the machine where they were produced, and go out to read them when called for. This is an important lesson because centralizing and indexing logs is very foolish unless you are a Splunk shareholder.

XorNot 11 months ago | root | parent | next [–]

I'd say retaining logs is the real mistake. You need to retain *some*, but it's going to be far less then you think. The more interesting data is usually "what's being logged *right now*" and none of the big logging systems have an answer as good as "tail -f somefile.log" or "journalctl -f -u some-service" on the machine it's running on...which sucks, because technically there's value to be found there! Connect me to the entire cluster I'm running and tail "just these logs" in as close to real-time as you can.

EDIT: I know it *can* be emulated in a bunch of ways, but it's certainly not a first class feature.

perpil 11 months ago | root | parent | prev | next [–]

I'm not sure that is a takeaway here. The processes evolved, but at AWS, centralized logging w/CloudWatch Logs was becoming more common than having tools that ran commands on end hosts against locally stored ephemeral logs. Sometimes an agent would aggregate the data locally before publishing, but using tools like CloudWatch Insights was distributed and fast enough (multiple gigabytes/sec) that no indexes were necessary to get quick answers to adhoc questions. My read is they are not advocating for publishing debug logs which gets expensive quick. They are suggesting you have the ability to enable verbose logging if needs be for a temporary use case. The two types of logs they describe are service logs which have one log entry per request (customer, request id, e2e latency, response code, etc) and applications logs that go into more detail about what happened on a particular request. Service logs are less expensive to store and more information dense. You are primarily querying service logs for operational visibility. Service logs are inexpensive and fast. You query application logs to trace a specific request and that is slower and more costly. For cost savings, I know of some teams that would reprocess applications logs after x weeks to strip information that only provided short term value.

wspeirs 11 months ago | root | parent | prev | next [–]

Centralizing logs can be a HUGE help... You just have to use a tool that doesn't cost what Splunk charges. At the risk of over-selling, log-store.com caps charges at $4k/month! That's still a lot of money, but way less than what Splunk and other SaaS providers charge!

jeffbee 11 months ago | root | parent | next [–]

I just think it's pretty foolish. All the resources you own are out at the edge of you infrastructure. You might as well just drop application logs right there and leave them there, because pushing out your search predicate to all the relevant machines is going to exploit all that CPU and IO bandwidth you already paid for.

hagen1778 11 months ago | prev | next [–]

Monitoring is a pain because we need to monitor more and more complex things. And not only that. Mostly, the problems described by OP are connected to the scale. So the main source of the "pain" is the volume of data we need to process with monitoring systems.

That's why Graphite is fading off - it just can't keep up with the growth of data volume.

Prometheus is a great tool, but now it is not enough of one server. So you are either looking for workarounds with federation, or running more complex monitoring systems like Thanos, Cortex, VictoriaMetrics.

mvdtnz 11 months ago | prev | next [–]

I just think that this is an incredibly whiny post. These are all solved problems but he brushes away the solutions with a hand wave in order to continue being relentlessly negative.

> Maybe you start with an ELK stack, but running Elasticsearch is actually a giant pain in the ass.

And that's the last he says of it. But I've seen ELK stacks in organizations from 30 employees / 5,000 users all the way to 5,000 employees / millions of users and it scales well. I'm sure it continues to scale above that.

Or for Prometheus scaling options,

> Adopt a hierarchical federation which is one Prometheus server scraping higher-level metrics from another server. [...] The complexity jump here cannot be overstated. You go from store everything and let god figure it out to needing to understand both what metrics matter and which ones matter less, how to do aggregations inside of Prometheus and you need to add out-of-band monitoring for all these new services. I've done it, it's doable but it is a pain in the ass.

It's not a pain, it's like an afternoon of work. I don't see why you need to be deciding which metrics matter - collect all of them. And if you don't know how to do aggregations in Prom, learn! You'll need to know this to use it anyway.

> lolinder 11 months ago | parent | next [–]
>
> A lot of the article is less "monitoring is a pain" and more "distributed systems are a pain". *Everything* is harder in a distributed system, not just monitoring, and the author's fixing of the blame on monitoring tools is a distraction from what's actually causing their headaches.
>
> lazyant 11 months ago | parent | prev | next [–]
>
> It can be both true that ELK scales well and it's a PITA to manage.

dbg31415 11 months ago | prev | next [–]

I think you can monitor things "as a developer" and you can, and should... but let's face it, nobody is really watching that stuff.

Better to tell the marketing teams to set up their own monitors and metrics.

It could be Google Analytics, it could be UptimeRobot. Or whatever they want to use.

