Incident management, from first alert to postmortem

An e-book on incident management, incident response, on-call, and escalation policies, plus how to decide whether to build or buy.

5 CHAPTERS · 58 MIN READ · UPDATED NOVEMBER 2026

Chapters

01 Incident management →

02 Incident response →

03 On-call management →

04 Escalation policies →

05 Build vs buy →

CHAPTER 01

Incident management explained: process, roles, metrics, and best practices

See how incident management works from the first alert to the postmortem, with clear roles, best practices, and a real-world example.

TL;DR

Incident management is how IT, DevOps, and operations teams handle service disruptions, from the first alert to the postmortem. It follows 6 steps: identify and log, categorize, prioritize, respond and investigate, resolve and recover, and close and review. The goal is to restore normal operations quickly, so the service is up and running again.

In this chapter, we'll dive into what exactly incident management is, how it differs from incident response, and the key steps involved in it. Also, we'll discuss the key metrics, best practices, and AI automation.

What is an incident?

An incident is any unplanned event that disrupts a service or lowers its quality for your users. For example: your website goes down, customers can't log in, or the checkout page takes a full minute to load.

However, events like planned maintenance (a database upgrade you announced in advance) or early warnings that aren't affecting anyone yet (a server running at 50% CPU) are not considered incidents. Nothing is broken for your users, so they don't need an immediate response.

How are incidents classified?

Incidents are classified by how critical they are and how urgently they need a fix. That gives you four combinations.

Critical is about impact, which covers how much of your system or how many users an incident affects. Urgent is about time, which covers how fast someone needs to act.

Type of incident	What it means	Example
Critical and urgent	Significant impact that needs action now	Login goes down at peak hours
Critical but not urgent	Significant impact with time to act	A data sync job has been failing silently, found during working hours
Urgent but not critical	Limited impact that still needs a fast answer	An enterprise customer can't access one feature
Neither critical nor urgent	Limited impact that can wait	A dev cluster's CPU jumps over the weekend

To learn more about these four combinations, read this guide: Four types of incident alerts every team should know

What is incident management?

Incident management is the process of identifying, managing, and resolving incidents that disrupt IT services. The goal is to restore normal operations quickly and then learn from what happened, so the same incident doesn't repeat.

Say a deployment breaks checkout. A quick rollback gets users paying again in minutes, which beats an hour of debugging on a live system. You can dig into why it broke once things are up and running.

Incident management is important because it gives your team a plan of action when an outage hits and helps restore the service quickly. The right engineer gets paged the moment something breaks, nobody wastes time asking who owns the fix, and customers hear from you before they start complaining.

Incident management vs incident response

People often use these terms interchangeably. However, they are not the same.

Incident response is the tactical work of finding the cause and resolving an incident. Whereas incident management is a more comprehensive process that handles every incident end to end, from the first alert to the postmortem.

Criteria	Incident response	Incident management
Focus	Find the root cause and resolve the incident	Manage the incident end to end, from the first alert to the postmortem
Scope	One live incident and its fix	Every incident, plus the on-call, severity, and review processes around it
Timeline	While the incident is live	From alert to postmortem
Who is responsible	On-call engineers and experts	The whole team, including support and leadership

Say your on-call engineer rolls back a bad deployment. That's incident response, while the schedule that paged them, the status page update customers saw, and the postmortem the next morning all fall under incident management.

To learn more about incident response, read this guide: What is incident response?

The incident management process in 6 steps

1. Identify and log

Identification is the step where you spot and record an incident. A monitoring alert, a user report, or a support ticket flags the problem. The incident then gets a record with a title, a start time, and the affected service.

Who does it: monitoring tools like Datadog, Grafana, and CloudWatch, or the person who spots the issue. An incident management tool creates the record on its own.

2. Categorize

Categorization is the step where the incident is sorted by type. The type can be infrastructure, application, network, or security. The affected service gets noted too, like payments or search, and this tells you which team owns the incident.

Who does it: the on-call engineer, or an Alert Routing rule that reads the alert.

3. Prioritize

Prioritization is the step where the incident is ranked by impact and urgency. It gets a severity from SEV1 to SEV3 and a priority from P1 to P5. The rank decides how fast the team acts and who gets woken up.

Who does it: the on-call engineer, or an Alert Routing rule that sets the severity and priority from the alert.

4. Respond and investigate

Response is the step where the right people are alerted and start working on the problem. The on-call engineer acknowledges the alert, and if nobody responds, the escalation policy pages the next person. The team then checks dashboards, logs, and recent deployments to find the cause, while customers get a status page update.

Who does it: the on-call engineer and subject matter experts. The incident commander leads the incident response, and the communications lead handles updates.

5. Resolve and recover

Resolution is the step where the issue is fixed and the service works normally again. This could be a rollback, a config change, or a code fix that minimizes the impact. Recovery confirms that users can work again and no data was lost.

Who does it: the on-call engineer and subject matter experts.

6. Close and review

Closure is the step where the incident is marked as closed and the team learns from it. The status page gets a final update, and the team can run a blameless postmortem after every incident to find what to change.

Who does it: the incident commander or on-call engineer closes it. The whole team joins the postmortem, and the scribe's notes guide it.

Incident management roles and responsibilities

Everything starts with the on-call responder, because they're the first to know. If the issue is small, they handle it end to end and nobody else needs to join.

When an incident grows, someone needs to steer it, and that's the incident commander. They decide what the next steps are.

The commander then pulls in subject matter experts who know the failing system, like the engineer who owns the payments database. At the same time, a communications lead takes over updates, so engineers aren't interrupted by questions from support or leadership.

In the background, a scribe keeps notes on decisions and timings, and those notes become the backbone of the postmortem. Stakeholders such as product leads only need to stay informed, so they get the important updates.

Real-world incident management example

To show the six steps in action, we set up a sample gas agency in Spike. It has an on-call schedule, an escalation policy, and a status page. Then we triggered a test leak report late at night. Here's how it moved through each step:

1. **Identify and log:** a leak report came in, and Spike turned it into an incident right away. The first alert went out as a phone call to the on-call engineer.
2. **Categorize:** an Alert Routing rule saw it was a home issue at night, so it sent the incident to the pipeline team.
3. **Prioritize:** another Alert Routing rule spotted the word "leak" in the title and set it to P1 and SEV1.
4. **Respond and investigate:** the on-call engineer didn't pick up the call. So the escalation policy called the next person in line, who acknowledged it. The team then ran a Leak Response playbook, which opened a war room and posted the issue on the status page.
5. **Resolve and recover:** once the leak was fixed, the team resolved the incident. A second playbook closed it on the status page too.
6. **Close and review:** the team went through the incident timeline. The main lesson was to answer the first call quickly, because the missed call sent the alert up the escalation chain.

