

DevOps 101 with Atlassian



CONTENTS

What is DevOps?	3
DevOps and Atlassian	6
Building Products, DevOps Style	9
Continuous Delivery for Infrastructure	16
Handling Incidents at Atlassian	24
It's Time for Your DevOps Story	28

What is DevOps?

Five years ago, Marc Andreessen proclaimed that software is eating the world. After all, what company isn't a software company? Case in point:

- Modern cars contain hundreds of millions of lines of code—far more than all of Facebook, from Zuckerberg's dorm years to today.
- Even pizza delivery has gone high tech. With advanced mobile applications for placing orders and tracking deliveries, Dominos Pizza has increased its IT workforce by 240%.
- Nike is turning footwear into a fully connected platform by integrating shoes with lifestyle and fitness applications.



Old-school development models just don't hold up to such high-demand, high-growth environments. Traditionally, Development and Operations teams work separately in silos, hindering the ability to move fast. The response to this contentious relationship was a movement called DevOps. It's a fancy phrase for a simple idea: your dev and ops teams work better together. It advocates for better

communication and collaboration so that developing, testing, releasing, and running software can happen more rapidly and reliably.

Instead of delivering big, infrequent releases (once every 3 to 9 months) like traditional development teams at major enterprises, DevOps takes a “continuous delivery” approach. This means releasing small, incremental improvements regularly—often even several times per day.

The results are enormous, and go far beyond the operational.

“ Companies that practice DevOps are twice as likely to exceed their goals for profitability and market share.

Puppet Labs’ 2015 State of DevOps Report

They also enjoy:

- 30x more frequent deployments
- 60% higher change success rates
- 60x fewer failures
- 160x faster recoveries

These results aren’t limited to major enterprises with billion-dollar dev teams, either. You can achieve them yourself, no matter how small your team is. The #1 success factor is teamwork. At Atlassian, the key to faster, higher quality releases is a strong relationship between your dev and ops teams, and the right tools and processes in place to support them.

So what does that look like at Atlassian, and how did we get started?

The #1 success factor is teamwork

“The #1 success factor is teamwork. At Atlassian, the key to faster, higher quality releases is a strong relationship between your dev and ops teams, and the right tools and processes in place to support them.”



DevOps and Atlassian

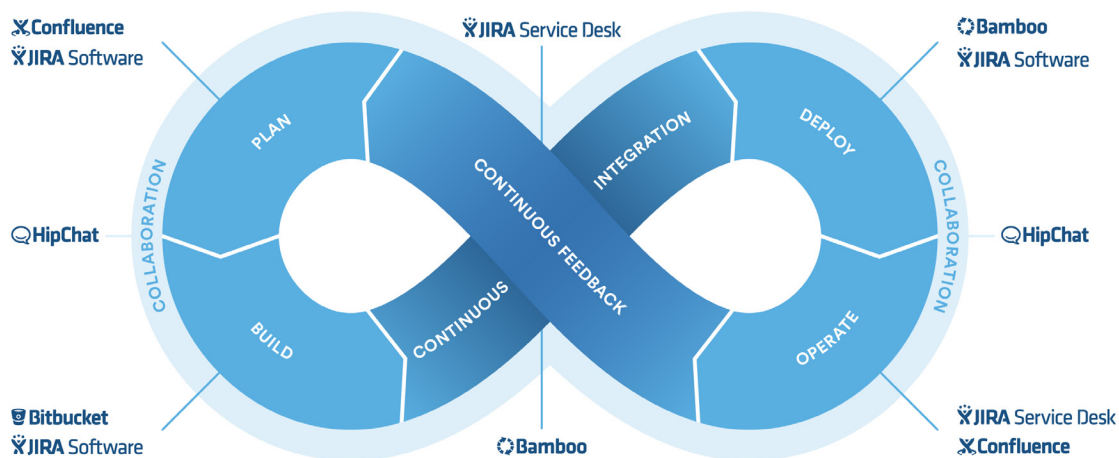


Christophe Capel
Head of Product
Marketing, JIRA
Service Desk

At some major big box retailers, really heavy items have “team lift” stickers on them to indicate when several employees need to help move the items from shelf to shopping cart. “Team lift” is actually a perfect analogy for the entire DevOps methodology, since DevOps isn’t any single person’s job—it’s everyone’s job.

At Atlassian, we use our own products to understand our uses, and provide additional testing before we release them to our customers. In short, we dogfood our own products.

In this ebook, we’ll cover each step in detail, and exactly how we use each Atlassian solution. For now, let’s start with our process, which looks a bit like a hot tasty pretzel:



1. First, we plan the features we will deliver to our customers. We use Confluence and JIRA Software to organize customer feedback and list requirements. We create issues in JIRA Software to start tracking the stories and epics we define for each software project.

2. Then, we build the software—writing code and running tests until we get it right. Bitbucket lets us create branches for each new feature we need to create, and it also allows us to code more collaboratively,

since we can use pull requests to facilitate faster reviews, and comment inline and hold conversations between our developers right within the code.

A cool feature we love: When a developer creates a pull request to flag new code for team review, Bitbucket automatically updates the related JIRA Software issue status.

3. We continuously integrate new features back into a master branch for deployment. Bamboo makes this easier, helping us automate builds, tests, and releases along the way. It really speeds up deploying to AWS, too—we love using Docker and Bamboo together for even faster, more efficient deployment.
4. JIRA Software's release hub also gives us full visibility across all our branches, builds, pull requests, and deployment warnings, so we can release with confidence.
5. Once we've deployed a new feature into production, it's time to run and operate it. At Atlassian, our developers are fully responsible for the features they build, so using JIRA Service Desk helps them track and resolve incidents faster. We use Confluence to manage run books, knowledge base articles, and related documentation at every step.
6. We deliver continuous feedback (via reports, tickets, etc.) to our development teams, so they can plan new releases, fix bugs, and deliver faster, more reliable software to our customers. With JIRA Service Desk, we can even request customer feedback from both internal and external users.