Good for them to set monitors of every page that gets more than 1% traffic.

Good to monitor for the HTTP Response Code (especially if there's a WAF involved), simple content check (like the page title, to make sure something didn't get redirected), "too many hops test" for more than 2 hops to get to a destination, page load speed, and of course uptime. Most tools also will alert you if your SSL Cert or Domain is about to expire. Kinda useful. Good to monitor on every major vanity URL redirect as well... so like mysite.com/some-promo.

Setting this stuff up is SO EASY... but few people do it correctly / consistently -- so it's good to ask them to check once a month if they have all the monitors they need. Just grab the GA data and set a monitor on any page URL that gets more than 1% traffic... easy to just drop the data from GA into a CSV and import it into UptimeRobot. 2 minutes a month and you have amazing monitoring. Really important for monitoring to be run from a 3rd party location -- not from the same server, or behind the same firewall as your corporate network, but run from a location similar to what your real users are seeing.

From the dev side... having good logs so you can diagnose issues. But Let's face it, nobody has time to monitor and respond to things unless it's been escalated. So just let the marketing team / ecommerce team escalate the things they care about, and just make sure you've got enough logs and ways to diagnose the issues they report.

fpanzer 11 months ago | prev | next [–]

OP needs to install naemon and chill. Monitoring means alerting, and not collecting metrics. Monitoring means being notified of outages. Metrics are for troubleshooting and post mortems.

If you *monitor* CPU or RAM usage, you are doing it wrong. Oh and by the way, you want your ram to be full.

compumike 11 months ago | prev | next [–]

> I'm running more complicated stacks to capture massive amounts of data in order to use it less and less. > […] > My experience is companies do not anticipate that the cost of monitoring an application can easily exceed the cost of hosting the application even for simple applications.

These problems resonate with me for sure, and after tackling them at other companies, they were part of our team's inspiration for building Heii On-Call [1] as a lightweight, minimal monitoring platform for apps / APIs / Cron Jobs / etc.

Instead of "ingest all the metrics/logs" we have guides showing, for example, how to set up the most minimal possible Prometheus / Alertmanager, with the smallest footprint of CPU/memory/disk requirements. [2]

And for many of our users they don't even use any of our Prometheus or Datadog integrations, and just stick with our dead simple out-of-the-box HTTP monitoring. [3]

Note that these address the "now" problem of monitoring: is it working now, or not? While most of the article talks about the "past" problem of monitoring (long term storage of logs, metrics; pretty charts).

The "past" part is the part that the author is pointing to as being rarely useful and the source of the cost and complexity. I think there's probably still value in dumping whatever logs and metrics you want into S3 just in case you need to go digging one day, but for the most part, this is pretty low value. The relatively higher value and lower implementation complexity is in the "now".

[1] https://heiioncall.com/

[2] https://heiioncall.com/guides/minimalistic-monitoring-and-al...

[3] https://heiioncall.com/docs

wspeirs 11 months ago | parent | next [–]

>Instead of "ingest all the metrics/logs" we have guides showing, for example, how to set up the most minimal possible Prometheus / Alertmanager, with the smallest footprint of CPU/memory/disk requirements. [2]

I'll bite... why _not_ "ingest all the metrics/logs"? I see 2 possible reasons: 1) don't have enough resources (CPU/memory/disk) to consume "all the logs"; 2) you won't be able to find what you need.

I think both are a failure of the tools in the space.

jrms 11 months ago | prev | next [–]

That's why I'm still using munin, even for kubernetes clusters, even after 20 years in the business, if I should need a plugin that I wrote back then it would still work!

And I can monitor a whole k8s cluster (around 70 nodes) using 100MB of mem and 3G for storage (up to 1 year).

Keep it simple!!

klysm 11 months ago | parent | next [–]

It's only simple because it's something you already understand - that word doesn't really mean anything

jrms 11 months ago | root | parent | next [–]

That's correct! Keep it simple, for yourself!!

milar 11 months ago | parent | prev | next [–]

I love munin and I'd rather do Perl with it than the other shit

tonymet 11 months ago | prev | next [–]

Monitoring is more about process than logs .

The logs (including text logging, telemetry) are just signals.

The way to improve monitoring is by continuing to refine s/n ratio & critical alert health through continuous review of signals and identifying leading indicators.

Every week you should identify the critical objectives and the leading indicators that represent healthy and unhealthy states for those. Also identify false positives and noisy alerts and work to reduce noise.

For example, how many people are checking out, adding to cart, viewing the website. What are indicators of health and error conditions. What are true signals and what are noise?

