RAYGUN

The On-Call Survival Guide

How to ensure your team is prepared for when disaster strikes

By Nick Harley and Jason Fauchelle

Published by Raygun.com - www.raygun.com. © 2017 Raygun Limited - First Edition

All rights reserved. No portion of this book may be reproduced in any form without permission from the publisher, except as permitted by copyright law. For permissions contact: support@raygun.com

CONTENTS

Introduction

In today’s technology-driven world, almost every business is using software to reach their customers.

This no longer applies to just businesses that offer technology solutions to technically savvy buyers. This captures all sorts of companies, industries, and sectors from small high street retailers to tech industry giants.

Over 2 billion people now have a smartphone and can use it to pull up information in a heartbeat. We are more connected than ever before.

Without investment into search, responsive marketing sites, mobile apps, online stores and up to date applications that perform well for end users, companies that are slow out of the gate can find themselves outmaneuvered by their competition.

If we take the example of Home Depot, the building supplies store has historically had to consider the simple business model of a customer entering a shop and handing over their hard-earned cash in exchange for products. As technology has moved on, however, the Home Depot business had to become a software company too.

As technology advanced, customers didn’t just use the Home Depot website for information on store hours and locations, they wanted to purchase products online too. Then, real-time stock level visibility for stores is required; a mobile app, online ordering and store collection, delivery fleet tracking and management. The list goes on.

As customers demand and expect to be able to have these options open to them when interacting with a large business, demand needs to be met. Without this innovation, customers would simply head elsewhere, to competitors that provided a smoother interaction between the online and offline world.

Over time, even a building supplies company has now morphed into a huge tech giant, hiring and employing hundreds or thousands of technical employees.

This is the case for the large majority of companies who have had to make the shift from non-tech to heavy-tech reliance as software “eats the world”.

Just look at Dominos share price over time for another company that is constantly pushing the boundaries of technology, yet their main line of business revolves around cooking round discs of dough (with plenty of toppings of course).

The “uptime” economy

Companies and businesses all over the world now have more demand than ever before from consumers wanting 24/7 access, functionality and information. Just a short outage can affect a brand’s reputation as well as losing out on significant sales.

After Amazon’s homepage was down for 49 minutes in 2012, it was reported, based on their $61.09 billion 2012 net sales this cost the company $116,229 in lost sales every minute. Just under $5.7M in total. On current annual sales figures of over $100 billion, you could likely double that figure should history repeat itself today.

Amazon is an extreme example but highlights the fact that downtime is a costly business for all companies that have a reliance on software to serve their customers and generate sales. It is, therefore, quite rightly feared and mitigated against in every tech-oriented organization.

High availability

It is an inevitability that your company will at some point experience a service disruption causing some level of downtime. It's just a reality of complex software systems that a bad release could cause considerable negative effects or the service can go down entirely.

Nor can you plan for everything. Your application may experience high traffic spikes or unexpected high volume usage from individuals stamping up and down on your API with crazy numbers of requests.

These issues often have no warning. What separates the best from the rest is how issues are managed and resolved within tech teams when things do turn sour. Minimizing the impact down to the smallest numbers of customers for the smallest amount of time is paramount.

How do you calmly address technical issues and ensure all users, your fellow team members and your peers are delivered solutions, not further problems or blind panic? Outages shouldn't mean you are awoken at 3 am only to try and solve problems in a cold sweat as the minutes tick by.

Not only are there many tools and services available at your disposal that make issue management and team communication a breeze, common practices leveraged by bleeding-edge tech companies can help keep applications stable, reliable, and available at any scale.

We hope that this guide helps you feel more prepared and therefore comfortable with being on-call by sharing insightful information on how you can make outages (or planning for them) less of a thing to be feared and part of a great on-call “culture of uptime” within your organization.

Why being on-call is so important for software teams

Any team that cares about producing software of quality that performs for customers should know the importance of being on-call. Teams that do not require engineers to be on-call (or rarely have incidents that require immediate fixes) haven’t simply got to that position by chance.

These engineering teams have been able to build scalable, reliable, robust software and infrastructure that can withstand various issues without falling over entirely or descending into a spiral of the knock on effects. These incidents may become critical events for teams that have not pre-planned their resolution options.

Anyone who is on-call shouldn’t be getting into situations where members of their team are unavailable for a critical service whilst senior management are barking at you to ‘just fix it’.

Those put in this position often have no choice except to hijack repository permissions and write patches just to stop the fire from spreading. Naturally then comes the inevitable reprimands for such heroic acts. This messy and anti-collaborative approach to being on-call can make engineers wary of the responsibility entirely, and without the guardrails of a well-defined on-call shift, this is what “no-on-call” organizations descend into.

In the large part, if things are set up in the right ways, being on-call is not a difficult job. It's really up to you and your colleagues to build processes and systems to make it that way.

Even with a barrage of testing in staging or QA, you will still find problems in your software that only exist in the production environment, and it won't be for lack of trying to unify fiddly things like configuration parameters or runtime versions.