Key point: Alert Routing rules and Playbooks can help you automate routine tasks like setting severity, creating tickets, etc. This way, you can focus on finding the root cause and resolving the incident.

Incident management metrics

MTTD (mean time to detect): the average time between a problem starting and your monitoring or team noticing it. A low MTTD means your alerts catch issues before users report them.

MTTA (mean time to acknowledge): the average time from an alert firing to someone on-call acknowledging it. It shows how quickly your on-call team reacts.

MTTR (mean time to resolve): the average time from the alert to the fix, when the service is back to normal operations. It's the clearest measure of how fast you recover.

MTBF (mean time between failures): the average time a service runs without problems between one failure and the next. You get it by dividing the total time the service was up and running by the number of failures. A higher MTBF means the service breaks less often.

MTTF (mean time to failure): the average time a part runs before it fails for good and needs replacing, like a disk. You get it by dividing the total running time of those parts by the number of parts that failed. It tells you how long a component usually lasts.

MTTA and MTTR are a good pair to begin with, and the others can follow as your process matures.

To learn more about these metrics and how to calculate them, read this guide: MTBF, MTTR, MTTF, and MTTA explained

5 best practices for incident management

1. Write down your severity levels with one example each. Then everyone on-call sets the same level for the same kind of incident, and nobody debates it in the middle of an outage.
2. Give every service an owner team. Alerts then reach the right people, so the first responder can start fixing the issue right away.
3. Match the alert channel to the severity, with phone calls for SEV1 and Slack for SEV3. That way, a phone call always means the incident is critical and the team needs to act immediately.
4. Update your status page within 15 minutes of a SEV1. Even "we're investigating" is better than silence, and it cuts the support tickets that pile up while you work on the fix.
5. Run a blameless postmortem after every incident and turn each action item into a ticket with an owner. Action items without an owner get forgotten, and the same incident repeats.

While dealing with incidents, you might face some challenges. And we have covered the common challenges and how to deal with them in our blog. Read it here →

How automation and AI help incident management

Some steps in incident management can be automated, like setting severity, paging the right person, and updating the status page. Alert Routing rules and Playbooks handle these steps for you, and AI pushes the boundary further by automating more of the work.

Alert Routing rules read each alert the moment it arrives. They set its severity and priority and send it to the team that owns the service. Playbooks then run the routine tasks for that incident: they open a Slack or Teams channel, create a Jira ticket, and post status page updates.

AI gives you a better picture of the incident. It gathers recent deployments, logs, etc., and gives you an analysis to work on. Also, once the incident is resolved, it creates a postmortem with all the details.

How to get started with incident management

1. List the critical services and set up alerts on those first. For an online store, that's checkout and payments.
2. Create an on-call schedule and add an escalation policy. The schedule gives every alert an owner, and the escalation policy pages a backup if that person doesn't acknowledge within a set time.
3. Agree on what is a SEV1, SEV2, or SEV3 incident. For example, SEV1 is "nobody can log in or pay", SEV2 is "search fails in one region", and SEV3 is "CSV export fails, but the API works".
4. Pick one Slack channel for incident updates and set up a status page. Conduct short reviews after every incident to catch more problems than one long review each quarter.

How Spike helps you manage incidents

Spike is an incident management and on-call platform. It handles every step of the incident management process in one place, from the first alert to the postmortem.

Connect monitoring tools like Grafana, Datadog, AWS CloudWatch, or plain email so every alert becomes an incident in Spike. Alert Routing rules then set its severity and priority and send it to the team that owns the service.

From there, on-call schedules and escalation policies make sure the right person hears about it. Spike reaches them by phone call, SMS, WhatsApp, Slack, Teams, or push notification, and pages the next person if nobody acknowledges.

While your team works on the fix, Playbooks handle the routine tasks for you. They open a war room, add runbook links, and update your status page, which comes with every plan.

Once it's resolved, the resolution notes and incident timeline give your postmortem the facts it needs.

Give your next incident a clear path

Incident management doesn't need a big rollout to start working. An alert on your most critical service, an on-call schedule, and agreed severity levels already give every incident a clear owner and a clear path. From there, each postmortem shows you the next thing to improve.

Spike puts alerts, on-call, escalations, and status pages in one place, so your first setup takes minutes, not weeks.

Go ahead and try it out with a 14-day free trial.

FAQs

What is the difference between incident management and problem management?

The main difference is intent: incident management restores service quickly, while problem management finds and fixes the underlying cause. Say your app crashes every week because of a memory leak. Incident management restarts the app to get users back, and problem management investigates and fixes the leak so the crashes stop.

What does an on-call engineer do during an incident?

The on-call engineer is the first responder. They acknowledge the alert, check how many users are affected, and set the severity. Then they either fix the issue themselves, like rolling back a bad deployment, or escalate to the incident commander and subject matter experts. Once the service works normally again, they resolve the incident and add notes for the postmortem.

How do you reduce alert fatigue?

Only alert on things that need action. Group duplicate alerts, send minor ones to Slack instead of a phone call, and review noisy alerts after every on-call week.

What tools do you need for incident management?

You need a monitoring tool to detect problems and an incident management tool for alerting, on-call schedules, and escalation. Add a chat tool like Slack or Teams for the war room and a status page for customer updates. Spike covers alerting, on-call, escalation, and status pages in one platform.

What are P1, P2, P3, and P4 incidents?

P1, P2, P3, and P4 are priority levels that set how fast an incident needs a response. A P1 needs action right now, a P2 within the hour, a P3 during working hours, and a P4 can wait until the next work shift.

Do small teams need incident management?

Yes, small teams need it too, even with two or three engineers. One alert, a shared on-call schedule, and an escalation policy already make sure nobody misses a critical incident.

CHAPTER 02

What is incident response? Lifecycle, best practices, and examples

When an outage occurs, a clear incident response gets things up and running fast. Read how to set up such an incident response and use AI to automate parts of it.

TL;DR

Incident response is the process your team follows to triage, fix, and resolve unplanned disruptions in production, like a crashed payment gateway or a failed deployment. Alerting reaches the on-call engineer immediately, an escalation policy moves the alert to the next person if they miss it, and on-call rotations share the load so nobody burns out. Prioritization separates urgent incidents from the ones that can wait, and automation runs playbooks for common incidents.

