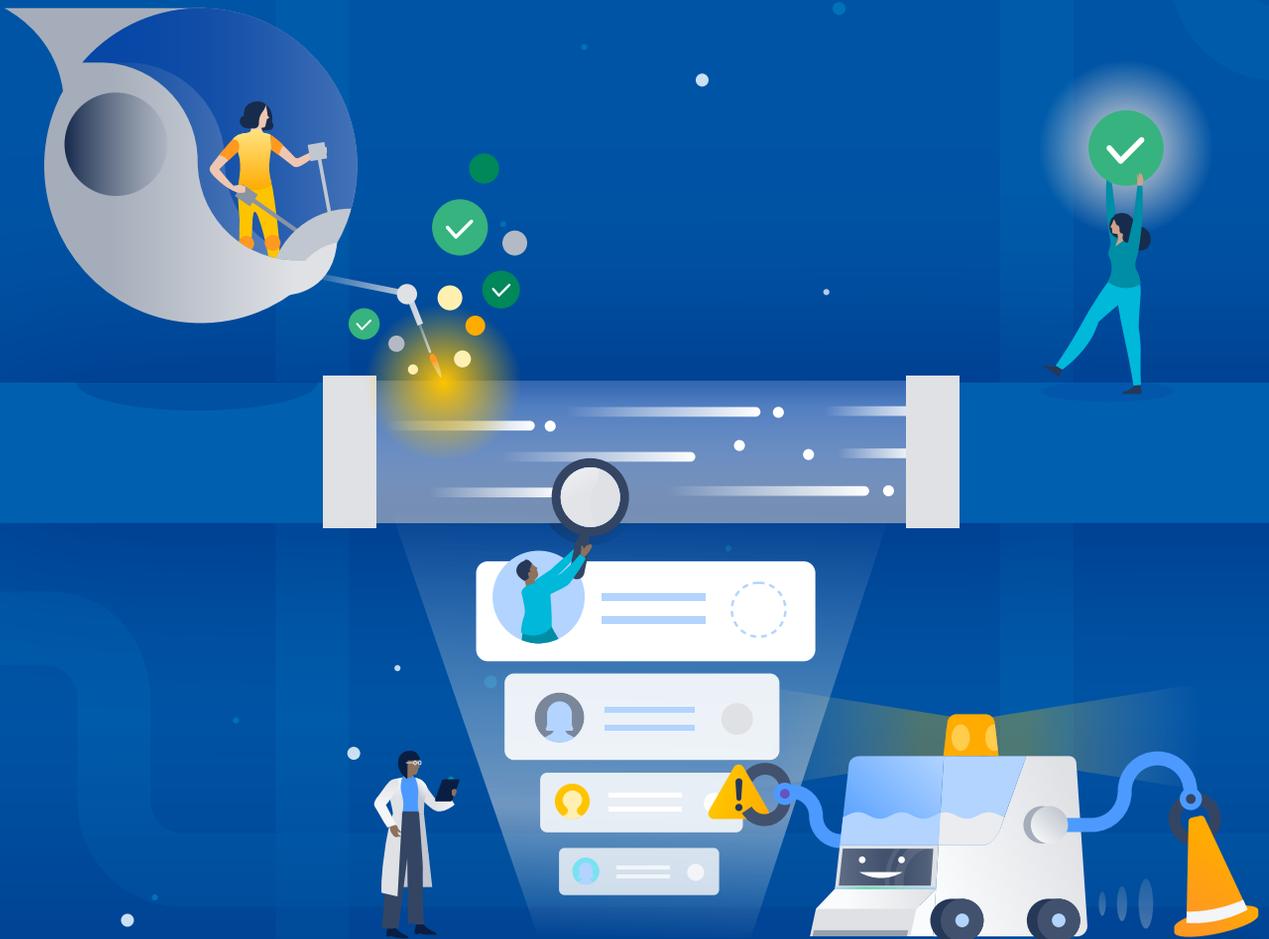


# Incident Management Handbook





# Table of contents

<b>5</b>	<b>Foreword</b>	<b>41</b>	<b>Chapter #3: Incident postmortems</b>
	Or: How I learned to stop worrying and love on-call	42	What is a postmortem?
		42	Why do we do postmortems?
		43	When is a postmortem needed?
<b>11</b>	<b>Chapter #1: Incident overview</b>	43	Who completes the postmortem?
12	Overview	44	Why should postmortems be blameless?
12	Who is this guide for?	45	Postmortem process overview
13	What is an incident?	47	Postmortem issue fields
13	Our incident values	54	Proximate and root causes
16	Tooling requirements	56	Categories of causes and their actions
17	A note of incident tracking	57	Root causes with external dependencies
18	Incident manager	60	Postmortem actions
		62	Postmortem meetings
<b>19</b>	<b>Chapter #2: Incident response</b>	65	Postmortem approvals
20	Detect	65	How are postmortem actions tracked?
22	Open communications		
25	Assess	<b>68</b>	<b>Major incident manager cheat sheet</b>
27	Send initial communications		
29	How well are you communicating to customers during an outage?		
31	Escalate		
31	Delegate		
33	Send follow-up communications		
35	Iterate		
36	Resolve		
38	A well-written incident report		



# Foreword

Or: How I learned to stop worrying and love on-call

**BY JIM SEVERINO, DEVELOPMENT SR. TEAM LEAD, ATLISSIAN**

In early 2014, driven by the broad industry shift to SaaS, the Atlassian Cloud suite was evolving from a cobbled-together set of our downloadable software into a true “cloud native” system. This was a big change that required careful planning and execution across most of the company. For eight months, a team had been working to make user and group management easier for our cloud customers by unifying them across all products. So, instead of having to manage users and groups separately in each product, there would be only one set per customer and one interface to manage them with.

**“ Things looked good at first...but then an interesting support case came in.**

Eventually we were ready. Over several weeks, we gradually rolled the change out to all customers. Things looked good at first...but then an interesting support case came in. A customer claimed that their users were able to see content that they shouldn't have access to. Upon investigation, we realized that in some edge cases the rollout had reconfigured customers' groups in a way that removed some existing access controls. This allowed some of those customers' users to access content that was supposed to be off-limits to them.

We immediately recognized this as a very serious problem and jumped on it right away. Although it affected a relatively small number of customers, given the impact we treated this problem as our highest priority. We released a fix in short order...at least we thought it was a fix (and I bet you can guess what happened next). It turned out that the “fix” actually made the problem worse: We had inadvertently broadened the impact by an order of magnitude. Now, instead of a few users or a few customers being able to see content they shouldn't have had access to, we had thousands of people across hundreds of customers in this situation. It was a very hairy knot of access control misconfiguration, and some of our best customers were understandably furious.

## “ Once the incident was resolved and the dust had settled, we started asking questions: What went wrong that led to this?

The ensuing incident took weeks of work from many teams, entailed several all-nighters, and pulled in staff from engineering to public relations, and everyone in between. Existing plans were trashed, vacations canceled, and meeting rooms cordoned off as “war rooms” for weeks on end. Once the incident was resolved and the dust had settled, we started asking questions: What went wrong that led to this? How could we have prevented it, or improved our response? Why hadn't we learned from similar incidents in the past? The answers to these questions led to the incident and postmortem processes that are set out in this book.

Many organizations discover the need for incident and postmortem processes in a similar way: They have a major incident that goes badly, and they resolve to never let it happen again. In this situation, it's tempting to design an incident process that has as its goal the complete elimination of incidents. After all, incidents are things to be avoided, right? If we had a choice, we wouldn't have them at all, so why shouldn't we try to prevent them all from happening?

In an attempt to eliminate incidents, organizations often introduce safety gates, checkpoints and other overheads to their software design, development and release processes. They try to prevent or block incidents by introducing things like change review boards and weekly release meetings with the intention of carefully scrutinizing every change in detail so that no bugs are allowed to slip through. This is an understandable reaction to incidents, but it's not the right reaction. Change gates and checkpoints slow the organization's rate of change down, which does tend to reduce the rate of incidents in the short term, but more importantly reduces your momentum. Often backlogged changes pile up behind the gates you constructed, leading to bigger batches of change less frequently, which makes them more (not less!) risky. The net effect is that your company is unable to make changes as fast as it was before, there are more overheads, and you still have incidents. This situation tends to frustrate everyone involved. The pendulum has swung from unawareness to over-caution.

The sentiment—wanting to reduce the impact of incidents—is correct, but aiming for zero incidents is a mistake. The only way to completely prevent incidents is to stop all changes. But organizations need to make changes in order to compete and survive, and all change entails risk—we aim to reduce risk in ways that let us continue to pursue progress. It's the same reason we invented air bags and smoke

detectors rather than stop making cars and buildings. The ability to make more changes with the same or less risk is a competitive advantage. We can adopt practices like canaries and feature flags that reduce the risk per change, but some risk always remains—it's never completely eliminated. "Risk tolerance" is the level of risk that's appropriate for your organization and its activities, and it varies depending on the nature of the business and services. The more changes an organization can make within its "risk tolerance", the more competitive that organization can be. This is expressed well in the Google SRE book's Chapter 3, "Embracing Risk."

