Ask HN: What is your company's on-call rotation like?

82 points by bobzimuta on Jan 13, 2016 | hide | past | web | favorite | 63 comments

Also looking for readings on best practices.

Curious about your company's on-call for details about:

* expected duties (only answer on-call, do other work, etc)

* how deep does your on-call dive into code to fix, vs triaging, patching, and creating follow up work to fix permanently?

* priority order of work (on-call tickts vs regular work vs existing on-call tickets vs special queue of simple tasks)

* what happens when the current on-call can't complete in time?

* how do you manage for other teams' risk? (ie their api goes down, you can't satisfy your customers)

* any other tidbits

jeletonskelly on Jan 13, 2016 [-]

To software developers in this thread who are on-call; I'd like to share some thoughts with you. I've worked at places that do have on-call rotations and others that have none. I will no longer work at a company that requires me to be on-call.

Why? It says a lot when a company doesn't put the effort into various forms of testing and QA to ensure that production software does not have critical issues that warrant at 2am call. Unit, functional, integration, load, and simulation tests should be written for every single piece of critical infrastructure. You should be hammering these things in staging environments with 10-100x of your normal peak load.Use something like Gore to replay live traffic against a version in staging or QA environments. Yes, that takes work, but to me it's better upfront than to wake me up in the middle of the night or to know that when I go home I have to have my phone around me at all times. The business should care about these things too; it's their product and they should care enough about you to make sure good processes are in place to ensure quality production software.

That said, when I was at non-on-call companies there are definitely times when something does happen that warrants immediate attention. Generally someone in operations would get the first call, they check logs, diagnose the issue, and call a developer familiar with the app that's causing the issues if that's the case. I don't mind waking up because I know it has to be something serious that slipped past our processes.

snewman on Jan 13, 2016 [-]

Minimizing 2am issues, and maintaining an on-call rotation, aren't contradictory. There's no substitute for having someone on call; but you can minimize the number of times they actually get called. This topic is near and dear to my heart.

You talk about testing - that's one side of the coin; the other side is careful alert tuning, (a) to minimize false positives at 2 AM, (b) to catch incipient issues while it's still business hours. (It's useful to think of alerts as just another phase of QA - the one that occurs after you hit production. The sooner you notice a problem, the less damage it causes, to both your customers and your sleep schedule.)

At my workplace, we run a fairly complex system, but we've been able to keep nighttime pager incidents down to I think less than once per quarter, including false alarms. I can't remember the last one. The QA effort isn't overwhelming, either. See http://blog.scalyr.com/2014/08/99-99-uptime-9-5-schedule/ if you're interested.

herval on Jan 13, 2016 [-]

honest question: have you ever managed to work on a company that's either successful or growing very fast and can guarantee "various forms of testing and QA to ensure that production software does not have critical issues that warrant at 2am call"?

I can imagine that being possible in consultancies or small scale/load products, but honestly never seen it on anything larger than that - including environments with a mind-blowing number of layers of QA and tests...

hirsin on Jan 13, 2016 [-]

Absolutely - not everything is a service (yet). It's not sexy, but if you only ship your bits every month or every two weeks then you shouldn't have 2am calls. Everything can wait til tomorrow (unless tomorrow you're shipping, in which case I have seen people still awake at 2am, but I consider that a failure).

coderjames on Jan 14, 2016 [-]

I work at a successful avionics company writing software that goes onto airplanes. We do not have critical issues that warrant 2am calls.

I'm in the camp of "will not work at companies requiring on-call or pager-duty."

bcbrown on Jan 13, 2016 [-]

I've worked at a late-stage startup that serves billions of requests per day at peak load, and there was no on-call for developers. I believe the ops team did have an on-call, but that was more at the infrastructure level, as everything was self-hosted at colos.

This was done by having a simple and resilient serving architecture. Every server was stateless. In addition, all the complicated logic was pre-computed into immutable lookup tables, offline. So if that task fails, it doesn't cause downtime, and can wait until the next workday.

We had a robust QA process, but it was far from stultifying.

nrr on Jan 13, 2016 [-]

… whereas I'd conversely argue that volumes are also spoken when a company does not require its developers to be on-call. Often, those volumes are written with negative undertones and a narrative that speaks to developers not owning or understanding their software stack. Worse, in an exercise reminiscent of companies offering unlimited PTO, "not on-call," when read between the lines, could really mean "always on-call." I currently work with teams that operate this way, and the amount of burnout is staggering.

As someone who has leapt between development and operations time and time again, I can anecdotally say that the systems where developers felt like they had ownership stake (including a desire to know "what went wrong," often only accurately conveyed via an OC shift) were better designed, better maintained, had longer MTBF and had dramatically shorter MTBR.

That doesn't mean that the development team that engineered the system in the first place should always be the first line of defense; rather, it means that there should still be some sort of escalation tier in case the operations team to which the service has been handed off needs some further assistance. I can recount stories for days wherein I was frustrated as someone in operations that I couldn't reach someone from a development team in the middle of the night for a critical service--where VP- and SVP-levels were barking at me to fix it--only to hijack repository permissions and write patches myself for their software. It should go without saying, naturally, that reprimands for such heroic acts (heaven forbid I actually fix the damn thing) were definitely forthcoming the next morning.

That whole mess is wrong and anti-collaborative. Without the guardrails of a well-defined on-call shift, this is what no-on-call organizations devolve to.

This all comes with a caveat though: Again, anecdotally, I've found that tossing the development team to the fire for the first six or so months of a service's production lifetime, before even allowing them to ask for operations handoff, pays dividends in terms of meeting (or exceeding) business goals, keeping operations' frustrations low, and delivering a quality service that other teams can rely on. This goes in concert with automated unit/functional/whatever testing, knocking parts over in production in a controlled manner, continuous reviews of documentation, commit-gating code reviews, monitoring that makes sense, and so on and so forth.