In this chapter, we'll dive into what incident response is, how it differs from incident management, and the four steps of its lifecycle. We'll also discuss the best practices and how AI is changing incident response.

What is incident response?

Incident response is the process your team follows when something breaks in production. It covers how your team triages the incident, finds the cause, and fixes it until the service works normally again. The goal is to get your services back to normal operations as quickly as possible and limit the damage to your users.

With an efficient incident response, every engineer knows their role, updates reach the right people on time, and the whole team follows the same playbook instead of improvising under pressure.

Real-world example of incident response

Say payments fail on your e-commerce store at 2 AM. With incident response in place, your team resolves the incident in 15 minutes instead of 5 hours.

Without incident response, nobody even notices the failed payments until morning. Support tickets pile up overnight, and your team spends the first hours of the day working out what broke.

With incident response, the on-call engineer gets a phone call at 2:00 AM, the moment the incident triggers. They check the payment service, find that the API connection failed and follow the response playbook for payment failures. By 2:15 AM, customers can pay again.

So with this, something that could have been 5+ hours of lost sales becomes a 15-minute disruption.

Why is incident response important?

Incident response matters because it keeps downtime short. Without it, a checkout failure plays out like this: Engineers debug the same service in parallel without knowing what the others found. Managers message them every 10 minutes for updates, and support has no answer for the customers writing in. A glitch that needed one config rollback turns into hours of downtime and a dent to your brand.

A clear incident response process changes three things:

- Your team spots issues early and fixes them fast, which cuts downtime and revenue loss.

- Customers keep trusting you, because a fast fix shows them you're reliable even when things break.
- Engineers stay focused when alerts fire at 2 AM, because they know their roles and follow a tested plan.

As your systems grow, the cost of each incident grows with them. A structured response lets your team handle that complexity.

Incident response vs incident management

Incident response is the action your team takes to find the cause of an incident and resolve it. Incident management covers the incident end to end, from the first alert to the postmortem.

Criteria	Incident response	Incident management
Focus	Find the root cause and resolve the incident	Manage the incident end to end, from the first alert to the postmortem
Scope	One live incident	Every incident, plus the processes around them
Timeline	From the alert until the service works normally again	Before, during and after each incident
Who is responsible	On-call engineer and incident manager	Engineering leads and SRE managers

To learn more about incident management, read this guide: What is incident management?

What are the 4 steps of incident response?

Incident response starts when an alert reaches your team and ends when the incident is resolved. Everything in between falls into four steps.

1. Triage

Triage is the step where you confirm the incident and decide how serious it is. The on-call engineer acknowledges the alert, checks how many users are affected, and sets the incident's severity and priority. For critical incidents, they also pull in the incident manager and the experts who own the affected service.

Who does it: the on-call engineer, with the incident manager joining for critical incidents.

2. Find the root cause

Finding the root cause is the step where you work out why the service broke. The team checks dashboards, logs, and recent changes such as deployments, config changes, and traffic surges. Recent changes are worth checking first.

Who does it: the on-call engineer and the subject matter expert who owns the affected service.

3. Apply the fix

Applying the fix is the step where you stop the damage to users. Common fixes are rolling back a deployment, restarting a service, scaling up capacity, or changing a config. A quick rollback gets users back faster than a full code fix, which can follow later.

Who does it: the on-call engineer and the subject matter expert, with the incident manager approving the change.

4. Resolve the incident

Resolving the incident is the step where you confirm the service works normally again and close the incident. The team runs real checks, watches the system, and posts a final update on the status page.

Who does it: the on-call engineer and the communication coordinator.

How do you get started with incident response?

Step 1: Map your critical services

The first step is usually a list of the problems that need an alert immediately. They point to your critical services, the parts of your system that customers and revenue depend on. For an e-commerce site, that's payment processing and the checkout flow.

Step 2: Set up alerts, on-call rotations, and escalations

Alerts that trigger when a critical service fails come next, and urgent ones work best as phone calls. An on-call schedule then decides who receives them and spreads the load across the team. An escalation policy makes sure the alert reaches a backup responder if the first person misses it.

Step 3: Define incident categories

Not all problems need the same response. Simple criteria based on impact and urgency help you classify incidents, and three levels (high, medium, and low) are probably enough at first. The categories get sharper as you handle more incidents. In Spike, every incident carries a severity and a priority, so you can route each incident to the right responders based on both.

Step 4: Write a response plan and test it

A short plan for each severity level says what to do and who gets involved. A low-severity incident stays with the on-call engineer, a medium-severity incident brings in a subject matter expert, and a high-severity incident pulls in the incident manager and the communication coordinator. A practice drill shows whether the plan works, and it's worth updating after every real incident.

5 incident response best practices

1. Document step-by-step response plans for your most common incidents, like payment outages or database failures, and save them where responders can find them in seconds.
2. It can be helpful to assign an incident manager, technical lead, or communication coordinator before an incident, so nobody asks who's in charge when it starts.
3. Message templates for common incidents and communication channels picked in advance make updates easier. Critical incidents often need an update every 30 minutes.
4. Automating alert routing, ticket creation, and status page updates leaves your team more time for the fix.
5. Auto-resolving alerts nobody acts on and grouping related alerts into one incident reduce alert noise, so every page means something.

How AI is changing incident response

AI speeds up incident response by reading the alert, logs, and chat for you and turning them into context you can act on. When a responder joins, it sums up what has happened so far in one message. It also checks the incident against recent deployments, config changes, and similar past incidents, so you know where to look first.

As the investigation goes on, AI suggests the next log, dashboard, or runbook step based on what the team has already tried. It groups related alerts into one incident, so a single failure doesn't page you again and again. Once things are stable, it writes the first draft of a status update or postmortem timeline for a person to edit before it goes out.

Three decisions still stay with people: approving changes to production, sending messages to customers, and running the review.

In Spike, you can ask Spike AI inside Slack to summarize what happened, point at possible root causes, and suggest what to check next. With the Spike MCP server, you can also bring incident details and on-call information into Claude Code or Cursor while you debug.

Set up your incident response today!

Incident response isn't about never breaking things. It's about how fast you recover when they do. That's why the order matters: stop the damage first, then fix the cause, and finally learn from what happened.

You don't have to set it all up at once. Start with an alert on your most critical service, so the right engineer knows the moment it fails. Once that works, add an escalation policy, severity levels, and a review after every incident.

FAQs

Is incident response the same as a SOC?

No, they're different things. A SOC (security operations center) is a security team that watches for threats like breaches and malware. Incident response is a process that both SOC teams and engineering teams follow: SOC teams use it for security incidents, and engineering teams use it for outages, failed deployments, and slow services.