**“ The sentiment—wanting to reduce the impact of incidents—is correct, but aiming for zero incidents is a mistake.**

So change means risk, and risk means incidents. If incidents are a fact of life, then, what should we do? Do we just accept them?

In a way, yes. You need to accept that in the presence of risk, incidents are bound to happen sooner or later, and having an incident doesn't mean that you have permanently failed. An incident should properly be regarded as the beginning of a process of improvement. Your goal should be to do that in the best possible way. Once you understand that, you can focus on two important things that are entirely in your control: How well your organization responds to incidents, and how well you learn from them.

A good incident process is fast and predictable. It quickly turns detection into response, escalates to the right people on the shortest path, makes communications clear and keeps customers in the loop. It's simple enough for people to follow under stress, but broad enough to work for the variety of incident types you will encounter. The primary goal of an incident process is fast resolution, so speed and simplicity are paramount. The incident process set out in this book is the same process that we've continually developed and used at Atlassian over the years.

## “ A good incident process is fast and predictable.

Your incident process is the “how well you respond” half of the equation. The other half—“how well you learn”—is about your postmortem process.

The primary goal of a postmortem process is to prevent repeats by learning from our incidents and near-misses. A good postmortem follows a structured process to comprehensively review the incident from several angles, extract learnings, and find the best places to apply mitigations. Postmortems can be tricky because humans are normally pretty bad at identifying systemic causes. We naturally tend towards 20/20 hindsight, post-facto rationalization, finding fault with people and blaming them, so it's often easier (and, ironically, safer) to do this and move on than it is to honestly examine your company's shortcomings and commit to repairing them. No company is perfect in this regard (Atlassian included), but the process set out in this book is a collection of the best ways that we've found to do it.

I've heard incidents described as “unscheduled investments in the reliability of your services.” I think this is accurate. Given that incidents are going to happen whether we want them or not, your job as the owner of an incident and postmortem process is to make this “investment” as cheap as possible (in terms of reducing incident impact and duration), and to extract every last drop of value from each incident that you have (in the form of learnings and mitigations). Like so much else in the world it boils down to a matter of minimizing the cost and maximizing the benefit.

**“ A good postmortem follows a structured process to comprehensively review the incident from several angles, extract learnings, and find the best places to apply mitigations.**

I hope this book gives you high value for low cost, and that one day you too can stop worrying and learn to love on-call.



## CHAPTER #1

---

# Incident overview

## Overview

Teams running technology services today are expected to maintain 24/7 availability.

When something goes wrong, whether it's an outage or a broken feature, team members need to respond immediately and restore service. This process is called incident management, and it's an ongoing, complex challenge for companies big and small.

We want to help teams everywhere improve their incident management. Inspired by teams like Google, we've created this handbook as a summary of Atlassian's incident management process. These are the lessons we've learned responding to incidents for more than a decade. While it's based on our unique situation and experiences, we hope it can be adapted to suit the needs of your own team. You'll see throughout the book areas where we suggest some other popular options for specific practices.

## Who is this guide for?

If you're on a development or operations team that looks after services for customers who require 24/7 availability, this handbook is for you.

## What is an incident?

We define an incident as an event that causes disruption to or a reduction in the quality of a service which requires an emergency response. Teams who follow ITIL or ITSM practices may use the term major incident for this instead.

An incident is resolved when the affected service resumes functioning in its usual way. This includes only those tasks required to restore full functionality and excludes follow-on tasks such as root cause identification and mitigation, which are part of the postmortem.

The incident postmortem is done after the incident to determine the root cause and assign actions to ensure it is addressed before it can cause a repeat incident.

## Our incident values

A process for managing incidents can't cover all possible situations, so we empower our teams with general guidance in the form of values. Similar to Atlassian's company values, our incident values are designed to:

- Guide autonomous decision-making by people and teams in incidents and postmortems.
- Build a consistent culture between teams of how we identify, manage, and learn from incidents.
- Align teams as to what attitude they should be bringing to each part of incident identification, resolution, and reflection.

Stage	Incident value	Related Atlassian Value	Rationale
1. Detect	Atlassian knows before our customers do	Build with heart and balance	<p>A balanced service includes enough monitoring and alerting to detect incidents before our customers do. The best monitoring alerts us to problems before they even become incidents.</p>
2. Respond	Escalate, escalate, escalate	Play, as a team	<p>Nobody will mind getting woken up for an incident that it turns out they aren't needed for. But they will mind if they don't get woken up for an incident when they should have been.</p> <p>We won't always have all the answers, so "don't hesitate to escalate."</p>
3. Recover	Shit happens, clean it up quickly	Don't !@#\$ the customer	<p>Our customers don't care why their service is down, only that we restore service as quickly as possible.</p> <p>Never hesitate in getting an incident resolved quickly so that we can minimize impact to our customers.</p>

Stage	Incident value	Related Atlassian Value	Rationale
4. Learn	Always blameless	Open company, no bullshit	<p>Incidents are part of running services. We improve services by holding teams accountable, not by apportioning blame.</p>
5. Improve	Never have the same incident twice	Be the change you seek	<p>Identify the root cause and the changes that will prevent the whole <b>class</b> of incident from occurring again.</p> <p><b>Commit</b> to delivering specific changes by specific dates.</p>

## Tooling requirements

The incident management process described here uses several tools that are currently used at Atlassian and can be substituted as needed:

- **Alerting and on-call management:** we use Opsgenie to manage on-call rotations and escalations.
- **Chat room:** a real-time text communication channel is fundamental to diagnosing and resolving the incident as a team. At Atlassian, we use Slack.
- **Video chat:** for many incidents, team video chat like Zoom can help you quickly discuss and agree on approaches in real time.
- **Incident tracking:** every incident is tracked as a Jira issue, with a follow-up issue created to track the completion of postmortems.

 Throughout the book, we'll include a few recommendations and tips worth exploring.

### **Recommended solution:**

Many of our customers use Opsgenie at the center of their incident response process. It helps them rapidly respond to, resolve, and learn from incidents. Opsgenie streamlines collaboration by automatically posting information to chat tools like Slack and Microsoft Teams, and creating a virtual war room complete with a native video conference bridge. The deep integrations with Jira Software and Jira Service Desk provide visibility into all post-incident tasks.

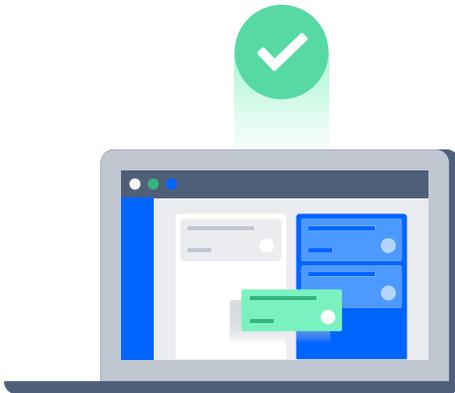
- **Documentation tool:** we use Confluence for our incident state documents and sharing postmortems via blogs.

- **Statuspage:** communicating status with both internal stakeholders and customers through Statuspage helps keep everyone in the loop.

## A note on incident tracking

Every incident is tracked as a Jira issue, with a follow-up issue created to track the completion of postmortems.

In Jira, we use a simple “Pending -> Fixing -> Resolved” workflow to track incidents. Having a simple workflow is good because we want to minimize complexity during incidents. Using a Jira ticket to track our incidents works well for us because it allows us to track incidents consistently but with a rich set of data, in a way that we can extend with automation, and with good reporting and visualization features.



## Incident manager

Each incident is driven by the incident manager (IM), who has overall responsibility and authority for the incident. This person is indicated by the assignee of the incident issue. During a major incident, the incident manager is empowered to take any action necessary to resolve the incident, which includes paging additional responders in the organization and keeping those involved in an incident focused on restoring service as quickly as possible.

The incident manager is a role, rather than an individual on the incident. The advantage of defining roles during an incident is that it allows people to become interchangeable. As long as a given person knows how to perform a certain role, they can take that role for any incident.



CHAPTER #2



# Incident response

The following sections describe Atlassian’s process for responding to incidents. The incident manager (IM) takes the team through this series of steps to drive the incident from detection to resolution.



## Detect

People at your company can become aware of incidents in many ways. They can be alerted by monitoring, through customer reports, or by observing it themselves. However an incident occurs, the first step the team takes is logging an incident ticket (in our case, a Jira issue).

