DeliveryHero Tech

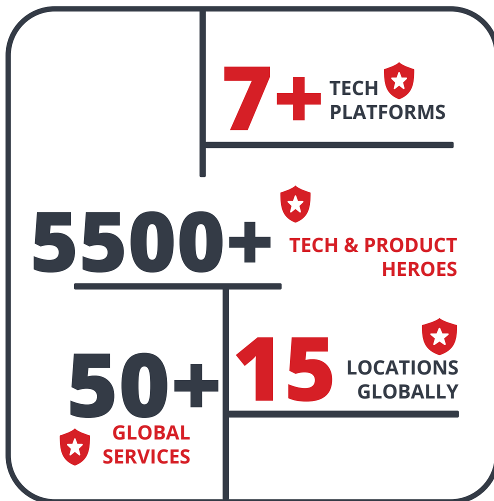
ARE WE ALL ON THE SAME PAGE?
LET'S FIX THAT

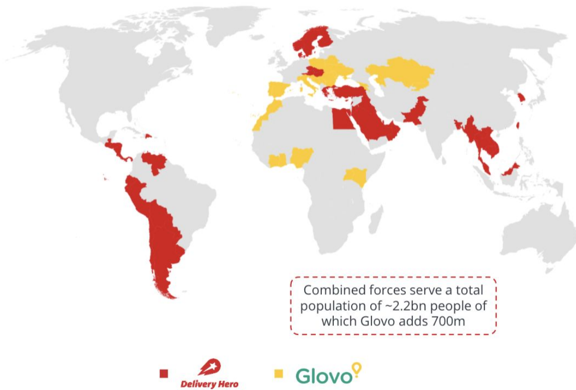
Luis Mineiro
@voidmaze / @voidmaze@techhub.social
SREcon23 Asia/Pacific

These are the tech platforms powering our worldwide brands and customer base.
You may be familiar with some of the Asian brands running on these platforms such as Foodpanda.

DH TECH & PRODUCT SCALE

DeliveryHero Tech





Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

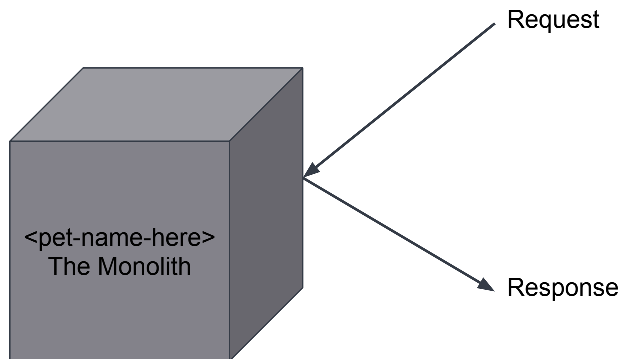
Over 2B customers being served by these platforms is a pretty impressive scale. We achieve that with a tech & product team, north of 5000 colleagues, working across 15 locations around the globe. We have one right here in Singapore.
This scale is relevant for this talk… I would like to start with a journey…

A long time ago… we used to run monoliths

THE AGE OF THE MONOLITH

DeliveryHero Tech

Single, large boxes
that did everything



Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

Monoliths handled every request. Request and response would be entirely the responsibility of the monolith…
There were some minor evolutions of this basic model, namely for redundancy and availability, but that's not relevant to the argument...
More important -- monoliths were simple. They were easy to reason about…
and monitor...

MONITORING THE MONOLITH

DeliveryHero Tech

Ops Monitoring

- Is the box alive?
- Is the monolith process up?

Devs Monitoring

- Are requests returning errors?
- Are requests reasonably fast?

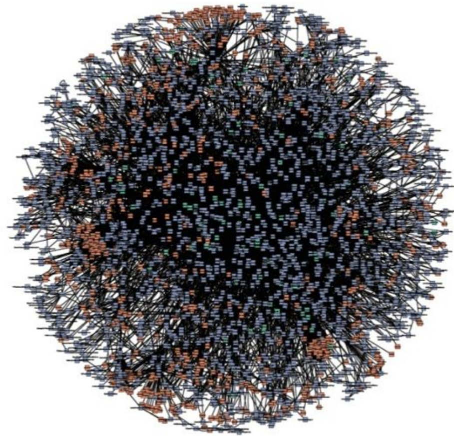
Photo by Deneen LT on Pexels

Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

This was the time of the Ops and Dev silos...
Trying to simplify things here, but I believe this is a fair description…
The Ops people monitored the hardware and checked if the monolith process was up.
The Devs monitored requests and the responses.
We all probably agree that this approach had its own share of problems, particularly as businesses grew and the approach didn't allow the business to scale further…
How did the industry react to these problems?

MODERN MICROSERVICES ARCHITECTURES

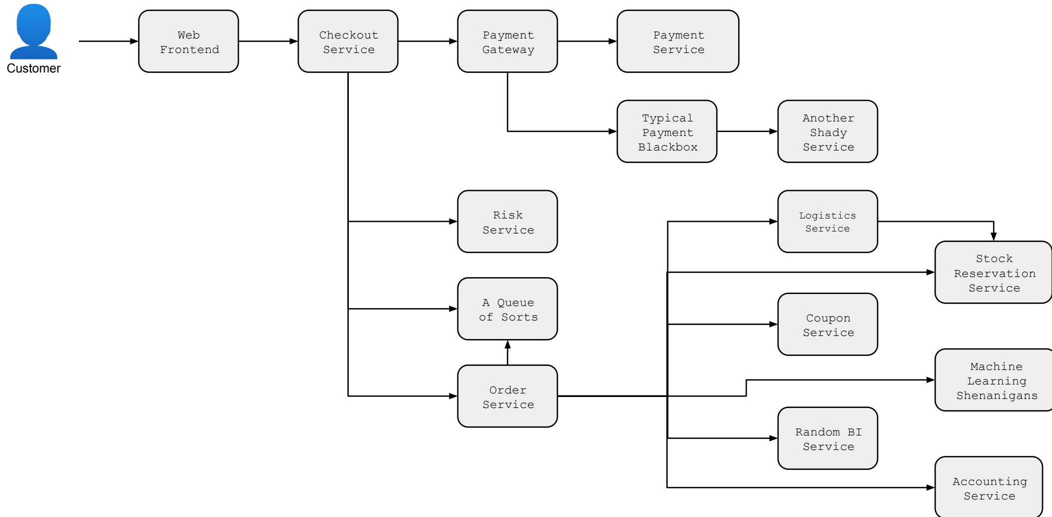
DeliveryHero Tech



Amazon internal microservice dependency visualization circa 2008

Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

Well, we solved it… ;-)
This is not another talk about microservices but I believe the context is very important when it comes to monitoring and, in particular, alerting.
So, let's try with a simpler example



This is a typical business operation for e-commerce websites. Placing an order...
To fulfill this business operation, it's common that the number of involved micro-services could be similar to this diagram.
In my experience, it is common to have something like 30 micro services involved in such a business operation, including some so called legacy ones.
For your organization this may look too complex or too simple… YMMV
How do we monitor and alert on this?