The biggest issue i see is people setting up logging and alerting without continuing to revise and improve the process.

jimbokun 11 months ago | parent | next [–]

Yes. I found the article jumped too quickly to specific tools and their shortcomings.

The process needs to be monitoring/alerting need -> best tools and processes to meet those needs.

8note 11 months ago | parent | prev | next [–]

I'm not sure if weekly is right, but at least when an alert goes off, it's worth checking if it was useful.

Checking for things that don't alert is much harder, and I *think* also needs a review of what you aren't measuring to make an update

aynyc 11 months ago | prev | next [–]

Monitoring is at the infrastructure level. CPU/Memory/Disk space/network.

Logging is at the application level. Log just enough that you can replicate a problem, usually it means just the input.

The rest is marketing.

wise0wl 11 months ago | parent | next [–]

I have to completely disagree. Application level metrics can also be emitted at the log level, but are much easier to work with as a metric. We have total request count, average request latency per route, and per HTTP response. This is extremely useful for finding performance regressions when new code is released.

crabbone 11 months ago | parent | prev | next [–]

This is misguided.

Think about, eg. Ceph. From the side of the user, it gives user the storage, from the side of Ceph admin, it's an application.

Now, imagine Ceph is used in Kubernetes to create PVC. Now, Kubernetes is your storage provider... and so on.

WFHRenaissance 11 months ago | parent | prev | next [–]

There are some basic performance metrics that come to mind that are also important:

Latency Uptime Throughput

And if you're working with a distributed system, you should really be using metrics to track the health of your system at the application level.

aynyc 11 months ago | root | parent | next [–]

Latency, uptime and throughput for an application are non-functional requirements that should be built into your application. People like to generalize that and pawn that off to logging dashboard but I personally think that's a mistake.

klysm 11 months ago | parent | prev | next [–]

Usually it’s input + state to repro - good luck logging that

bob1029 11 months ago | prev | next [–]

We decided to outsource this pain to Azure. Az Monitor & Application Insights are far from perfect (I can't stand their UI for querying logs), but it really does "just work" out of the box unless you are using something very non-ms with it. If you are throwing a happy path thing like .NET6 on top of Az function runners, you can expect to not have to configure *anything*, aside from an option at app creation time.

Our primary mission around monitoring is to proactively respond to customers before they call us regarding exceptions in their environments. Az Monitor w/ basic alerts are all we really need to get that job done.

After spending 20 minutes with these tools, I felt really dumb for rolling our in house solution the last ~8 years. All we had to do was take a look and try. There is absolutely no way in hell that my team can compete with the resources of a trillion dollar hyperscaler like this.

The long game I am looking towards is the partnership between MS & OpenAI. "AIOps" is already listed as a "Respond" option under Az Monitor in their documentation. Anyone who has been standing on principle against Microsoft has to at least see some of the light coming through this tunnel by now, train or no train.

waffletower 11 months ago | prev | next [–]

This article focuses on the author's dislike of logging. Never does the author mention log storage expiration, an obvious and effective tool to reduce the largest complaint, storage cost. The author also doesn't mention differences of scale for services -- the log resolution for service X, may be inappropriate for service Y, etc.

0xbadcafebee 11 months ago | prev | next [–]

This isn't an article, this is just whinging that things aren't perfect.

"Log Levels Don't Mean Anything"

If you're not handling log levels correctly, you've just thrown away half the utility of logging. The author doesn't even make an argument here, he just lists a bunch of different log levels and moves on. You should absolutely rely on logging levels in your

applications.

"No consensus on what logs are even for"

They're for an application to tell you what is going on. Come on dude.

"Maybe I'm wrong about monitoring."

You're not wrong (well, about being a pain), you just should get a different job. Monitoring is hard, yes. But it's very beneficial, despite being highly imperfect. It just doesn't benefit you to build your *own* monitoring stack. Pay someone else to do it, accept that it's not going to be the Platonic ideal of computer science theory, and move on with life.

crabbone 11 months ago | parent | next [–]

I can absolutely get behind the "nobody knows what log levels are for".

First of all, logs come from application components, and the bigger your application is, the more levels are in this hierarchy (components of components etc.) Authors of individual components may not have any idea about how performance of their component is going to affect the whole application.

So... log level needs to be re-interpreted? Logs need to be collected hierarchically? Logs need to become part of the application public interface? -- This is too high price to pay, and, in practice, nobody is going to do this. So, in reality, outside of very simple cases, it's just easier to ignore the level of the log message.

