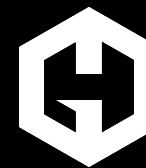


CHATOPS/AUTOMATION

HOW TO GET THERE WHILE EVERYTHING'S ON FIRE

Fran Garcia
hostedgraphite.com



\$ WHOAMI

Fran Garcia, Ops Engineer @hostedgraphite

No twitter account (I know, right?)

(We're hiring! hostedgraphite.com/jobs)



CHATOPS: NOT JUST FOR BIG TEAMS ANYMORE!



Usually the automation success stories we see in blog posts or in talks at conferences come from either big or dedicated engineering teams like Twitter, Facebook, etc. They are inspiring and usually open source software comes out of it but apart from that, they're not all that relevant when you're just a one-person, two-person ops team with your hands full just keeping things operational.

We're starting to see a bit more content on that topic now that's applicable to a wider range of teams, and this is my attempt at helping close that gap.

THINGS I WANT TO COVER TODAY

- WTF is this chatops thing anyway?
- Hosted Graphite: a one-year automation journey
- Key lessons learned on building automation



I'd like to talk about three things today:

- I want to take a quick look at this "chatops thing" the kids are talking about, and see why it could be something you might want to invest time on.
- I'd like to talk a little about our own journey over the last year, what was our starting point and where we're at now, mostly so we can have a frame or reference for the third part of the talk...
- ...Which is the key lessons we learned along the way on trying to build up our own automation.

"CHATOPS IS GOOD, CHATOPS IS LIFE"

Before we start, let's introspect a little...



OK, so before we start, a little show of hands...

"CHATOPS IS GOOD, CHATOPS IS LIFE"

Before we start, let's introspect a little...

Who doesn't know what chatops is?



"CHATOPS IS GOOD, CHATOPS IS LIFE"

Before we start, let's introspect a little...

Who doesn't know what chatops is?

Who is currently implementing chatops?



"CHATOPS IS GOOD, CHATOPS IS LIFE"

Before we start, let's introspect a little...

Who doesn't know what chatops is?

Who is currently implementing chatops?

Who is planning to implement chatops or would like to?



"CHATOPS IS GOOD, CHATOPS IS LIFE"

Before we start, let's introspect a little...

Who doesn't know what chatops is?

Who is currently implementing chatops?

Who is planning to implement chatops or would like to?

Who hates the word "chatops" with the burning rage of a thousand dying suns?



"NIHILIST ENGINEERS DON'T BELIEVE
IN CHATOPS"



The truth is that chatops is mostly a marketing term.

A lot of the content that gets pushed to us feels rather empty, there's a lot of talk of bots that have personalities, or a full AI and that makes people feel like they're just looking at a fad that will go away soon or an attempt to sell them something.

And I think that's harmful because if you look past all that you can find some really useful ideas.

SO WHAT IS
THIS CHATOPS
THING ANYWAY?



In its essence, as a concept it's so simple it doesn't really deserve or need a fancy name. Just put your tools right where the conversation is happening, make "doing the thing" and "telling people you're doing the thing" the same action.

I don't want to jump back and forth between windows during a major incident to tell people that I'm recalibrating the flux capacitor, flux capacitors are hard enough as it is. If everything happens in the chat then everybody has the appropriate context, today and 6 months from now. A new hire has 90% of the context on how you solve that weird incident 6 months ago.

You are sharing knowledge by doing work. That's basically it.

HOSTED GRAPHITE, CIRCA JUNE 2015

1 ops engineer (Hi Dan!)

Growing quickly, lots of scaling bottlenecks.

Very immature automation, nonexistent in some areas.

Ops putting out fires, pushing new features



I joined HG roughly a year ago. Before I joined there was only one dedicated ops engineer and boy was he busy!

We were growing fast, and hitting scaling bottlenecks left and right, which meant lots of fires to put out while a solution was figured out.

We didn't have much in the way of automation, though we did have some basic foundations.

HOSTED GRAPHITE, 2015 DEPLOYMENT PROCESS

- "Just modify this yaml file in this server here"
- "Then just trigger the deployment using ssh in a for loop"



As an example, let's look at the deployment process we had in place.