Culture of "uptime"

The trick to being on-call is to be effective and efficient. Resolving issues quickly is your job. It's on you to fix the issue swiftly before support requests flood your customer support channel and social media lights up with angry users.

Colleagues and managers begin to angrily seek answers as infrastructure queues start to spiral out of control impacting more and more users. You need to be made aware of and recover from this problem as quickly as possible.

Unless you are a global operation with senior technical employees within every time zone, naturally there are going to be periods within the day when your core team is not sitting in front of their computers ready to fix problems at a moment's notice.

Approximately one-third of your day may be spent at your workstation, but issues can tend to arise at the most inconvenient of times - During your evenings and weekends or in the middle of the night.

For teams that are based in other areas of the world but serve a mainly US based customer base, this problem is made even worse due to the time differences. Your highest traffic volumes could be seen in the middle of the night rather than your local timezone, so infrastructure falling over can lead to problems needing to be fixed in nocturnal hours regularly.

Being on-call is not only important for your customers to ensure they are not disturbed by outages or serious issues affecting their user experience, it’s also hugely important for technical teams to have strict and well-organized processes around issue management when disaster does strike.

How prepared would you be if your website unexpectedly went offline at 3 am tonight?

Improvement & innovation

Being on-call shouldn’t be seen as an inconvenience or task that nobody wants to do in the organization. Take pride in your team’s ability to solve issues quickly, no matter what time of day or night they happen. Constantly strive to improve not only the infrastructure and code base but the response time and methods towards the recovery of disruptions as well.

Customers demand “life-improving” enhancements and experiences from the products and services we build and support.

Working out your priority levels

Different priorities fall into different categories. You should know how to recognize a critical outage. However, just because it's critical, you may not know how to fix. Critical outages sound terrible, but if a server has fallen over it should be as simple as jumping in and fixing it quickly. This list is a guideline to categorizing the different issues and knowing how urgently you are expected to solve it, and if it needs to be an issue that is addressed by the on-call team.

Critical outage

The highest priority incident is that which causes customer data loss.
For anyone on-call outside office hours, this is the kind of incident that warrants immediate escalation to one or more engineers. During office hours, all engineers stop what they are doing to focus on resolving the issue immediately. Customer data loss can occur by all entry API nodes going down, or by a server running out of memory with nowhere to put new incoming data.

Loss or degradation for subset

One or more subsystems have gone down, causing customer data processing to halt to some degree that customers will notice. No customer data is lost however due to buffering the data in queues between subsystems. You want to resolve this issue immediately as it can greatly damage the user experience, and can cascade to further issues if not resolved.

Loss of minor functionality

A process has encountered an issue that is not part of the critical customer data flow, but instead is responsible for tasks off to the side that may be triggered by a user action such as generating a custom report. If left unattended this can do no further damage to the system, but affects customer experience and so will trigger an on-call alert and should be fixed swiftly.

No customer-visible impact

A redundant system fell over and failed to recover. Due to redundancy, this has no impact on the user's experience beyond an ideally unnoticeable degradation of processing data. This kind of incident wouldn't raise any alerts (unless of course, all related redundant systems fell over) and can be resolved during office hours.

All others

A rare bug that is enough to knock down a system that quickly auto restarts itself and in the end requires no on-call action. This is not enough to raise an on-call alert, and just gets noted down to be fixed when it happens enough to be annoying.

How do you know when something went wrong?

Monitoring and detecting are the processes of using one or more systems that run checks, receive results and checks them against configured thresholds, triggering an incident.

Monitoring services are used in conjunction with an Incident Management service to ensure the fastest path to recovery from a service disruption. Based on configurable rules, the Incident Management system applies a logical operation to determine a number of things:

- Is the incident critical enough to require action?
- what team or individual should be alerted to an incoming incident
- who within that group is currently on-call
- what method that individual is to be contacted
- perform automation to provide the first responder with valuable context and resources to recover as quickly as possible

The system

System monitors tend to be more interesting to operations teams (i.e. Ops, SRE, System Administrators)

No matter how prepared you think you are, there is always more that can be done to prepare yourself for jumping into action when times require it. We'll assume you have the basics covered and have your laptop or a desktop set up nearby with internet access (you'd be amazed how many developers can be called upon and they yet don't have the ability to work remotely from anywhere other than their workstation!)

Ensure that things are set up and ready to go, a phone is close by and charged, a laptop is also close by and running out of battery is not a risk you're unprepared for. If you're stumbling around in the dark half asleep, getting online should be as easy as opening the lid of your laptop.

As a habit, you should test your setup frequently. We'd even go as far as saying you should incorporate a little test of your setup or checklist into your going to bed routine. This will give you absolute confidence that when you're woken in the night, you will be ready to go.

Tip: Your mobile phone will have a do-not-disturb function which is worthwhile enabling, even when on-call. Getting a good night's sleep is important and you don't want to be woken by late night Facebook or Twitter notifications. Just make sure you add any paging system phone numbers or apps to the exceptions list so you do get on-call alerts.