Another aspect is that log levels aren't really a sequence. Messages can belong to several categories at the same time, but these categories need not occupy a continuous sub-sequence. And the more detailed you make it, the more obvious this problem becomes.

Yet another aspect: people writing logs may rely on another (dumb) program processing them. So, they will leave semantics of logs aside, while concentrating on the desired side effect caused by the application processing them.

My personal experience with, eg. Kubernetes is that its authors consistently underestimate severity of some conditions and overestimate severity of others. I often find that what was labeled as "warning" was the reason the whole cluster doesn't function anymore, while something that was reported as "error" was a completely expected condition that the program absolutely knew how to recover from.

0xbadcafebee 11 months ago | root | parent | next [–]

> logs come from application components, and the bigger your application is, the more levels are in this hierarchy (components of components etc.) Authors of individual components may not have any idea about how performance of their component is going to affect the whole application.

There is no way to solve this problem. If you have 50 unrelated sub-components in an application, there is no way to know if a given event is critical or just informational, because it requires specific context to understand what the event is actually impacting in that moment.

That doesn't mean the log level is useless. It just means it is one aspect of the signal you are getting from a sub-component. You can then filter on that specific signal from that sub-component in cases where the sub-component might provide a different log with a different log level signal. Is it perfectly accurate? No. Does that make it useless? No.

> Another aspect is that log levels aren't really a sequence. Messages can belong to several categories at the same time, but these categories need not occupy a continuous sub-sequence. And the more detailed you make it, the more obvious this problem becomes.

This just means you have a low-quality signal. Is every low-quality signal useless? No.

> Yet another aspect: people writing logs may rely on another (dumb) program processing them. So, they will leave semantics of logs aside, while concentrating on the desired side effect caused by the application processing them.

Again, low quality signals aren't useless, and you don't have to do this if you don't want to. Throwing away log levels is literally throwing away more signal, which is going to make life harder, not easier.

crabbone 11 months ago | root | parent | next [–]

> You can then filter on that specific signal from that sub-component

I already answered why this is a bad idea, but I will repeat: this makes logs part of the public interface. Which, in turn, imposes a lot more restrictions on component providers than they currently have / than is plausible to expect from them. And if you (the mothership application developer) decide on your own to rely on the feature that is not a part of public interface (i.e. you decide to filter logs from components and translate them somehow) anyways -- well, you've done a crappy job as an engineer by essentially planting a time bomb in your application.

So, your plan is not good.

This also means you cannot automate response to log levels. As a human reading the logs, in most if not even all cases, you could probably get down to the bottom of why a particular log message had the level it had, but it's not humanly possible to write a program to do that. This is what makes log levels worthless (in the context of monitoring).

> low quality signals aren't useless

You seem to confuse the goal OP set for logs (use them for monitoring, i.e. in an automated way) with other possible goals (eg. anthropological, where you are studying how human understanding of log messages evolved over time).

shadowgovt 11 months ago | root | parent | prev | next [–]

> Throwing away log levels is literally throwing away more signal, which is going to make life harder, not easier.

Signal that can't be used is just noise, and noise complicates solving a problem by interfering with the signal.

In projects I work on, I generally only use two logging levels: info and error. Error indicates unrecoverable conditions that mean an execution context is terminating. Info is everything else. "Warn" is useless because something is only a warning in a particular context and I don't have that context when I'm building the logs. "Debug" is a lie; logging isn't debugging, and if I need to debug I need to slap an actual debugger on the binary with source code available.

I couple that with the ability to turn on and off logging at a fine-grained module level and (if I'm living my best life) being able to instrument the production code for breakpointing and logging on the fly (via systems such as Google Cloud Debugger).

morelisp 11 months ago | root | parent | prev | next [–]

Not only logs - that year I spent where CPUThrottlingHigh was an "error" alert (on a cluster with mandatory CPU limits) was awful.

KaiserPro 11 months ago | parent | prev | next [–]

> They're for an application to tell you what is going on. Come on dude.

I mean yeah, but the format is unique to every app. trying to rationalise them to reduce them to a tertiary value (ok, degraded, fucked) is as unique to each app.

Plus, with every update, it'll change, means that its loads of work

wspeirs 11 months ago | root | parent | next [–]

You just need a tool that's adaptable and will let you parse-as-you-search. Without a schema, your logs can change whenever, and you can still easily derive value from them.

KaiserPro 11 months ago | root | parent | next [–]

but thats debugging, not monitoring.

