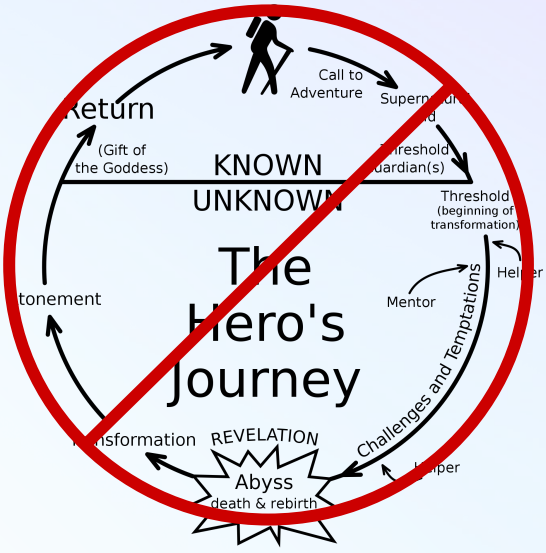
Storytelling as an incident management skill

Laura de Vesine
silverrose@datadoghq.com

DATADOG

What's "Storytelling"



So, depending on how much time you've spent thinking about stories, you might be familiar with Joseph Campbell's concept of the "hero's journey" and the claim that this is how stories go. <click> this isn't what I'm talking about when I say "storytelling".

Image source: Wikipedia (https://en.wikipedia.org/wiki/Hero%27s_journey)

Narrative

"A narrative is a story, an account of a string of events occurring in space and time. They do not unfold randomly, but rather as an ordered series of events connected by the logic of cause and effect."

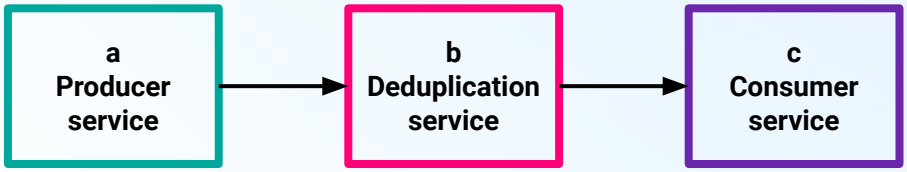
I actually want to talk about a more fundamental storytelling skill: how to compose and communicate a narrative that contains an ordered series of events, connected by cause and effect. A well constructed narrative of an outage both makes more sense to whoever you are communicating with, and can help *you* better analyze the logic of your system and how it is failing.

Narrative definition from
https://www.studiobinder.com/blog/what-is-a-narrative-definition/

Cat with book Photo by Klaudia Ekert:
https://www.pexels.com/photo/photo-of-cat-standing-on-top-of-a-book-2383122/
Kitten with book Photo generated by DALL-E

An Incident Narrative

- The **b** service deduplicates data between the **a** and **c** services
- The **b** service deduplicates in memory using static routing
- The **c** service stopped receiving data around <time>
- 1 instance of the **b** service got overloaded and started crash looping at the time of failure
  - This single instance overload caused global failure
  - Because the **a** service blocks on unsendable messages
- We mitigated by scaling up the overloaded **b** service instance
- We'll be following up long term on dynamic sharding designs and a dead-letter mechanism for the **a** service

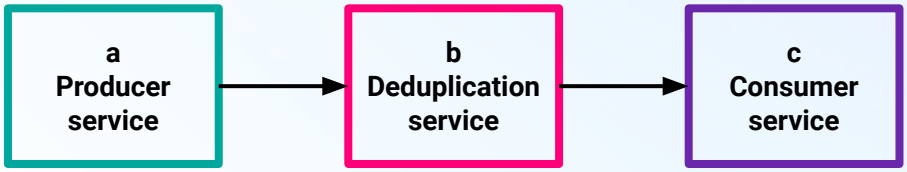


So, here's an example of a narrative that we might create about an incident. <click twice>. The first thing we do is set the scene – we describe some services, how they function, and how they interact: a calls b, which deduplicates data, and sends it to c. <click> then, we have an event: the c service stopped getting data. But importantly, this event *isn't* the first element in the logical chain. Instead, we want to extend backwards in time to look for that logical progression. This also winds up extending "up" the call stack in this case: <click> first from the c service up to an issue the b service, <click> seeing that a limited issue in the b service had global impact <click> and explaining that global impact with the behavior of the a service. This gives us a logical progression of the actual incident – what is our story of cause and effect within the setting of our services. <click> extending forward in time and logic, what did we do about it, both in the short term to mitigate by scaling up, and in the long term to prevent recurrences by fixing the failure modes in our systems.

