DevOps: Шаблон Post Mortem

https://incident.io/blog/incident-post-mortem-template

Key information

The key information section is a recap of the data behind the incident. It’s the most succinct summary of the incident, which is useful to orient anyone reading the remainder of the document.

Think of this as the metadata of your incident. Core information such as incident type, severity, leads and more should be outlined here.

Here’s what this section could look like:

```
□□ Tagged Data!

    Incident Type: Platform
    Severity: Major
    Affected services: Login, Cart, User Preferences

□ Team

    Incident Lead: Chris Evans
    Reporter: Steven Millhouse
    Active participants: Rebecca Song
    Observers: Michael Scott

□ Useful Links

    Jira issue
    Slack channel

□□ Key Timestamps

    Impact started at: November 10, 2022 2:12 PM
    Reported at: November 10, 2022 2:12 PM
    Identified at: November 10, 2022 2:18 PM
    Fixed at: @November 10, 2022 2:20 PM (UTC)
    Closed at: November 10, 2022 2:30 PM
```

Incident summary

The summary should be a concise and accessible overview of the incident. It’s important that this summary is exactly that, a summary. If you try to throw everything in here, it becomes very difficult for someone to come in later on and easily parse what happened.

Your incident summary should clearly articulate what happened, who was involved, how you responded, and the overall impact of the event. It’s helpful for the summary to be written such that someone who wasn’t there can use it to develop a high-level understanding of the situation.

We find it useful to pitch it so your boss’s boss would understand!

Here’s a real-life incident post-mortem summary we wrote up internally:

```
 On 2022-11-18 from 15:39 to approximately 16:23, an asynchronous event caused an error which took one of our Heroku "dynos" down (referred to as servers for simplicity below). Our infrastructure logic is such that if an event has an issue being processed, it retries. In this case, it meant that another server then attempted to process the same malformed/dangerous event, before also crashing. This repeated until all our servers were down, at which point our app became completely unavailable. This type of error is commonly known as a "poison pill".

The poison pill was caused by a consumer that returned an error, which wrapped a nil error. This meant our consumer tried to unwrap the error and got a nil pointer dereference when trying to access the root cause. Normally this would be fine, however only the actual app-code section of our queue consumers were within the panic recovery boundary, so by panicking in the pubsub wrapper, we'd crash the entire app.

We quickly rebooted the servers, but unfortunately, the event was retried again, taking down each server in quick succession after a few minutes. This continued for around 25 minutes.

We deployed a fix to make sure all of the code in our consumers were covered by a "recover" statement (meaning we handle errors like this more gracefully), which then quickly caught the error once we deployed it and also prevented it taking the app down again.

From there, we fixed the bad code and stayed online. We also made a number of other improvements to our app and infrastructure setup.
```

Incident timeline

The incident timeline exists to provide a narrative of the incident; essentially retelling the story from start to finish. It should outline key events and developments that took place, investigations that were carried out, and any actions that were taken.

It’s tempting to go into huge amounts of detail here, but we’d typically advise the detail to remain in an incident communication thread (i.e. a Slack channel), and have the timeline contain just the significant events and developments.

Typically we’d include the timestamp (in the dominant timezone for the incident or UTC if spanning many), the time since the notional start of the incident, as well as any details about what was happening.

```
Timestamp   Event
12:00:00    Chris reported the incident
12:07:00 (+7mins)   Incident lead assigned
12:10:00 (+10mins)  Issue identified
12:12:00 (+12mins)  Incident closed
```

Contributors

We think it’s helpful to enumerate the contributors of an incident, where contributors are thought of not as ‘root causes’ but as a collection of things that had to be true for this incident to take place, or that contributed to its severity.

This can include technical contributors (e.g. the server’s disk filled up), human contributors (e.g. the on-call engineer did not respond the first time we paged them), and any other external contributors (e.g. we were simultaneously running a marketing event).

It’s useful to enumerate these items to fully explore the space of the problem, and avoid overly fixating on a singular cause.

Mitigators

Where contributors constitute the set of things that helped the incident occur, mitigators can be thought of as the opposite. More precisely, they’re anything that helped to reduce the overall impact of the event.

Like contributors these can come in many guises, like the incident happened during working hours, or the fact that the person who knows the most about the system at the heart of the incident also being on call.

It’s useful to explicitly call these out, as they often highlight positive capacities that are helpful to double down on, or socialize further.

Learnings and risks

This section is a little harder to capture, but it should answer questions like: what did we learn, how can we streamline our response in the future, and what broader risks did this incident point towards?

For example, you might have learned that one of your teammates was the only person who knew the details of the system that failed during this incident, and that might point at a more general “key person risk.” If other incidents point at similar risks, that’s useful learning for teams.

Follow-up actions

Follow-up actions are here to convey what’s being done to reduce the likelihood and/or impact of

similar events in future.

In the context of this document, we think it’s most useful to highlight any key actions items or deliverables, rather than the full set. Ultimately, this will vary depending on the intended audience, but if the document is being written to be read and learned from, this section should close the loop on what’s being done to mitigate the major themes that have been identified.

Optional: Post-mortem meeting notes

Many post-mortems can happen async but others may require a dedicated meeting to talk through what happened—especially incidents that were of higher severity. In cases like these, it’s a good idea to set up some time with incident response team members and relevant stakeholders and use the post-mortem as your meeting agenda.

While it may seem a bit redundant, having the space to have a retrospective and talk through issues in real-time ultimately encourages better team collaboration and communication that’s nearly impossible to capture over an async document.

It’s a good idea to brainstorm and collect talking points to help guide your post-mortem analysis meeting. In the end, an effective-post mortem meeting ensures that your team is rallying around tactics that will drive continuous improvement of your response processes and ultimately your product.

post-mortem