(As an aside, folks on the operations side have enough to worry about in terms of integrating all of the infrastructure to make everything appear to Just Work™. Adding the burden of having to reverse engineer tossed-over-the-wall "It's an ops problem now" garbage is akin to the trend in the initial days of the devops movement for operations folks to toss systems automation over the wall to developers. It's disrespectful. Work together. Show trust and solidarity by carrying a pager alongside the ops guys to say, "Yes, I'm right here with you in case you need me.")

Tom Limoncelli somewhat recently put out "The Practice of Cloud System Administration"[0], and I implore that you give it a cover-to-cover read. Even folks who align with the development side of the house will get some benefit from it.

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

0: http://the-cloud-book.com/

bcbrown on Jan 13, 2016 [-]

> Worse, in an exercise reminiscent of companies offering unlimited PTO, "not on-call," when read between the lines, could really mean "always on-call." I currently work with teams that operate this way, and the amount of burnout is staggering.

I've found the best way to screen for those sorts of problems is to ask everyone in the interview about their personal habits, instead of about process. Don't ask about vacation policy; ask when's the last time your interviewer went on vacation. Don't ask about the on-call rotation, ask about what issues came up outside work hours in the last month.

jeletonskelly on Jan 13, 2016 [-]

I don't disagree with anything you said here. Organizations should try to hire developers that care about what they build and understand that, as an implied part of the nature of the job, your phone may ring at some unknown time because of an issue. In operations it is certainly implied that you will likely be the first one notified of production issues and you will most likely be the first to know which developer(s) need to be contacted. I am certainly not advocating for a developer to throw their hands up and say "not my problem, I'm not on call" or "it's an ops issue." Those would be very junior or childish reactions and certainly not the qualities of a senior developer.

What my post was getting at more of an observation I have made during interviews where teams had on-call rotations for developers. When I ask "how often does your phone ring during your time on-call" I would get answers that hinted at a deeper issue. Maybe it wasn't always that way, maybe the on-call rotation started with the purest of intentions like you have highlighted, but somewhere along the line management saw that as an opportunity to take shortcuts with testing and quality. So, am I saying that if a company were to make me an excellent offer, but required on-call rotations are they automatically ruled out? No. I am, however, going to be asking some very pointed questions and probing that arrangement quite a bit.

existencebox on Jan 13, 2016 [-]

Your points are valid. Stronger than that, I stand more to your view than the parents. Although I certainly understand the desire of the parent, I've been on both the Ops and Dev side of that fence, and on call REALLY SUCKS (I almost feel dirty in this post agreeing with a statement of "I should be on call" but it is what it is :)) but it sucks worse if as you say it's entirely on the backs of the ops who are also handling sysadmin style work, and are kept more isolated from dev as is typical in "throw it over the wall" shops. (In writing this I realize I want to clarify that I have other thoughts entirely about having ops specialists instead of suggesting a merge to totally unified devops, but that's an entirely other discussion). Also as you say, ownership/responsibility/etc is also an ancillary benefit.

What I actually wanted to say from all this however, is that despite the truth of your argument, I'd take a third angle. No on call is bad, but similarly, _unpaid_ OT is bad. The common rhetoric from this soapbox is that if employers are accountable for this time, it'll incentivise them to have processes and hold values that don't abuse an expensive resource. If something is on fire; the on call employee still _will be there to handle it_, but the fact that it's become a pro-bono assumed part of many of our jobs in the tech sector is the part I take issue with more than doing the task itself.

vkjv on Jan 13, 2016 [-]

This. tl;dr: No on-call schedule sometimes means always on-call.

brndn on Jan 15, 2016 [-]

> Use something like Gore to replay live traffic against a version in staging or QA

environments.

What is Gore?

biot on Jan 13, 2016 [-]

For those doing on-call and required to be responsive to alerts at all times during your shift, one thing to think about is your pay for this burden. Some cardiologists get over $3K/day to just be on-call (whether or not anything comes up) which means keeping within range of the hospital, being sober, not being more than a few minutes away from their phone, and so on. While generally nobody's life is on the line should a server go down, if you're expected to go beyond "if I hear my phone alert and I'm sober and am near my laptop" level of effort, ensure you are compensated appropriately.

grecy on Jan 13, 2016 [-]

>*Ensure you are compensated appropriately.*

I agree 100%

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.

freehunter on Jan 13, 2016 [-]

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.

pbang on Jan 13, 2016 [-]

3k a day? do you have a source for that?

milkcircle on Jan 13, 2016 [-]

Appears to be less than 3k/day on average. Source: http://www.acep.org/Clinical---Practice-Management/Survey--M...

biot on Jan 13, 2016 [-]

Thanks for looking this up. I did some research many years ago when I was in an unpleasant on-call situation (unreliable third-party dependency) and negotiated pay for it as a result. I recall seeing a $3500 daily rate figure at the time for a cardio surgeon, but that may have been an outlier.

While the numbers may be off, one point I didn't make in my initial post but should have is that anybody doing on-call is providing a valuable service. It's no different than paying a babysitter to watch TV while your kids sleep. Both babysitters and devops/IT/whatever are compensated for their ability to act in case the unexpected happens. If someone's not getting paid for that, then it must not be valuable to the company to have that coverage.

angelbob on Jan 13, 2016 [-]

Yeah, the article linked there is nicely specific. From the article (and these are median):

Neurological surgeons had the highest median daily rate for providing on-call coverage, about $2,000 a day. Near the top of the pay scale were neurologists ($1,500), cardiovascular surgeons ($1,600), internists ($1,050), and anesthesiologists ($800).