MONITORING MICROSERVICES

DeliveryHero Tech

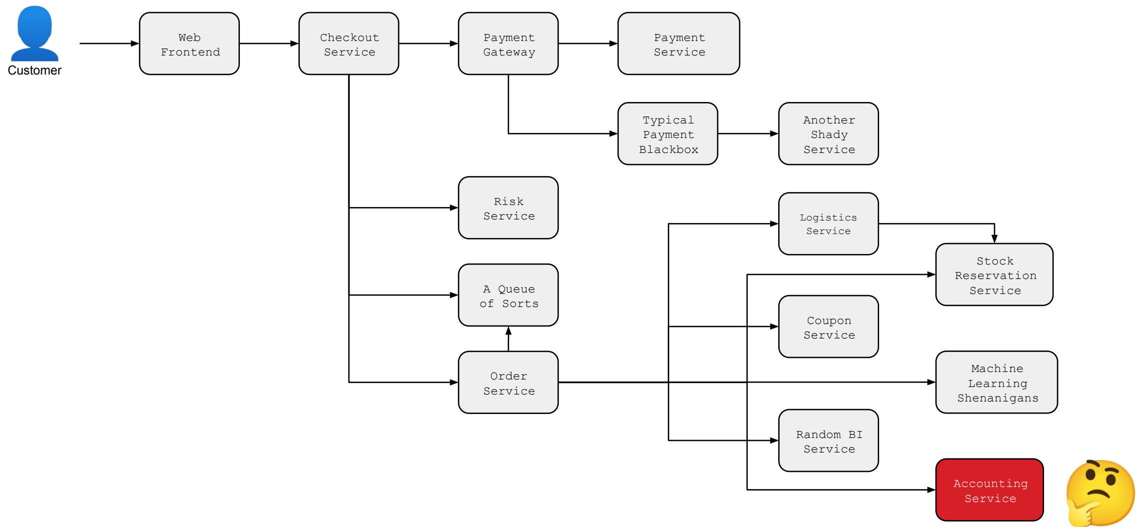
"DevOps" Monitoring

- Is the box alive?
- Is the micro-service process up?
- Are requests returning errors?
- Are requests reasonably fast?

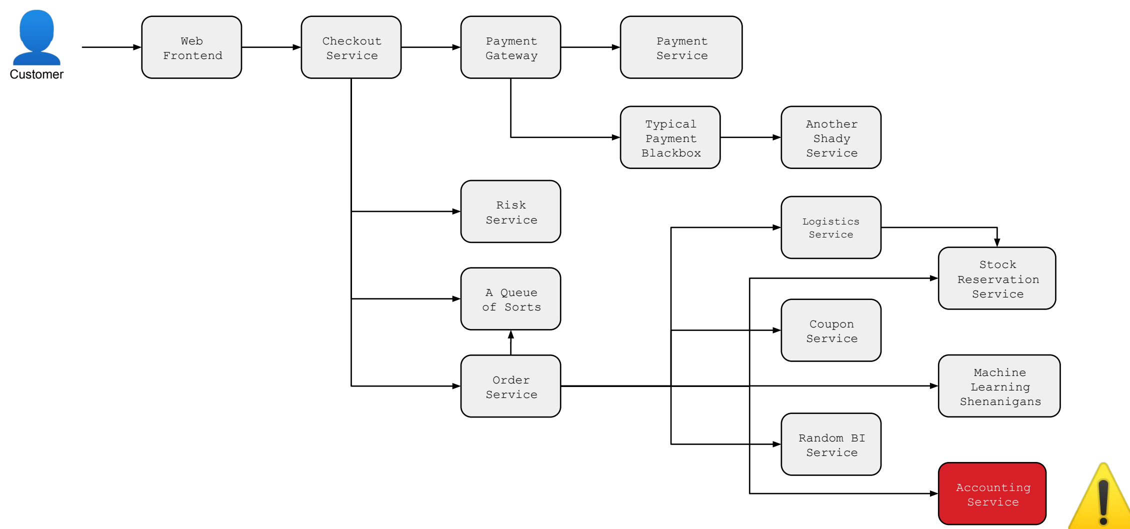
Photo by Antoine Plüss on Unsplash

Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

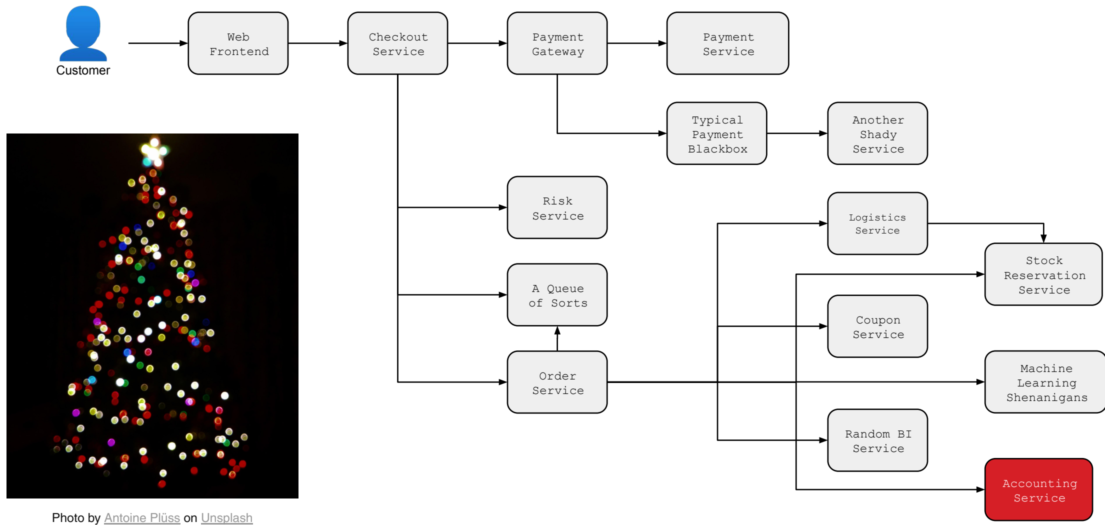
We came up with new roles, some call them DevOps, some call them SRE, but the name is not important… We could call them Cupcake Fairies… it doesn't matter. What matters is… How we monitor didn't change much though… We still check if boxes are alive, if processes are responsive and if individual micro services succeed and, if we're lucky, if they are fast enough.
When it comes to monitoring, I'd say that we're just monitoring a distributed monolith.
What about alerting?