If you don't know that your service is down, then you don't know where and what to look for. for monitoring, you need to be looking for a specific parameter or string. if that changes, its very difficult to generate an automated alert.

wspeirs 11 months ago | root | parent | next [–]

Fair enough... if you're monitoring "response_time", and a developer changes that field to "time_taken_to_send_the_bits" you'll probably have a tough time monitoring the service. However, if the dev communicates that the value has changed, with the right tool it isn't hard to have something that covers *both* fields.

KaiserPro 11 months ago | root | parent | next [–]

100% but in the real world its a bit hard to coordinate.

Ideally you'd have a schema that you agree company wide. Before an app is deployed it has to pass the schema test. At least that would cover most of the basic checks.

but, For most small places, logs are perfectly fine for monitoring, as you imply

shadowgovt 11 months ago | parent | prev | next [–]

To translate: I believe the author is saying logging isn't a scalable, transferrable skill (and therefore a bit of an ill-fit for -as-a-service software stacks to provide benefit). Due to lack of standardization and the fact that every application is different (and logs are an application-level concern), the rules differ completely from app to app on what should be logged, what logs matter, and what error levels even mean.

... the problem with logs has always been that logs aren't needed until you need to deep-inspect the behavior of a running system, and then they're necessary. But since you really never know how your system will break down until it's in production, all logging is in some sense crystal-balling future failure modes.

http://thecodelesscode.com/case/73

http://thecodelesscode.com/case/74

http://thecodelesscode.com/case/103

(... while it can be very expensive to set up the infrastructure to support it, I'm a *huge* fan of the ability to enable live-tracing: in essence, I've seen systems that let you do something equivalent to breakpoint debugging on a live service via

injection of novel code to be run when a specific line of code is hit in the service. That live code can then log whatever is relevant, dump a stack trace, dump variable values into a log somewhere, and so on. Of course, the downside is that your runtime has to be very specially constructed to support that kind of hot-patching on a production service).

0xbadcafebee 11 months ago | root | parent | next [–]

> Due to lack of standardization and the fact that every application is different (and logs are an application-level concern), the rules differ completely from app to app on what should be logged, what logs matter, and what error levels even mean.

This has nothing to do with logging or metrics. Every application *actually is different*. There is no such thing as a universal application error. Even when somebody tries to standardize errors, they still need to be interpreted on a case-by-case basis.

Let's say you change your logging method to just output HTTP status codes. Someone sent a request, you send back a 408. What timed out? What caused it to time out? To figure that out, we need more context; at least the body of the request that timed out, but also network connection details, and the details of whatever server app was supposed to process the request. We need context outside of the application to make sense of the error.

The application literally doesn't have that context, so it literally cannot communicate to you "enough" information for that log to be "useful". The log is just a signal, which needs to be processed with a lot of other signals, correlated, and interpreted, on a case-by-case basis.

Yes, the rules are different from app to app. Because the context is different, the operations are different, virtually everything is potentially different. Could there be a little bit more standardization of errors, a la HTTP status codes? Sure. But would that solve the problem of actually understanding the context and implications of what's actually going wrong? No way.

radiator 11 months ago | prev | next [–]

I don't like this article.

- *we are all doing monitoring wrong*? No, sorry.

- *You set up these tools with the mentality of "set and forget" but they actually require ever increasing amounts of maintenance*? The tools and your monitoring rules require maintainance like everything in IT.

- *Logs* OK, don't use logs for monitoring.

- *Metrics* Yes, do use time series for monitoring.

- *Prometheus is really designed to be running on one server. You can scale vertically ...* Actually you can operate more than one Prometheis behind a load balancer, avoiding the situation *"when there is a Prometheus problem you lose visibility into your entire stack at once"*, and without needing to federate them.

slaymaker1907 11 months ago | prev | next [–]

In terms of controlling what data is actually logged, SQL Server takes a completely different approach instead of using log levels. Extended Events are strongly typed and reduce data by having predicates that can look at the data of the event to determine if it should be saved or not. While log levels are handy, I think it would be great if more log frameworks had this kind of flexibility since it lets you log some specific “DEBUG” style events in production rather than turning on all DEBUG logs for a particular logger (for example, logging all client connections could be expensive, so maybe you just log the ones which aren’t system accounts).

solatic 11 months ago | prev | next [–]

If you're not excluding debugging and info logs at the SaaS level, you're doing logging with a SaaS provider wrong. Log everything including debug, exclude debug and info, set up alerts for error logs, wait until an error comes in, re-enable debug logging for 15 minutes if you need to, then enable the exclusion again.

SaaS observability gives you the tools to control these costs. It's much, much cheaper to log up to debug if you're not indexing them most of the time. If you don't understand features like exclusion rules, you should go learn them.