Among the specialists earning lower median daily rates for on-call compensation were: psychiatry ($500), general surgery ($500) gastroenterology ($500), ophthalmology ($300), and family medicine without obstetrics ($300), according to the MGMA survey data.

darkr on Jan 13, 2016 [-]

They probably get a lot of repeat work when they tell the patient how much it's costing them.

freehunter on Jan 13, 2016 [-]

If you're calling an emergency cardiologist after-hours, you're probably not immediately worried about how much it costs.

gnur on Jan 13, 2016 [-]

Current rotation for me is: 1 out of 3 weeks from 7 am till 12 midnight as a systems administrator. Our SLA to customers is non-existent, so on-call duties are best-effort (if you are in the shower and it takes 30 minutes to respond, no real problem). Furthermore, this also means that it is up to the person on call to determine what needs to be done.

Outside of the regular work hours you are only expected to work when an alert / call comes in. If at all possible you should fix it, if it would take > 1 hour you start looking at temporary workarounds that will hold until you are back at the office. During regular work hours on-call tickets always get priority. When problems arise with external dependencies we open tickets in their support system and wait for their response. Most important in our case is to keep the support desk up to date so customers don't open too many tickets and to let people know we are working on it.

Depth of on call work depends. It usually consists of rebooting / restarting services and checking if they come up as expected and trying to find the root cause of the problems. If the root cause can be easily identified and fixed, fix it immediately. Otherwise, open a ticket to fix it asap.

dnnrly on Jan 13, 2016 [-]

I started being on-call a few months ago. For us it works like this:

- 1 week in 4 Monday 00:00 to Sunday 23:59, with a fixed pay adjustment (same fixed amount added to our salary every month, not a % increment).

- If you get a call, you're on the clock and get paid regular over-time rate. Theoretically this only applies if you get called and are doing something for more than 30 minutes though.

- I know most of the on-call managers personally so don't mind picking up the phone for a quick opinion for free, if it's at a reasonable time.

- Online within an hour once we've picked up the phone.

- Senior on-call manager gets woken up by Ops first and then will decide which parts of the system are affected. After going through known issues and work arounds will then work out which part of the system needs to be looked at and makes the call to the on-call developer that is appropriate (ie. which team).

- Duty is primarily to diagnose and provide knowledge to the on-call manager to make a decision so service can come back up again.

- Don't do code fixes (J2EE on my team, mix of Java and C++ on others) - we have far too much to go wrong for hacking around like that, but may do config changes and some scripting. The line can get a bit blurred.

- In our team we should raise a ticket/Jira for every time we get a call so that it's visible to everyone what happened and see where our pain points are.

- During office hours, Production issues are distributed in the team in the same way as bugs, according to domain knowledge and individual capacity.

- We tend not to get called out except for when we do installation in to Production. We have a fairly heavy weight process for releases - which we are working on.

- I am at least 4 levels of communication away from any customers, including 1st and 2nd line teams.

Edit: formatting

rodgerd on Jan 13, 2016 [-]

On-call is paid at a weekly rate for being the on-call person. There's an additional bump if you're actually called out, pro-rata on an hourly basis to your salary.

You are required to respond to calls within the SLA, and work on problems until they're fixed (definition of fixed being service restoration - root cause canalysis for complex problems and hence fixes will happen during normal hours) or until you've been working long enough you need to be spelled.

If you're on-call you're still expected to be doing your day job, but production takes priority, and projects are expected to accomodate that.

On-call rotation is generally rotated around teams, so one week in 4 - 8 depending on team size.

samuellb on Jan 13, 2016 [-]

We use a very similar system where I work, except that we have one "extra" person on-call in case the first one gets sick, has a hard disk failure or likewise, or just needs extra help for some reason.

Both people are paid a fixed amount per week + a fixed hourly rate if called out. Weeks with holidays give you double pay. Simple but it seems to work well.

notmeknees on Jan 13, 2016 [-]

Funny that I decided to finally create an account to answer this.

Our team is heavily silo'ed, and each developer is responsible for a specific set of code. Since we have gone through a bunch of reductions and reorganizations, this ends up meaning that often one person is responsible. Our customer is a 24/7/365 operational unit, and it is our only customer.

So, the end result is any given developer is on call all the time, with no pay compensation for that at all. Not only that, but you should expect to be called about not only a defect, but in cases where the customer is trying to use our software in a way it was never designed to be used. So, get a call at 2 AM on the weekend and design and implement a new feature, release to production to debug and test.

I really dislike the way we operate, and recently the entire QA team was let go so I have a feeling things are only going to get worse.

FLUX-YOU on Jan 13, 2016 [-]

Get out ASAP. If that 2am feature causes issues because of an honest mistake from coding at 2am, they will blame you, not their incredibly horrid practices. It is doubly worse they do not compensate you.

Amezarak on Jan 13, 2016 [-]

Yeah, I have to say I envy a lot of the other people in this thread.

I had a very similar deal. I was on call 24/7/365 for two years. Generally something came up at least twice a week. Usually in the evening or on holidays. Often a production issue or an "urgent" ad-hoc query request (they always needed some crazy slicing of data RIGHT AWAY because of customer X or Y or...) but we did, once or twice have a "WE NEED THIS CRITICAL FEATURE WE NEVER EVEN HINTED AT NOW NOW NOW" deal.

No extra pay and the job itself paid only 45k.

I recommend you get out. It just wears too much on you to effectively always be at work.

Now I don't get to do basically any development and the work is boring, but at least I get paid a lot more and I'm not on call. Sigh. Guess you can't have everything.

acveilleux on Jan 13, 2016 [-]

Holy crap, I thought places like that only existed as straw mens in articles about bad practices. It'd unlikely that any other place would be worse, you have little to lose by changing jobs.

A company that behaves like that is actively trying to find ways to cut your job too and they will just as soon as they think they can get away with it... Or maybe they'll cut someone else and have you do their job too, for efficiency.

angelbob on Jan 13, 2016 [-]

The really good and really bad places tend to exist and not get talked about. That is, a surprising number of "strawman" workplaces, good and bad, tend to actually exist.