Systems to be monitored include:

- Servers
- Switches
- Routers
- Instances
- Containers
- Clusters
- Databases

The application

Application monitors tend to be more interesting to developers.

The most important part of being on-call is having access to the information you need in order to diagnose and fix issues. Without the relevant information on hand, you'll be floundering and wasting a lot of time.

Make sure the information is real-time or if in a document form then it is kept up-to-date. Create a shared folder with your on-call team using a secure online cloud storage provider. Store any relevant documents here and ensure you have it synced to your local machine for ease of access. Some of the types of documents you might want to create would be:

- An infrastructure list - List the servers in your environment and where services are located. If there are secondary services for failover, then list where are these located. IP addresses and names of servers can be useful too when looking at logs, you may need to match one to the other

- Your on-call policy. This should include guidelines and expectations for everyone who is on-call. Expected response times and guidelines on how to classify the urgency of an issue. Who else to get involved when you are unable to deal with the issue at hand

- A phone list of all people involved in the on-call roster or key personnel responsible for production systems

- A troubleshooting or how-to guide where you document common issues and how to resolve them

- A setup guide or checklist for new people going on-call. This should include checklists of permissions needed for each system.

- How to configure firewall rules so you can access your environment remotely and anything else that may be needed such as generating SSH keys

Bookmark links to infrastructure dashboards that allow you to get a high-level overview of your system health. Datadog is a great product that allows you to build custom dashboards of both high-level and more detailed system monitoring and metrics. They also provide a number of predefined plug-and-play dashboards for commonly used systems such as AWS, Redis, RabbitMQ, MySQL, Postgres and much more. Datadog can also be used to configure threshold alerts and monitors based on metrics. These can be a great addition to any existing monitoring you have in place or can provide an early warning system for your software.

When trying to diagnose an issue, you want as much information at your disposal as possible. Many standard products also come with monitoring dashboards you can plug-in to get detailed information on a component of your system. We’d recommend installing these and exposing them externally via a secure channel so you can access them remotely when it matters the most.

As with hardware preparation, software preparation is just as important. Make sure that you can access the system dashboards from your remote location. Make sure you have access to all the information and tools to be an effective incident responder.

Before you join the on-call roster, get your remote location fully set up and practice logging into each system to ensure it's all ready to go. Make sure your home on-call environment is comfortable, in particular, ensure you are able to keep yourself warm during the night.

Any good operations person will want to lock down a production system by minimizing who can access it and where they can access it from. This, however, conflicts with the need to provide on-call support from a remote location. So how do you provide access to production infrastructure for on-call staff?

One option would be to set up a Virtual Private Network (VPN) to your office network. Generally, there will be a trusted relationship between your place of work and your production infrastructure. If you can provide a mechanism that allows someone on-call to join the work environment then they can utilize the existing trusted relationship to access production infrastructure as though they were in the workplace.

Another option is to allow the on-call person direct access from their remote location to the production infrastructure on a temporary basis. This would generally involve setting up firewall rules to allow Remote Desktop (RDP) or Secure Shell (SSH) connections through to your production servers from a specified IP address. However, this is generally much harder to manage due to the dynamic nature of IP addresses, that an IP address can change, or that the on-call person may be remotely accessing the production infrastructure from a shared location.

If you do allow this sort of access then ensure there is a good auditing policy and there are procedures in place to reverse out any changes to the firewall rules after the incident is resolved.

A third option that involves more work would be to develop or use a third-party agent that is installed on production servers. This agent can be configured to communicate with a known central server over a secure SSL channel. The agent can then perform a limited set of tasks on behalf of an authenticated user. This could include the ability to download, search and tail log files, view config files, view and search Windows Event Viewer, restart applications or services and track server and application changes.

The business

Business monitors tend to be more interesting to management.

What is the core service the company is providing and is that being monitored? The company exists to make a profit. All metrics supporting that goal are important. Still, if the company isn't making money, senior leadership should be aware and able to make necessary course corrections to recover and improve the overall direction of the business.

With more companies moving to cloud and serverless, they care less about CPU load or Memory consumption and more about "Is our service available and working". As a result, they monitor for business metrics that are as important (if not more) than any other data measured (i.e. are we making money?)

The individuals

Deciding who will be on-call on your own team can be a frustrating and complex task at times. If you're a CTO or Lead Developer, it can often be expected that you are the one to be on-call. It's your job to deliver on the company goals with technology, and that includes doing whatever it takes to keep the application running for customers.

After all, you're going to be the one in the firing line should things be falling over on a regular basis. But are others on the team able to assist in recovering from a service disruption? Of course, no highly available complex system survives a single point of failure, including a single person responsible for the "uptime" of a service.

High-performing teams identify and establish roles as they shore up their on-call duties. Clearly defined roles help teams to organize quickly and begin on their path to service recovery.

On-Call Roles & Responsibilities

Regardless of if your organization has been structured by the breakdown and assignment of responsibilities to numerous teams (Dev & Ops), understanding who should be on-call can be extremely complex.