We use an easy-to-remember short URL that redirects Atlassians to an internal Jira Service Desk portal. Atlassians can check if there’s an incident already in progress by looking at a Jira dashboard or our internal Statuspage. Teams such as our customer support teams have dashboards set up at well-known locations to monitor incidents in progress.

We fill in the following fields for every incident:

Jira field	Type	Help Text
Summary	Text	What's the emergency?
Description	Text	What's the impact on customers? Include your contact details so responders can reach you.
Severity	Single-select	(Hyperlink to a Confluence page with our severity scale on it) Choosing Sev 2 or 1 means you believe this must be resolved right now - people will be paged.
Faulty service	Single-select	The service that has the fault that's causing the incident. Take your best guess if unsure. Select "Unknown" if you have no idea.
Affected products	Checkboxes	Which products are affected by this incident? Select any that apply.

Once the incident is created, its **issue key** (eg "HOT-12345") is used in all internal communications about the incident.

Customers will often open support cases about an incident that affects them. Once our customer support teams determine that these cases all relate to an incident, they **label** those cases with the incident's issue key in order to track the customer impact and to more easily follow up with affected customers when the incident is resolved.

When the incident issue has been created but hasn't yet been assigned to an incident manager (IM), the incident is in a "pending" state. This is the initial status in our Jira incident workflow.

We have a service that receives Jira webhooks and triggers a page alert when a **new** major incident issue is created. This alerts an on-call IM based on the "faulty service" that was selected when the ticket was created. For example, an incident affecting Bitbucket will page a Bitbucket incident manager.



**Another option:**

In Opsgenie, you can use Incident Templates for specific teams or services to set up the communication flow to incident commanders, responders, and stakeholders.

The page alert includes the incident's issue key, severity, and summary, which tells the incident manager where to go to start managing the incident (a link to the Jira issue), what's generally wrong, and how severe it is.

## Open communications

The first thing the IM does when they come online is assign the incident issue to themselves and progress the issue to the **"Fixing"** state to indicate that the incident is being actively managed. The Jira issue's assignee field always indicates who the current IM is. In an emergency response, it's very important to be clear who's in charge, so we're pretty strict about making sure this field is accurate.

Next, the IM sets up the incident team’s communication channels. Similar to how we trigger pager alerts with Jira webhooks, we use a webhook sent by an “Incident Setup” transition to automatically create these channels. The goal at this point is to establish and focus all incident team communications in well-known places.

We normally use three team communication methods, each of which are represented by a field on the Jira issue, for every incident:

- **Chat room** in Slack or another messaging service. This allows the incident team to communicate, share observations, links, and screenshots in a way that is timestamped and preserved. We automatically give the chat channel the same name as the incident issue key (e.g., #HOT-1234), which makes it easier for responders to find.
- **Video chat** in a conferencing app like Skype, Zoom or similar; or if you’re all in the same place, gather the team in a physical room. Face-to-face synchronous communication helps teams build a shared understanding of the situation and make decisions faster.
- **A Confluence page** called the “incident state document”, where we track changes (for example, a table of who changed what, when, why, how to revert, etc), multiple streams of work, or an extended timeline. When people simultaneously edit the same page, they can see what info is being added and changed in real time. An incident state document is extremely useful as the source of truth during complex or extended incidents.

We’ve found that using both video chat and chat room synchronously works best during most incidents, because they are optimized for different things. Video chat excels at creating a shared mental picture

of the incident quickly through group discussion, while text chat is great for keeping a timestamped record of the incident, shared links to dashboards, screenshots, and other URLs. In this way, video and text chat are complementary rather than one or the other.

The dedicated chatroom should also be used to record important observations, changes, and decisions that happen in unrecorded conversations. The IM (or anyone on the incident team) does this by simply **noting observations, changes, and decisions** as they happen in real-time. It's okay if it looks like people are talking to themselves! These notes are incredibly valuable during the postmortem when teams need to reconstruct the incident timeline and figure out the thing that caused it.

Most chat systems have a **room topic** feature. Our communications automation updates the room topic with information about the incident and useful links, including:

- The incident summary and severity.
- Who is in what role, starting with the IM.
- Links to the incident issue, the video chat room, and the incident state document.

Remember that we automatically named the chat channel based on the incident's issue key (e.g., HOT-1234), and our page alerting automation includes this issue key in page alerts, and all of our internal communication about the incident (covered later) includes the same issue key. This consistency means that anyone with that issue key can easily find the incident's chat room and come up to speed on the incident.

Finally, the IM sets their own **personal chat status** to the issue key of the incident they are managing. This lets their colleagues know that they're busy managing an incident.

## Assess

After the incident team has their communication channels set up, the next step is to assess the incident's severity so the team can decide what level of response is appropriate.

In order to do that we ask:

- What is the impact to customers (internal or external)?
- What are customers seeing?
- How many customers are affected (some or all)?
- When did it start?
- How many support cases have customers opened?
- Are there other factors, e.g., public attention, security, or data loss?

Based on the business impact of the incident, we assign one of the following **severity levels**:

Severity	Description	Examples
1.	A critical incident with very high impact	<ol style="list-style-type: none"><li>1. A customer-facing service, like Jira Cloud, is down for all customers.</li><li>2. Confidentiality or privacy is breached.</li><li>3. Customer data loss.</li></ol>
2.	A major incident with significant impact	<ol style="list-style-type: none"><li>1. A customer-facing service is unavailable for a subset of customers.</li><li>2. Core functionality (e.g., git push, issue create) is significantly impacted.</li></ol>
3.	A minor incident with low impact	<ol style="list-style-type: none"><li>1. A minor inconvenience to customers, workaround available.</li><li>2. Usable performance degradation.</li></ol>

Once you establish the impact of the incident, adjust or confirm the severity of the incident issue and communicate that severity to the team.

At Atlassian, severity 3 incidents are assigned to the responsible teams for resolution as appropriate (normally during business hours), whereas severity 1 and 2 require an immediate response and continuous 24/7 management through to resolution. The difference in response between severity 1 and 2 is more nuanced and dependent on the affected service.

Your severity matrix should be documented and agreed among all your teams to have a consistent response to incidents based on business impact.

## Send initial communications

When you're reasonably confident that the incident is real, you need to communicate it internally and externally. The goal of initial internal communication is to focus the incident response in one place and reduce confusion. The goal of external communication is to tell customers that you know something's broken and you're looking into it as a matter of urgency. Communicating quickly about new incidents helps to build trust with your staff and customers.

We use Statuspage for incident communications both internally and externally. We have separate status pages for internal company staff and external customers. We'll talk more about how to use each one later on, but for now, the goal is to get initial communications up as quickly as possible. In order to do that, we follow these templates:

	Internal Statuspage	External Statuspage
<b>Incident name</b>	<i>&lt;Incident issue key&gt; - &lt;Severity&gt; - &lt;Incident summary&gt;</i>	Investigating issues with <i>&lt;product&gt;</i>
<b>Message</b>	We are investigating an incident affecting <i>&lt;product x&gt;</i> , <i>&lt;product y&gt;</i> and <i>&lt;product z&gt;</i> . We will provide updates via email and Statuspage shortly.	We are investigating issues with <i>&lt;product&gt;</i> and will provide updates here soon.

**Tip:**

In Statuspage, you can create incident communication templates to use as a starting point. Fields like the name and message will be pre-filled and ready for your quick review before you send off to your customers. This saves time and relieves some of the stress involved when in the midst of an incident.

In addition to creating an incident on our internal Statuspage, we send an email to an incident communications distribution list that includes our engineering leadership, major incident managers, and other interested staff. This email has the same content as the internal Statuspage incident. Email allows staff to reply and ask questions, whereas Statuspage is more like one-way broadcast communication.

Note that we always include the incident's Jira issue key on all internal communications about the incident, so everyone knows how to find out more or join the incident (remember, the chat room is automatically named for the Jira issue key).

# How well are you communicating to customers during an outage?

**BY NICK COATES, PRODUCT OWNER AT SYMANTEC**

Your company's portal goes down. You're immediately in firefighting mode, trying to fix the bug to restore the portal. But are you being transparent with your customers? Are your support teams starting to see queues fill up with tickets and Tweets?

Good incident response isn't just about getting services back up quickly—it's about being upfront and frequently updating your customers.

We have over 30 enterprise cloud products here at Symantec. Our Incident Communication Team works with more than 20 engineering teams. There is a lot at stake, and the potential for poor communication to have an impact on our business is real.

We recently decided to take a step back and have a look at how well we're doing with incident communication and learn where we can improve.

I've been fortunate enough to have an Atlassian help run several Atlassian Team Playbook plays for our service teams here in Reading, UK. Based on the positive reaction, I knew playbooks were the way forward. We decided to run the Incident Response Communications play to analyze our incident communication practices.