ChrisArgyle on Jan 13, 2016 [-]

So many red flags, time to find a new job. The big one I recognize from a previous job is the 2AM feature implementation. In our shop it generally went like this:

* Sales, having trouble closing a sale, promises a non-existent feature

* Customer tries to use imaginary feature, calls sales to complain

* Developers have to drop what they're doing and implement imaginary feature full steam

edit: formatting

jeletonskelly on Jan 13, 2016 [-]

> So, get a call at 2 AM on the weekend and design and implement a new feature, release to production to debug and test.

Oh no... just no. Production is not where debugging and testing should ever happen, obviously. If that's what your company expects I'm not surprised QA was let go when your customers are doing it for you. This makes me sad.

poooogles on Jan 13, 2016 [-]

>Oh no... just no. Production is not where debugging and testing should ever happen.

Debugging sure. Testing in production however is incredibly valuable.

RankingMember on Jan 13, 2016 [-]

Are you kidding? Testing in a copy of production (integration, staging), sure. You should never be testing anything in production if you value your clients and reputation (and sanity).

toomuchtodo on Jan 13, 2016 [-]

That's what a canary is for, where you only send it a percentage of traffic.

I guarantee you that you'll never be able to accurately reproduce production traffic in staging or development.

scardine on Jan 13, 2016 [-]

In Brazil the labor relations are heavily regulated. Each hour you are on-call you make 1/3 of a regular hour of work, and while you answer a call and work on weekends or at night you are paid twice the regular hourly wage.

You are expected to answer the phone promptly, be sober and do anything to remedy the problem including reaching for other people that can fix it (does not have to be a permanent solution).

throwawayoncall on Jan 13, 2016 [-]

Work at a well funded Series B startup

On-call is unpaid and no additional bump if called out or public holidays. No additional vacation in lieu either :(

- Expected duties are to fix the issue and bring production systems back up, all engineers have context on production systems and can escalate if any additional context is needed. We have an established workflow of steps to take to determine root cause. We can also directly ping all engineers to provide additional help should there be a critical issue with a system you are unfamiliar with.

- If you are on-call, the on-call duties come first and foremost above everything else. Expected response time is 10-15 minutes from issue raised at all times.

- Mitigating risk is via communication and getting enough context to know the bad outcomes and how to mitigate any issues.

- On-call rotation is very arbitrary and undefined (we have five engineers who can do on-call duty), we try to share the duties equally but some get assigned way more on-call than others. (I was on-call most weekends last couple of months and all throughout the Xmas period for example)

biot on Jan 13, 2016 [-]

As long as the 10-15 minute response time is within work hours, all good. For a well funded company to require that level of responsiveness outside of work hours for no additional pay is simply them taking advantage of you.

tluyben2 on Jan 13, 2016 [-]

I would not consider working like that; 10-15 min response time? If on-call is only during office time then there is something about it although 10-15 minutes is mental. But outside office hours you need to get paid and paid a lot. Why would you do that unpaid? During the xmas period? You are getting screwed here.

toomuchtodo on Jan 13, 2016 [-]

> 10-15 min response time?

Unfortunately, yes. I'm (devops/sysadmin) in the same boat. <10 minute response time expected when on-call, regardless of time of day.

Below market pay, but I work from home. It works for now.

cookiecaper on Jan 13, 2016 [-]

Everyone takes a turn being on-call in one week cycles. We have an on-call day and an on-call night. Any infrastructure alarms go to the on-call person and he/she is expected to address the issue, which can include escalating it to the person who knows the most about it if the severity warrants that, or deciding it's not important and silencing for some time period.

On-call fixes until the crisis is mitigated. If something can be addressed the next day during normal hours, it is.

On-call tickets are the most important. In our company, "on-call" just means you get the infrastructure alarms for that week and time period, and gotta deal with them when you get them. There is no alteration in pay for on-call weeks.

If the current on-call can't respond within 10 minutes (or forgets to ack in PagerDuty), the next person on the escalation schedule will be notified and expected to deal with the issue.

Other teams' risk is mitigated because if it's breaking production, we just call them and make them fix it. The whole company is "always on-call" in some sense, because if your thing is breaking production, you're going to get a call and get asked to fix the problem.

We feel like the tooling on this, even with PagerDuty whose whole job is managing on-calls, is subpar and are constantly talking about creating an in-house replacement.

lucaspiller on Jan 13, 2016 [-]

A few years ago I worked at a company that paid £30/day for being on-call and

£30/hour (or part thereof) while investigating / fixing incidents. We were usually on call for a week at a time with a rotation of around 6 developers.

We got SMS or phone call (we had a non-technical ops team that worked early/late monitoring systems and tried to do preliminary fixes) and were expected to be available and investigate 24/7. If you didn't acknowledge the SMS alert within 30 minutes everyone else on the team got it, so it was in your best interest to do so to avoid waking up everyone else :)

Some of the services we had were pretty critical, so the fix in that case was 'as long as it takes to get it working', and then putting in a permanent fix during business hours. For lower priority services, we'd just leave it broken and fix it during business hours. If you were up fixing something in the night you weren't expected to be in at 8am the next day.

In terms of escalation if there was a major problem that couldn't be handled we'd usually get in touch with a few other people on the team to get their input. This hardly ever happened though, the only time I can remember is when the ventilation system failed in a data centre which was !FUN!

After that I worked at a startup that didn't pay for any overtime or on call, which is one of the reasons why I left. Since then I've been contracting, so haven't had to worry about on call but I'd be happy to do it again for the right price.

fweespeech on Jan 13, 2016 [-]

1) We don't keep a formal on call but 3 of us are tied to an automated alert system and whoever has a chance to take care of it, does. We are all full stack devs so generally we can fix it at the time. If its complicated, we can help it hobble and fix it later.

2) We get 3-4 alerts a year that have to be handled before the next business day.