Some of your team may have children, dependant family members or other commitments whilst others on the team don't have any of these things to work around. As much as it would be ideal to throw the on-call hospital pass to your newest and youngest junior developers like most other jobs that you feel keen to delegate, getting the company's infrastructure back online in the middle of the night may not be one task you want to delegate.

Sharing the responsibility around your fellow team members helps take the onus off the same individuals having to deal with critical issues. Empathy is at the heart of DevOps and load-balancing the responsibility of being on-call is a sure fire way to begin building it.

Some incidents can be more swiftly resolved with more heads - such as needing to shut down all processes across multiple machines that make up a cluster, in order to restart it all up again.

Each member can focus on a subset of the processes. In these cases, the Primary, or at least the person who acknowledges the incident should be responsible for coordinating who does what. Having this responsibility logic in place can help avoid individuals tripping over each other trying to take charge in what they do. Including guidelines for how the primary contact can coordinate these incidents in some kind of playbook is a good way to educate anyone that will be a primary contact in future.

Primary contacts

At any point in time, there is always a Primary on-call developer who will be the first to receive incident alerts. When it’s their turn, the Primary will be the one responsible for keeping things under control outside of work hours.

It’s generally wise for the Primary to avoid going to movies (where you’d not be able to, or not want to receive an alert) or go anywhere that you don’t have quick access to any means of resolving an incident.

Secondary contacts

There is also always a Secondary on-call developer available for two main purposes. Firstly in the event that the Primary misses an alert for whatever reason, the Secondary acts as a backup to pick up the slack. Secondly, high priority incidents are best managed by multiple engineers, so the Secondary is the go-to engineer for immediate assistance. As such, it’s best for the Secondary to be readily available at all times just like the Primary.

Teams & Rosters

Even if the structure of your teams are divided into “developers” and “operations”, delegating the on-call responsibilities to more across all teams is the best way to make sure that on-call responsibilities are shared, tribal knowledge is disseminated, empathy is built, and problems are solved faster. Creating a roster amongst the wider team of who is on-call, for what types of services or issues, as well as when and who else is available, is where you should start.

Having multiple Subject Matter Experts (SME) from various domains on-call together helps to spread vital information and expertise across teams. Gaining a deeper knowledge of the code and infrastructure happens naturally as on-call responders are alerted to, investigate, diagnose, and resolve issues as a team. This type of collaboration and response leads to building really high-performing IT teams. But it all starts with building a roster for your teams.

Rotations & Scheduling

At Raygun, we have a rotating schedule of several on-call developers that will either be a Primary or a Secondary, one week at a time. Each developer will first be a Secondary for a week, before then becoming the Primary the following week, then having a break until their turn comes back around.

The length of an on-call shift will vary for different organizations. You want to pick a routine that will work best for your team. If you have developers all around the world, you may be able to take advantage of this by maximizing how much on-call work is done during daylight. Eight or twelve-hour shifts are common for this. We recommend that on-call shifts are no longer than two weeks to avoid exhaustion and to better share the load, giving each on-call developer a good balance between gaining on-call experience and having a break.

One week shifts work well, as we can then have an on-call hand-off meeting once per week immediately after the usual weekly team meeting. At on-call meetings, the Primary lists the incidents that occurred, we discuss any actions that we want to take throughout the week to mitigate or resolve commonly occurring incidents, and finally make sure everyone knows who is Primary and Secondary.

The time at which hand-off occurs is also important to consider. Our on-call hand-off occurs first thing Monday morning during work hours, which happens to coincide with our weekly meeting. At the very least, handing off during work hours is a wise move. Whatever schedule you decide on, keeping things as consistent as possible makes it easy to remember when it’s everyone’s turn.

Paging Policies

As mentioned previously, the current Primary will be the first to receive incident alerts. Upon receiving an incident alert, the Primary must then mark the incident as acknowledged, and then set to work on resolving the issue.

In the case that the Primary misses the alert, you don't want the issue to just sit there. Instead, the incident will automatically escalate to Secondary if it remains unacknowledged for a short amount of time. You want this amount of time to be long enough for the Primary to acknowledge it, but short enough for potentially critical issues to not be left for too long.

In the unlikely event that something happens to the Primary after acknowledging, such as stumbling down the stairs or dozing off, the incident will become unacknowledged after some time and then escalated to secondary if not re-acknowledged.

ChatOps

Issue management and resolution don't need to involve panic, blame and miscommunication. The days of scrambling to resolve a problem in a cold sweat are behind us.

If teams can embrace ChatOps within their own organizations and automate many processes with the use of bots, third-party plugins and tools, they stand to make big gains in the efficiency of their wider team, and at the same time, make the management of issues in their software applications stress free, collaborative and fully transparent.

It’s one thing to be alerted to an issue, that’s just the notification part, but without the diagnostic details about the problem, we can spend a whole bunch of time just trying to work out the cause whilst our users continue to see problems, support queues start to fill up and the issue grows into a major company-wide incident.

If we can get this information immediately at the time the incident notification comes through, fixing and debugging can be done at record speed and your whole company can benefit, as well as your end users.