GnarfGnarf 11 months ago | prev | next [–]

I write my app's logs in HTML. It allows me to format as tables, large fonts for alerts, colour for errors, embed images, all sorts of visual aids. Easy to scan and identify the important stuff at a glance.

There is one common function to format as low-level HTML, and a trivial markup code to simplify calling it.

ducharmdev 11 months ago | parent | next [–]

Are the logs actually written as HTML, or are you logging objects (like structured logs) that are transformed into HTML when displayed?

If the former, I would imagine that'd make searchability harder for larger volumes of logs.

GnarfGnarf 11 months ago | root | parent | next [–]

The logs are written as raw HTML. They can be searched by including HTML id's, tags, etc. They are plain text so they are searchable.

The color option really makes things stand out. The formatting possibilities has made debugging so much easier.

ducharmdev 11 months ago | root | parent | next [–]

I guess what I was getting at, when you are logging lots of data in something like Azure app insights, Splunk, etc., you don't have the luxury of being able to just scan your logs with your eyeballs, because there's simply too much information. Instead you log objects, and are querying logs almost like you would SQL.

iDemonix 11 months ago | prev | next [–]

Using Prometheus in federated mode has been fine for me, I'm running it almost exactly as depicted in the article and I'm not sure why OP describes it as a complex on - you just need to have some idea of what metrics you'd like to store longer term.

acaloiar 11 months ago | prev | next [–]

I generally agree with the sentiments here, yet I think it's important to understand the value of the ".0001%" of logs that _are_ used.

Just because log utilization rates are often very low, doesn't meant that the .0001% isn't immensely valuable.

paulvnickerson 11 months ago | prev | next [–]

Check out Grafana's suite of tools, especially Mimir [1] for a simple and easy to use monitoring platform.

[1] https://grafana.com/oss/mimir

hagen1778 11 months ago | parent | next [–]

It consists of 6 separate components to run, plus 4 additional optional components [1]. And I don't even count 5 additional caches [2]. How is that "simple and easy"?

[1] https://grafana.com/docs/mimir/latest/references/architectur...

[2] https://github.com/grafana/mimir/blob/9a52522a05f25d6d6bc2e8...

paulvnickerson 11 months ago | root | parent | next [–]

Currently we run it in "monolithic" mode, in which all components are grouped together, and it works just fine: https://grafana.com/docs/mimir/latest/references/architectur...

hagen1778 11 months ago | root | parent | next [–]

How is it different from running a Prometheus server then? Prometheus is monolithic as well. It is cool and not cool in the same time. With monolithic approach, the process becomes very simple to run and efficient at the same time, since there is no penalty on interprocess communication and network is not involved. But you can't scale monolithic applications horizontally. Horizontal scaling is very likely the reason why Mimir exists at all. If Prometheus supported horizontal scaling out of the box, there may be no need in Thanos, Cortex or Mimir.

Jedd 11 months ago | root | parent | next [–]

Have a read about the design intent and architecture to get deeper info on how it's different - but basically, indexes aren't retained locally, which means you've got a MUCH bigger scale (several orders of magnitude) than Prometheus can cope with. You're right that you can't 'scale monoliths horizontally' but Mimir was designed to handle much more data than Prometheus was, on the same compute/memory footprint (obviously storage is a direct function).

Mimir will consume multiple Prometheus with remote_write's and de-dup those series. Because it puts older (default 2h) data out to block storage, costs plummet without impacting performance too savagely - the vast majority of queries are going to be hitting fresh data.

We've been running monolithic (Nomad, docker) but are experimenting with their 'in-between model' which has 3 components - read, write, backend - which should help us scale more easily than juggling the full microservices deployment method you mentioned above.

> If Prometheus supported horizontal scaling ...

Well, yes, precisely so.

hagen1778 11 months ago | root | parent | next [–]

> Have a read about the design intent and architecture to get deeper info on how it's different - but basically, indexes aren't retained locally

Would appreciate if you refer to link where this explained. But from what I was able to find, both Mimir and Prometheus store indexes within data blocks. It doesn't matter where data blocks are: locally or on S3. Because you still need to read the index and fit it in the memory in order to locate the needed data. So I don't understand how S3 helps here, unless you run an additional store-gateway which does this job instead.

> but Mimir was designed to handle much more data than Prometheus was

Do you have any numbers to refer? Isn't Mimir based on Prometheus source code? The storage, indexes, query engine - that should be the same or mostly the same. I'd be really surprised if Mimir could outperform significantly Prometheus in monolithic mode.