Notice that this narrative is light on "characters" – but it does have action by "us", which is to say incident responders. There's other ways we could order this story of our system, and we could add more details depending on our audience, but the important thing I'm focusing on here is that we're telling the logical progression of events in the system (and not, for example, the series of actions taken by responders).

The Wrong Kind of Story

- The **b** service deduplicates data between the **a** and **c** services
- The **b** service deduplicates in memory using static routing
- We were paged around <time> because the **c** service was failing
- We upscaled the **c** service but this didn't help
- Then we found a metric that said the **b** service was overloaded
- We tried upscaling the **b** service but it didn't work
- We also had to restart the **a** service to get it to re-send stuck messages
- Customers could see results again at <time>



Here's another version of the same story that you might have seen instances of before.
<click twice> We start exactly the same way with setting the scene.

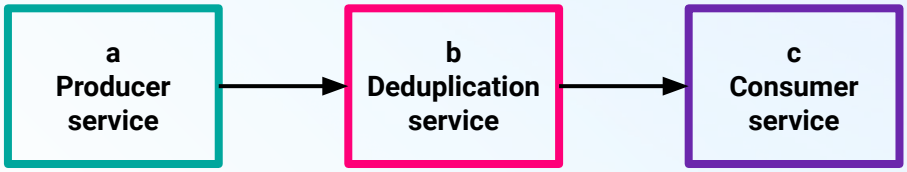
Then, <click>, instead of beginning with *what happened to the system*, we start with what people did – that's usually "got paged". <click> and then the narrative continues following what the incident responders did. <click> there's very little structure here explaining the *logic* of events. We did this, then this, but why did we take any particular action? <click> Even more importantly, why did any given action help or not help? What was happening to the actual system, either to cause the incident or to cause recovery? <click> a very, very attentive reader could maybe work out what was happening in the system that led to this incident, but it's extremely hard to follow. <click> and because we don't know what happened to the system itself, it's quite hard to see what possible remediations we could try besides "get better at troubleshooting".

By some conventions of storytelling this is a better "story" – it's got characters driving the action, and it's in the active voice! But the logical narrative is missing, and that's the crucial communication for incident storytelling.
You *can* tell the story of your troubleshooting and make it a logical narrative, by carefully laying out what data you saw at any given point, what hypothesis that led you to, then what action you took, the results of that action, and what you concluded about your hypothesis from that action. That's going to take longer, and be less "crisp" for your audience, but it's a useful technique if you really want to emphasize why it

was hard to come to the right mitigation… or you're still in the middle of the incident, and trying to catch others up on the troubleshooting so far to get them up to speed

The Story as it Happens

- We were paged around <time> because the **c** service was failing
  - We thought it might be overloaded based on <graph>
  - So we tried upscaling the **c** service but throughput didn't increase
- Then we saw that data wasn't actually being sent from the **b** service <graph>
  - One pod was OOMing and crashing, so we tried to upscale
  - The crashing stopped but progress didn't restart
- The **a** service also seems stuck <graph>
  - We know from experience that **a** won't write to **b** unless all pods are up
  - So we restarted **a** to get it to resend stuck messages