Many developers may automatically assume this only extends to servers falling over, database timeouts and more, but it can also include production errors your users are encountering every day. If you knew that a recent update has broken your billing system and customers couldn’t pay you, surely you’d be scrambling to fix that pretty quickly!

If you already use team chat applications in your business, you might already have several pieces needed to get an awesome issue management workflow operational, connecting these pieces together can create a world-class incident management workflow and ensure our entire team is following along from the first alert to resolution.

Read the complete guide to ChatOps in our ebook

When The System Goes Down

Everyone owns and is responsible for the application code as well as the infrastructure and its availability. The overall service is where our focus is. Customers or end users ultimately don't care who is responsible for what. They only know that the service they are attempting to use isn't behaving as expected or is completely unavailable, and that's a problem for them.

What is important for organizations is that anomalies or trends towards an imminent problem should be monitored and alerted to as quickly as possible.

Engineers and system administrators who may be solely responsible for infrastructure should be part of regular on-call rotations. Everyone owns and is responsible for all of it.

Alerting

Based on monitor status, an alert will be sent to the current Primary when enough has gone wrong to require attention. At Raygun, we have a simple scheme in where an alert will be triggered when any two monitors are tripped. The monitors are set up in a way that any real issues will always cause two monitors of some kind to trip.

If a process falls over, then that's one monitor. The fact that the process has fallen over means that the queue that the process is responsible for will start building up which will trip another monitor.

If a process stalls or slows down, then again the queue will build up and trip 2 monitors at different thresholds. This simple scheme helps to weed out false positives. If a process is knocked over, it can usually auto restart a little while later. Sometimes extraordinary load could trip a queue monitor for example.

Triage

Regardless of if an incident occurs during office hours or not, it's best for the current Primary to resolve any issues that occur. This simple logic removes the question of who should fix the issue during office hours, and also gives the current Primary the opportunity to fix issues they've never resolved before - useful experience to have during a 3 am crisis. If you absolutely don't know what to do, don't be ashamed to escalate the incident or ask for help, as long as the issue is resolved and you learn what to do for next time.

Identification

This is the process of figuring out what's gone wrong. If you've set up the monitors well, then simply the name of the monitors that triggered an alert, or any description attached to them will be enough to point you in the right direction.

Investigation

Sometimes you may need to do further investigation to aid in resolution. This may involve double checking any customer facing aspects related to the incident in order to update a public status page. Another step may be to check related dashboards to drill down on the specifics of what's gone wrong.

If the incident that has occurred has never happened before and hence no documented resolution procedures are applicable, plus if any generic procedure such as restarting a server or process does not work, it's always good to start by asking "What has recently changed".

Resolution

As with most things in life, communication is a very important part of resolving incidents. At Raygun, we set up a single Slack channel to pipe monitor trips and incident alerts, and then we send notes about everything that we check, attempt or do throughout the resolution of an incident. Keep Slack open in a separate window or device so that you can quickly jump between sending a note and performing on-call resolution procedures.

When there are multiple hands on an incident, it's necessary to know who's going to do what, and what's been done. If you escalate an incident to Secondary, they'll be grateful for anything you've noted down so they don't waste time repeating any investigation from scratch. Then right at the end of resolving an incident, all the notes you've sent creates a timeline that is useful for writing up a report later on.

New never before seen scenarios are generally caused by something that has recently changed in an otherwise stable system. A change could be a recent deployment, a server being replaced with a new one, a process config change, a new customer who heavily uses your system or many other things.

Postmortem

After the incident is resolved, or while waiting for monitors to go green again, it’s good to do a quick further investigation into what caused the issue. This could include looking at exceptions logged to an exception logging service and identifying anything related. Or checking any metrics history to spot anything that changed such as memory and CPU usage on the related servers.

Documentation

An organized timeline of what happened and what actions were taken should be written up during the next business day for examination. Writing down notes in Slack while investigating and resolving the issue really helps out here.

This write-up can then be discussed to brainstorm ideas for improvements to monitors, metrics, logs or exception reporting that would make incidents of this nature easier to resolve. Ideas for on-call developers to do things differently that will speed up incident resolution, or ideas that will prevent the issue happening in the future. If the procedure for resolving this incident has not been documented, then now is a great time to add it to the playbook while it’s fresh in your mind. Or if it has been documented, update it with any resolution improvements you’ve discovered.

Other considerations

Building a culture of on-call is important.

Without complete buy-in from all teams and individuals, the response time to both major and minor incidents will increase. In most cases, a disruption of service is costing the company real money. The longer it takes to recover from a problem, the harder impact it has on both customer or user sentiment and the bottom line.

It's important that your team is paid for their time whilst on-call, otherwise, there isn't much incentive for doing so for employees.