What is the difference between an incident response plan, a playbook, and a runbook?

An incident response plan sets who does what across all incidents. A playbook covers the response to one type of incident, and a runbook lists the exact steps for one task inside it, such as restarting a service or failing over a database.

When should you notify customers about an incident?

Notify customers early about critical incidents that affect many users, with a short "We're aware and working on it" message. For minor issues, wait until you know the impact and the expected fix time, so you don't cause unnecessary concern.

What metrics should you track to measure incident response?

Track mean time to detect (MTTD), mean time to acknowledge (MTTA), and mean time to resolve (MTTR). MTTD shows how fast you discover incidents, MTTA how fast someone starts working on them, and MTTR how fast you fix them.

CHAPTER 03

What is on-call management? All you need to know

This chapter explains on-call management from the ground up: schedules, rotations, five on-call models, and a 4-step setup that keeps the on-call load fair.

TL;DR

On-call management decides who responds to incidents outside office hours and how the alert reaches them. A good setup covers five things: who's on call and when, how alerts escalate, which channels they arrive on, how shifts hand over, and how fairly the on-call load is shared. When it works well, every hour has an owner and a backup. The on-call load also stays light enough for people to get proper rest.

What is on-call management?

On-call management is the process that decides who responds to an incident at any hour and gets the alert to them. A team of five might take turns being on-call for a week each, with a second person as backup. When checkout fails at 2 AM, that week's person on-call gets a phone call and the backup is ready if they miss it.

The person on-call doesn't have to fix every problem alone. Their job is to acknowledge the alert, assess the severity, and bring in help when it's needed.

For every shift, the setup settles two things:

- Who carries the pager, and for how long
- Who the backup is if the first person doesn't respond

Why is on-call management important?

It matters because problems don't wait for office hours. Without it, an alert either reaches nobody or reaches everybody, and the fix starts late either way.

Say your food delivery app stops taking orders at 11 PM on a Saturday. With nobody named for the weekend, the alert lands in a group chat that everyone has muted. The team finds out on Monday, after a weekend of lost orders. With a weekend rotation in place, one engineer gets a phone call as soon as the incident triggers and has orders flowing again within the hour.

A good setup pays off in three ways. One named owner acknowledges faster than a busy group chat, which shortens mean time to acknowledge (MTTA). The fix starts minutes after the incident triggers instead of hours later, so downtime stays short. And fair rotations keep the load within Google SRE's limit of two incidents per 12-hour shift, so the team stays rested.

What are the core components of on-call management?

1. **Scheduling and rotations:** An on-call schedule sets which hours need cover, and a rotation decides who takes each turn. Rotations usually run in daily or weekly blocks.
2. **Escalation policies:** An escalation policy moves an unacknowledged alert to the next person after a set wait time. A common path runs from the primary to the secondary and then to the team lead.
3. **Multi-channel alerts:** Alerts that flow through more than one channel give the person on-call more chances to pick up an incident. An email or a Slack message is easy to miss at night. If nobody acknowledges it within 5 minutes, a phone call follows, so the incident still gets picked up.

4. **Handoffs:** A handoff passes context to the next person on-call: open incidents, systems acting strangely, and alerts that fired for no reason. A short note at shift change saves them from starting cold.
5. **Runbooks and documentation:** A runbook is a short guide for a common issue: what to check, how to fix it, and who to call. Up-to-date runbooks and a simple map of how the system fits together let the person on-call fix problems at 3 AM without waking the expert.
6. **Health tracking:** MTTA, escalation rate, and night pages per person show whether on-call is working and whether the load is fair. Spike's shift reports show who carried the pager and for how long.

Which on-call rotation model fits your team?

If your team has	The model that fits	Why it works
5 or more people in one time zone	Primary and secondary	Every shift has a backup, and newer engineers learn with help nearby
3 or 4 people with similar skills	Rotation-based	Simple turn-taking keeps the load even
Offices in 3 or more time zones	Follow the sun	Each region covers its own daytime, so nobody works nights
Part-time staff or fixed hours	Fixed schedule	Each person keeps the same slot every week
Specialists who own separate systems	Skills-based	Alerts go straight to the person who knows that system

Each model comes with a trade-off. Follow the sun depends on clear handover notes between regions. Primary and secondary keeps two people on-call at once, so each person spends more weeks on-call than in a simple rotation. Skills-based rotations can wear out the experts. Rotation-based and fixed schedules are the easiest to run, as long as the shift length suits the team.

Setups often mix models as they grow, and Spike's article on offbeat on-call rotations shows a few less common patterns that work.

How do you set up on-call in 4 steps?

Step 1: Pick the services that need 24/7 cover

A good starting question is "Which alert would I want to wake up for at 3 AM?" For an online store, that's payment processing and checkout. For a SaaS product, it's probably login and the core API. Services where downtime affects users or revenue are the ones to cover first.

Step 2: Choose who goes on-call

The people on-call need enough knowledge of the system to fix simple problems and to know when to ask for help. Team size matters here. Google's SRE book treats eight engineers as the minimum for a single-site team that runs a primary and a secondary, so each person is on-call one week in four.

Location matters too. A team in one city needs night shifts, while a team spread across time zones can share the day between regions. Smaller teams can still run on-call by covering only their most critical services at night. Before adding anyone to the rotation, it helps to have runbooks, clear escalation paths and tool access in place. Spike's post on who should be on-call explains why these matter more than seniority.

Step 3: Choose a rotation pattern

A weekly rotation is a good place to start for groups of five or more. Smaller groups might find a 2-day or 4-day rotation easier to sustain. Each shift works well with a secondary who gets the alert if the primary doesn't respond within 5 minutes.

Step 4: Connect your tools and set the escalation

Monitoring tools like Datadog or Prometheus are what spot a problem in the first place. On their own, they don't know who is on-call. Spike fills that gap as an incident management tool with on-call built in. Once your monitoring tools are linked to it, each alert goes straight to the person on shift. If that person doesn't respond, the escalation policy passes the alert to the backup. It can also be helpful to give everyone access to these tools before their first shift, so nobody hunts for a password at 3 AM.

How do you keep on-call sustainable?

A fair on-call load

Nights, weekends and holidays should be shared evenly across the rotation. If someone has a family event, an override lets a teammate cover those hours without changing the whole schedule. Holiday shifts work best when they're planned early, as Spike's post on on-call during holidays explains.