What do you think happens when the Accounting Service has an outage?



What almost always happens is that dozens (or hundreds) of alerts come up.



I call this the Christmas Tree effect. Lots of blinking things… It's almost the same…except…

The happiness levels are different, and definitely no one is getting any presents!

...

This approach almost always leads to an excessive number of alerts and results in alert fatigue. Only one of those teams can actually do something about it - the one operating the Accounting Service.

The alternative to this is to alert on symptoms instead (this is something the industry already accepted - in theory)

SYMPTOM BASED ALERTING

DeliveryHero Tech

Symptoms are a better way to capture more problems more comprehensively and robustly with less effort - "**symptom-based monitoring**," in contrast to **"cause-based monitoring"**.

Rob Ewaschuk, "My Philosophy on Alerting", 2013

Keep alerting simple, **alert on symptoms**. Aim to **have as few alerts as possible**, by alerting on symptoms that are associated with end-user pain rather than trying to catch every possible way that pain could be caused.

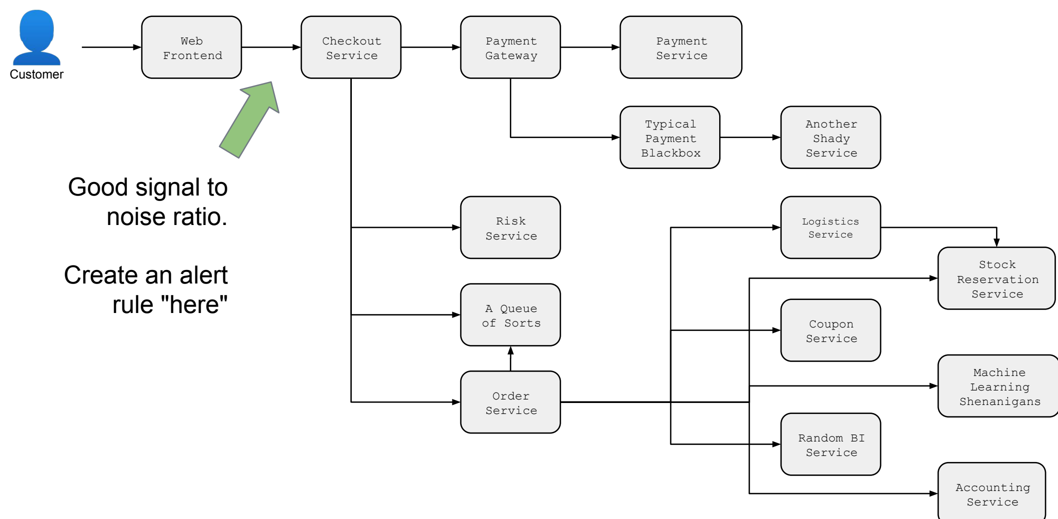
Prometheus Best Practices 2015, https://prometheus.io/docs/practices/alerting/

Alerting should be both **hard failure–centric** and **human-centric**.

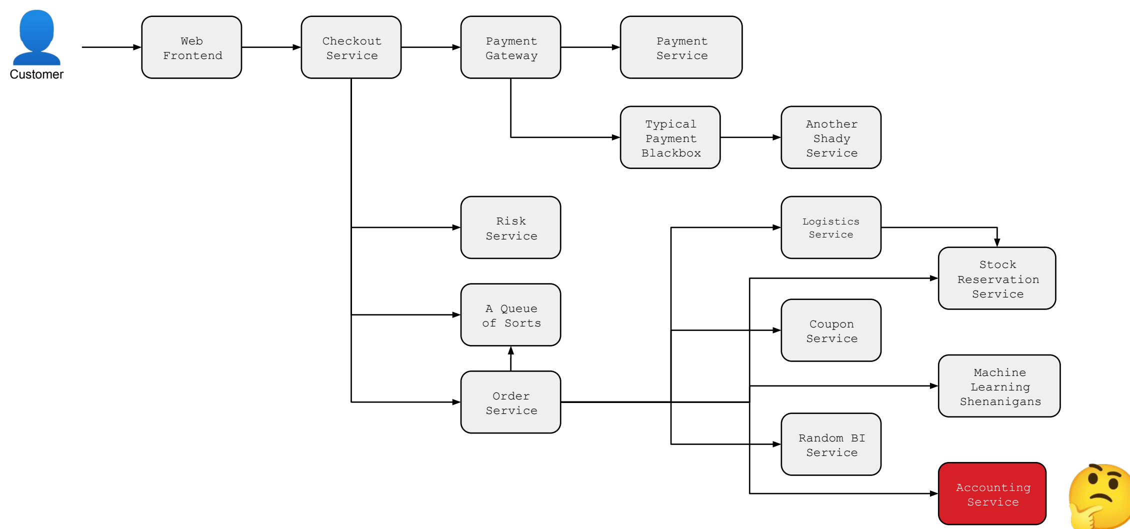
Distributed Systems Observability 2018, Chapter 2: Monitoring and Observability

Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

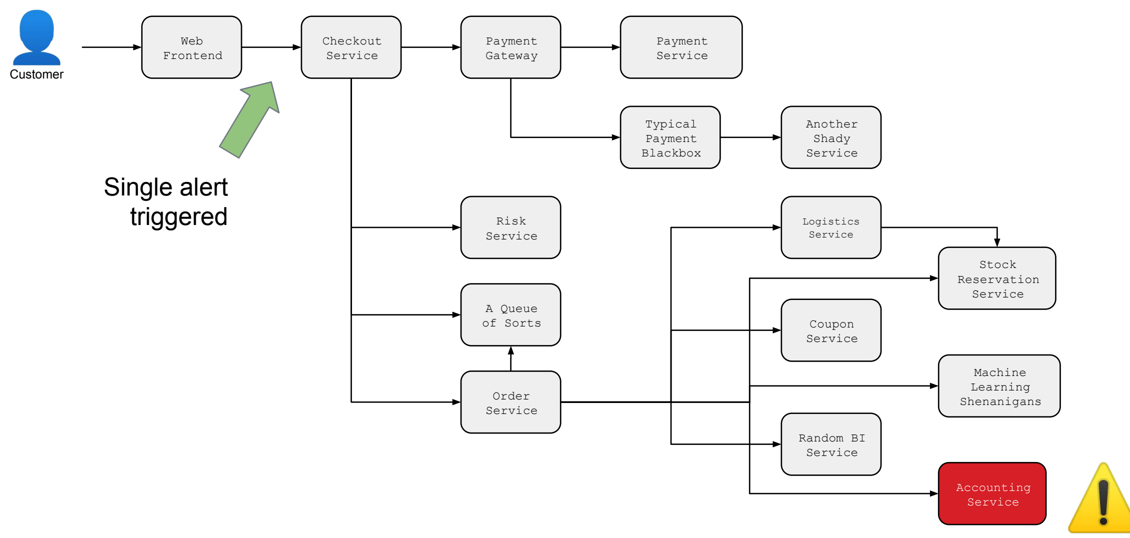
While one can proxy certain things such as number of 500s in the HTTP service closer to the users, this practice, replicated across the entire distributed system can be a key contributor to the christmas tree.
How would our example look if we were alerting on symptoms?