3) As such, there is no real work priority, triage, etc. You resolve it immediately. There is no other priority. [Any on-call event == lost money]

> * how do you manage for other teams' risk? (ie their api goes down, you can't satisfy your customers)

Asynchronous processing. I buffer until their API is back up. I do this for literally dozens of companies from small manufacturers to Amazon.

There really isn't any other good way to handle it and if you need to do it otherwise that really is a fundamental architectural problem that should have been resolved at the design phase.

acveilleux on Jan 13, 2016 [-]

Has your buffering got any back-pressure at all? Or do you buffer until the disk is full?

fweespeech on Jan 13, 2016 [-]

It would need to be down for ~96 hours under our heaviest 4 day period in our history for it to fill the disk. All of them would have to also go down simultaneously.

There is backpressure at 85% disk fill (this also is an on-call trigger event since it shouldn't ever happen in practice). Suffice to say, this never happens in the real world without hardware failures.

Disks are cheaper than fatiguing engineers with on-call events and this is accomplished by basically having a nearly empty 3 VM cluster with 512 GB SSD (Raid 1 pair, so 2 disks) each. The load on the rest of the VM host is negligible given its being asynchronously processed from this cluster already, so its just really filling the extra disks on the VM host we dedicated to this purpose.

Just realize we build to 4 9s.

YTD failures 3662/75631129 = 0.00004841921

This doesn't trigger an on-call event because it recovered automatically but the cluster does fail every so often for ~3k events. This is for a single API call.

michaelt on Jan 13, 2016 [-]

Overview: On call one week in six; pay £300/week for being available (extra for national holidays), and time-and-a-half for any time spent dealing with calls, rounded up to the next 15 minutes.

Expected duties: Respond to major problems outside of office hours, in response to a phone call. Be online fixing the problem within 15 minutes of being called. Typical workload 2 hours per week. Be somewhere with a reliable internet connection, and where you can stay even if an issue takes a few hours to resolve (no mobile internet from campsites). Be sober.

Fixes: Because of the need for code review and second testing, emergency code releases never happen out of hours. Instead, all code deployments include a backout plan, which reverts to a known-good version of the code (not a bug-free version, there's no such thing, but a version that hasn't caused major problems when run for several weeks). Data in the database may need to be manually fixed.

If you're patching an issue by deleting some bad data, and in your judgement deleting the data might delete evidence needed to identify the root cause of the problem, try to identify the root cause if time allows.

Priority: On call is only called for major problems - either their software has raised an alert saying they should be called, or there is a problem costing thousands of pounds a minute such as "website down", "customers cannot place orders" or "customers will not receive orders".

There isn't a formal SLA, but as it costs thousands of pounds a minute the expectation is to fix as fast as you safely can, without mistakes that make the problem worse.

Escalation: To their team leader, or to some other senior software engineer on their team (our team leaders have all been senior software engineers on the team they lead)

APIs down: In a rather old-fashioned design move, all critical components are maintained in house. Contact the out-of-hours on call for whatever team maintains the broken system. If it's a non-critical component.

ogsharkman on Jan 13, 2016 [-]

Programmer at a Hospital -- Only have to do stuff if I get paged (rotation is 1 every 4 weeks for 7 days, 5PM-8AM). If the issue(s) affects a clinical caregiver then the issue needs to be resolved asap no matter what, priority is only regarding the issue you were paged for, if the on-call person can't resolve they will usually page additional people or escalate to a manager who can find the right people. If another team's issues are causing us problems we just need to inform the customer about what's going on and then keep them updated.

dripton on Jan 14, 2016 [-]

I work at a big famous company that strongly believes in devops.

I'm on call about one week in 6. (About 12 people in the rotation, with two on-call at a time.)

When on call, the ticket queue is supposed to have priority, but managers have an unfortunate tendency to say "that guy's on call so he's not doing anything so let's have him do it." So what ends up happening is that high-severity tickets get priority, then whatever random stuff the manager wants done, then low-severity tickets. So low-severity tickets tend to remain open for way too long because the oncalls aren't making them a priority.

For high-severity tickets, we escalate if we can't fix things quickly. First bring in the other oncall, then bring in the manager, then bring in anyone else on the team who can help.

When a dependency breaks, we can ticket its owners to fix it. If it gives us a high-severity ticket, then they get a high-severity ticket too. Ultimately there's a tradeoff between using fewer dependencies in the name of controlling your own fate, versus using more dependencies to avoid reinventing the wheel. Each team has to find that balance for itself.

I don't enjoy being on call, but I think it's the best solution we have. Stuff breaks and someone has to fix it. If you don't pick someone, then everyone has to fix it. If the same people who build something also have to fix it, they have strong incentives to do things correctly.

elliottcarlson on Jan 13, 2016 [-]

Previous role:

* On-call is a week long (Tuesday to Tuesday 10am) rotation

* Primary duty on a page is to triage and be a point person during the incident

* Proper procedures during day-to-day coding should make the on-call shift be uneventful 99.9% of the time

* Managers are escalation point, then Directors, then VP of Eng

* Revert to previous state and call it a night is an acceptable fix - the primary issue can be resolved the next day

* Blameless post-mortems

spotman on Jan 13, 2016 [-]

Hi,

My company does 24/7 devops for some clientele. We are a team of 6 for reference. We have been doing this for many years since before "devops" was the term for it. Some of the platforms we have also built for these clients, and some we simply manage, or we have only built+manage the automation.

In terms of answering your question:

- expected duties. In terms of what we are selling the client there is a long specific list of what exactly we can provide the client. In practice however the list is exhausting, and intended 1/2 for CYA, and we will fix whatever issues arise. Some of that is seeing things that need work/adjustment/tuning/optimizing before problems arise, and then as you can imagine the whole being-on-call thing means that you must be an expert at resolving issues that no one saw coming. In terms of SLA stuff, we are supposed to be on-scene (digitally) in 15 minutes, but we are never that slow.