Alerts worth waking up for

An alert that rings a phone at night should need a human. Alerts that fix themselves, repeat for the same problem or don't affect users are the first ones to tune or turn off. Grouping related alerts also means one failure sends one alert, not ten. Spike's guide to reducing alert noise covers this step by step.

Recognition for the extra work

On-call is extra work outside normal hours, so it's fair to reward it. Companies usually pay a fixed amount per shift, pay overtime for time spent on incidents, or give time off in lieu, like a day off after a week on-call. The right choice depends on your company and local labour laws, so it's worth agreeing on it before the first shift.

5 on-call management best practices

1. Alerts work best when they carry context: the service, the latest deployment, and the runbook. That saves the person on-call the first ten minutes of digging.
2. Written expectations make on-call much easier to settle into. A short on-call policy can cover how fast to acknowledge an alert, what to keep nearby during a shift, and when to call for help.
3. New engineers usually find their feet faster with a shadow shift or two before they go on-call alone. They get the same alerts as the primary but only watch and ask questions.
4. Handovers go more smoothly on a weekday morning than on a Friday evening. Both people are awake and online, and any open issue can be passed on with time to talk it through.
5. It can be useful to review the on-call load once a month by looking at alerts per person and night-time alerts per shift. If one person carries far more than the rest, the rotation probably needs a tweak. Spike's shift reports make this check quick. A test alert outside office hours each quarter is worth adding to the same routine, since it shows whether phones ring and escalations still fire.

How does AI help with on-call?

AI takes the small, repeated questions off the person on-call and gets them up to speed faster when they're paged.

In Spike, anyone can tag @Spike in a Slack channel like #on-call and ask "who's on call for payments?". Spike replies with the primary and secondary and when their shifts end. Overrides work the same way, and a plain-English request creates the override and confirms it in the channel. Engineers in Claude Code or Cursor can ask the same questions through the Spike MCP server.

When someone is paged into an incident that's already underway, Spike AI gives them a short summary of the alerts, the timeline, and what's been tried.

What should you look for in on-call management software?

Feature	Why it matters	In Spike
Rotations and layers	Covers every hour without manual planning	Yes
Overrides	Lets people swap shifts in seconds	Yes
Phone call alerts	Useful for critical alerts	Yes
Escalation policies	Calls the backup when the person on-call doesn't respond	Yes
Slack and Teams	Lets people check and change on-call from chat	Yes
Shift reports	Shows who carried the pager and for how long	Yes
Free trial	Lets you test it on your own alerts	Yes, 14 days

Integrations matter as well. On-call software works best when it receives alerts from the monitoring tools you already use (like Datadog or Prometheus) and keeps tickets in sync with tools like Linear. Check out Spike's integrations.

Set up your on-call schedule today!

On-call management comes down to one promise: when something breaks, the right person hears about it and knows what to do. The rotation models, escalation policies, and best practices in this chapter all serve that promise. None of it needs to be perfect on day one. A single rotation for the most critical service with a backup on every shift is already a working setup.

Spike lets you build that first rotation, connect an escalation policy, and test phone call alerts in one place. Try it free for 14 days.

FAQs

How long should an on-call rotation be?

A week is the usual length. Groups of fewer than five people may prefer 2-day or 4-day shifts, and anything longer than two weeks is hard to sustain.

What does an on-call schedule look like?

A common one is a weekly rotation with two people on-call each week: a primary who gets alerts first and a secondary as backup.

How many people does an on-call rotation need?

Five or more people make a weekly rotation comfortable, since each person is on-call about one week in five. Google's SRE book suggests eight for a single-site team that runs a primary and a secondary.

Should non-technical team members be on call?

Yes, for certain incidents. Support teams can handle customer updates during an outage while engineers work on the fix.

Is on-call management the same as incident management?

No, it's one part of incident management. It decides who responds to an alert, while incident management covers everything from detection to the review after the fix.

What metrics should you track for on-call?

Mean time to acknowledge (MTTA), mean time to resolve (MTTR), escalation rate, and shift load per person. Together, they show how fast alerts get answered and whether the load is fair.

CHAPTER 04

Escalation Policy Guide: Levels, Time Intervals, and Best Practices

Alerts that nobody answers turn small problems into long outages. This chapter shows how an escalation policy fixes that, and shares five best practices.

TL;DR

An escalation policy is a set of rules that passes an alert to the next person when the first responder doesn't acknowledge it. It's built from four parts: escalation triggers, time thresholds, an escalation chain of levels, and communication channels. A separate policy for each severity keeps phone calls for critical incidents and sends minor ones to Slack.

What is an escalation policy?

An escalation policy is a set of rules that decides who gets alerted about an incident, in what order, and how long each person has to respond before the next person is alerted. For example, when the primary on-call responder doesn't answer a phone call within 5 minutes, the secondary responder gets a call.

Without a policy, every alert rests on one person. If they miss it, the incident stays open and unresolved. An escalation policy sends the alert to the next person after a set time, so someone always responds and nobody has to decide at 2 a.m. who to call. It also lowers the mean time to acknowledge (MTTA), which is how long it takes someone to respond to an alert.

Escalation policies and on-call schedules are often confused, but they do different jobs. An on-call schedule decides who is responsible for responding to incidents now. An escalation policy decides who gets alerted next if that person doesn't respond. The schedule changes with every rotation, while the policy changes only when the team or the services change.

What are the types of escalations?

There are three types of escalations.

1. Hierarchical escalation: the issue moves up to someone more senior, such as a team lead or an engineering manager. It is for incidents that need a decision only a senior person can make, like taking a service offline.

2. Functional escalation: the issue moves to the team with the right skills. A database error raised by support goes straight to the database engineers.

3. Automatic (time-based) escalation: rules pass the alert to the next level when nobody acknowledges it within a set time automatically. If the primary doesn't respond in 5 minutes, the secondary gets a call.

Automatic escalation makes sure someone responds. Hierarchical and functional escalation then bring in the people who can fix the problem.

What are the core components of an escalation policy?

Component	What it decides
Escalation triggers	When an escalation starts
Time thresholds	How long each level has to respond

Component	What it decides
Escalation levels	Who gets alerted, in what order
Communication channels	How each person gets the alert

1. Escalation triggers: the conditions that start an escalation. A missed response time, a high-severity alert or a drop in payments can all be triggers.

2. Time thresholds: how long the policy waits before it alerts the next level, such as 5 minutes. The acknowledgement timeout is a second timer. It moves the incident on if someone responds but can't fix it within a set time.

3. Escalation levels: the people or teams alerted in order. Level 1 is the primary on-call, level 2 the secondary and level 3 an engineering manager. Spike's post on L1, L2 and L3 escalation covers how these levels work.