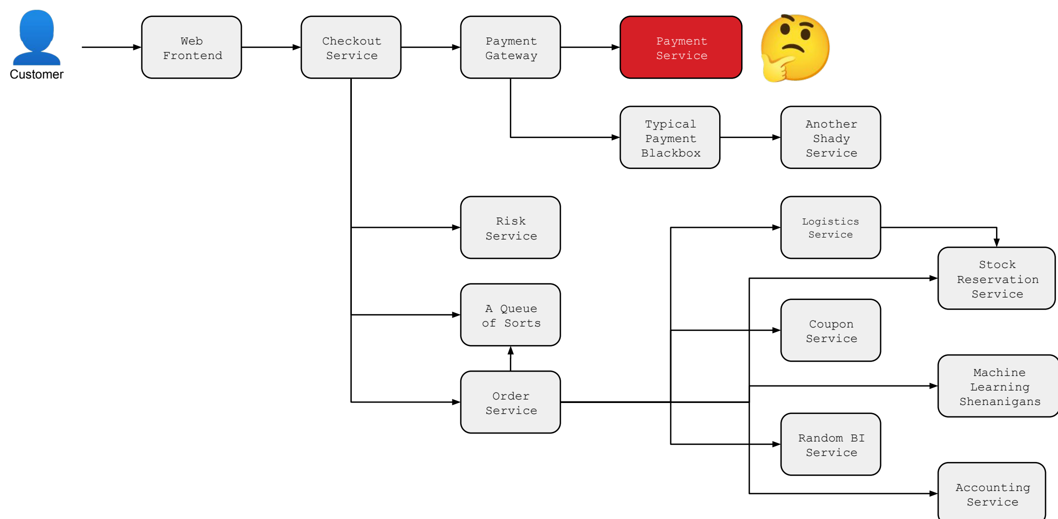
A good place to measure when placing an order, where the signal-to-noise ratio is optimal, would be as close as possible to the customer pain
We can measure signals like latency and errors here.



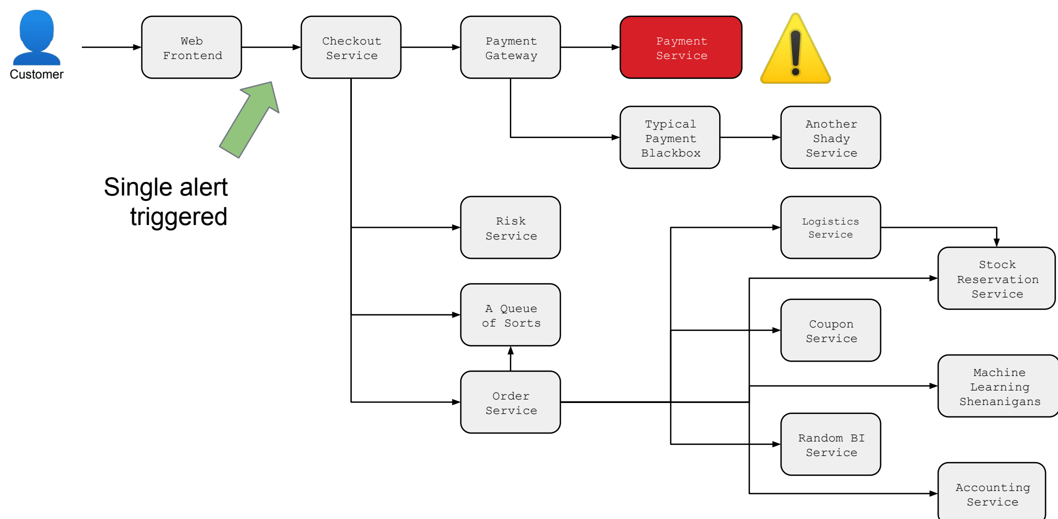
What happens in this case if the Accounting Service has an outage?



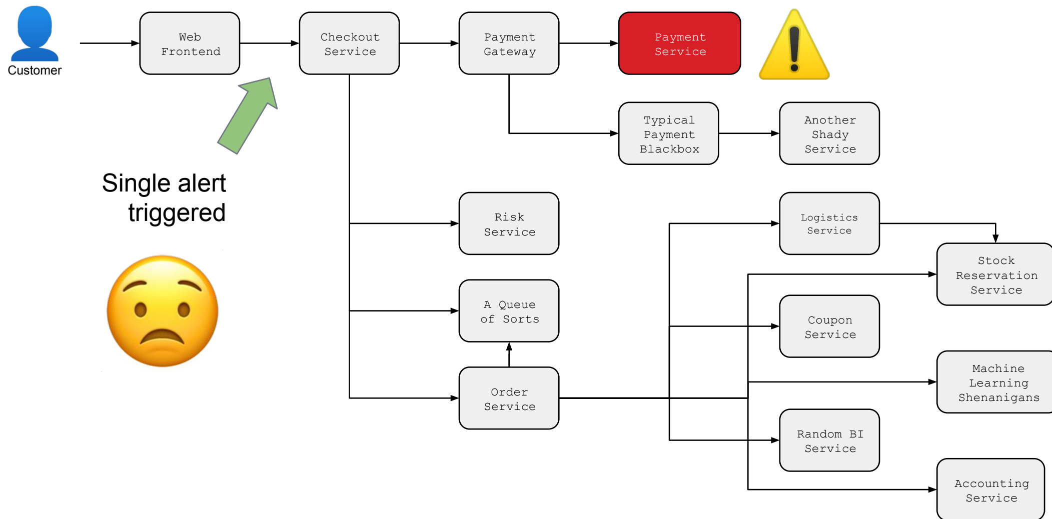
The alert we created based on the symptom will be up.
This looks better, doesn't it?
Is there anything wrong with the approach?
Let's try a different example...