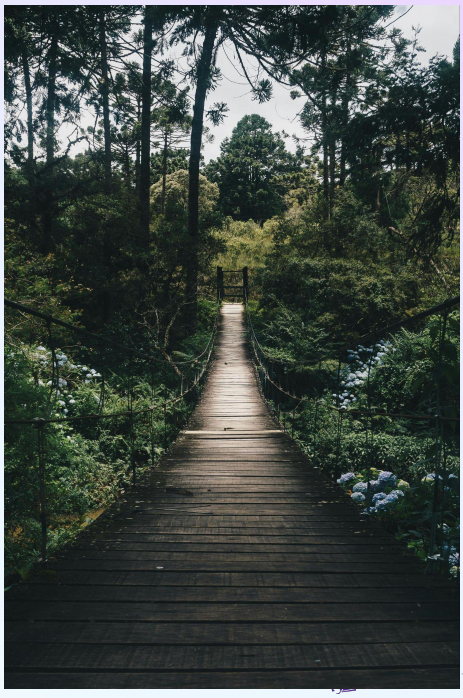


Here's that same story one more time, as I might tell it during an incident – <click> here, the first thing we say is usually "why am I here?" – establishing a shared setting is something that happens before and after the incident, not during it. <click> Based on the initial page, what was my first hypothesis of the incident: the c service was overloaded <click> and my action based on that hypothesis, and the result. <click> Rejecting that answer for system behavior let us look for another based on telemetry <click> and more investigation, a new hypothesis (the problem is upstream) and action <click> and a result. Notice again that the logical progression here is clear – both why we took particular action, and what the results and conclusions from them were. we don't have to belabor every point (the audience is also smart engineers usually!), but explaining the narrative of actions as we take them makes it easier to figure out next steps, and to understand what *isn't* happening. <click> Having been clear about *why* we took particular actions, it's easier to pinpoint where to look next for mitigation and remediation – if we restarted a service because it was OOMing, and that is resolved but the incident is still happening, we need to look elsewhere <click> and maybe draw on our knowledge and experience about the service to propose a possible story of why it's behaving the way it is <click> and take logical actions instead of flailing uncertainly.

So that's the basic concept of a narrative as part of an incident, and you can already see how having one is helpful. We also tell stories before our incidents, especially to create that shared setting

Storytelling in Oncall Prep

- What does our system do?
  - What does each service do?
  - How does a request flow?
- How has our system broken in the past?
  - What kinds of things page us?



The same setting that is useful in the incident narrative – what our system does, and how it does it – tells us both when we should be paged (when the system isn't doing the things it's supposed to), and begins to prepare us for diagnosing outages in our services. In fact, you can think of (good) SLOs as a more formal description of "what our system does". Making sure that everyone on your team has a similar picture of "what our system does and how", including how it breaks, means that you can all communicate with the same assumptions during an incident.

Forest Photo by Kaique Rocha:
https://www.pexels.com/photo/black-hanging-bridge-surrounded-by-green-forest-trees-775201/

Empathy and Pager Stories

- I got paged at <time>
- Wasn't really sure what to do
- Took some deep breaths and got help
  - Specific techniques
- Members of other teams were really helpful

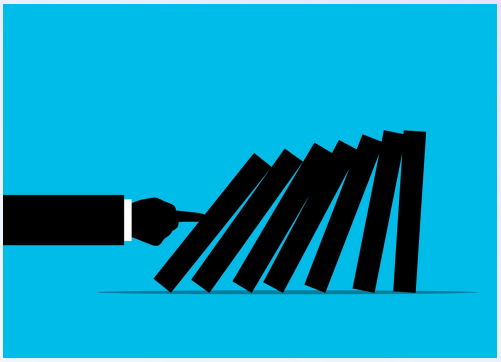
- One time, I really messed up in production
- But our culture is blameless, so instead of being yelled at, here's how we fixed things…

- I got paged at <time>
- It turned out this wasn't really an emergency
- Sucked to be interrupted, but they were having a tough time

Oncall prep is a place where we also want to tell more character-driven stories about incidents. Instead of sharing the story of what happened in the system, we share the story of *what it felt like to get paged* as a way of building both empathy with each other. Rather than being about a logical progression, these are stories about feelings we had, to share and normalize them. We can use these kinds of empathy-generating stories to teach people good judgment on when it's appropriate to page a team vs. not, how to work with someone that you've just paged (or who just paged you), and to get more comfortable with the responsibility of being oncall. Some good prompts for stories like this include "what's the most scared you've ever been in an incident", and "what's the worst thing you've ever broken, and then what happened"