To start the process, I worked with one of our Incident Communication Managers to look at the past 30 days worth of incidents. We wanted to narrow down on one completed incident that we could focus on and analyze. Out of the handful of incidents, we chose one and started to gather the incident details by collaborating on a Confluence page.

Being a remote team, we pulled the incident timeline into a shared screen and collaborated on the screen using the notes feature in our video conference software. One of the team members was the scribe and took notes, adding them to the timeline on the screen.

The team focused on the entire incident from start to finish: from the point that engineering received the first monitoring alert to our last customer update.

Once the team assessed the incident, we put an action plan in place which included people, process and technology actions. We assigned each action to an “owner” and a recommendation on the next steps. A few weeks later we re-grouped and the owners provided an update on the actions which has now improved our future communications.

But this isn't a one-off exercise. It's something that we will continue to repeat every few months to help refine and improve our incident communication process. A good incident communications plan is a journey, not a destination. It's something you can constantly improve and iterate on. If you're ready to kick off that journey, this exercise is a great start.

Nick Coates is a Product Owner at Symantec. To find this and other Team Playbook plays, visit [Atlassian.com/team-playbook](https://atlassian.com/team-playbook).

## Escalate

You've taken command of the incident, established team communications, assessed the situation, and told staff and customers that an incident is in progress. What's next?

Your first responders might be all the people you need in order to resolve the incident, but more often than not, you need to bring other teams into the incident by paging them. We call this **escalation**.

The key system in this step is a page rostering and alerting tool like **Opsgenie**. Opsgenie and similar products allow you to define on-call rosters so that any given team has a rotation of staff who are expected to be contactable to respond in an emergency. This is superior to needing a specific individual all the time ("get Bob again") because individuals won't always be available (they tend to go on vacation from time to time, change jobs, or burn out when you call them too much). It is also superior to "best efforts" on-call because it's clear which individuals are responsible for responding.

We always include the incident's Jira issue key on a page alert about the incident. This is the key that the person receiving the alert uses to join the incident's chat room.

## Delegate

After you escalate to someone and they come online, the IM **delegates a role** to them. As long as they understand what's required of their role then they will be able to work quickly and effectively as part of the incident team.

At Atlassian, we train our responders on what the incident roles are and what they do using a combination of online training, face-to-face training, documentation and hands-on “shadowing” experience.

The roles we use at Atlassian are:

- **Incident Manager**, described in the Overview.
- **Tech Lead**, a senior technical responder. Responsible for developing theories about what’s broken and why, deciding on changes, and running the technical team. Works closely with the IM.
- **Communications Manager**, a person familiar with public communications, possibly from the customer support team or public relations. Responsible for writing and sending internal and external communications about the incident.

The IM can also devise and delegate roles as required by the incident, for example, multiple tech leads if more than one stream of work is underway, or separate internal and external communications managers.

In complicated or large incidents it’s advisable to bring on another qualified incident manager as a backup “sanity check” for the IM. They can focus on specific tasks that free up the IM, such as keeping the timeline.

We use the chat room’s topic to show who is currently in which role, and this is kept up-to-date if roles change during an incident.

## Send follow-up communications

You already sent out initial communications. Once the incident team is rolling you have to update staff and customers on the incident, and as the incident progresses you need to keep them looped in.

Updating internal staff regularly creates a consistent shared truth about the incident. When something goes wrong, information is often scarce, especially during the early stages, and if you don't establish a reliable source of truth about what's happened and how you're responding, then people will tend to jump to their own conclusions, which creates confusion.

For our **internal communications**, we follow this pattern:

- We communicate via our internal Statuspage and via email, as described under “Initial Communications” above.
- Use the same convention for incident name and email subject formatting (<*Incident issue key*> - <*Severity*> - <*Incident summary*>).
- Open with a 1-2 sentence summary of the incident's current state and impact.
- A “Current Status” section with 2-4 bullet points.
- A “Next Steps” section with 2-4 bullet points.
- State when and where the next round of communications will be sent out.

Following this pattern for internal communications makes it easier and more reliable to produce and consume information about incidents. In an emergency this is important. Before sending, we review the communications for completeness using this checklist:

- Have we described the actual impact on customers?
- Did we say how many internal and external customers are affected?
- If the root cause is known, what is it?
- If there is an ETA for restoration, what is it?
- When & where will the next update be?

During an incident, information is often scarce, and it's important to be clear about what we do and don't know. We encourage our incident managers to be explicit about unknowns in their internal communications. This reduces uncertainty. For example, if you don't know what the root cause is yet, it's far better to say "the root cause is currently unknown" than to simply omit any mention of it. This makes it clear that we don't know the root cause, as opposed to letting our readers guess (e.g., maybe we do know, but just aren't saying).

Updating external customers regularly is important because it builds trust. Even though they might be impacted, at least they know we're fixing it, and can get on with other things, knowing we'll keep them up-to-date.

For **external communications** we simply update the incident that we opened on the external Statuspage earlier, changing its status as appropriate. We try to keep updates "short and sweet" because

external customers usually aren't interested in the technical details of the incident—they just want to know if it's fixed yet and if not, when it will be. Generally, 1-2 sentences will suffice.

Incident communications is an art, and the more practice you have, the better you'll be. In our incident manager training, we role-play a hypothetical incident, draft communications for it, and read them out loud to the rest of the class. This is a good way to build this skill before doing it for real. It's also always a good idea to get someone else to review your communications as a "second opinion" before you send them.

## Iterate

There's no single prescriptive process that will resolve all incidents—if there were, we'd simply automate that and be done with it. Instead, we iterate on the following process to quickly adapt to a variety of incident response scenarios:

- **Observe** what's going on. Share and confirm observations.
- **Develop theories** about why it's happening.
- **Develop experiments** that prove or disprove those theories. Carry those out.
- **Repeat.**

For example, you might observe a high error rate in a service corresponding with a fault that your regional infrastructure provider has posted on their Statuspage. You might theorize that the fault is isolated to this region, decide to fail over to another region, and observe the results. Process aficionados will recognize this

as a generalization of the Deming “Plan-Do-Check-Act” cycle, the “Observe-Orient-Decide-Act” cycle, or simply the scientific method.

The biggest challenges for the IM at this point are around maintaining the team’s discipline:

- Is the team communicating effectively?
- What are the current observations, theories, and streams of work?
- Are we making decisions effectively?
- Are we making changes intentionally and carefully? Do we know what changes we’re making?
- Are roles clear? Are people doing their jobs? Do we need to escalate to more teams?

In any case, don’t panic—it doesn’t help. Stay calm and the rest of the team will take that cue.

The IM has to keep an eye on team fatigue and plan team handovers. A dedicated team can risk burning themselves out when resolving complex incidents—IMs should look out for how long members have been awake for and how long they’ve been working on the incident for, and decide who’s going to fill their roles next.

## Resolve

An incident is resolved when the **current or imminent business impact** has ended. At that point, the emergency response ends and the team transitions onto any cleanup tasks and the postmortem.

Cleanup tasks can be easily linked and tracked as issue links from the incident's Jira issue.

At Atlassian, we use Jira custom fields to track every incident's start-of-impact time, detection time, and end-of-impact time. We use these fields to calculate time-to-recovery (TTR) which is the interval between start and end, and time-to-detect (TTD) which is the interval between the start and detect. The distribution of your incident TTD and TTR is often an important business metric.

We send final internal and external communications when the incident is resolved. The internal communications have a recap of the incident's impact and duration, including how many support cases were raised and other important incident dimensions, and clearly state that the incident is resolved and there will be no further communications about it. The external communications are usually brief, telling customers that service has been restored and we will follow up with a postmortem.

The final responsibility of the incident manager is to get accountability for completion of the postmortem. See the next chapter for how we do that...

# A well-written incident report

**BY JOHN ALLSPAW, CO-FOUNDER OF ADAPTIVE CAPACITY LABS**

Below is an answer from an “Ask Me Anything” session with John Allspaw on the Atlassian Community. John is the co-founder of Adaptive Capacity Labs and the former Chief Technology Officer at Etsy.

## **QUESTION:**

What tips would you suggest for writing a good postmortem report? Structure of postmortems is something many companies seem to struggle with e.g., what to include, what really would make a difference?

## **JOHN ALLSPAW:**

I think you’ve captured a core challenge people doing incident analysis face: there is really no end to the data and context that can be put into a written exploration of an incident! What to include, what to focus on or highlight, what to skim over or dismiss as valuable... all of these are judgment calls that need to be made by the author(s) of these documents.

My short answer is:

- Consider the audience for this particular document. Is it hands-on practitioners who are in the “trenches” of production systems? Is it the general public or customers under contract? Is it executive leaders, board members, or investors? The answer to this influences almost every part of the document’s construction.