What happens with this approach if the Payment Service has an outage?



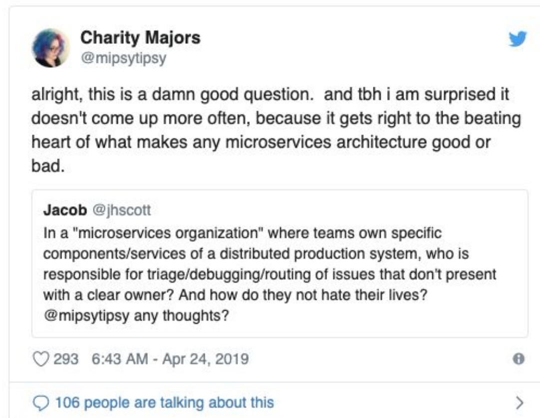
The same alert will be up… and the same team, the owner of the alert rule, gets the paging alert
...
For **each and every** possible failure in the distributed system!



This sort of pivoting is a serious problem that hasn't been addressed properly AFAIK. Alerting on all the layers of the distributed system is not healthy and the alternative, alerting on symptoms, can result in overwhelming the team owning the client-facing service…

ALERTING FOR MICROSERVICES

DeliveryHero Tech

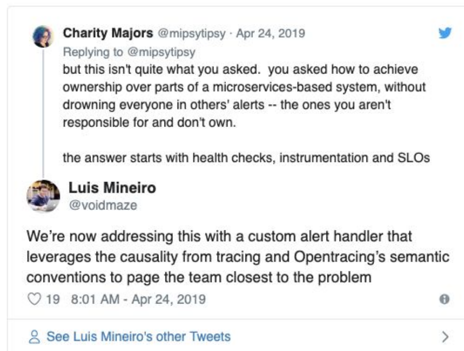


Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

I think this tweet captures the essence of the problem. The so called "microservices organizations" struggle to figure this out.
Not a coincidence, that's exactly what this talk is about

ADAPTIVE PAGING

DeliveryHero Tech



Adaptive Paging is an **alert handler** that leverages the **causality from tracing** and **semantic conventions** to page the team **closest to the problem**.

Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

From a single alerting rule, a set of heuristics can be applied to identify the most probable root cause, paging the respective team instead of the alert owner.
Before I dive into the implementation and explain how it works, I want to describe the main ingredients - tracing and its semantic conventions.

DISTRIBUTED TRACING

DeliveryHero Tech

- A trace tells the **story of a transaction or workflow as it propagates** through a distributed system.
- It's basically a directed acyclic graph (DAG), with a **clear start** and a **clear end** - no loops.
- A trace is made up of **spans** representing contiguous segments of work in that trace.
- **OpenTelemetry** is made up of an integrated set of APIs and libraries as well as a collection mechanism via an agent and collector. It also does **distributed tracing**

Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

This is just a quick introduction about DT
(read #4)
You're able to find out a lot of details by checking its origins, like the Dapper paper.
Opentracing and Opencensus were some of its predecessors
In this talk I think it is important to emphasize some of the details.

OPENTELEMETRY CONCEPTS: SPANS

DeliveryHero Tech

Span: represents a unit of work or operation, usually a remote procedure call. They have a name and record the **duration** of the operation, with optional **Attributes** and Events.

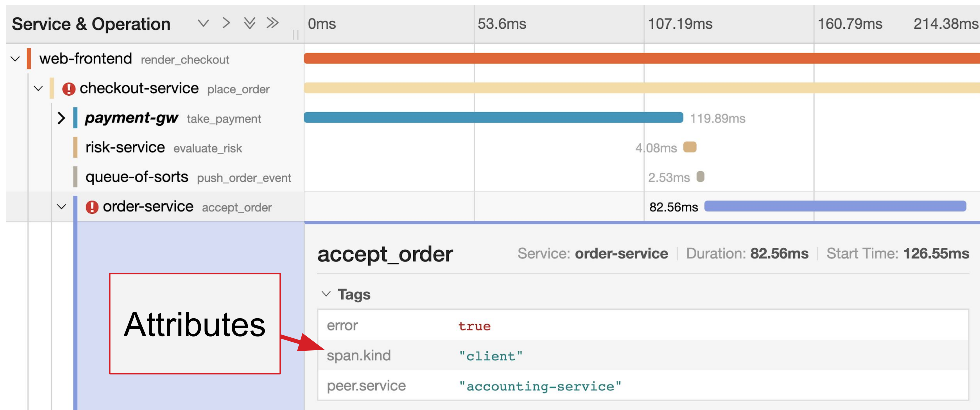
Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

(read)
Probably the most important element of opentelemetry - spans. Spans are the building blocks of Traces. In other words, a trace is a collection of spans.
Operations can trigger other operations and depend on their outcome. for ex., place_order triggers and depends on all the other operations, including update_account in the accounting-service.
This is CAUSALITY - this is important.
…
I deliberately highlighted the other most relevant element for this talk - Attributes.
What's an attribute?

OPENTELEMETRY CONCEPTS: ATTRIBUTES

DeliveryHero Tech

Attribute: Attributes are **key-value pairs** that contain metadata that you can use to annotate a Span to carry information about the operation it is tracking.



Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

(read)
Every span can have its own set of attributes.

Consider them metadata that enrich the operation abstraction (the span) with additional context.

Attributes are key-value pairs that contain metadata that you can use to annotate a Span to carry information about the operation it is tracking.

For example, if a span tracks an operation that adds an item to a user's shopping cart in an eCommerce system, you can capture the user's ID, the ID of the item to add to the cart, and the cart ID.

This is a very important ingredient of Adaptive Paging.