We use puppet, and an external node classifier (ENC) to tell what hosts should have what classes and drive our configuration. Our ENC lives in a git repo, but changes needed to be made in the puppet master itself before committing them (assuming you didn't forget to commit them when you were done), so we'd update our ENC, and then we'd need to trigger the deployment in the affected servers.

The solution? SSH in a for loop to the rescue.

The process was awkward and it could either be done quickly, masking some errors, or in a more elaborate way, actively tying up an engineer for some amount of time.

Due to this deployments for our web projects would only happen during a couple of days a week.

HOSTED GRAPHITE, PRESENT DAY

3x growth on ops team (3 of us now! and going up).

Still growing! Traffic has doubled over the last year.

Much more mature automation, not limited to use inside of ops.



Today: We're already three ops engineers, and looking for a fourth one.

We're still growing fast and hitting the occasional scaling bottleneck, but we're better prepared to face them.

Our automation has matured and is not only used by ops: New developers deploying to production on their first day (stats from a random week show roughly 20 *web* prod deployments on that week).


Developers provisioning and decommissioning servers straight from a contingency hardware pool without intervention from ops.


HOSTED GRAPHITE: PRESENT DAY

 **fran** 17:48
glitter: addtolb webserver-0028 render wrestled it from rob's cold dead fingers, and back to prod

 **rob** 14:56
glitter: justdeploy cop_carbonpickle cops@production --version=0.2.9.11

 **glitter** 14:57
Triggering puppet run in cops@production with --tags=aptupdate,cop_carbonpickle and a batch size of 10. Rundeck job id is 482 (<http://localhost:4440/project/main/execution/show/482>)

 **fran** 11:41
glitter: incidentnote 765 while trying to update the config haproxy was stopped in some aggs

 **glitter** 11:41
Annotation for incident #765 added.

 **fran** 15:15
glitter: agg drain agg-0013,agg-0026,agg-0029,agg-0030,agg-0032,agg-0033 draining more old aggs

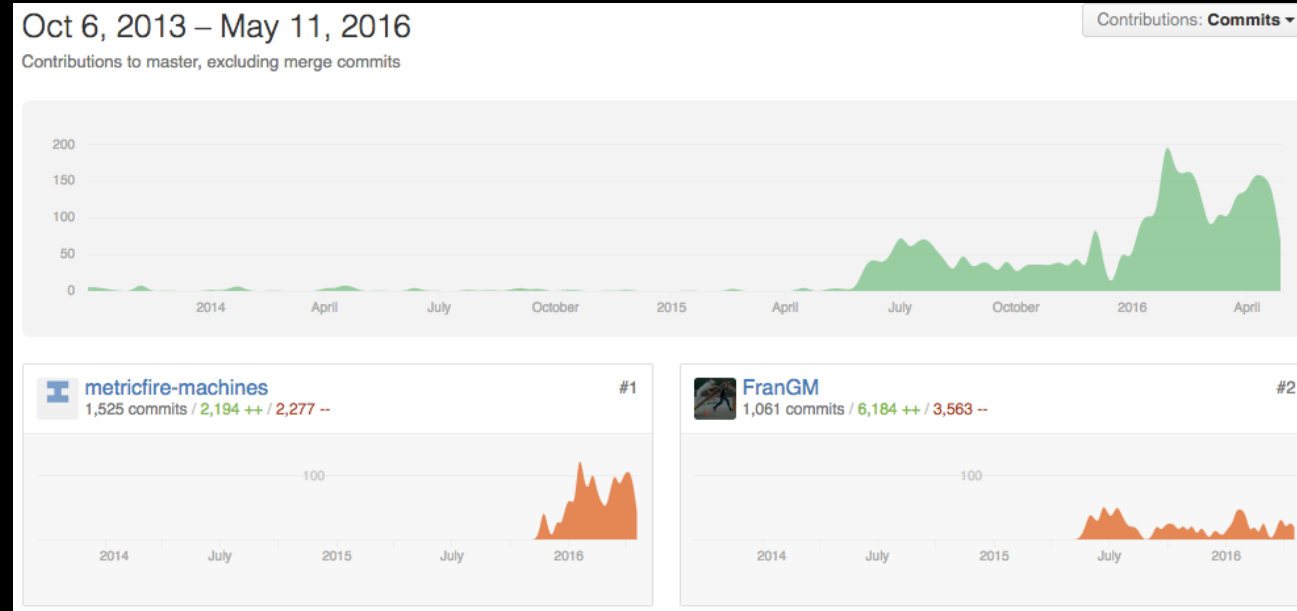
glitter: ratelimitset production haproxy_api=1750