> My last company is notorious for official-unofficial on-call duties. You must remain within 30 minutes of the office, be near your cell phone and sober, 24x7. Because there is no official policy, there was no rotation, so technically I was always on-call. Their official stance on compensation is that salaried employees can't get overtime, and it's covered by the annual bonus anyway. Yep, unlimited on-call & overtime covered by a ~7% bonus that pays out about 50% of the time. I was once out of the country on a statutory holiday and didn't answer my personal cell phone (roaming would have killed me), which caused me to get reprimanded and lose my bonus for the entire year. After some back and forward I quit soon after.

- Grecy

> I worked for a small company with a >25 person IT team once, and everyone participated in the on-call. When you were on-call, you were on-call for the entire department. I worked IT security, but I was on-call for networking, servers, code, database, and even power outages. The shifts were for two weeks straight and were non-transferrable, so we couldn’t trade off. We needed to be sober, ready to go any time of day or night, and able to be on-site at any of the locations around the city within half an hour. Even if the phone rang at 3am. I quit after seven months. I wasn’t getting paid nearly enough to go through that.

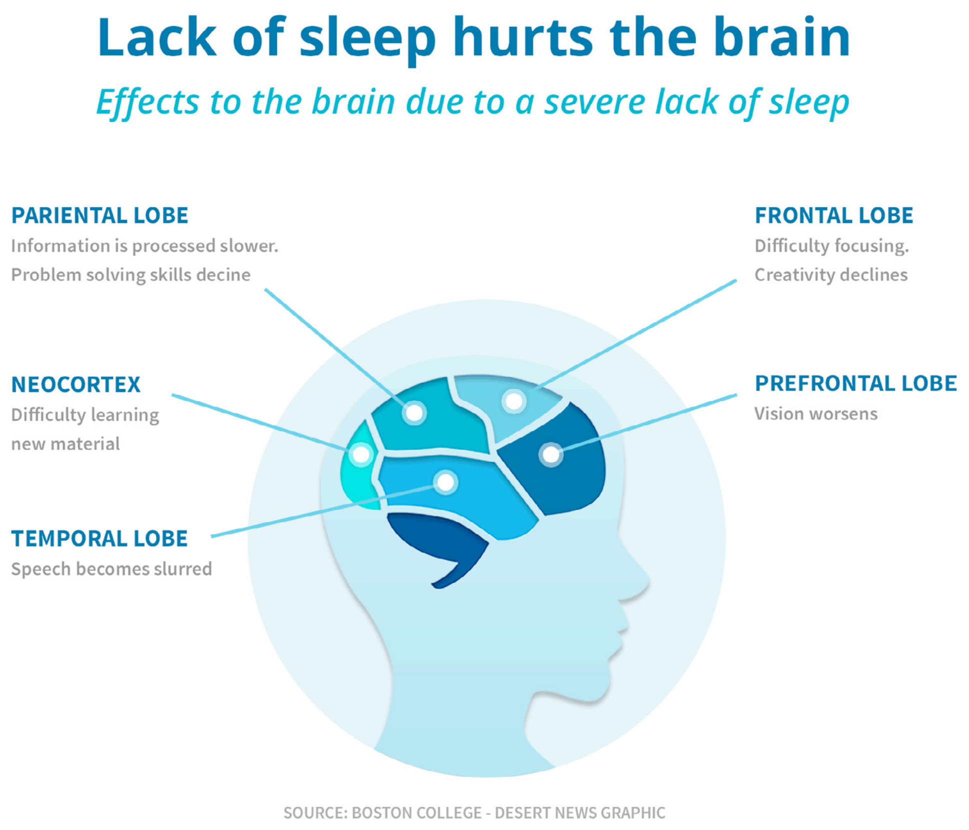
- Freehunter

Senior staff must ensure that being on-call as an employee is seen as a task that is respectfully compensated in your organization and something given to engineers as a sign of trust, rather than a non-incentivised tasks that the team has to do alongside their day to day jobs.

Teams that do not reward staff for on-call hours, seeing it as a task that employees should be doing anyway, may be thinking they are trimming costs or encouraging responsibility, but they stand to lose a lot more when employees find on-call a thankless task, experience burnout, or leave for better deals elsewhere.

Sleep deprivation

Copious amounts of coffee and Red Bull may get you through, but you're going to crash at some point when the caffeine wears off.


Lack of sleep hurts the brain
Effects to the brain due to a severe lack of sleep
PARIENTAL LOBE
Information is processed slower.
Problem solving skills decine
FRONTAL LOBE
Difficulty focusing.
Creativity declines
NEOCORTEX
Difficulty learning
new material
PREFRONTAL LOBE
Vision worsens
TEMPORAL LOBE
Speech becomes slurred
SOURCE: BOSTON COLLEGE - DESERT NEWS GRAPHIC

Often the times you can't sleep are when you know you may have to get up in the middle of the night. This apprehension could come from a lack of preparedness for incidents that may occur. If you are confident that you have things covered when alerted to issues in the middle of the night, you may find it easier to relax and get the hours of sleep you need.

You shouldn't treat being on-call any differently to your normal routine.