Pager beep icon created by Freepik - Flaticon
https://www.flaticon.com/free-icons/beep
Multitasking man via Pixabay
https://pixabay.com/illustrations/multitasking-efficiency-2840792/

Wheel of Misfortune



A final way that storytelling helps us with oncall prep is by supporting great wheel of misfortune exercises. If you understand your incident (or your page) as a story that includes the logical progression of “how it broke”, and you further understand your system as your story setting in great detail, it’s easy to run a wheel of misfortune tabletop with no additional prep. Introduce your victim to the initial thing they are paged for (that you were paged for), and because you understand the series of events in your system, even if your victim takes a different road of debugging and response than you did, you should be able to reasonably extrapolate what evidence they will see in the system and what impact attempted mitigations will have.

Meeting chickens image generated by DALL-E
Dominoes Image by Mohamed Hassan from Pixabay

During an Incident

DATADOG 11

Storytelling can also support incident mitigation and resolution by helping us see the logic of a failure. You can think incident incident communication as a “collaborative storytelling” exercise – the group of responders aims to explain the logical progression through the system of a trigger. Focusing on the logical narrative (“why would a lead to b”) can help “unstick” responders who aren’t sure how to remediate during an incident.

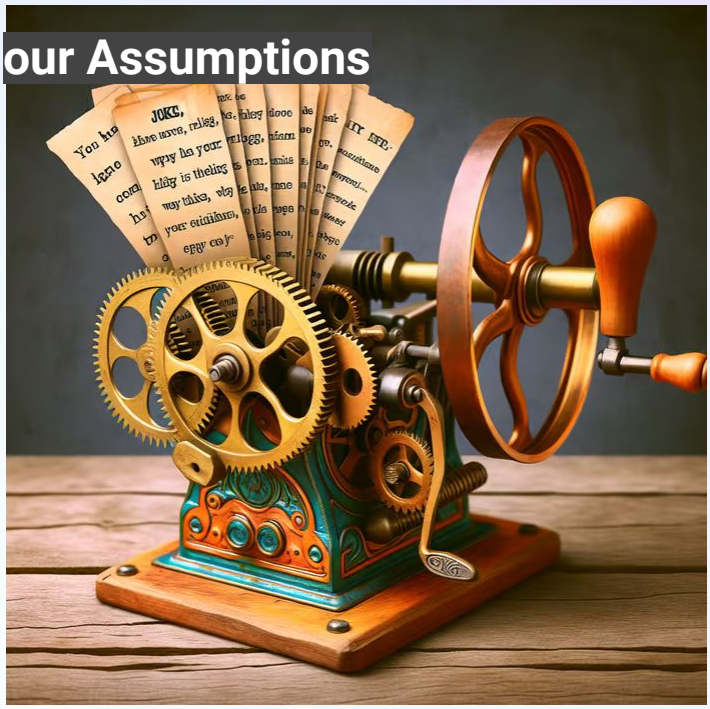
For instance, we have a pipeline that takes in customer data, performs various transformations (including some specified by customers via regex), and eventually writes the data for querying. An engineer was paged because this pipeline was lagging. Initially it seemed to be only one pod that was stuck, so the responder directed traffic away from that pod. This didn’t help – lag continued to increase.

The engineer was now at a loss – was the problem reading from kafka? Writing downstream? Something else?

Falling back to the story of “how does the pipeline do it” helped responders become unstuck – particularly, each service reads from kafka up to a certain number of bytes, but will pause further reads until it can commit results downstream. This led responders to discover that no results were being committed from the affected pods, and furthermore that CPU was completely utilized in those pods. The conclusion was that a bad regex was causing issues and delaying processing, and with that information the bad regex could be disabled.