OPENTELEMETRY SEMANTIC CONVENTIONS

DeliveryHero Tech

Span attribute name	Type	Notes and examples
service.name	string	Logical name of the service that generated the associated Span. E.g., "checkout-service".
status.code	string	**Error** if and only if the application considers the operation represented by the Span to have failed
peer.service	string	**Remote service name** (for some unspecified definition of "service"). E.g., "accounting-service"
span.kind	string	Can be "**CLIENT**" or "**SERVER**" for the appropriate roles in a synchronous RPC.
... and more		

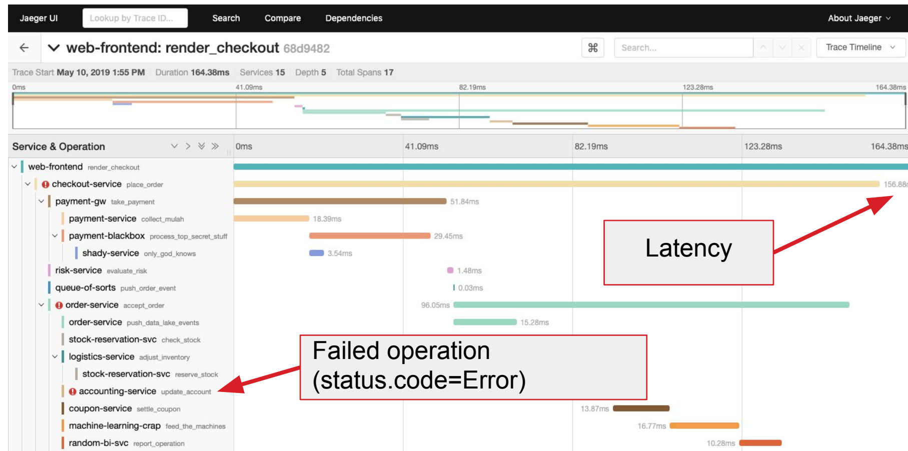
OpenTelemetry semantic conventions

Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

OpenTelemetry's semantic conventions establish certain tag names and their meaning.
This is strong enough to set expectations and what machines can do when analysing tracing data.
How, you may be wondering...

OPENTELEMETRY MONITORING SIGNALS

DeliveryHero Tech



The Four Golden Signals
SRE Book, Chapter 6: Monitoring Distributed Systems

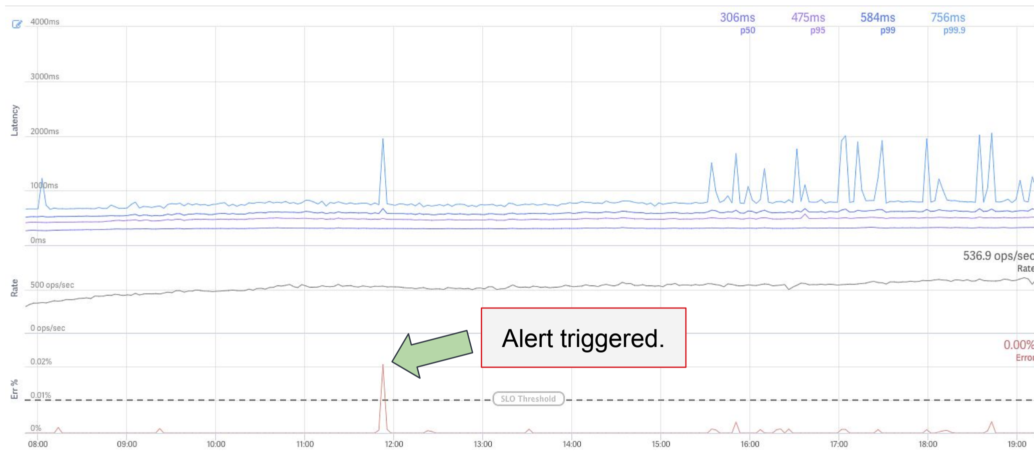
Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

Distributed tracing gives us, implicitly, latency and throughput (ops per second).
Through the semantic conventions we also get errors - spans with the status.code attribute set to Error
This is a good opportunity to refer to the Four Golden signals.
The four golden signals of monitoring are latency, traffic, errors, and saturation.
If you can only measure four metrics of your user-facing system, focus on these four.
They are great for alerting.

For the next example we'll focus on one concrete signal, alerting on the error rate.

ERROR RATE ALERTING RULE

DeliveryHero Tech



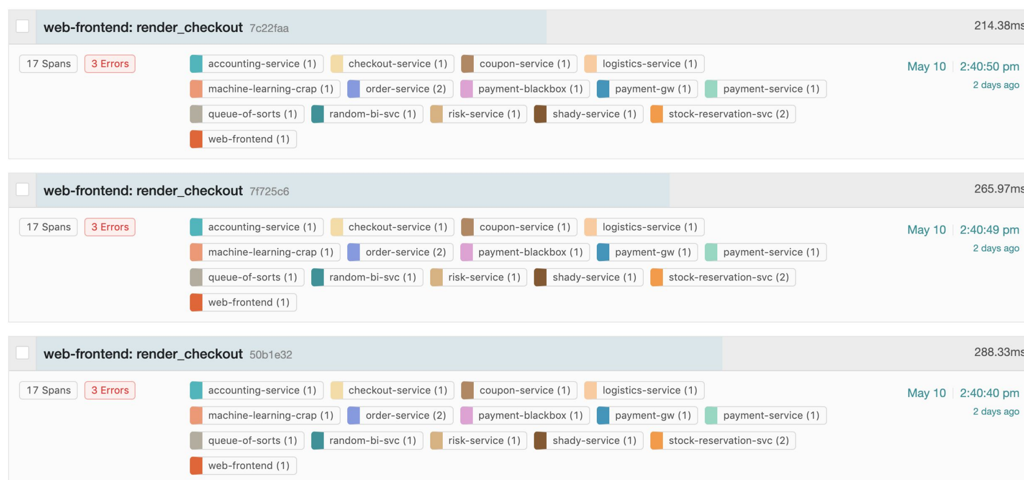
component: **checkout_service** && operation: **place_order**

Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

Assume that you configured an alert for your place order operation which has an SLO of 99.9 success rate.
A typical way to catch those would be some sort of criteria like "component: checkout_service && operation: place_order" - this is where we want o measure customer pain - the symptom
How to configure the alert itself depends on the tool you'll use. There are many open source and commercial solutions for it. That's not in scope of this talk.
Adaptive Paging is an alert handler and we're going to focus on that.

ALERT PAYLOAD

DeliveryHero Tech

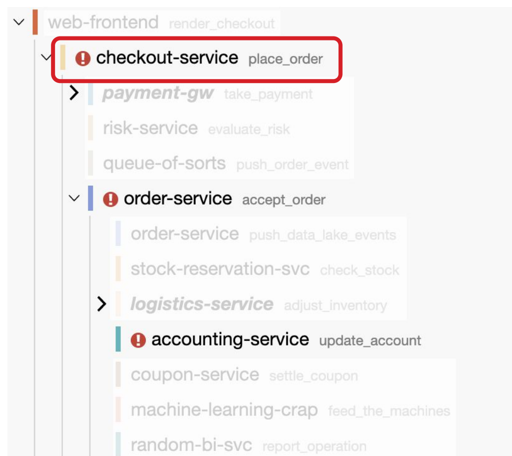


Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

The first thing Adaptive Paging does is to collect exemplars. The collected exemplars will be representative of the situation and that's the data that will be analyzed.
This can be done already by your OT tool and be part of the alert payload. If it isn't, you should be able to query the OT backend for traces that match the same criteria of the alert rule during the time of the incident.