Should you need to be awake fixing issues during the night, ensure that you allow some time to recover. It's all too easy to be awake in the early hours and decide to roll on through a normal workday - don't do it! Your organization should have expectations in place to allow you time to come in late, or leave early to get some rest. Sleep deprived staff members who are there in body, but not in mind due to lack of sleep may do more harm than good. Organizations need to reward staff with acceptance of time to recover from on-call incidents or face employee burnout.

Very few people can stay awake for 24 hours and even when they stay awake, their mind really loses problem-solving skills. Drivers who are sleepy simply hit other cars. Accountants who are sleepy, simply make mistakes in their calculations. Also, many programmers when sleepy, write less quality code.

Would you let a severely jet-lagged developer loose on your infrastructure? Probably not, so don't expect staff to be putting in a full day at the office if they've spent their nights fighting fires.

Customer communications

As organizations strive to become more successful and grow user bases and customers exponentially over time, such success can also bring more complexity and consequences when outages occur.

It's a relatively simple fact that the more people relying on your service, the more will be affected when it is unavailable. In November 2015, Slack, the hugely popular team chat application suffered an outage for approximately two and a half hours.

I myself remember it vividly as our team's communication ground to a halt and everyone was asking 'Is Slack down?'. Suddenly a service we relied upon (probably more so than we realized) was unavailable, and it wasn't until it suddenly wasn't there did we find a sense of frustration.

This kind of incident is something that teams try to minimize as much as possible. It's the entire reason for the creation of this guide, but in the case of Slack, almost every tech news site was covering the outage, social media was blowing up with people commenting on it. The influx on their support team was probably just as concerning as the outage itself for their operations teams. Communication to users on what is happening is so important in order to keep customers informed.

Depending on your audience, managing the social media fallout can be tricky. If your customers do not tend to be heavy social media users, then things could be limited publicly. On the other hand, if your user base is heavy social media users, an outage could lead to a lot of publicly visible information needing to be addressed.

Your marketing or support teams may be the ones in your organization that manages the social media accounts, and having to pass through up to date technical information that informs users that the issue is being worked upon, may lead to inaccurate or plain wrong information being given publicly. The ability for those on the front line of things to manage announcements is something all teams should think about.

If you are on-call and responding to a major outage in the middle of the night, other team members may not be available, so alongside fixing the issue, on-call teams need the tools available to them to communicate the status of issues themselves.

Many people seem to want to head straight to social media when they have issues with a service they are relying upon. Usually, because they feel they might be able to skip or avoid any support queue. This is often a problem for software teams, as many times the people who monitor social media channels still need to find answers with internal teams before they can respond, yet customers still want to cram detailed support requests into 140 characters. With posts being public and visible to others, organizations feel obliged to respond in a timely manner.

You can minimize public backlash for outages and issues if you have a plan and team education in place. Teams must think about these situations and put the tools in place before they occur, rather than confusion and inaccessible team members stopping the right information getting to customers quickly.

Affected customers

I had an email from a company recently that apologized profusely for their recent period of downtime. They were so sorry that they had let me down. They were going to try harder in future and hoped that this kind of incident wouldn't happen again.

I had no knowledge of this outage, I wasn't using the tool at the time it occurred, so if they had not told me of their outage, I wouldn't have known otherwise.

Although I can commend this company for being so transparent about the outage, all it did was damage my view of the reliability of their service and confidence in their brand. Suddenly my views of them had been lowered, and completely unnecessarily. In today's data-driven world, there is honestly no need for this to happen.

Monitoring your applications with error and crash reporting software or having customer analytics services attached can have many associated benefits rather than just what they are fundamentally designed for. Many services that can monitor affected users for specific errors and crashes or poor user experiences can allow you to contact only users that have been affected by a bug or underlying issue. Tools that can show users that logged in between specific time periods can also be used to build segments and cohorts of users.

By pulling out these segments of users and then those email addresses handed off to marketing or support, we need only communicate recent outages to those that were impacted, saving us having to notify our entire user base and risk a wider loss of confidence than is necessary.

Final thoughts

Having redundant systems in place is a great way to minimize on-call incidents. If a single instance of a system is enough to handle the load in a reasonable manner, then an alert does not need to be raised unless all instances of that system goes down. This way, if a redundant system goes down that doesn't affect end users, an alert isn't going to wake up on-call developers during the night unnecessarily.

Each morning, developers can check if any redundant systems went down and start them back up during office hours. Further investigation should be done to identify the cause of redundant outages to see if any work can be done to prevent or minimize them occurring in the future.

Fewer incidents waking up developers during the night means they can get better sleep and do more work during the day. Cost, however, needs to be considered when setting up redundant systems.

As new types of incidents occur, it’s good to keep updating a playbook of resolution procedures. This is useful for existing on-call developers to reference while resolving incidents that they have never done, or not done for a while. This is also extremely valuable when onboarding developers to the on-call roster.

Everything should be automated where possible. The resolution of some issues can be automated to avoid on-call incidents occurring. Another thing that can help with on-call procedures are tools in the form of a console application or a page in an admin site that does common investigative work for you. It’s important not to rely on these things too much in the case that issues occur within them. Make sure that training and documentation are in place for on-call developers to perform low-level manual operations if necessary.