- how deep. Think of it in phases. When something needs fixing, you must quantify the urgency. If there is an urgent fix needed this is ground zero. You do whatever you can, if that involves waking people up, or checking out code bases that are new to you and patching some developers code in the middle of the night, its all on the table. In ground zero mode, you do whatever it takes. Once the bandaid/fix/solution is in place to "stop the bleeding" as I like to say, then there is follow ups, that may include working with the client to have them implement the elegant permanent fix, or if they are in over their head, sometimes that lands on us too. But it serves no-one to have a bandaid that is going to just get ripped off every night. So we see the problem from A-Z, even if our offramp is on B, or M. If it's not urgent, we will just wait for tomorrow to discuss with the client, it falls out of the scope of on-call.

- priority. Well, on-call work is billed and agreed different to ticket work since we are primarily a consulting company. So these are different buckets. But we also have our own products that we own 100%, and the priority is quite easy. If something is broken/down/about-to-break, it trumps everything else. Regular tickets are great, but they are meaningless if the platform for which your developing them against can't stay functional. On-call work rarely has a "queue" or even really a ticket system.

- there is no "complete in time" it's either done or you failed. I say this, having failed too, and it sucks, but it is what it is. If something breaks, and you can't fix it, you don't just go home. But.. sometimes you do, but you walk away from the scene with your tail in-between your legs, no one plans for that.

- managing other teams risk. Communication. Putting energy in ahead of time, and bringing things up before they break is huge. Also if you say "Hey you should turn left, there is a cliff!", and the client is insistent on turning right, this can do two things. A - they know and hopefully its recorded in an email or in a meeting that you wanted to turn left. B - if your absolutely certain they are going to run over a cliff, but your still on the line / have to support the darn thing anyway, you can quietly put a bunch of pillows at the bottom of the ravine, and prepare for the inevitable. When the car goes over the cliff, and everything almost grinds to a halt, and you manage to have been correct about it, but also managed to bail them out, you score a lot of gratitude from the client, and ongoing future relations.

This is all from the point of "consulting" mostly, but I have done this same type of work for many large companies directly on their payroll too, it all applies, and the bigger the company the more it's like consulting in some way also, because bigger companies are more compartmentalized. I have also done this in many small startups where your all just one small team.

Holidays and vacations are important, but they will never truly be the same after years of this. We are pretty good at it now, and we really do try to keep everyone up to speed with where "the bodies are buried" with all of our clients infrastructure. That is the hardest part. Everyone can look at a perfectly groomed wiki or set of 100% refined chef/puppet cookbooks/modules, but the world isn't perfect. So the hard part is learning how to take the punches with elegance, and people need a break. It really does take at least 3-4 people to have a not-insane 24/7 schedule.

We generally plan about 1-3 weeks ahead depending on the time of year and put it into our scheduling system, which is also the system that does our paging. We rotate weekends, and work it out amongst ourselves. Some people have things like date nights that we know about and try hard to not screw up.

Don't build your pager system yourself, you have enough stuff to do. I won't plug them because I don't want this to sound like an advert, but do yourself a favor and pay the money to a company that specializes in bubbling up alerts to notifications that can do phone/sms/email/ios/android/push/etc. These have really helped us manage the insanity & schedule.

If there is any advice, simply don't be frightened to say when you don't know something, even if your tasked with fixing it. There is little room for ego when your the one that ultimately has to be responsible. Being on call means you can't pass the buck, and the sooner you know what you don't know, the sooner you're able to learn it!

Edit: Typos

fractalcat on Jan 13, 2016 [-]

Last time I was in a role which involved on-call rotation:

> expected duties (only answer on-call, do other work, etc)

Only answer pages. My employer did shifts a bit differently from most companies - only six hours per shift, no fixed schedule (decided a week in advance) and only outside of work hours (pages during work hours were handled by whichever sysadmins were on duty), which worked quite well to avoid burning out sysadmins. On-call shifts were paid, and shortages of volunteers were rare.

I'd expect to spend maybe fifteen minutes per shift fixing things, on average (this is in managed hosting, so a page could be any of our customers' services).

> how deep does your on-call dive into code to fix, vs triaging, patching, and creating follow up work to fix permanently?

In my case (sysadmin for a managed hosting company) the code involved was often not under our control; the standard practice was to escalate to the customer if the cause of the outage was a bug in the application. The usual process when suspecting a bug was to track it down if possible (the codebases were usually unfamiliar, so this wasn't always the case), work around it as best we could (e.g., temporarily disable a buggy search indexer which was leaking memory, et cetera), and then get in touch with the customer (by email if the workaround was expected to last until work hours, by phone if not). Occasionally I'd fix the bug in-place and send the customer the patch, but this was technically outside scope.

> priority order of work (on-call tickts vs regular work vs existing on-call tickets vs special queue of simple tasks)

The only priorities were resolving the pages at hand and arranging followup where needed (usually raising a ticket to be followed up during work hours).

> what happens when the current on-call can't complete in time?

Generally the on-call sysadmin would resolve whichever pages they had acknowledged; in the event of an extended outage the acking sysadmin was expected to brief and hand over to the person on the next shift.

> how do you manage for other teams' risk? (ie their api goes down, you can't satisfy your customers)

In practice, we could escalate to anyone in the company for a serious outage we were unable to handle ourselves. This was pretty rare, as a small ops-heavy company, but everyone had access to everyone else's cell phone number and an outage-inducing bug was usually sufficient cause to wake someone up if it couldn't be worked around.

foxyv on Jan 14, 2016 [-]

Developer on-call every 8 weeks or so. 90% of the job is checking logs and saying "Yup it's working" or "Nope $VENDOR is down again." Get paged on average 3 times a cycle. One time I actually got to fix some code. That was awesome. Just feels really weird that no one in the tier 1 center can check logs at 6AM. But then again there is a lot of turnover.