- There is no “One True and Complete Structure <sup>TM</sup>” of a post-incident review document; all incidents have different insights to shed light on. Therefore, it’s up to the person analyzing the incidents to recognize what the case has the potential to reveal about the teams, the context, the systems, etc. and focus on those.
- Getting good at this (not just writing post-incident documents, but data collection/collation, group interviews, etc.) takes time and practice.
- I will assume that the audience is internal hands-on engineers for the rest of my answer. :) Quite often, these reports are seen as just a simple vehicle upon which “remediation” or follow-up action items are placed. This is a mistake and a missed opportunity.

## “ Capturing tasks that might help in the future is important, but not sufficient for real learning.

A key thing to remember is that these documents need to be written with the reader in mind.

What will the reader want to know about the context of the incident? We could imagine that indicating when an outage happened (Cyber Monday? Same day as your IPO? When your CEO is on stage giving an interview?) might be important in some cases and not really important in others, for example.

What about the handling of the incident might be important to highlight to the reader? For example: sometimes, resolving incidents

requires the arcane knowledge of long-tenured engineers who understand the weird behaviors of that legacy stuff that usually “just works.” In that case, noting who those engineers are and what they know might be very important. Maybe all the engineers needed to respond to the incident are attending a conference at the time? Capturing this data in a document can be really valuable for readers of the report who wasn’t there at the time or who might want the backstory on why some piece of tech is the way it is.

Effective incident analysis requires developing (and maintaining!) the skills required to not only writing these narratives but also knowing what data to collect, what people to interview, what diagrams to include, etc. Organizations that acknowledge this expertise and invest in getting better at it will find that it’s a competitive advantage.



**CHAPTER #3**

---

# Incident postmortems

The blameless **postmortem** is the alchemy that turns bitter incident tears into sweet resilience sauce. We use it to capture learnings and decide on mitigations for the incident. Here's a summarized version of our internal documentation describing how we run postmortems at Atlassian.

## What is a postmortem?

A postmortem is a written record of an incident that describes:

- The incident's impact.
- The actions taken to mitigate or resolve the incident.
- The incident's causes.
- Follow-up actions taken to prevent the incident from happening again.

Opsgenie can automatically generate postmortem reports using prebuilt templates. At Atlassian, we ultimately track all follow-on actions and postmortem reports using Jira Software.

## Why do we do postmortems?

A postmortem seeks to maximize the value of an incident by understanding all contributing causes, documenting the incident for future reference and pattern discovery, and enacting effective preventative actions to reduce the likelihood or impact of recurrence.

If you think of an incident as an unscheduled investment in the reliability of your system, then the postmortem is how you maximize the return of that investment.

## When is a postmortem needed?

We always do postmortems for severity 1 and 2 (“major”) incidents. For minor incidents they’re optional. We encourage people to use the postmortem process for any situation where it would be useful.

## Who completes the postmortem?

Usually the team that delivers the service that caused the incident is responsible for completing the associated postmortem. They nominate one person to be accountable for completing the postmortem, and the issue is assigned to them. They are the “postmortem owner” and they drive the postmortem through drafting and approval, all the way until it’s published.

Infrastructure and platform-level incidents often impact a cross-section of the company, making their postmortems more complicated and effort-intensive. For this reason we sometimes assign a dedicated program manager to own infrastructure or platform-level postmortems because these staff are better suited to working across groups, and they are able to commit the requisite level of effort.

## Why should postmortems be blameless?

When things go wrong the natural human reaction is to ask “who is to blame” and to insulate ourselves from being blamed. But blame actually jeopardizes the success of the postmortem because:

- When people feel the risk to their standing in the eyes of their peers or to their career prospects, it usually outranks “my employer’s corporate best interests” in their personal hierarchy, so they will naturally dissemble or hide the truth in order to protect their basic needs;
- Blaming individuals is unkind and, if repeated often enough, will create a culture of fear and distrust; and
- Even if a person took an action that directly led to an incident, what we should ask is not “why did individual X do this”, but “why did the system allow them to do this, or lead them to believe this was the right thing to do.”

As the person responsible for the postmortem you need to actively work against this natural tendency to blame. The postmortem needs to honestly and objectively examine the circumstances that led to the fault so we can find the true causes and mitigate them. We assume good intentions on the part of our staff and never blame people for faults.

In our postmortems, we use these techniques to create personal safety for all participants:

- In the meeting, make an opening comment stating that this is a blameless postmortem and why;

- Refer to individuals by role (e.g., “the on-call widgets engineer”) instead of name (while remaining clear and unambiguous about the facts); and
- Ensure that the postmortem timeline, causal chain, and mitigations are framed in the context of systems, processes, and roles, not individuals.

Our inspiration for blameless postmortems and the useful concept of “second stories” comes from John Allspaw’s seminal article\*.

## Postmortem process overview

For postmortems to be effective, the process has to make it easy for teams to identify causes and fix them. The exact methods you use depend on your team culture; at Atlassian, we’ve found a combination of methods that work for our postmortem teams:

- **Single-point accountability** for postmortem results makes responsibilities clear. The assignee of the postmortem ticket is always the person accountable.
- Use **face-to-face meetings** to speed up analysis, quickly create a shared understanding, and align the team on what needs fixing.
- **Postmortem review and approval** by our engineering leadership team helps to set the right level of rigor and priority.
- Significant mitigations have an agreed **Service Level Objective (SLO)** for completion (8 weeks in most cases), with reminders and reports to ensure they are completed.

\* Blameless PostMortems and a Just Culture by John Allspaw:  
<https://codeacraft.com/2012/05/22/blameless-postmortems/>

Running the postmortem process includes completing a postmortem issue, running a postmortem meeting, capturing actions, getting approval and communicating the outcome.

The postmortem owner follows these steps:

1. Create a postmortem ticket and link it to the incident ticket.
2. Edit the postmortem issue, read the field descriptions and complete the fields (the fields we use at Atlassian are listed under “Postmortem issue fields” below).
3. Use the “Five Whys” technique to traverse the causal chain and discover the underlying causes of the incident. Prepare a theory of what actually happened vs. a more ideal sequence of events, and list proposed mitigations for the causes you identify.
4. Schedule the postmortem meeting. Invite the delivery team, impacted teams and other stakeholders using the meeting invitation template.
5. Meet with the team and run through the agenda (see “Postmortem meetings” below).
6. Follow up with the responsible engineering managers to get time-bound commitment to the actions that were agreed in the meeting.
7. If the participating teams haven’t done so already, raise a Jira issue for each action in the backlogs of the team(s) that own them. Link them from the postmortem issue.
8. Add the engineering manager(s) from the appropriate group(s) to the “Approvers” field on the postmortem.

9. When you feel the postmortem ticket is ready for approval and publication, select the “Request Approval” transition to request approval from the nominated approvers. Automation will comment on the issue with instructions for approvers.
10. Follow up with approvers and other stakeholders as needed. This often involves answering questions and driving decisions.
11. Once the postmortem is approved we want to multiply its value by sharing what we learned with the whole company. To this end we have automation that creates a draft blog post in Confluence when the postmortem ticket is approved. Follow the “Draft Blog Post” link on the postmortem ticket to edit and publish the blog post.

## Postmortem issue fields

Our postmortem issue has an extensive series of custom fields to collect all the important details before holding the postmortem meeting.



### **Another option:**

If you’re tracking and collaborating on incidents within Opsgenie, you can automatically generate a postmortem report using a predefined template. All critical actions that occurred during the incident are documented in a timeline and related Jira tickets are included as reference and remediation.

Below is a list of those fields and some examples of how we fill them out.