4. Communication channels: how each person gets the alert. Phone calls suit critical incidents at night. SMS and push notifications work as a backup, and Slack or Microsoft Teams suit daytime and low-priority alerts.

What does an escalation policy look like during a real incident?

Here's how an escalation policy works for a login service. It has three levels: the primary on-call, the secondary on-call and the owner of the login service. Each person has 5 minutes to respond before the next one is alerted. Once someone responds, they have 30 minutes to fix the issue before the policy brings in the next level.

At 1:10 a.m., login fails and the monitoring tool raises an alert. The primary gets a phone call and an SMS.

Let's say the primary is asleep and doesn't respond. Then the policy calls the secondary at 1:15 a.m. The secondary picks up and starts looking into it.

By 1:45 a.m., the secondary still hasn't found the cause. The 30 minutes are up, so the policy calls the owner of the login service. The owner finds an expired security certificate and renews it. Logins work again at 2:05 a.m., 55 minutes after the first alert.

Nobody had to decide who to call next. The policy alerted the next person automatically.

What happens if nobody responds?

When Repeat Escalation is enabled, the policy starts again from level 1 after a short pause of about 15 minutes. It repeats until someone responds or it reaches a set limit, such as 5 rounds. If an alert goes unanswered for five rounds, the on-call schedule probably needs fixing.

How do you create an escalation policy in 5 steps?

1. **Determine your triggers:** Triggers decide which alerts start this policy. Severity is the simplest trigger, so all SEV 1 alerts from the payments service could start it. Alerts can also be matched by their title, such as any alert with keywords like login fail, database, etc., in it. Spike's post on severity and priority-based routing shows how this works.
2. **Add an on-call schedule or a person at each level:** When a level points to the on-call schedule instead of a named person, the policy always alerts whoever is on-call. If the rotation hands over on Monday, the new person starts getting alerts that day and nobody has to edit the policy. Later levels can also point to a specific person, such as the owner of a service.
3. **Choose the alert channels for each level:** Phone calls work best for critical incidents. A simultaneous Slack message at level 1 tells the rest of the team at the same time. Low-priority policies can skip phone calls and use email or mobile app alerts.

4. **Decide on the time intervals:** There are four time intervals to set: the wait before the first alert, the time to acknowledge, the time to fix after acknowledging and how often the policy repeats if nobody responds. Critical incidents need short time intervals: no wait time, 5 minutes to acknowledge, 30 minutes to fix and a repeat every 15 to 30 minutes.
   Low-priority incidents can have longer time intervals. A 3 to 5 minute wait before the first alert lets issues that fix themselves close without alerting anyone. After that, a few hours or the next working day to acknowledge is usually enough and the policy rarely needs to repeat.
5. **Send a test alert:** A test alert shows whether the calls go through, whether the time intervals are right and whether the alert reaches the last level. Consider running one during working hours and again after any change to the schedule.

5 escalation policy best practices

1. Define your severity levels before building the policy. Severity describes the incident's impact on the users. Critical incidents could be when users can't use a core feature such as login or payments. A major incident might mean a feature is broken for some users, and a minor incident means little or no user impact. Each level then gets its own policy. Spike's posts on escalation policies for critical incidents and low-priority incidents cover both in detail.
2. Pass context along with the alert. When an incident moves to the next level, the new responder shouldn't have to start from scratch. Short notes on what's been checked and a link to the runbook let them pick up where the last person stopped.
3. Set response times that match the severity. Critical incidents usually get an immediate alert. Whereas minor incidents can wait a few hours, they still need a time limit so they don't remain unattended.
4. Match the alert channel to the urgency. A critical incident needs an alert that's hard to miss, so a phone call works best. Sending a call and a Slack message together makes it even less likely to be missed. A minor incident can go to Slack, Microsoft Teams or email instead, because it can wait.
5. Use an on-call schedule at level 1. That way the first responder isn't always the same person, and the load is shared evenly across the team.

What should you look for in escalation policy software?

Feature	Why it matters	In Spike
Phone call alerts	Reaches the on-call person for critical incidents, even at night	Yes
Response time limits	Alerts the next person if nobody acknowledges, or if someone acknowledges but can't fix it in time	Yes
Repeat alerts	Starts the policy again when nobody responds, so the alert doesn't get missed	Yes
Separate policies by severity	Critical incidents can call people while minor ones go to chat	Yes
On-call schedule sync	Alerts whoever is on duty, so nobody has to edit the policy after each rotation	Yes
Slack and Microsoft Teams alerts	People can see and acknowledge alerts in the chat tool they already use	Yes

In Spike, you build an escalation policy one level at a time. Each level says who gets alerted and how: a person or an on-call schedule, by phone call, SMS, email, push notification, Slack or Microsoft Teams. If nobody acknowledges, the policy moves to the next level and keeps repeating until someone does. On top of these basics, Spike adds four features that make policies easier to run:

- **Slack mentions:** an alert can mention @channel, @here or one teammate, or even no mention.
- **Escalation history:** every change to a policy is saved, along with who made it.

- **Wait time:** a short pause before the first alert, so issues that fix themselves don't alert anyone.
- **Simultaneous alerts:** one level can alert several people and channels at the same time.

Set up your first two escalation policies today!

An escalation policy doesn't have to be perfect on day one. Two policies are enough to start: one for critical incidents and one for low-priority ones. The critical policy calls people and moves to the next level within minutes. The low-priority policy sends a Slack message or email and gives people more time to respond. Link both to the on-call schedule and send a test alert. Real incidents will then show what to adjust.

Spike lets you set up that first policy and hear a real test call within minutes. Create your first escalation policy today!

FAQs

How many levels should an escalation policy have?

Three or four levels suit almost any service: the primary, the secondary, a specialist or service owner, and a manager.

When should managers or executives be included in escalations?

Managers and executives should be included when an incident goes beyond what the on-call team can handle. This covers incidents that affect critical business operations, need a decision above the team's authority, or need several teams to work together to resolve.

What if an incident resolves itself before anyone responds?

The policy stops as soon as the incident resolves, so nobody else gets alerted. A wait time of 3 to 5 minutes before the first alert also keeps these short-lived issues from waking anyone up. The incident should still be logged and reviewed, though.

How often should you review an escalation policy?

Every quarter is a good rhythm. A team change, a new service or a major incident are good reasons for an extra review in between.

Do support and IT teams need an escalation policy too?