glitter: retire_host riakts-0001 riakts-0002



Here are some examples of tasks we perform today for which we had no tooling a year ago. There are quite a few more, and others that have been created and deprecated on that time frame.

THE ROBOTS ARE TAKING OUR JOBS!



Here's an example of how profound the change has been for us. We use puppet heavily in our infrastructure, so the great majority of our configuration changes involve modifying our ENC in some way or another. As our ENC is stored in a github repo, every change results in a commit.

You can see in this graph how, despite the fact that we only started tracking glitter's commits in late 2015 it's already by far the largest committer to that repo. Not only that, but the rate at which we commit to the repo has seen a massive increase as well, which means we're getting a lot more done while doing a lot less work.

So the balance so far has been positive, but how did we manage to get here?

HOW DID WE GET THERE?



At the beginning, we didn't really have the time to spend on automation with everything else going on, which was just as well because we didn't really know what we wanted, or even where to start. Everything seemed important!

Initially our automation efforts were more of a side project than anything else. We knew that having solid automation was really important for us but it was hard to justify investing the amount of time we thought it would take us to start things off.

Looking back at the past year we can see a set of principles/lessons that we think can be applied to achieve better automation in the context of a team that doesn't have a lot of resources.


DREAM BIG



OK, so this one sounds almost like a joke. We're talking about not having time and resources to automate all the things and our first suggestion is to dream big? The truth is, direction is a really important thing to have. If you don't know where you'd like to go, you can only get there by moving in random directions. So sit down and ask yourself, how would this tool behave if time and resources were not a problem? What's the ideal scenario here?

DREAM BIG: EXAMPLE

glitter 14:25
superagg-0003 seems to be acting up (bad meatgrinder canary values) <http://i.mfhg.io/render-api/5de74f77/c6c5c58e4836cdc.png> (10KB) ▾



, attempting to remove
superagg-0003 has now been removed from the backend! ✓

rob 14:25
nice one, glitter

dan 14:32
Good man



So this is an example of having what at the time seemed like an ambitious goal. Our data ingestion pipeline is composed of several layers, and in our ideal scenario our automation would be able to detect that a server is misbehaving at that layer and remove it from service, with the on call person not being paged but notified in the morning.

The only way we could actually achieve this was by tying it with the second lesson...

DREAM SMALL



If you just remember one thing, remember this one, everything else is filler to get this idea out.

Get your crazy goal, and think of the first 2-3 tiny little steps that would help you get there. Ignore the massive leap of faith between step 3 and the desired result, and focus on 2-3 tiny steps that are easy to achieve and would get you a little bit closer. This is important, every step needs to result in something that's useful, even if just slightly.

This should be something that requires a very small investment on your part, so it's easier to justify. You just need to do a couple of small tasks, and by that point you've already improved your tooling and gotten one step closer to your goal.

This is going to allow you to experiment, see what works and discard what doesn't without fear, as opposed to spending two months designing a solution that might be the wrong one.

At that point you can figure out what the next 2-3 steps are and try again. Sometimes you'll be surprised how quickly you'll get to your "ideal" state.