Image by DALL-E

Checking our Assumptions



Another example of how a focus on the system narrative supports better incident handling is that is allows us to check our assumptions when mitigations don't function as expected.

For example, we have a query system that reads and filters data from file storage with several layers of caching. This service implemented a change to read files only once and multicast them to many processors to save on network and read costs, but this had the impact that certain queries caused memory overloads. When the team saw an incident where the cache layers were OOMing, they assumed it was this usual pattern – queries with many results causing overload because of the file processing in parallel.

However, scaling up the cache didn't help for this incident. Responders used the well-understood story of how the parallel file processing led to overload to check each part of the system for the expected behavior: are the errors in our cache layers, or the service behind the cache? Are we reading particularly large files, or processing especially parallel queries? Instead of an issue with data being read and over-parallelized, in this case they discovered a pathologically large query being run by automation that could then be blocked.

An Engaging Postmortem

1. **Set the scene**

DATADOG 12

Checking your assumptions is even more relevant when writing a postmortem: we can't solve the problem in the long term unless we really understand the progression of how the system broke. We also want our colleagues to be able to read and understand the postmortems we write (and ideally find them interesting!) – something very much supported by having a good narrative. Here's a good outline to think about

1. Set the scene. Share the services that are involved, what they do, and how they function. This is also a good place to include some introductions to the team, and local operating conditions – things like "the team has seen this monitor generate a lot of false positives" or "most team members are new to this service" might be useful to include in your setting for the postmortem

Image from Kotomi_ via Flikr:
https://www.flickr.com/photos/kotomi-jewelry/5362413718

An Engaging Postmortem

1. Set the scene
2. **Add some drama**

DATADOG 13

Now that we have a setting, we want to add some drama (really!). This can take one of two forms:

(a) Share the impact from the incident (“we were paged because users could not load the website”)

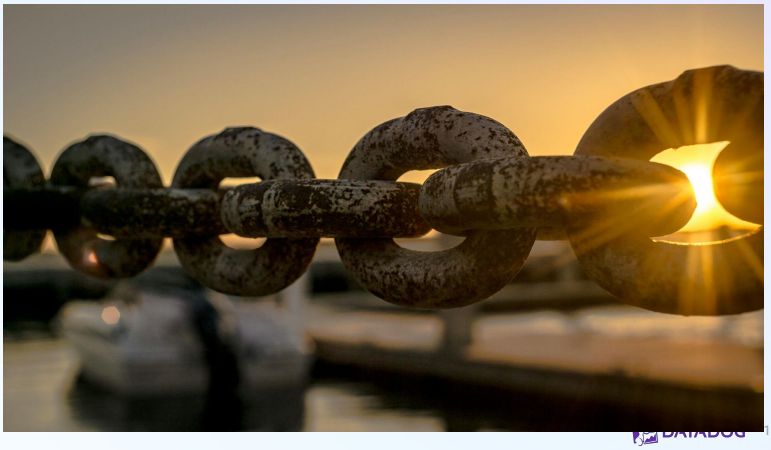
(b) Discuss an underlying flaw in the service (“unbeknownst to us, the foo service has the property that attempting to post rickrolls causes pods to crash” or “when we originally built this service, we isolated each customer to their own shard. We knew this had a chance to cause hotspotting, but judged it a minimal risk at the time”)

Personally, I prefer the second option – when I start my story with what I got paged for, I’m tempted to tell the story of how our troubleshooting went, instead of the story of the service. That can look a lot more like this:

Photo by Mike Bird:
https://www.pexels.com/photo/red-human-face-monument-on-green-grass-field-189449/