Yes. Support and IT teams handle urgent issues too, such as a customer who can't log in or an office network that goes down. Without an escalation policy, these issues can sit with nobody assigned and miss their service level agreement (SLA).

What metrics show an escalation policy is working?

A working policy means fast responses, few false alarms and a team that isn't burdened. Mean time to acknowledge (MTTA) shows how fast people respond. The escalation rate shows how often alerts pass the first level, where a high rate means missed alerts. The false alarm rate shows how many alerts needed no action, and out-of-hours alerts per person show whether the load is fair.

CHAPTER 05

Build vs buy incident management: how to decide

Not sure whether to build your own incident management tool or buy one? This build vs buy incident management chapter compares both on real 3-year costs, setup time and reliability.

TL;DR

Buying is usually the better choice in the build vs buy incident management decision. Alerting, on-call schedules and escalations are solved problems, and vendors have spent years making them reliable. Building only makes sense if you sell incident tools yourself or if strict rules stop you from using an outside vendor. The first version is quick to build, but keeping it running costs far more over time.

Why do teams consider building their own incident management tool?

1. It fits your own workflows and terms. An in-house tool can follow your incident process exactly. It can use your own severity labels and incident stages, and it can connect to your internal tools.

2. There's no per-user bill. The engineers who build it are already on the payroll, so there's no subscription to approve. The cost doesn't disappear, though. It moves into engineering hours and hosting, which the next section covers.

3. You control the features and the data. You decide what gets built and when, with no feature request waiting on a vendor's roadmap. You also decide where incident data is stored, which matters for companies with strict data residency rules.

4. AI makes the first version quicker to write. AI coding tools can produce a working Slack bot or alert handler in days rather than weeks. That makes building look cheaper than it used to. The first version is still the smallest part of the work, though.

What are the hidden costs of building in-house?

Development time and resources

A basic version that opens a Slack channel and alerts one person can be ready in a few weeks. A tool people trust at 3 AM takes several months of building and testing. After that comes the ongoing work of bug fixes, new features and questions from the people who use it. That time comes from engineers who would otherwise work on your product.

Complexity over time

Alerting someone is only one part of incident management. A complete tool also needs:

- Escalation policies that call a backup when nobody responds
- On-call schedules with rotations, overrides and time zones
- Alerts by phone call, SMS, push, Slack and email
- Integrations with monitoring tools like Prometheus, Datadog or AWS CloudWatch

Alert delivery is the hardest of these to get right. A phone call can go unanswered, a carrier can filter an SMS and the calling provider can have its own outage. An in-house tool needs the same safeguards before anyone can rely on it at night.

Incident risks for your incident tool

What happens if the tool you built to manage incidents has an incident? If it runs on the same servers or login system as your product, it can go down in the same outage. The team then has two problems at once and no alerts for either.

Keeping the tool up means hosting it separately. It also needs its own deployment process, its own monitoring and someone on call for the tool itself.

Opportunity cost

Every hour spent on an internal incident tool is an hour not spent on the product your customers pay for. For a company that doesn't sell incident tooling, that trade is rarely worth it. The engineers best placed to build reliable alerting are usually the same ones your roadmap depends on.

The tool becomes a second product

Once people rely on it, an in-house tool becomes a product of its own. It needs an owner, updates, documentation and support. When the engineer who built it moves on, the knowledge often leaves with them if not documented well. Changes start to feel risky, so the tool stays as it is while the services around it keep changing.

How much does it cost to build vs buy incident management software?

Building usually costs several times more than buying over 3 years. Buying is a predictable monthly fee per user. Building means paying engineers up front, then paying for servers, security and maintenance for as long as the tool runs.

Here's a 3-year example for a team of 20:

Cost	Build in-house	Buy (Spike Business, 20 users)
First version	$75,000	None (set up in hours)
Maintenance per year	$45,000	Included
Servers, phone and SMS per year	$5,000	Included
Subscription per year	None	$3,360
3-year total	$225,000	$10,080

Buying Spike saves this team about $214,920 over 3 years, which is roughly 95% less than building. Spike prices are list prices as of October 2026.

What does the first version cost?

An incident tool needs more than a Slack bot. The first version must route alerts to the right team, follow escalation rules, make phone calls and send SMS, and keep a timeline for each incident. Building all of that takes months.

A simple way to estimate the cost is engineers × months × monthly salary. For example, one engineer earning $150,000 a year spends 6 months on it. That comes to about $75,000.

Big software projects also often grow beyond their plan. A McKinsey study of large IT projects found they ran 45% over budget on average and delivered 56% less value than expected.

What does it cost to keep it running?

The costs don't stop after launch. Three of them come back every year.

- **Maintenance:** bug fixes, security updates and new features usually take a quarter to a third of one engineer's time. At 30% of a $150,000 salary, that's about $45,000 a year.

- **Servers:** the tool has to run apart from your main product, so it keeps working when your product goes down. That means separate hosting, backups and monitoring, plus a phone and SMS provider like Twilio. This example budgets about $5,000 a year for it.
- **Security:** incident data often describes weak spots in your systems. That makes security reviews and strict access controls necessary, and both add cost.

Outside tools keep changing as well. When Slack or Twilio updates its API, your team has to fix the connection.

What does buying cost?

Buying is a subscription priced per user each month. Spike costs $7 per user a month on the Standard plan and $14 on the Business plan, which includes unlimited phone and SMS alerts. For a team of 20 on Business, that's $280 a month or $3,360 a year. Maintenance, servers and phone delivery are all part of the subscription.

Why is buying usually the better choice?

1. Fast setup. A bought tool can be running in hours. Connecting a monitoring tool, adding an on-call schedule and setting an escalation policy can usually be done in an afternoon.

2. Reliability that doesn't depend on your servers. A vendor runs the tool apart from your product, so it keeps working when your systems go down. A public status page shows how well it has held up. Spike's status page tracks phone call and SMS alerts as separate parts.

3. Continuous improvements. New features, security fixes and performance updates arrive without any work from your team.

4. Ready-made integrations. Spike connects to 180+ monitoring tools, cloud platforms and error trackers, including Prometheus, Datadog and Sentry. It also works with Slack, Microsoft Teams, Zendesk and Freshdesk from day one.

5. More time for your own product. Engineers stay on the work customers pay for, while the vendor's team looks after the incident tool.

What are the downsides of buying?

Subscription cost

A subscription is a bill that arrives every month and grows with every user. It helps to compare what each plan includes, since phone and SMS alerts are often limited on cheaper plans. Spike's Standard plan caps phone and SMS alerts at 100 a month, while Business has no limit.