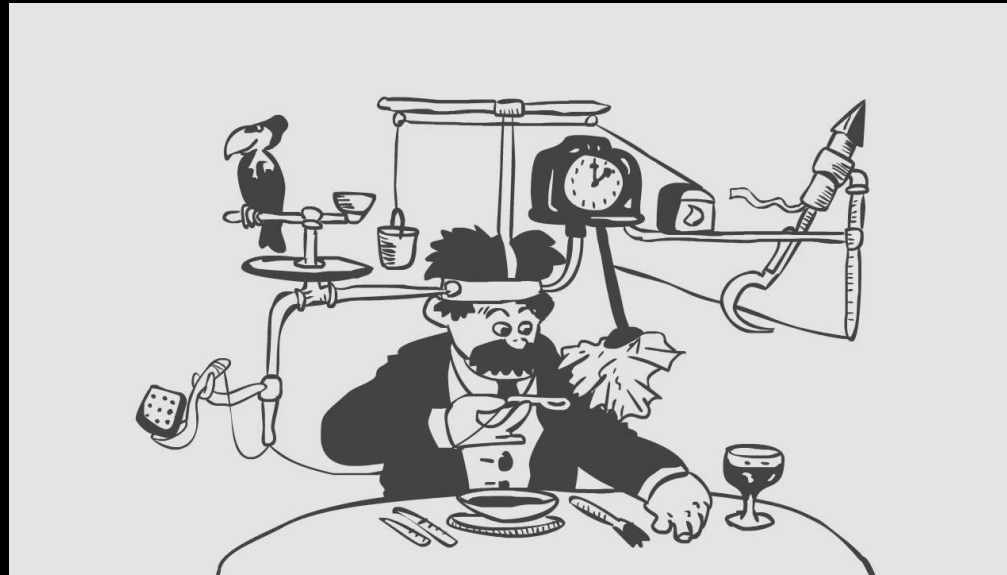
DREAM SMALL: EXAMPLE



Back to our previous example, we only got to fully automating this process by breaking it down into tiny problems and solving each of them separately. For example, we have a tool that allows us to make sure a server is removed from the backend, and another tool that will tell us if a given server is not receiving all the canary data it should over the last few minutes. It's easy to see how combining these two we can achieve automatic removal of bad hosts. These two commands themselves are also built upon smaller building blocks: like code for making generic requests to graphite and evaluating the results, or an API of sorts to interact with our ENC.

BE CONSISTENT WITH YOUR INTERFACES

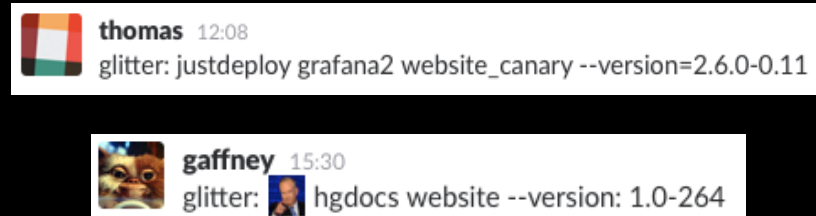
(Not just for code or APIs)



This is not really about your API or the way the user interacts with your tool (but those are really important too!). Everywhere a component interacts with another component an interface is created.

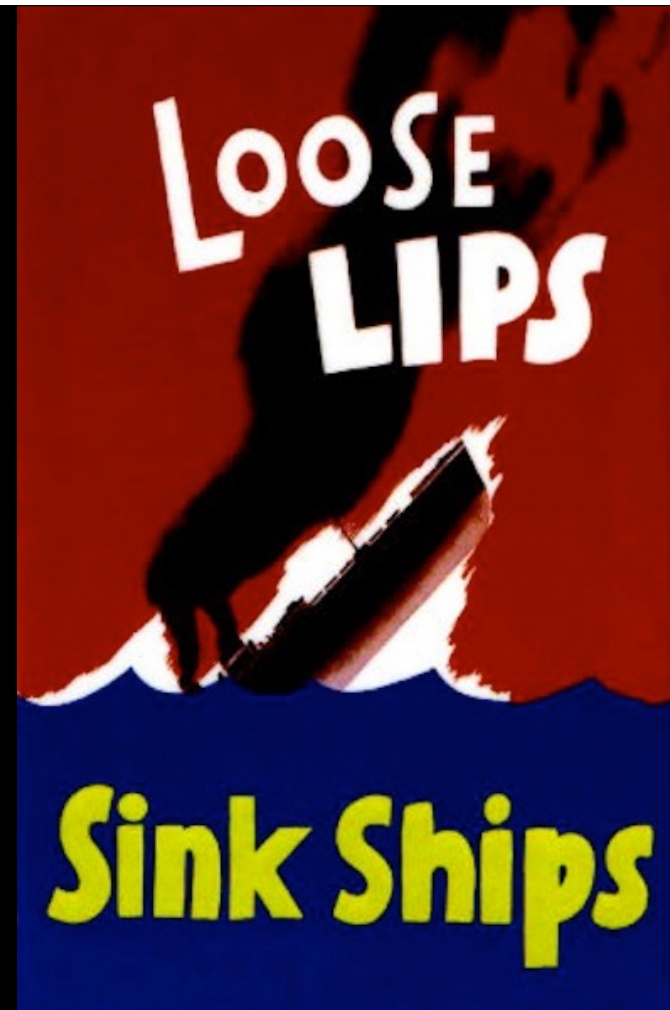
This is about the underlying plumbing that your automation is going to use. The more consistent you are the more impact you can have with your automation changes (remember, you're trying to make your own life easier here)