If I understood correctly, the whole "Mimir was designed to handle" is related to microservice architecture, to its horizontal scaling capabilities and ability to shard read queries across multiple components.

Jedd 11 months ago | root | parent | next [–]

Sure, no problem - I typed 'mimir architecture' into my search engine to get the link I was thinking of:

https://grafana.com/docs/mimir/latest/get-started/about-graf...

Perhaps indexes wasn't the optimum word. There's some references to Prometheus' memory usage if you go looking, typically if you're searching around 'why does prometheus use so much memory'.

I'm struggling to find the comparative metrics on capacity that you're seeking, though I do recall seeing them previously, and having run up multiple prometheus instances (in vanilla and agent-mode) and fed those into mimir, I'm aware that prom has some non-negotiable limits on time series.

geoffbp 11 months ago | prev | next [–]

The key is to develop a monitoring framework. pick the tool/s you want, enrich and standardise what can be done with them (dashboards, alerting, etc) then make the path to using them easy

waffletower 11 months ago | prev | next [–]

Would be useful to add data presence/freshness monitoring to the taxonomy of tools here.

JohnnyTsunami 11 months ago | prev | next [–]

Well, I built technology that alleviates this pain. :0

JohnnyTsunami 11 months ago | parent | next [–]

Either:

If you are using $XYZ software, take a look at Aronetics' Thor. This technology is currently running within the a NCR Government office for evaluation with great success. We're leveraging this blessing from Uncle Sam as another go-to market strategy.

Thor has completely eliminated the threat presented to your data, it's a defense and counter-offense tool. As further exposure, when could you set aside time to learn more?

Or

Our digital hammer Thor is a defensive technology that closes gaps utilizing Zero Trust providing precise tamper-proof assurance to take action against tomorrows unknown threats so you have operational truth and trust of your compute device today.

jrockway 11 months ago | prev | next [–]

I have achieved zen with logs.

Levels are easy. You have three, DEBUG, INFO, and ERROR. DEBUG is stuff that's interesting to you, the author of the code. INFO is things that are interesting to the person/team that runs your software in production. ERROR is for things that require human attention.

My opinion is that INFO logs are the primary user interface for your server-side software, and demand the level of attention that you would expect a frontend designer to devote to your brand or whatever. That's where the software gets to interface with the person that's responsible for its care and feeding. Garbage in there wastes their time. Everything in INFO should be logged for a reason. (Yes, sometimes there's spam. Usually request/response data. Most of the time the requests go great, but you don't know they've gone great until you've logged the fact that they were made. It's a necessary evil.)

Meanwhile, DEBUG should probably be pretty chatty. If your program is consuming resources like CPU, it better justify itself in the DEBUG logs. A program sitting there using 100% of the CPU and not logging anything at level DEBUG is very very suspicious. My philosophy is to "show your work" on every critical operation. Log when something starts, what decisions it makes along the way, and when it's done. A message like "failed to contact server example.com: i/o timeout after 10 minutes" is 10 minutes too late to be interesting. "connect to example.com: started", "connect to example.com: i/o timeout after 10 minutes" is much better, because 10 minutes before the error you can read the logs and realize example.com is NOT where your database is hosted. (You can really go overboard here and I would never stop you. connect to example.com: resolve DNS: start, connect to example.com: resolve DNS: A 1.2.3.4, connect to example.com: dial 1.2.3.4: start, ... In production, I compromise and log "start", "ongoing long HTTP request" (at 10 seconds), and "finished" with the status code.) Every branch is a place to log which branch you selected and why. If there's a bug and you only know the outcome, but not the software's "thought process" that led you there, you have nothing to go on. "Hey, the FrobTheBaz endpoint is returning 400s." That is useless information. You have to read the code and guess. You do not want to be guessing during an outage.

Errors are a whole can of worms. If someone reading the logs can't fix the problem, it's not an error. "Error: cannot write data to the database". That's a big problem. Disconnect that popcorn popper and plug the power cable to your database back in! "Error: login for root from spambot@1.2.3.4: invalid signature" is a problem for your firewall. What is someone reading the logs going to do about that? Call up 1.2.3.4 and ask them to stop guessing SSH keys? At best, that's an INFO log. There are 3.4 with 38 zeros possible keys. They aren't ever going to guess it. If you decide "this isn't even worth logging", I can definitely agree with you. (I'd log it, though. An increase in failure attempts is good ancillary information when something else draws your attention to the system.)

Finally, newline-delimited JSON ("jsonl") is the only correct format for logs. Nobody wants to write a parser to figure out "how many 400s did we serve today". Just let JQ [https://jqlang.github.io/jq/] do it for you. If you want pretty colors for viewing, use something like jlog: https://github.com/jrockway/json-logs. You don't need to write VT100 control characters to your log file. I know for a fact you don't even *have* a VT100.