Adapting to the vendor's setup

A bought tool comes with its own way of working. Your team might need to adapt to how the vendor handles severities, rotations or postmortems. Before choosing one, it's worth checking whether you can set your own severity levels, rotation patterns and escalation rules. Settings like these usually cover what people think they need to build.

Vendor lock-in

Once your team relies on a tool, leaving it gets harder. Moving incident history and settings to another tool takes work. If you need a niche workflow the tool doesn't support, you also wait on the vendor's roadmap instead of building it yourself. A tool with an open API, webhooks and an easy way to export data softens both problems, so these are good things to check early. In Spike, a REST API and webhooks let you add your own steps on top.

The vendor shuts down the product

It does happen. Atlassian stopped selling OpsGenie in June 2025, and support ends in April 2027. Any data that hasn't been moved by then gets deleted. OpsGenie customers now have to migrate whether they like it or not. And if you happen to be an OpsGenie user looking to migrate, we've got you covered with OpsGenie migration resources.

When does it make sense to build your own incident management tool?

1. Your needs are unique. Some workflows are so specialized or so tied to your own systems that no ready-made or open-source tool can adapt to them. Strict rules are one example: some defence, government and banking systems run on networks with no internet access or must keep all data in one country. Consider checking these rules with your security team first, since they're sometimes assumed rather than written down.

2. You operate at a very large scale. Companies the size of Google or Meta run incidents across thousands of engineers and services. At that size, per-user pricing adds up fast and a custom tool built around their own systems can work better.

3. You have a dedicated team to run it. Large companies sometimes have DevOps or SRE teams whose full-time job is internal tools. With that team in place, the tool always has an owner and the work doesn't take time away from your product.

4. Cost isn't a concern. Building only makes sense on budget grounds if you can pay for years of maintenance, updates and redesigns, not just the first version. With that budget in place, building gives you full control over the code and data.

Build vs buy incident management: a side-by-side comparison

Factor	Build in-house	Buy
Initial cost	Low cash outlay, paid in engineering hours	A monthly subscription per user
Maintenance effort	High and ongoing: bug fixes, API changes and feature requests	Handled by the vendor
Customization	Fully customizable	Configurable within the vendor's settings
Time to value	Several months for a reliable version	Hours to days
Scalability	Needs more development as you grow	Grows with your plan
Reliability	Depends on your own hosting and on-call cover	Runs apart from your product
3-year cost for 20 people	About $225,000 in engineering time and hosting	About $10,080 on Spike Business
Integrations	Each one built and maintained by your team	Ready-made for monitoring and chat tools
When the builder leaves	The knowledge often leaves too	Documented product and vendor support

Key considerations before you decide

Is incident management part of your core business?

Incident tooling is core only when it's what your company sells, like an alerting or monitoring product. For everyone else, it supports the product rather than being the product. Alerting the on-call engineer, opening a Slack channel and updating a status page are standard jobs that ready-made tools already do well. Building only fits strict needs no vendor supports, such as an air-gapped network with no internet access or a regulatory framework unique to your company.

How soon do you need it?

A bought tool can be set up in hours or days. Building one can take 6 to 12 months of design, coding and testing. Until it's ready, your team handles incidents without a proper workflow.

What will it cost to keep it running?

An in-house tool is never really finished. Each time your chat app, monitoring tools or cloud setup changes, the tool's connections can break and someone has to fix them. If the engineer who built it leaves, that work falls on people who never planned for it. A subscription is easier to plan for, and it usually costs far less than the engineering time a built tool needs each year.

How reliable does it need to be?

Your incident tool has to keep working when your main systems go down. A vendor runs it on separate servers with backups built in. If you build it, its reliability depends on how much time your own team can give it.

Which tools does it need to connect to?

A bought tool comes ready to connect with monitoring tools, ticketing systems, deployment pipelines and chat apps. Building means writing and maintaining a new connector every time your team adopts a new tool.

Spend your engineering hours where they count

The build vs buy incident management question usually comes down to one thing: where your engineers' time is best spent. Unless your company sells incident tooling, that time is worth more on your product than on an internal tool. A practical next step is to list your must-have features and run the cost example above with your own numbers.

If buying comes out ahead, Spike lets you test it on real alerts first. You can connect a monitoring tool, add an on-call schedule and hear a real test call within minutes.

FAQs

What is the build vs buy decision in incident management?

It's the choice between creating your own tool for alerts, on-call and incident response, or paying for a ready-made one. The decision weighs cost, setup time, upkeep and how much control you need.

Is it cheaper to build or buy incident management software?

Buying is usually cheaper once engineering time is counted. A 20-person team pays $3,360 a year for Spike's Business plan, while upkeep alone on an in-house tool can take a quarter of an engineer's time.

How long does it take to build an incident management tool?

Usually 6 to 12 months for a tool your team can rely on. A basic Slack bot that alerts one person can be ready in a few weeks, but escalations, on-call schedules, reliable phone and SMS alerts and separate hosting take much longer. The work doesn't stop at launch either: keeping it running takes a quarter to a third of an engineer's time every year. A ready-made tool like Spike can be set up in a few hours.

Are there free, open source incident management tools?

Yes, a few are still in active development. GoAlert was built at Target and handles on-call schedules, escalations and phone or SMS alerts. OneUptime combines monitoring, status pages, on-call, and incidents in one platform. Keep groups and filters alerts before they reach the on-call engineer. All three are free to download but still need hosting, upkeep and your own phone and SMS provider.

What happens if an incident management vendor shuts down?

You move to another tool before the deadline. When Atlassian ended OpsGenie, customers got nearly two years' notice and a way to export schedules, escalation policies and alert history. Spike's OpsGenie migration guide walks through the move.

Does team size change the build vs. buy decision?

Yes, but mostly at the far ends. Small and mid-sized teams rarely have spare engineers to build and maintain an incident tool, so buying almost always fits. Very large companies with thousands of engineers and a dedicated platform team are the exception. At that size, per-user fees add up and a custom tool can pay for itself.

Can a bought incident management tool meet compliance needs like SOC 2 or HIPAA?

Yes, in most cases. Established vendors usually publish their security and compliance details: SOC 2 reports, ISO 27001 certificates or HIPAA support. It can be useful to ask for these during a trial and to check where the vendor stores your data. Building only makes sense when a rule goes beyond what any vendor offers, such as a network with no internet access. Spike's security page lists its own measures, including encryption at rest and GDPR compliance.

Need help setting this up?

Write to support@spike.sh, or book a call with our team at spike.sh/demo, and we will help you set up Spike.

Read this online: spike.sh/guides