CONSISTENT INTERFACES: EXAMPLE



Example: All our puppet modules try to be consistent in how we pin installed versions for each service. That means that when implementing a command that would handle deployments of a given service, I realised that the same command would work with 90% of our services with not modifications required (except the name of the command).

KEEP IT
LOOSE(LY
COUPLED)



We know loose coupling is great when designing systems, and same as in the previous case, everywhere two components interact a dependency is born. This is a bit of a pain as we're trying to experiment with a lot of things and if we're not careful we might end up needing to support something that doesn't quite work out just because we didn't realise we were introducing a dependency at the time.

As an example, when we started on this magical automation journey we started using mcollective to do orchestration behind the scenes. While it served us well, we felt like we were not getting the reliability we needed out of it, so we decided to replace it with rundeck for some tasks. This would have been a daunting task as it affected almost every single command. Fortunately the relevant code had been abstracted away to two methods: one for triggering an action through mco, and another one to parse the results, making the transition reasonably easy and seamless.

ALWAYS HAVE A PLAN B

Or "don't get rid of the old way of doing things just yet"



You're going to be building something new, and we're going to try to move fast, so things are liable to get explodey at some point. When that happens you don't want to have your whole team/company paralysed, you want a plan B.

ALWAYS HAVE A PLAN B

[Personal](#)[Open source](#)[Business](#)[Explore](#)[January 28th Incident Report](#)

Our early response to the event was complicated by the fact that many of our ChatOps systems were on servers that had rebooted. We do have redundancy built into our ChatOps systems, but this failure still caused some amount of confusion and delay at the very beginning of our response. One of the biggest customer-facing effects of this delay was that [status.github.com](#) wasn't set to status red until 00:32am UTC, eight minutes after the site became inaccessible.

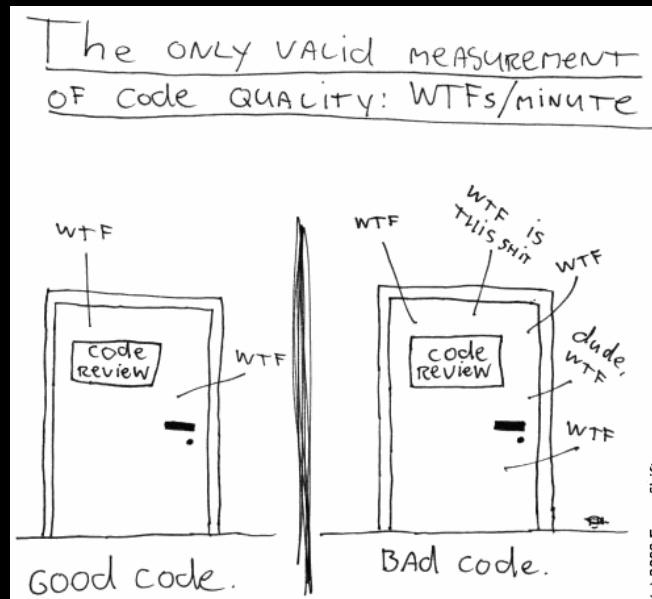


A good example of this is github's incident report from back in January, the loss of the servers running their "chatops" systems was a problem for them, and there's a lesson to be learnt there, but there's another important lesson: they have redundancy. They realise those servers are a critical part of their infrastructure and, therefore, need to have proper redundancy.