Field	Instructions	Examples
<p><b>Incident summary</b></p>	<p>Summarize the incident in a few sentences. Include what the severity was, why, and how long impact lasted.</p>	<p>Between <i>&lt;time range of incident, e.g., 14:30 and 15:00&gt;</i> on <i>&lt;date&gt;</i>, <i>&lt;number&gt;</i> customers experienced <i>&lt;event symptoms&gt;</i>. The event was triggered by a deployment at <i>&lt;time of deployment or change that caused the incident&gt;</i>. The deployment contained a code change for <i>&lt;description of or reason for the change&gt;</i>. The bug in this deployment caused <i>&lt;description of the problem&gt;</i>. The event was detected by <i>&lt;system&gt;</i>. We mitigated the event by <i>&lt;resolution actions taken&gt;</i>. This <i>&lt;severity level&gt;</i> incident affected X% of customers. <i>&lt;Number of support tickets and/or social media posts&gt;</i> were raised in relation to this incident.</p>
<p><b>Leadup</b></p>	<p>Describe the circumstances that led to this incident, for example, prior changes that introduced latent bugs.</p>	<p>At <i>&lt;time&gt;</i> on <i>&lt;date&gt;</i>, (<i>&lt;amount of time before customer impact&gt;</i>), a change was introduced to <i>&lt;product or service&gt;</i> to ... <i>&lt;description of the changes that led to the incident&gt;</i>. The change caused ... <i>&lt;description of the impact of the changes&gt;</i>.</p>

Field	Instructions	Examples
<b>Fault</b>	Describe what didn't work as expected. Attach screenshots of relevant graphs or data showing the fault.	<Number> responses were incorrectly sent to X% of requests over the course of <time period>.
<b>Impact</b>	Describe what internal and external customers saw during the incident. Include how many support cases were raised.	For <length of time> between <time range> on <date>, <incident summary> was experienced. This affected <number> customers (X% of all <system or service> customers), who encountered <description of symptoms experienced by customers>. <Number of support tickets and social media posts> were raised.
<b>Detection</b>	How and when did Atlassian detect the incident? How could time to detection be improved? As a thought exercise, how would you have cut the time in half?	The incident was detected when the <type of alert> was triggered and <team or person paged> were paged. They then had to page <secondary response person or team> because they didn't own the service writing to the disk, delaying the response by <length of time>. <Description of the improvement> will be set up by <team owning the improvement> so that <impact of improvement>.

Field	Instructions	Examples
<b>Response</b>	<p>Who responded, when and how? Were there any delays or barriers to our response?</p>	<p>After being paged at 14:34 UTC, KITT engineer came online at 14:38 in the incident chat room. However, the on-call engineer did not have sufficient background on the Escalator autoscaler, so a further alert was sent at 14:50 and brought a senior KITT engineer into the room at 14:58.</p>
<b>Recovery</b>	<p>Describe how and when service was restored. How did you reach the point where you knew how to mitigate the impact? How could time to mitigation be improved? As a thought exercise, how would you have cut the time in half?</p>	<p>Recovery was a three-pronged response:</p> <ul style="list-style-type: none"> <li>• Increasing the size of the BuildEng EC2 ASG to increase the number of nodes available to service the workload and reduce the likelihood of scheduling on oversubscribed nodes</li> <li>• Disabled the Escalator autoscaler to prevent the cluster from aggressively scaling-down</li> <li>• Reverting the Build Engineering scheduler to the previous version.</li> </ul>

Field	Instructions	Examples
<p><b>Timeline</b></p>	<p>Provide a detailed incident timeline, in chronological order, timestamped with timezone(s). Include any lead-up; start of impact; detection time; escalations, decisions, and changes; and end of impact.</p>	<p>All times are UTC.</p> <p>11:48: K8S 1.9 upgrade of control plane finished</p> <p>12:46: Goliath upgrade to V1.9 completed, including cluster-autoscaler and the BuildEng scheduler instance</p> <p>14:20: Build Engineering reports a problem to the KITT Disturbed</p> <p>14:27: KITT Disturbed starts investigating failures of a specific EC2 instance (ip-203-153-8-204)</p> <p>14:42: KITT Disturbed cordons the specific node</p> <p>14:49: BuildEng reports the problem as affecting more than just one node. 86 instances of the problem show failures are more systemic</p> <p>15:00: KITT Disturbed suggests switching to the standard scheduler</p> <p>15:34: BuildEng reports 300 pods failed</p> <p>16:00: BuildEng kills all failed builds with OutOfCpu reports</p> <p>16:13: BuildEng reports the failures are consistently recurring with new builds and were not just transient.</p> <p>16:30: KITT recognize the failures as an incident and run it as an incident.</p> <p>16:36: KITT disable the Escalator autoscaler to prevent the autoscaler from removing compute to alleviate the problem.</p> <p>16:40: KITT confirms ASG is stable, cluster load is normal and customer impact resolved.</p>

Field	Instructions	Examples
<b>Five whys</b>	Use the “Five Whys” root cause identification technique. Start with the impact and ask why it happened and why it had the impact it did. Continue asking why until you arrive at the root cause. Document your “whys” as a list here or in a diagram attached to the issue.	<ol style="list-style-type: none"> <li>1. The service went down because the database was locked</li> <li>2. Because there were too many database writes</li> <li>3. Because a change was made to the service and the increase was not expected</li> <li>4. Because we don't have a development process set up for when we should load test changes</li> <li>5. We've never done load testing and are hitting new levels of scale</li> </ol>
<b>Root cause</b>	What was the root cause? This is the thing that needs to change in order to stop this class of incident from recurring.	A bug in <i>&lt;cause of bug or service where it occurred&gt;</i> connection pool handling led to leaked connections under failure conditions, combined with lack of visibility into connection state.
<b>Backlog check</b>	Is there anything on your backlog that would have prevented this or greatly reduced its impact? If so, why wasn't it done? An honest assessment here helps clarify past decisions around priority and risk.	Not specifically. Improvements to flow typing were known ongoing tasks that had rituals in place (e.g., add flow types when you change/create a file). Tickets for fixing up integration tests have been made but haven't been successful when attempted.

Field	Instructions	Examples
<b>Recurrence</b>	Has this incident (with the same root cause) occurred before? If so, why did it happen again?	This same root cause resulted in incidents HOT-13432, HOT-14932 and HOT-19452.
<b>Lessons learned</b>	What have we learned? Discuss what went well, what could have gone better, and where did we get lucky to find improvement opportunities.	<ol style="list-style-type: none"> <li>1. Need a unit test to verify the rate-limiter for work has been properly maintained</li> <li>2. Bulk operation workloads which are atypical of normal operation should be reviewed</li> <li>3. Bulk ops should start slowly and monitored, increasing when service metrics appear nominal</li> </ol>
<b>Corrective actions</b>	What are we going to do to make sure this class of incident doesn't happen again? Who will take the actions and by when? Create issue links to issues tracking each action.	<ol style="list-style-type: none"> <li>1. Manual auto-scaling rate limit temporarily used to limit failures</li> <li>2. Unit test and re-introduction of job rate limiting</li> <li>3. Introduction of a secondary mechanism to collect distributed rate information across cluster to guide scaling effects</li> <li>4. Large migrations need coordinated since AWS ES does not autoscale.</li> <li>5. Verify search is still classified as Tier-2</li> <li>6. File a ticket against pf-directory-service to partially fail instead of full-fail when the xpsearch-chat-searcher fails.</li> <li>7. Cloudwatch alert to identify a high IO problem on the ElasticSearch cluster</li> </ol>

## Proximate and root causes

When you're writing or reading a postmortem, it's necessary to distinguish between the proximate and root causes.

- Proximate causes are reasons that directly led to this incident.
- Root causes are reasons at the optimal place in the chain of events where making a change will prevent this entire class of incident.

A postmortem seeks to discover root causes and decide how to best mitigate them. Finding that optimal place in the chain of events is the real art of a postmortem. Use a technique like Five Whys to go “up the chain” and find root causes.

Here are a few select examples of proximate and root causes:

Scenario	Proximate cause & action	Root cause	Root cause mitigation
“Red Dawn” squad’s services did not have monitors and on-call alerts for their services, or they were not properly configured.	Team members did not configure monitoring and alerting for new services. Configure it for these services.	There is no process for standing up new services, which includes monitoring and alerting.	Create a process for standing up new services and teach the team to follow it.

Scenario	Proximate cause & action	Root cause	Root cause mitigation
The site was unusable on IE11 due to an upgrade to Fabric Editor that doesn't work on this browser version.	An upgrade of a dependency. Revert the upgrade.	Lack of cross-browser compatibility testing.	Automate continuous cross-browser compatibility testing.
Logs from Micros EU were not reaching the logging service.	The role provided to micros to send logs with was incorrect. Correct the role.	We can't tell when logging from an environment isn't working.	Add monitoring and alerting on missing logs for any environment.
Triggered by an earlier AWS incident, Confluence Vertigo nodes exhausted their connection pool to Media, leading to intermittent attachment and media errors for customers.	AWS fault. Get the AWS postmortem.	A bug in Confluence connection pool handling led to leaked connections under failure conditions, combined with lack of visibility into connection state.	Fix the bug and add monitoring that will detect similar future situations before they have an impact.

## Categories of causes and their actions

At Atlassian, we find it useful to group the causes of incidents into categories. This helps point us in the right direction when deciding on mitigations and lets us analyze incident trends. For example, if we see a high number of scale-related incidents in one group, that might prompt us to compare scaling strategies with other groups.

The categories we use are tailored to our own business as a software company. You may find that different categories work better for your business.