Don't get paid for On-Call, but the job is super flexible with hours so we get flex time which is well worth it. I like being able to take a nice 2 hour lunch every once in a while.

gambiting on Jan 13, 2016 [-]

I work for a major games developer/publisher. Our submission packages are being built this week and will be submitted next Monday. We are supposed to be on-call this weekend just in case any problems in any of the systems we were responsible arise. No, I am not paid for being on-call, just like I am not being paid overtime. From what I can tell people are going to come in this weekend just to play the game and make sure everything works even though they are not getting paid for it.

goldenkey on Jan 13, 2016 [-]

Given that this weekend work is scheduled into the process, not paying you is intentional and well, grossly deviant. Grow a backbone or find another job. For all of us, stop letting this shit slide.

laumars on Jan 13, 2016 [-]

I'm basically expected to be 24/7. Which gets a bit much some weeks.

toomuchtodo on Jan 13, 2016 [-]

What stops you from looking for another job?

laumars on Jan 13, 2016 [-]

Nothing. :)

But for all the long hours and high stress levels, I do still quite enjoy my job. Which makes it all the more harder sometimes, because as bad as my current place might get on occasions, I know there's very few positions out there that are sufficiently better to encourage me to leave.

But the plus side is that it puts me in the fortunate position where I can turn down jobs. A luxury many people don't have.

toomuchtodo on Jan 13, 2016 [-]

Completely understand. I'm in the same position. Thanks for replying!

jjp on Jan 13, 2016 [-]

In a past life covering out of hours (typically 9pm through 6am) for a core application for an insurance company the set-up we had was that we had a ladder of 4-6 staff providing on-call support. Whom ever was top of the list was first paged and if they didn't acknowledge within x minutes we kept going down the list until somebody responded. Once you got called you owned all incidents for that night and the next morning you would be moved to the bottom of the list.

When we got called out we were well recompensed at minimum of two hours of 3x hourly rate. If you resolved incident in 10 minutes you still got 6 hours of pay. If you got another call within those first two hours then you didn't get anything extra until you were working for 2 hours and 1 minute , but if the 2nd call came in after 3 hours you started the clock again. During times of instability (new code release) we often had management agree to you working from first call-out until the day shift came in to order to minimise downtime. When we were working during the night the on-call always had priority, but if you were sat around waiting for the system to do something then we all did something else to keep us awake. But there was no expectation that it was other tickets or project work. We made personal choices of doing day work or it could be crossword puzzle etc.

We all had areas of expertise and if you caught something you didn't know how to resolve then we had the authority to ring anybody. And that person would also receive the same compensation. In 5 years doing this I never had a complaint from calling somebody else out. Did we get tired yes? Most of us had project roles or team lead roles during the day but we all knew that the production system had priority and we and our own line management would deal with the project work accordingly. Did we get burned out on this? No because the rota meant that you still had a life. If you were top of the list and had other commitments that night you'd negotiate coverage with somebody else further down the list to see who could go top for the night.

My employer did this because the cost of not doing and having downtime during the day was a lot higher. If the system failed out of hours then it could impact our business and we could lose c.8,000 hours of day shift working time that would still require salary payment. Plus reputation damage etc. Most I ever accumulated over a weekend was 32 hours at 3x (situation so bad required to restore database to last known good and then catch-up log files).

One night I also got it wrong and in trying to correct the incident forced us into a situation when we had to do a database restore. That caused about 4,000 man/hours of downtime. When my bosses boss came in at 6:30am (he got called out by my boss) I got sent home being told I was too tired to continue support, day shift staff would take over and the missive to ring him when I had slept. I made the phone call expecting a significant dressing down only to be told, everybody else who looked at the diagnostics said they would have done exactly the same, we've all learned, and you stay top of the list tonight so that we know you're not scared to get back on the horse. Truth be told it did knock my confidence and anything that was a little out of the ordinary after that I always ended up calling somebody else for a consult. After a while I engineered my self off the on-call list.

romanhn on Jan 13, 2016 [-]

- Every team is free to choose the on-call practices (escalations, ramp-up for new members, etc) that work for them, although many practices are shared. Tooling is the same across the board. I'll speak for my team.

- Rotations are mixed, consisting of weekdays and weekends. In other words, if you're on-call M-F this week, in a few weeks you'll go on-call over the weekend.

- If the on-call engineer does not pick up, the incident is escalated to the manager, then their manager, all the way up to the CTO. Some teams have secondaries before managers are involved.

- The on-call engineer would normally work off the regular backlog. We have talked about pulling them out to work on on-call + tech debt exclusively, but there hasn't been enough on-call churn to justify this.

- The on-call is on the hook for resolving the immediate issue. In many cases this does not mean actually fixing the underlying problem. Those get written up as tickets to be prioritized as part of the backlog.

- The priority is determined as a team. As a manager, I encourage the team to aggressively tackle on-call issues, don't want to be a blocker for that. If something consistently pages us, it is guaranteed to make it to the top of the backlog fast. We have also chosen to prioritize product work over on-call tickets when we felt our pace was good and the on-call churn not too terrible. Kanban makes priority changes pretty easy.

- Not sure what you mean by "on-call can't complete in time". On-call-related tickets end up on the backlog so anyone can tackle them, not just the person that filed it.

- We have a pretty good on-call culture and the teams are very sensitive to other teams' pain. If we're being paged about an issue with another team's code (shouldn't happen too often), there's always the option to page their on-call and triage together.

- We track operational churn, send a weekly on-call hand-off email to the team with notes about the pages and steps taken, and have operational review meetings where these emails are reviewed (+ any other operational matters) and next steps are determined. Maintaining transparency around operational pain and building structure around the follow-up process has been really helpful in reducing our weekly churn.

- I think the ideal number of team members that should be on-call is around 5 to 8. Anything less than that and rotations become very frequent and a burden on the individuals. Anything more and the rotations are so rare that every on-call feels like the first time.