We actually have two bots running on different servers, ostensibly for testing changes to the software, but very useful for redundancy purposes as well.

We even have a plan C: All our automation focuses into augmenting what was already in place, not fully replacing it. I can deploy any project without access to Slack, it might be a little slower but it would work.

DON'T BE AFRAID OF WRITING BAD CODE



This one might be a bit controversial, but don't be afraid of writing terrible code. We all like to talk about how our code is clean, elegant, full of tests and wonderful abstractions but the truth is, particularly when dealing with ops tooling, reality is messy, and you can either choose to release code that works today and makes everybody more productive, or wait two months before realising you're trying to solve the wrong problem anyway.

DON'T BE AFRAID OF WRITING BAD CODE

Crappy code in the repo is more productive than great code in your head.

Prototype, discard what doesn't work and refactor what does, it'll be grand.

Recommended watch: "You code like a sysadmin"
LISA14 presentation (<https://youtu.be/ZjdqP-REoqc>)



The best way to achieve what we want is through small iterations and experimenting. You're going to throw out a lot of code and that's fine, just make sure you don't go refactor-crazy before you have the right solution.

I'm currently refactoring some of our code and I can tell you, it's pretty terrible, but all the effort spent writing it has already paid off, and even if I decided to rewrite it, it's easier to rewrite a working prototype than to write everything from scratch.

HELP PEOPLE
BUILD
CONFIDENCE IN
YOUR TOOLS



You can build great tooling but it's not going to have the impact you want unless people actually use it, and sometimes that doesn't necessarily depend on purely technical factors.

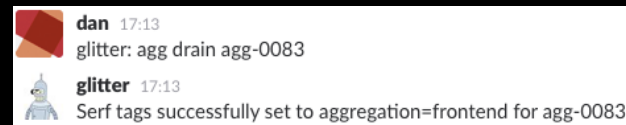
One way of looking at automation is as a productivity multiplier, which means that while it can help me be more productive and avoid mistakes, in a lot of cases it can also help me screw up 10x worse than otherwise.

There's also the psychological aspect to it, if a new hire makes a mistake and brings a server down most people would think it's normal and wouldn't forbid them from touching production ever again, but if an automated system does a similar thing people will want to disable the system making it harder to keep developing it.

This means that we need to be particularly careful when trying to introduce automation, especially in companies that are somewhat risk averse.

BUILDING CONFIDENCE

Build the tool before the automation.



Tom Limoncelli, author of "The Art of System and Network Administration" makes a distinction between automation and tool building. A tool is something that helps you do your job, but that needs you to be there and execute it, while automation is the system being fully autonomous and deciding when to run a given task without human input.



Building trust is a slow process, so you need to go step by step. Even if you think you can build the full-on automation at first, just build a version of it that needs to be triggered by hand, and slowly introduce it as an easier way to do something. If it's reliable enough people will get used to it and won't think twice when you expand its responsibilities.

The drain command we introduced earlier is an example of this. We made sure the command was reliable and everybody was familiar with it before even proposing to automate it.

BUILDING CONFIDENCE

Allow humans to verify actions before applying them

```
glitter: dnsplan  
 glitter 15:05  
OK, Generating plan for dns changes...  
Terraform plan generated for DNS: https://gist.github.com/bc2671313c8be753070b7716dfa9de97
```

```
 bruno 15:05  
glitter: dnsapply  
 glitter 15:05  
OK, applying a previously generated plan, this might take a while, put the kettle on or something...  
Terraform apply for DNS done: https://gist.github.com/ccdb15f728d7371bae8f164832dcc83d
```



Sometimes even building a simple tool seems scary. Our DNS is managed by route53 and our tools generate the DNS entries based on our inventory by generating terraform files on the fly.

In theory, our tool is ready to automatically regenerate our DNS anytime something changes, but that's scary. It could also generate a plan and apply it on a single command, but that's also scary.

Solution: Do it in two steps. Show the user what you would do, and allow them to apply the plan. At some point people will get tired of verifying the plan because it's always correct, and then you can merge both commands.