WALKING THROUGH A TRACE

DeliveryHero Tech



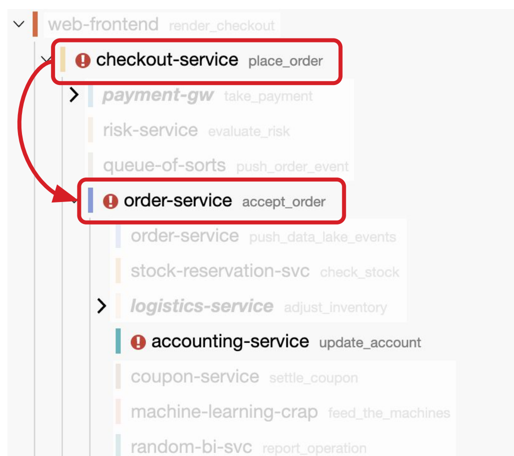
1. Starting at the span which was defined as the signal - **place_order**

Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

For a given trace, starting at the span which was defined as the signal, AP walks through the trace in a recursive way.
For every child span, its attributes and respective values are checked to decide which path to take.
In the example, the payment gw, risk svc or the queue operations are not tagged with errors.

WALKING THROUGH A TRACE

DeliveryHero Tech



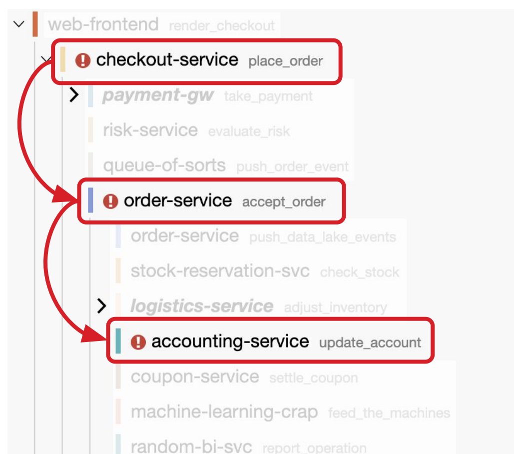
1. Starting at the span which was defined as the signal - **place_order**
2. Inspect every child span's attributes
3. Follow path with failed operation **status.code=Error**

Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

The accept_order operation in the order-service is. This allows the algorithm to take that path.
The same process is repeated. None of the operations of the order-service, stock-reservation-svc, logistics-svc or the others which were triggered by accept_order were tagged with errors.

DeliveryHero Tech

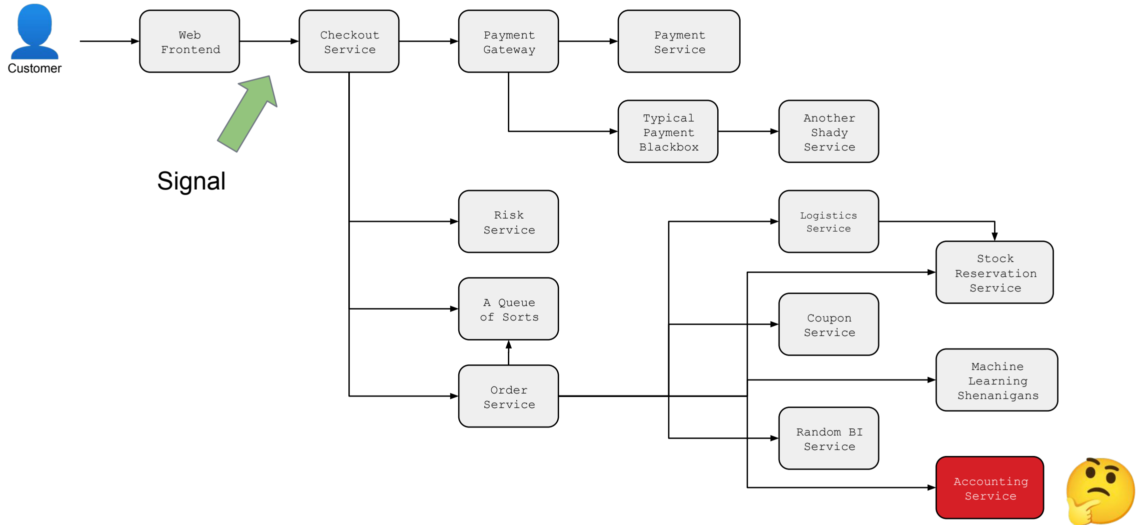
WALKING THROUGH A TRACE



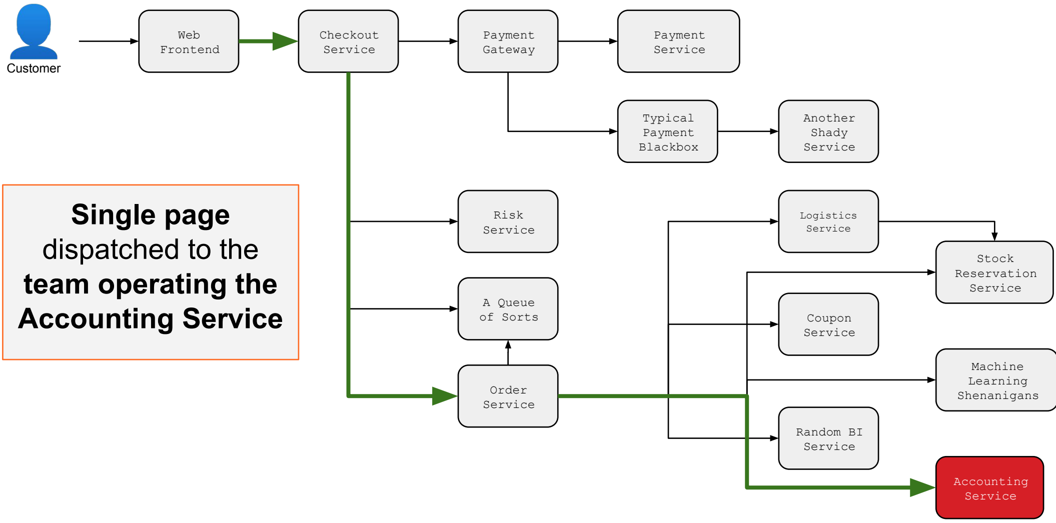
1. Starting at the span which was defined as the signal - **place_order**
2. Inspect every child span's attributes
3. Follow path with failed operation **status.code=Error**
4. Rinse and repeat until no more child spans

Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

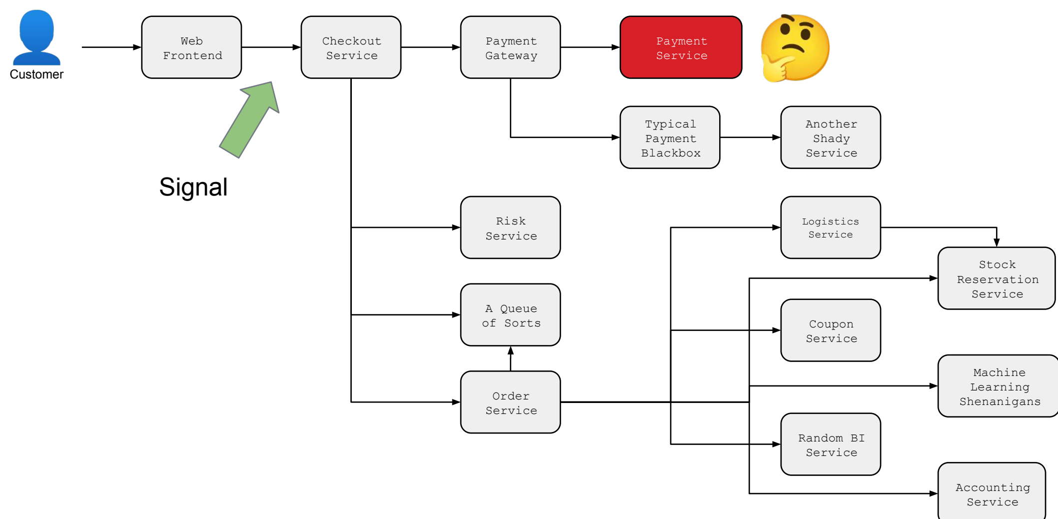
Except the update_account operation in the accounting-service.
...
Without any child spans to continue the traversal, the accounting-service is selected as the most probable cause.
Having the operation which is identified as the probable cause, the only thing left is to map the service to the respective team or on-call escalation
Let's revisit the previous examples to see how it'd work.



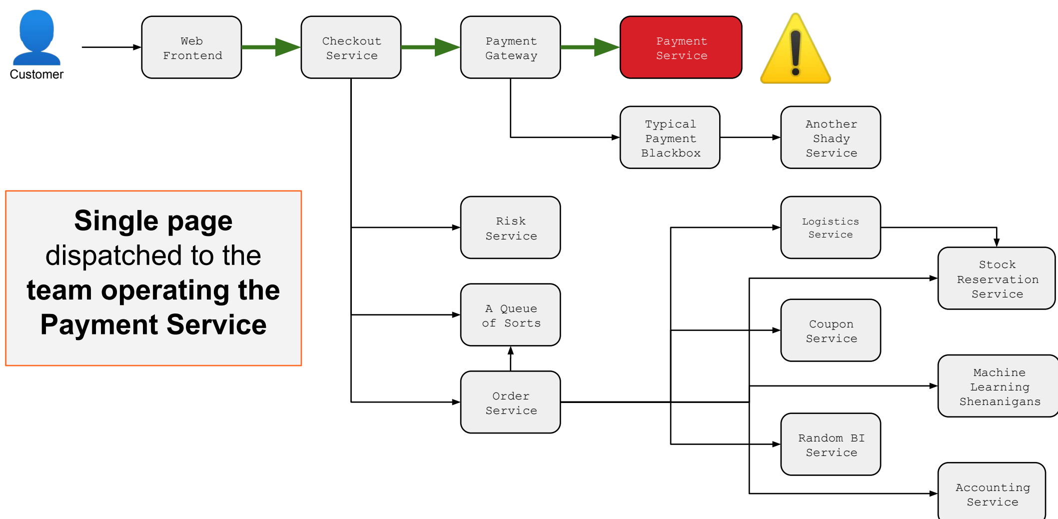
The signal is captured exactly in the same place - where it should. Where the signal-to-noise ratio is optimal, and as close as possible to the customer pain. What happens in this case if the Accounting Service has an outage?



Using adaptive paging as the alert handler, it will instead page the Accounting Service team.



What happens with this approach if the Payment Service has an outage?



In this example, the alert rule is exactly the same. The alert handler will, instead, identify the Payment service team as the one to be paged.
No more page concentration.

ADAPTIVE PAGING

DeliveryHero Tech

Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

We think this is a smart way to solve this particular problem.
It did not happen without challenges...

CHALLENGES

DeliveryHero Tech

- Multiple child spans with error:
  - Follow each path, attribute the probable cause a score
  - Analyze more exemplars and adjust the scores
  - Worse case scenario, page multiple probable causes and measure how often this happens
- Missing instrumentation or circuit breaker open for RPC
  - Use the **peer.service** and **span.kind=CLIENT** attributes to suggest which dependency would be the (missing) target
- Mapping services to on-call escalation
  - Owning team may not have their own on-call escalation. Fallback to closest

Delivery Hero SE This document is strictly confidential and may not be copied, used, made available or be disclosed to third parties without prior written permission.

The detection algorithm can adopt many different strategies. Our current implementation uses a couple of heuristics that are easy to reason about, like when we find multiple child spans tagged as errors. Paging 2 teams is probably still better than paging everyone.
The biggest problem we've found was missing or poor instrumentation. Having our own semantic conventions helped.
For ex., the usage of the peer.service attribute is a good incentive for the target team to put some effort to avoid being the page recipient.
Another one was mapping services to an on-call escalation. We added that information to our application registry.
Finding probable causes due to Latency is a challenge of its own.
We've also started looking at some excellent work from LinkedIn - MonitorRank from 2013, which fits nicely into AP

Photo by Patrick Tomasso on Unsplash

Observability still has ways to go. Monitoring and Alerting are not exactly the same thing.
Critical User Interaction (CUI) is something going on at Google today - cool stuff very similar to these concepts
With this talk we hope to contribute to improve the alerting situation, in particular paging alerts that burnout humans.
Are we all on the same page?

DeliveryHero
Tech

THANK YOU

Luis Mineiro
@voidmaze@techhub.social / @voidmaze

QUESTIONS?