A useful chunk of knowledge that on-call developers should learn is where each process runs and which processes communicate with another. If a cluster system needs to be restarted, you may want to shut down other systems that heavily query it so that there is not so much load on it while it’s being restarted.

The cause of an issue may sometimes not be within the system that tripped the monitors, but instead, originate from another system that communicates with it. Knowing how each system communicates can help with investigating the cause of an issue. Just like any on-call knowledge, such communication infrastructure should be documented, ideally in a diagram form that is easy to digest, with notes about the form of communication and structure of payloads.

Any work during work hours that will help minimize on-call incidents should be high priority. Not only does this mean fewer issues for your users, but also means better sleep for on-call developers which can result in better work during the day or avoid needing to take a bit of time off to recover.

All code that developers write should be well reviewed by at least one other developer. Code changes should be as minimal as possible to minimize the risk of breaking anything, but sometimes a colossal code change is required when changing/migrating the way that systems work.

Such change could include changing the way data is stored, or the way messages are communicated between multiple processes.

When these kinds of changes occur, it is recommended to schedule the deployment of these changes while the lead developer of those changes is the Primary, or change the on-call schedule so that the lead developer is Primary for a while after the change is deployed. The lead developer will know all the intricacies of the changes and is better equipped to resolve any early issues.

If the on-call developer does not know the extent of the code changes well and is woken up to resolve a crisis, the lead developer could feel guilt of causing someone else to be woken for the issue, and the on-call developer could feel angry about the situation. This can cause tension in the on-call culture which you want to avoid wherever possible.

Survival

We hope this guide has been helpful, or at least given you a few ideas for how you can survive on-call activities in your own organization.

The key is to stay organized, be fully set up and prepared before the time comes to fix issues and document processes as much as possible.

Reliable software for end users is something all teams strive for, but it’s no coincidence that high availability services are ‘lucky’ to stay online and not experience severe outages.

It comes down to your team, and the processes you have in place.

So be prepared if you want to survive any major incidents without end users noticing.

Take a free trial

Try Raygun free for 14 days

Book a demo

Take a few minutes to see the magic!

Learn with Raygun's eBook library

Check out these free ebooks published by Raygun

RAYGUN

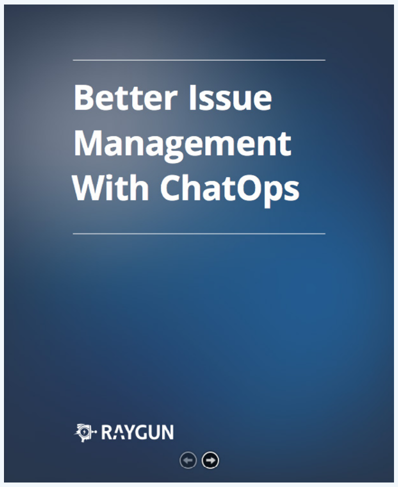
Software intelligence

The secret to building flawless web and mobile apps

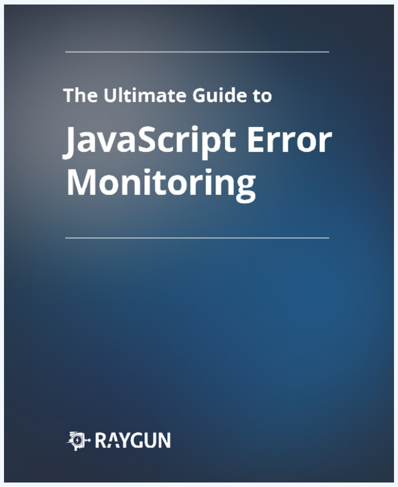
By Nick Harley and Freyja Spaven

Published by Raygun.com - www.raygun.com. © 2017 Raygun Limited - First Edition

All rights reserved. No portion of this book may be reproduced in any form without permission from the publisher, except as permitted by copyright law. For permissions contact: support@raygun.com


Better Issue Management With ChatOps

RAYGUN


The Ultimate Guide to

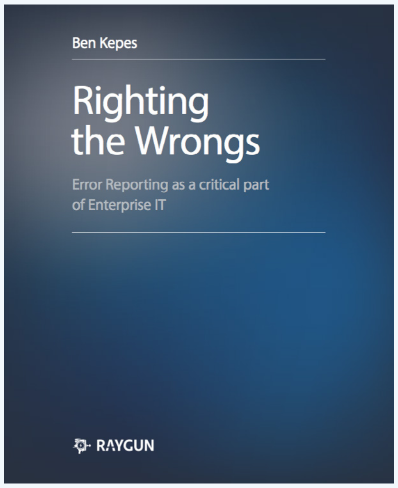
JavaScript Error Monitoring

RAYGUN

Error Monitoring Drives Innovation

Automating software quality liberates teams. Development is proactive rather than reactive.

RAYGUN.IO


Ben Kepes

Righting the Wrongs

Error Reporting as a critical part of Enterprise IT

RAYGUN