BUILDING CONFIDENCE

Add the heuristic and have the tool tell you what it would do



Once you have figured out when you'd like the tool to act, have it tell you when it thinks there is a problem. Sending a metric every time the tool would have acted is a good way of doing it, if it's supposed to be infrequent, you can also log it to chat, so you'd see what it'd look like.

Here's a previous iteration of our server removal tool, bitterly complaining that it's not allowed to make changes in production.

Once everybody is confident that there are no false positives it's a lot easier to make the case to fully automate this process.

TEACH THEM HOW TO FISH
INSTEAD.

DON'T GIVE
PEOPLE FISH
(BECAUSE IT'S
WEIRD)

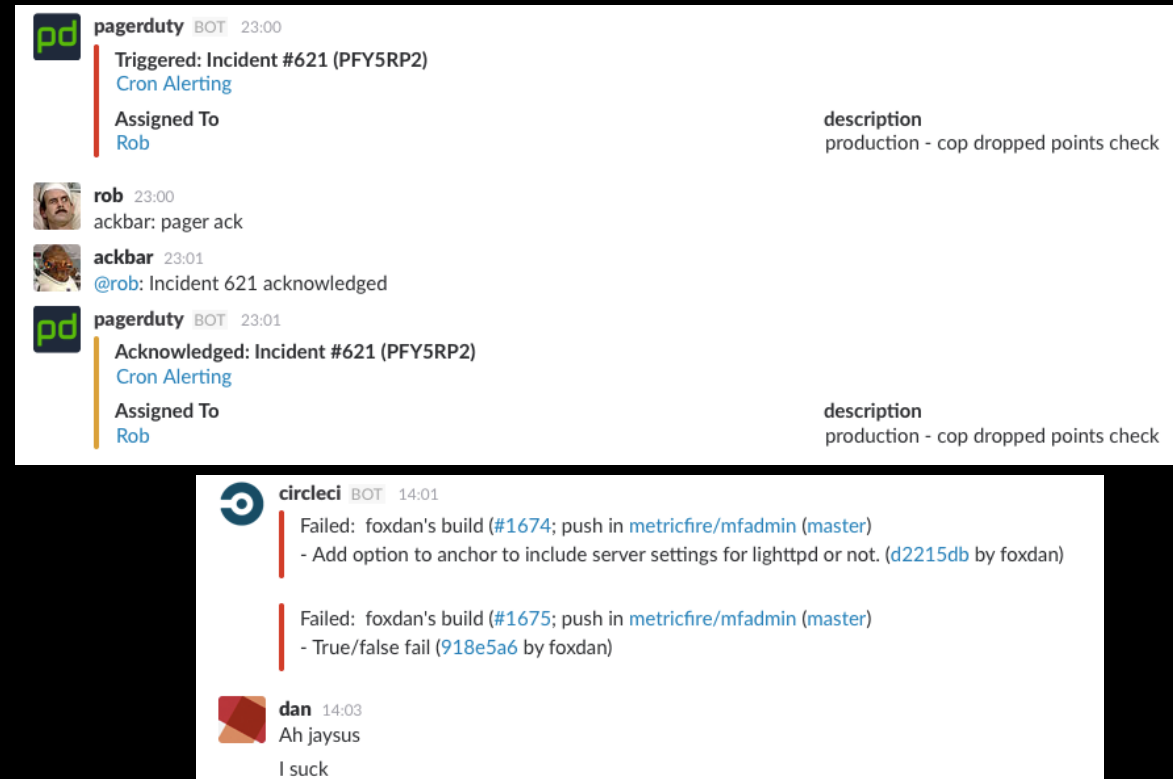


Even if people trust our tools to do the right thing, we still need to teach people what features are available to them, and how they can use them to accomplish their tasks.

In my experience this has been a surprisingly tough challenge, especially as we've been trying to move fast and experiment a lot, so at times new commands might be showing up on a weekly basis. Even if you kept completely perfect and up to date documentation, people are not going to check it that frequently, so they're still going to be missing out.

Something that works is to flat out skip that step. Don't try hard to document features or to teach people how to use them, just use them publicly yourself. When they ask you to provision a server and you know the bot can do it, just go to your main channel and ask the bot to provision the server. Maybe make a show of asking for the command's help before you invoke it so people can see what parameters are accepted. At that point, not only everybody knows that the bot can do the task, they have searchable history showing it at work!