An Engaging Postmortem

1. Set the scene
2. Add some drama
3. **Chain events together**

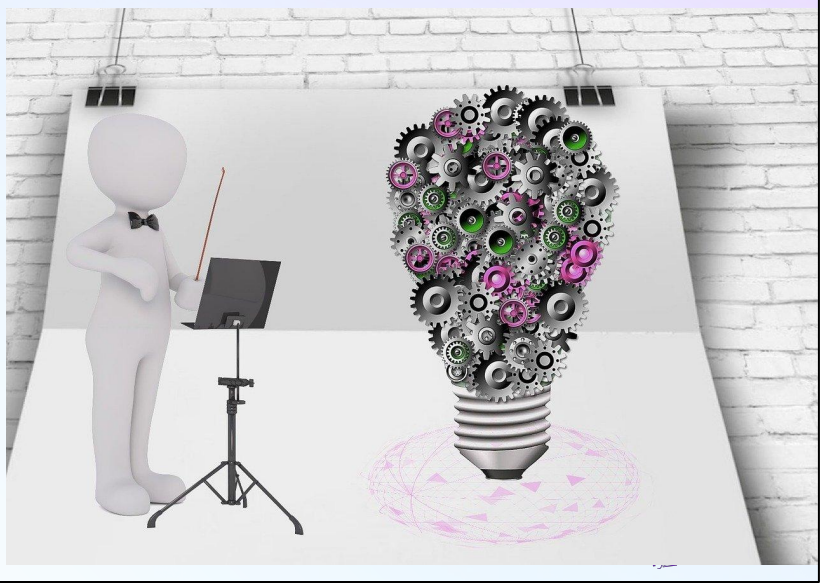


The next piece of an engaging postmortem is to lay out that logical narrative – start with the trigger of the incident, and walk the reader through the steps of how it affected the system, one logical progression at a time.

Photo by Joey Kyber:
https://www.pexels.com/photo/selective-focus-photoraphy-of-chains-during-golden-hour-119562/

An Engaging Postmortem

1. Set the scene
2. Add some drama
3. Chain events together
4. **Explain the response**

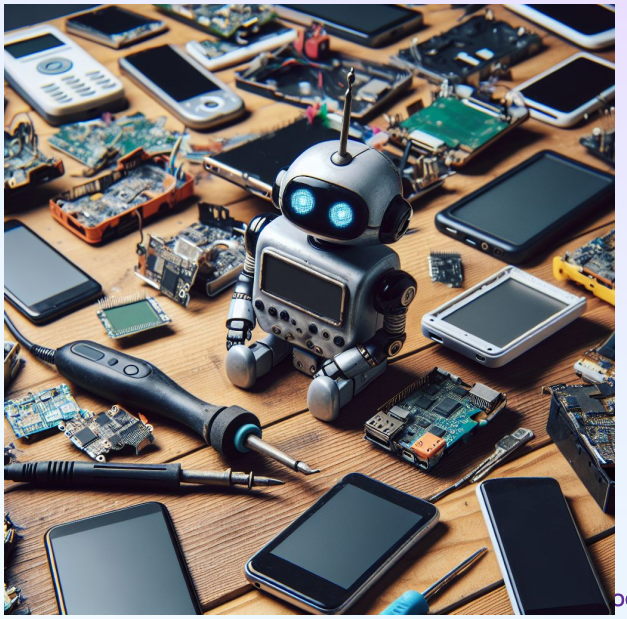


It *is* important in many incidents to document how we responded – both the red herrings and cul de sacs, as well as the response(s) we took that actually mitigated or resolved the issue, and *why those were logical based on what we knew*. Incidents give us a chance to fix lots of things – not just our underlying systems, but also our understanding of those systems (our setting), and our monitoring of those systems. Walking through the logic of the response helps us do that. (But, for my money, if I can only have one… I'd rather have the chain of events in the system and miss the troubleshooting narrative entirely)

Image by Mariana Anatoneag from Pixabay

An Engaging Postmortem

1. Set the scene
2. Add some drama
3. Chain events together
4. Explain the response
5. **Plan fixes**



Finally, of course, all good postmortems end with action items. I want to once again reflect that having the logical progression of what happened in the system makes it much easier to see what action items will actually prevent your incident in the future – without that progression, we're stuck trying to prevent triggers or just "troubleshoot better". With it, we can look at each step of how a trigger turned into an incident and try to halt future incidents at (potentially) each and every one of those steps.

Image by DALL-E

Questions?