- Last piece of advice is to FIX THE PROBLEMS (or change your alerts). Build whatever process you need to make sure that you have plenty of leeway to address anything that pages you constantly. Don't get overly attached to the alerts - you might find that changing sensitivity thresholds or deleting the alert altogether might actually be the right answer (please don't do that for the wrong reasons though :)). If you're paged for a bunch of things that you can't fix, you're probably doing it wrong. Just like technical debt, if you don't tend to on-call issues they WILL get out of hand. Picking away at it slow and steady will almost certainly help reduce it from a torrent to a trickle.

N.B. I work at PagerDuty. A bunch more suggestions here: https://www.pagerduty.com/blog/tag/best-practices/.

eecks on Jan 13, 2016 [-]

We have no developers on call

caw on Jan 13, 2016 [-]

I've worked two different on-call systems. The first was an enterprise company with a very defined process. 1 hour of pay for every 10 on-call (even if you're salaried, they figure out your hourly pay for this). Must be able to respond and/or get into the office in 1 hour, otherwise they call your manager to get someone working on the issue. Our 24/7 support would triage all issues before escalating on the on-call engineer--they were able to resolve most issues. Rotation period was roughly 1 week in 3 or 4 on a volunteer basis. We did have a group of software engineers who had an on-call rotation for the internal applications that powered the business. If I needed additional support for the application I could tell our 24/7 support folks to page/call them and they would hop online, which was needed on rare occasion when I couldn't fix an application error or a bug slipped the manual and automated QAs.

My current on-call system is considerably different as a startup. Most weeks there will be no pager alerts, some weeks will be particularly bad because something fell out of stability due to some other changes. There is a primary and secondary on-call. Each level has 20 minutes to respond and get online before it escalates (no office requirement since we're cloud based). The rotation goes through devops and all the software engineers, so you're on call for 2 weeks in 8-10 with no extra pay. I wouldn't recommend this method of scheduling because it's mandatory for all and some people don't take the duties seriously because nothing bad will happen if you let your pager slip to the next line, other than irking your coworker. There's no incentive to learn how to do the repairs or do them. I've seen a lot of "Oh my phone was off, I didn't realize I was on-call" that never happened at my previous job with the volunteers for extra pay. Having the secondary is nice because it is a guaranteed person you can escalate to for help on a complex issue, and they are available if you need to be indisposed for a short period of time.

About your specific questions: Your on-call duties are to communicate, fix what's broken, coordinate any additional escalations that need to happen, and most likely host the postmortem. Nothing more. Update your status page, send out an alert internally that X is broken, etc. When it's fixed and normal service has resumed, communicate that as well. You're the point of contact for anyone with questions about the issue, not anyone you had to bring online for a fix because if they're interrupted they're not fixing.

Don't start non on-call work. Fix to the extent that you know how and will make it stable until business hours. Not everyone has coded every system and knows the "permanent fix" for every issue. Your priority is based on what is broken and the business criticality of it. If multiple systems have failed, you fix the most critical ones first, which are normally the customer facing ones. There should be no "existing on-call tickets" because on-call bandaids and makes a high priority issue in the normal work queue.

If on-call can't complete the work in a reasonable amount of time, it may make sense to raise other people who can help. If it's going to take you 4 hours to get the $CRITICAL_FUNCTIONALITY back online, but getting Joe to help you will only make it take 1, then by all means try to get Joe if he's available. Again, based on your rotation he may not intend to be available, may not have his phone on, and may not want to take the call.

If you're dependent on 3rd parties, your application needs to take that into account. If you wake up because the external API is down and your functionality is down because of it, all you're going to be doing is losing sleep waiting for it to come back up to send out the all-clear. This changes somewhat if your external API comes with a SLA and a telephone number -- by all means call and start their triage chain.

A good thing to have is a sync up meeting with the on-call folks so you can establish patterns in the alerts that may not be evident by any one person.

INTPenis on Jan 13, 2016 [-]

* expected duties (only answer on-call, do other work, etc)

Expected duty is to solve the problem but there are often escalation paths to take. If a problem is not solved within a required SLA period then the company can be forced to pay the client penalty fees.

* how deep does your on-call dive into code to fix, vs triaging, patching, and creating follow up work to fix permanently?

They go as deep as necessary, or as deep as is dictated by an operational manual pertaining to that particular client/environment.

* priority order of work (on-call tickts vs regular work vs existing on-call tickets vs special queue of simple tasks)

On-call is always higher priority since it's an add-on service that clients pay for.

* what happens when the current on-call can't complete in time?

See above, penalty fees mostly.

* how do you manage for other teams' risk? (ie their api goes down, you can't satisfy your customers)

Not sure I understand the question, if an API goes down and affects our services then that API needs to be monitored and handled by our on-call team.

* any other tidbits

I'm not in the on-call team but I stay available for specialized expertize if the 1st line can't solve an issue.

I know how they work though so here's one example. It all depends on the clients SLA but let's say the client has 99% uptime, 24/7 on-call duties in their contract.

In that case one person out of 5 rotates an on-call device (phone) every 5 weeks. In the strictest of SLA they're required to respond within 15 minutes and sometimes have a solution within 4 hours. This varies wildly from contract to contract.

Of course an incident manager is available, redundantly, and is tasked with coordinating skills between departments to solve an issue within the designated SLA.

Alerts come into the device and the tech can respond to alerts directly via the device to acknowledge, force re-check or simply disable. There is also a more featured web interface for the alerts to access via a browser.

Alerts are sent with SMS through a self-hosted gateway. Directly attached to the monitoring server, not using any e-mail translation API.

Alerts are logged, and in some cases a ticket is created for an alert.

Preferably a manager should work out the on-call schedule, but techs often trade weeks and are more than capable of handling it themselves.

They receive an added monthly compensation including overtime. So any work must be reported in a time reporting utility to eventually lead to payed overtime depending on the contract it pertained to and the time when it happened.