All other formats of telemetry are just logs with fancier user interfaces. Metrics? Just write them out to your logs: "meter:http_bytes_sent delta:16384 x-request-id:813db394-75a1-4b22-94ad-e7850d9cdd25". You still have the ability to tail your logs and convert the metrics to a proper time series database (with less cardinality than "per request" of course), but you can now also do investigations in context. For example, I've implemented exactly this system at work. A customer shows up, saying that we're using too much memory. I ask for the logs and see something like "span start: compaction", "meter:self_rss_bytes delta:2000000000", "span end: compaction, error:nil", "meter:self_rss_bytes delta:-1000000000". Hey, we leaked a gig of RAM in compaction. You would never figure out the memory leak with "compaction done after 5 minutes" and a graph of memory usage across your fleet.

While you weren't looking I also threw traces in there! Have a logging primitive like Span("operate(foo)", func { ... }) that logs the start and the end, and throw a generated ID in every message. Concatenate the span IDs which each recursive call to Span (I abuse context.Context for this, sorry Rob Pike), and log them with every message in the span. Now you have distributed

tracing. As always, sure, parse those logs and throw the log-less spans in to Jaeger so you can efficiently search for rare events and have a nice data flow graph or whatever.

Log sare so simple and so great. The problem you run into is by not producing enough of them. Disks cost nothing. You can delete the debug logs after a short period of time, or not even produce them until you suspect an issue. Don't sweat the log levels; pick the 3 I picked and try to enforce the rules in code review. Aim the log messages at the right audience; dev team, ops team, emergencies. Show your work. Log contextual information. Put metrics in the logs, and aggregate them for dashboards and monitoring. Annotate logs with trace IDs, and put those into Jaeger. You'll never have to *wonder* why your system is broken again; you printed out everything it's doing. The aggregation will tell you you're not meeting your error budget, and the logs for a representative request will tell you why that one failed. Once you know what's broken, the fix is always easy.

Whenever I tell people my system, they tell me it can't scale. When I worked at Google, I wrote exactly this system for monitoring Google Fiber (well, we didn't have log levels really...), and we could answer questions about the entire CPE fleet as easily as we could answer questions about one device or one bug. Over many years, I've finally gotten this into the "real world" (like weeks ago), and it's been amazing.

Good logs are the best thing you can do for someone that has to run your software. For a lot of folks, that's you! For other folks, that's your customers. Can you customer set up Prometheus, ElasticSearch, and Jaeger, and dump that all out and send it to you in Slack? Unlikely. Can they dump a file that's 10MB of JSON lines into Slack? They sure can. And now you can diagnose the bug and fix it without bothering them for more info. It's good for everyone.

AndrewKemendo 11 months ago | prev [–]

Solving this is the entire purpose of Solarwinds and we see what kinds of risk trade offs you get…

I lived this for years and there’s just no easy solutions here.

I don’t care what kind of application you’re building, if you want to be able to reliably deliver good service and iterate in any type of scientific fashion, you must-have great observability. This is not a suggestion. It is simply creating the situation where you can run experiments and understand the impact of changes to your system.

But that means you need to significantly invest in not only the technical tooling, but the people, education and procedural things that are necessary to think in a data oriented way. That is something that is excessively hard to convince people to do because it’s so seemingly boring.

The real siren here is push button observability, and automation like with solar winds. Everyone would love to be able to have State machine level determinism, and monitoring to tell you which distinct state you’re in. I’ve yet to find an org that can comfortably manage and exploit that level of granular observability.

This is an amazing article that every engineer should read.

> SteveNuts 11 months ago | parent [–]
>
> I’ve never before heard Solarwinds being discussed even in the same realm of observability.
>
> Last time I used it it _barely_ can tell you up/down status of a hardware device via ICMP or SNMP.
>
> Has it gotten 10,000x better in the last few years?
>
> > AndrewKemendo 11 months ago | root | parent [–]
> >
> > I didn’t know that it was possible not to connect those two given that it’s literally the entire Raison Detre of Solarwinds.
> >
> > It is on every single part of their marketing materials and is exactly what they are selling to and promising CIOs and CISOs.

I mean just look at: https://www.solarwinds.com/

SteveNuts 11 months ago | root | parent [–]

Have you ever... used it?

AndrewKemendo 11 months ago | root | parent [–]

I'm not saying it works...just that's the whole problem they claim to be solving, which they clearly dont