DON'T UNDERESTIMATE INTEGRATIONS



The screenshot shows a Slack conversation with two main messages. The top message is from the Pagerduty bot (pd BOT) at 23:00, stating 'Triggered: Incident #621 (PFY5RP2)' with a link to 'Cron Alerting' and 'Assigned To Rob'. The description is 'production - cop dropped points check'. Below this, a user named 'rob' (23:00) says 'ackbar: pager ack', and 'ackbar' (23:01) responds '@rob: Incident 621 acknowledged'. The second message is also from the Pagerduty bot (pd BOT) at 23:01, stating 'Acknowledged: Incident #621 (PFY5RP2)' with a link to 'Cron Alerting' and 'Assigned To Rob'. The description is the same. Below this, a message from 'circleci BOT' (14:01) shows two failed builds: '#1674' and '#1675', both pushed by 'metricfire/mfadmin (master)'. The first failure is 'Add option to anchor to include server settings for lighttpd or not. (d2215db by foxdan)' and the second is 'True/false fail (918e5a6 by foxdan)'. Finally, a user named 'dan' (14:03) says 'Ah jaysus' and 'I suck'. A small hexagonal logo is visible in the bottom right corner of the screenshot.

pd BOT 23:00
Triggered: Incident #621 (PFY5RP2)
[Cron Alerting](#)
Assigned To [Rob](#)
description
production - cop dropped points check

rob 23:00
ackbar: pager ack

ackbar 23:01
[@rob](#): Incident 621 acknowledged

pd BOT 23:01
Acknowledged: Incident #621 (PFY5RP2)
[Cron Alerting](#)
Assigned To [Rob](#)
description
production - cop dropped points check

circleci BOT 14:01
Failed: foxdan's build (#1674; push in [metricfire/mfadmin \(master\)](#))
- Add option to anchor to include server settings for lighttpd or not. ([d2215db](#) by foxdan)

Failed: foxdan's build (#1675; push in [metricfire/mfadmin \(master\)](#))
- True/false fail ([918e5a6](#) by foxdan)

dan 14:03
Ah jaysus
I suck

The best thing about integrations is that they're code you don't have to write, which is the best kind of code there is, period. I can easily go to zero from "pagerduty alerts integrated in our Slack workflow" in no time.


Pagerduty integration is particularly great because it allows everybody to be aware when something exceptional happens, which makes it everybody's business instead of "just an ops thing". Since we started using the pagerduty integration we've had people that usually wouldn't react to pages jumping in and helping resolve an incident, sometimes even before the on call person gets the chance to!

One of the oldest integrations that we have enables us to set the message on our status page right from IRC/Slack. You don't know how useful this is until you need to update your status page at 4 am.


The problem is that they're so easy that people tend to overdo them sometimes, flooding everybody with data and making them ignore them altogether, so experimenting with them is recommended, but just because an integration exists doesn't mean we *have* to use it.


TESTIMONIALS


BONUS SLIDE

 **cian** 21:02
Oh, nice **feature**

 **cian** 21:02
<3 **glitter**


 **eman** 15:02
glitter is so polite, it's nice


 **charlie** 14:36
glitter has many more abilities than I thought.


 **dan** 14:36
glitter is evolving

 **rob** 14:36
glitter is sentient

 **fran** 12:03
glitter is no fun

 **fran** 17:46
glitter is the ghetto blaster of software development

 **fran** 12:49
glitter is stuck in the 90s

 **fran** 11:01
glitter is a massive mess, and I apologize for that

WRAPPING UP



Wrapping up, we have access to a lot of technologies that weren't quite so commonplace a few years ago, and which allow even really small teams to get pretty good results and a really interesting productivity multiplier. As long as you have your end goal in mind at all times and move forward in small but firm steps you'll improve things, and every improvement you make will only free you to make further improvements, so the only logical conclusion is that, five years from now, your work week will be -20 hours a week!

THANKS!

- Thoughts? Questions?
General abuse?
- @hostedgraphite
- fran@hostedgraphite.com

- (We're still hiring!
hostedgraphite.com/jobs)