Category	Definition	What should you do about it?
<b>Bug</b>	A change to code made by Atlassian (this is a specific type of change)	Test. Canary. Do incremental rollouts and watch them. Use feature flags.
<b>Change</b>	A change made by Atlassian (other than changes to code, which are bugs)	Improve the way you make changes, for example, your change reviews or change management processes. Everything next to “bug” also applies here.
<b>Scale</b>	Failure to scale (e.g., blind to resource constraints, or lack of capacity planning)	What are your service’s resource constraints? Are they monitored and alerted? If you don’t have a capacity plan, make one. If you do have one, what new constraint do you need to factor in?

Category	Definition	What should you do about it?
Architecture	Design misalignment with operational conditions	Review your design. Do you need to change platforms?
Dependency	Third party (non-Atlassian) service fault	Are you managing the risk of third party fault? Have we made the business decision to accept a risk, or do we need to build mitigations? See “Root causes with dependencies” below.
Unknown	Indeterminable (action is to increase the ability to diagnose)	Improve your system’s observability by adding logging, monitoring, debugging, and similar things.

## Root causes with external dependencies

With the advent of SaaS and IaaS, companies are more dependent than ever before on their suppliers. Companies like AWS, Google, Microsoft, and Salesforce provide much of the infrastructure that powers modern services. When they fail and you are impacted, how can you get value from the postmortem?

When your service has an incident because an external dependency fails, where the fault lies and what the appropriate mitigations are depend on **what your reasonable expectation of the external service is**. This is sometimes referred to as “Service Level Expectation” or SLE.

Why do we say “expectation” or “SLE” here as opposed to “commitment” or “SLA”? In reality, the SLAs published or agreed to by suppliers are almost always too low or toothless to be useful. In other words, the things a supplier will legally agree to will be either so minimal as to be useless in practice, or the consequences of breaching the SLA will be so minimal that they don’t actually deter breach. This is simply because suppliers, like the rest of us, have to survive in a competitive world and seek to minimize their exposure. Finally, even if you get a supplier to agree to terms that you believe are suitable to you, you may frequently find yourself in a “gray area” where an impact didn’t technically affect the agreed SLA but impacted you as a customer nonetheless.

Establishing and communicating an SLE is the responsibility of the person who “owns” the external service. This might be a head of IT, a lead engineer, or the person who signed the contract. Just like the owner of an internal service, this person needs to tell the rest of the organization what to expect from the external service.

Once you have settled what the “reasonable expectation” is, you can use it to decide where to apply mitigations after an incident:

In this incident, the SLE was...	...so we should:
<p><b>Exceeded:</b> The external service performed worse than we expected it to.</p>	<p>Get and review their postmortem. We may determine that they’ve correctly identified and mitigated the causes of the incident, in which case we’re done.</p>
	<p>Alternatively, we may need to adjust our expectations downward, and find ways to increase our resilience to external failures in line with our reduced expectations.</p>
	<p>Finally, if our newly-adjusted expectations aren’t acceptable, then we need to resolve the disconnect somehow, for example by changing service providers.</p>
<p><b>Met:</b> The external service performed as expected, but we were impacted anyways.</p>	<p>Increase our resilience to failures of this type. The external service behaved as expected, so we should either be resilient to that or accept the risk and plan for occasional outages.</p>
<p><b>None:</b> We don’t really have an expectation!</p>	<p>The person responsible for the use of the 3rd party dependency needs to establish an SLE and share it with teams so they know what level of resilience they need to build into their dependent services.</p>

## Postmortem actions

Sue Lueder and Betsy Beyer from Google have an excellent presentation and article on postmortem action items, which we use at Atlassian to prompt the team. This section is based on their suggestions.\* Work through the questions below to help ensure the postmortem covers both short- and long-term fixes:

Category	Question to ask	Examples
<b>Investigate this incident</b>	“What happened to cause this incident and why?” Determining the root causes is your ultimate goal.	logs analysis, diagramming the request path, reviewing heap dumps
<b>Mitigate this incident</b>	“What immediate actions did we take to resolve and manage this specific event?”	rolling back, cherry-picking, pushing configs, communicating with affected users
<b>Repair damage from this incident</b>	“How did we resolve immediate or collateral damage from this incident?”	restoring data, fixing machines, removing traffic re-routes
<b>Detect future incidents</b>	“How can we decrease the time to accurately detect a similar failure?”	monitoring, alerting, plausibility checks on input/output

\*Visit <https://www.usenix.org/conference/srecon17americas/program/presentation/lueder> for the full source material.

Category	Question to ask	Examples
Mitigate future incidents	“How can we decrease the severity and/or duration of future incidents like this?” “How can we reduce the percentage of users affected by this class of failure the next time it happens?”	graceful degradation; dropping non-critical results; failing open; augmenting current practices with dashboards or playbooks; incident process changes
Prevent future incidents	“How can we prevent a recurrence of this sort of failure?”	stability improvements in the code base, more thorough unit tests, input validation and robustness to error conditions, provisioning changes

“Mitigate future incidents” and “Prevent future incidents” are your most likely source of actions that address the root cause. Be sure to get at least one of these.

We also use Lueder and Beyer’s advice on how to word our postmortem actions:

The right wording for a postmortem action can make the difference between an easy completion and indefinite delay due to infeasibility or procrastination. A well-crafted postmortem action should have these properties:

- **Actionable:** Phrase each action as a sentence starting with a verb. The action should result in a useful outcome, not a process. For example, “Enumerate the list of critical dependencies” is a good action, while “Investigate dependencies” is not.

- **Specific:** Define each action’s scope as narrowly as possible, making clear what is and what is not included in the work.
- **Bounded:** Word each action to indicate how to tell when it is finished, as opposed to leaving the action open-ended or ongoing.

From...	To...
Investigate monitoring for this scenario.	(Actionable) Add alerting for all cases where this service returns >1% errors.
Fix the issue that caused the outage.	(Specific) Handle invalid postal code in user address form input safely.
Make sure engineer checks that database schema can be parsed before updating.	(Bounded) Add automated pre-submit check for schema changes.

## Postmortem meetings

The postmortem owner convenes a **postmortem meeting** once they have completed the issue fields and developed a strong theory as to the incident’s causes and appropriate mitigations.

At Atlassian, we find that meeting face-to-face results in more effective and faster communication, deeper analysis and more useful learning. We encourage teams to do this when possible. Sometimes it’s impossible to get everyone at the same time due to geography or time constraints, though, so sometimes you may need to schedule multiple meetings or get input asynchronously.

Although we use the term “face to face” these meetings are often over video conference due to our distributed teams. Sometimes they involve large audiences of interested stakeholders.

Our standard postmortem meeting agenda is:

1. Remind the team that all postmortems need to be blameless, and why (see “Why should postmortems be blameless” above).
2. Walk through the timeline of events. Add or change it as needed.
3. Present your theory of the incident’s causes, and where you think mitigations could best be applied to prevent this class of incident in the future. Discuss with the room and try to build a shared understanding of what happened, why, and what we generally need to do about it.
4. Generate actions using “open thinking” (e.g., “What could we do to prevent this class of incident”) and try to avoid “closed thinking” (e.g., “Why weren’t we monitoring system x”). Open thinking helps generate novel, “out of the box” ideas because you’re not constraining the room’s thinking to the current state of things.
5. Ask the team “What went well / What could have gone better / Where did we get lucky?” These questions help catch near-misses, learnings, mitigations and actions that are outside the direct blast radius of the incident. Remember, our goal is to extract as much value as possible given that we’ve already paid the price of having an incident.
6. Wrap up by thanking everyone for their time and input.

Here's a template you can use when you invite participants:

Please join me for a blameless postmortem of *<link to incident>*, where we *<summary of incident>*.

The goal of a postmortem is to maximize the value of an incident by understanding all contributing causes, documenting the incident for future reference and pattern discovery, and enacting effective preventative actions to reduce the likelihood or impact of recurrence.

In this meeting we'll review the incident, determine its significant causes and decide on actions to mitigate them.

A good postmortem meeting will generate lots of mitigations and other actions. This is good, but often postmortem meetings are a poor context for significant prioritization decisions because you don't have the right context (for example, the rest of the backlog and other planned work that will compete for time).

For this reason, it's usually more effective in practice to follow up with the responsible managers **after** the meeting to get firmer, time-bound commitments to the actions that you identified in the meeting, and then note that on the postmortem issue.

## Postmortem approvals

Atlassian uses a Jira workflow with an approval step to ensure postmortems are approved. Approvers are generally service owners or other managers with responsibility for the operation of a service. Approval for a postmortem indicates:

- Agreement with the findings of the post-incident review, including what the root cause was; and
- Agreement that the linked “Priority Action” actions are an acceptable way to address the root cause.

Our approvers will often request additional actions or identify a certain chain of causation that is not being addressed by the proposed actions. In this way, we see approvals adding a lot of value to our postmortem process at Atlassian.

In teams with fewer incidents or less complex infrastructure, postmortem approvals may not be necessary.

## How are postmortem actions tracked?

For every action that comes out of a postmortem, we:

- Raise a Jira issue in the backlog of the appropriate team. We track all of our postmortem actions in Jira.
- Link the actions from the postmortem issue as “Priority Action” (for significant mitigations) or “Improvement Action” (for other actions coming out of the postmortem).

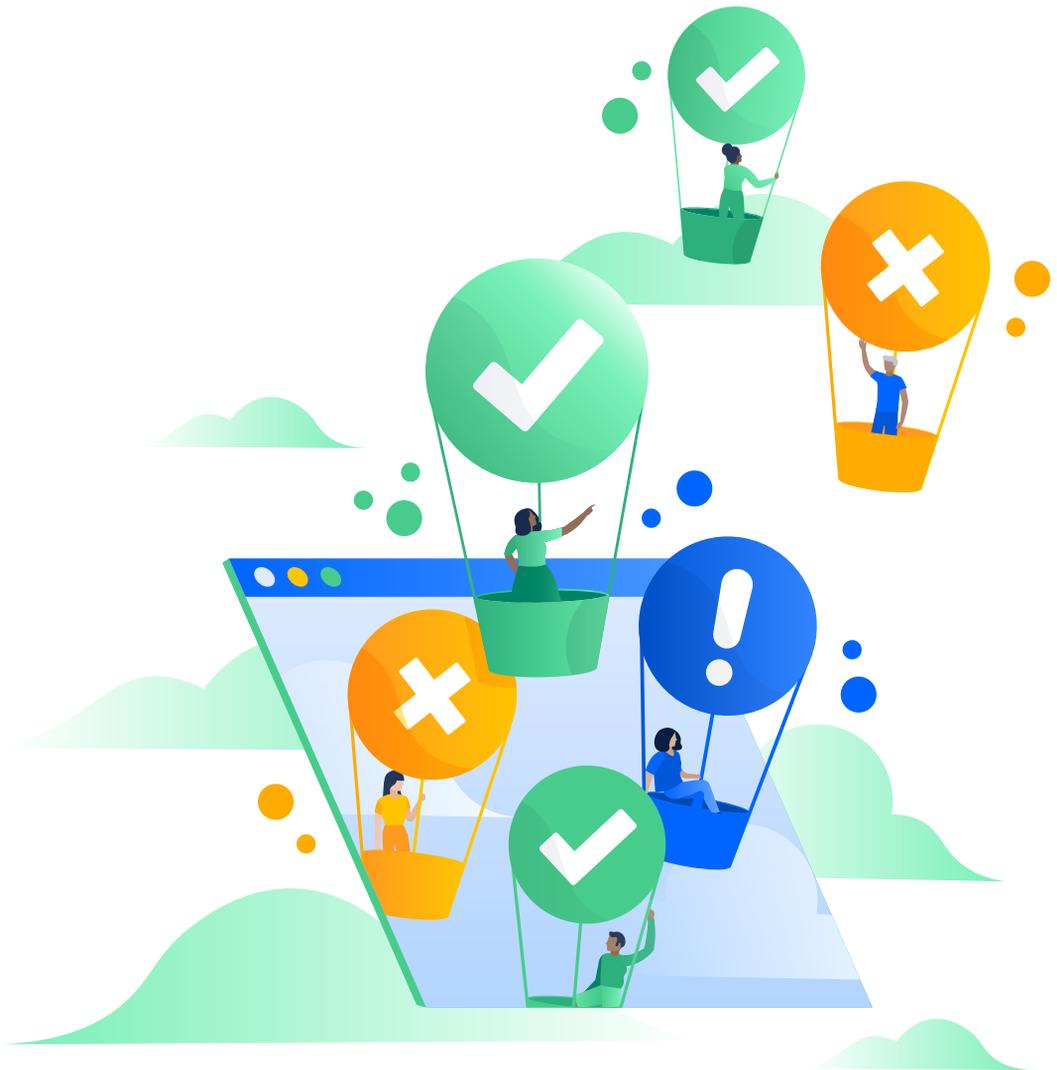
We distinguish “Priority” from “Improvement” actions so we can tell, at any point, which incidents have had their significant causes mitigated and which ones still represent a risk of recurrence. This gives us an idea of how much repeat-incident risk our different groups are carrying.

We use Jira issue links to track actions because our teams use many different instances of Jira Cloud and Server, but we still have to track and report on this data. To do this we built some custom tooling that spiders remote issue links and populates a database with the results. This allows us to track state over time and create detailed reports.

### **Keep calm and carry on...**

You’ve reached the end of our incident handbook. Thanks for reading!

If you have any feedback or suggestions, please send us an email at [incident-handbook@atlassian.com](mailto:incident-handbook@atlassian.com).



# Major incident manager cheat sheet

We'd like to leave you with a version of the cheat sheet we provide for our Major Incident Managers. It outlines the responsibilities of the incident manager in the simplest way possible and points to other relevant documentation and resources.

We use internal short URLs to provide quick access and memorable URLs. For example, the short URL for this page is [go/mimcheatsheet](#). Most of these short URLs redirect to an internal Confluence page. Many of our incident managers have this cheat sheet printed out and hung by their desk.



# Major incident manager cheat sheet

## Connect

- Connect to VPN
- Resolve the Page
- Assign the HOT ticket to yourself  
(We use 'HOT' as the issue key for incident tickets in Jira. For example an incident ticket might be 'HOT-1234.' Thus Atlassians will often refer to incidents as HOTs)
  - Incident systems down?  
GOTO Backups

## Open Communications

- Click Incident Setup on the HOT
- Get team into Slack and Zoom

## Assess

- Assess severity *go/severity*
  - What is the impact to customers?
  - What are they seeing?
  - How many customers are affected?
  - When did it start?
  - How many support tickets are there?
  - Any other factors eg Twitter, data loss, security?
- Confirm HOT "Affected Products"
- Confirm HOT "Severity"
- Post Initial Communications quickly (<=30m)
  - Internal: *go/internalcomms*
  - External: *go/externalcomms*
- Security Incident? Page Seclnt *go/pagesecint*

## Escalate

- Who else do we need to resolve this?
  - Platform or Infrastructure teams, other products? *go/oncall*
  - Need a dedicated External Communications Manager? *go/externalcomms*
  - Security Incident? *go/pagesecint*
  - More MIMs needed? *go/imoc*
  - AWS? *go/awshelp*
  - Head of Engineering/CTO/Founders?
- Anyone else?
- Directory
- Opsgenie's "Users" list.
- Vendors are listed in Service Central

## Delegate

- Explicitly delegate these roles:
  - Tech Lead(s)
  - Internal Communications
  - External Communications

## Periodic Tasks

- Update communications GOTO Communicate
- Has impact changed?
  - What was observed? Record with timestamps
  - Has severity changed?
- Incident duration: who needs to be relieved?
  - Who will hand over which roles to who, when?
  - GOTO Handover

## Communicate

- Post update communications regularly
  - Max 180m interval
- Follow this five-point checklist for communications:
  - What is the actual impact to customers?
  - How many int and ext customers are affected?
  - If a root cause is known, what is it?
  - If there is an ETA for restoration, what is it?
  - When and where will the next update be?

## Fix

- What is the observed vs expected behavior?
- Any changes (eg deploys, feature flags) go/changedashboard
- Share observations: graphs, logs, GSAC tickets etc.
- Develop theories of what is happening
- Work to prove or disprove theories
- What are the streams of work?
- Who is the Tech Lead for each stream?
- What are current steps and next steps for each?
- How do those steps move us towards resolution?
- Track changes, decisions and observations

## Resolve

- Confirm with CSS that service is restored
  - Customer feedback
  - Firsthand observations
  - External vendors
- Final update internal and external communications
  - Resolve Statuspage incidents
  - Close incident in go/multistatuspageupdater
  - Resolve the HOT; set impact and detection times
  - Create PIR and get accountability for completion

## Handover

- Outgoing MIM: 1-1 call with incoming MIM
  - Who is out and in of which roles
  - What are the current theories and streams of work
  - When next handover is expected and to whom
- Incoming MIM: Come up to speed first
  - Ask lots of questions!

## Backup Systems

- Communications: Zoom IM; Google Hangouts for video
- Docs: “MIM Offline Material” in Google Drive
- ISD: Google Docs
- HOT ticket: Hello/J IMFB project

## IM Responsibility and Behavior

A major incident is higher priority than any other activity Atlassian is undertaking. As a Major Incident Manager you can and must preempt all current and planned work to assemble the best team and drive them to resolve the incident. **Your role is to command, take charge, bring order to chaos.**

If you're on a development or operations team that looks after services for customers who require 24/7 availability, this handbook is for you.

Have questions?

Contact us at [incident-handbook@atlassian.com](mailto:incident-handbook@atlassian.com)