Throughout the entire lifecycle, HipChat is the secret salty coating to our pretzel. It adds an additional layer of collaboration on top of our already collaborative processes and technology by letting our teams swarm on incidents, wherever they are, via desktop, mobile apps, and even wearables.

That's just the basics, though, and you came here for details. So let's dive in.

PRO TIP

**DevOps isn't
any single
person's job**

—

it's everyone's job.



Building Products, DevOps style



Tanguy Crusson
Product Manager,
HipChat

Let's say your engineering team has gone Agile. They work in sprints, collaborate, and are building a lot of great features. But there's just one catch: you still have to wait for the release train to leave the station, and customers aren't getting value fast enough.

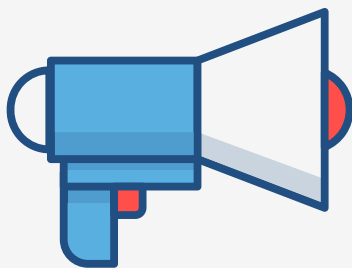
We'll show you our best practices for building products, DevOps style. Let's start with feedback; because no matter the product, your success is solely based on your users.

How to gather feedback—and use it to shape and build features

We've learned over the years that the easiest way to make our product better is to listen to the people that use it. Thousands of companies use HipChat, and thousands of Atlassian use it internally, too.

You can collect feedback from just about every source imaginable.

- Ask for in-product feedback
- Collect user feedback from JIRA Service Desk
- Monitor social media channels like Twitter and Facebook



- Use Apdex scores to monitor whether your users are satisfied with your service's response times
- Gather monitoring data from third party solutions like Datadog and New Relic.

What do we do with all that feedback? Here's what we do with it:

Keep in mind: this may or may not work for your team, but is nonetheless a useful starting framework that you can tweak.



We send all feedback to HipChat as notifications. For example, we get a ton of tweets:



hey @hipchat, any news about deeper JIRA integration? issue links!

Eric Wood @ejwood79

We route them, along with all our other social media mentions, bug reports, etc. into dedicated HipChat rooms where the whole team can discuss each notification and help shape our backlog.

Important feedback, like bugs, is then converted into a JIRA Software ticket—which we then prioritize into the backlog. If there's a new feature, we'll typically create a Confluence page to spec out goals and requirements.

In either case, we make sure to always listen to our customer feedback, wherever they are, and take action when possible.

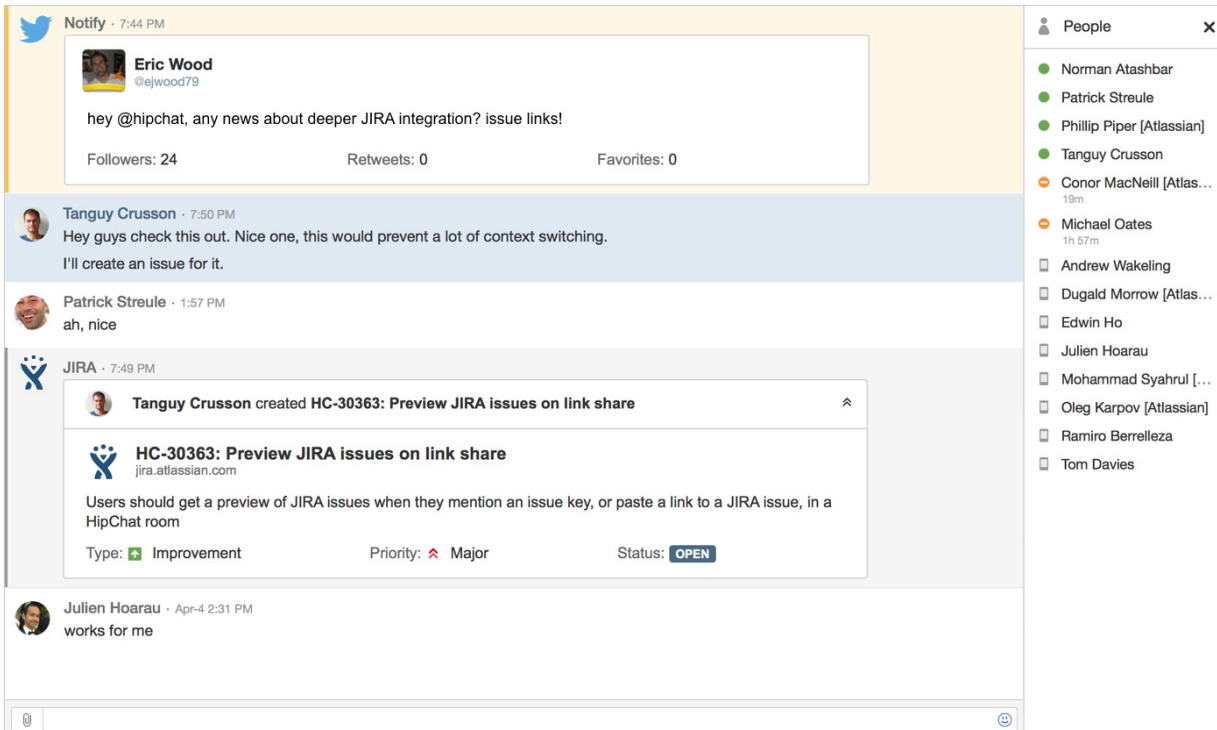
Plan together in sprints

So, how exactly do we plan what we're going to build?

Our small development teams regroup and meet for an hour every week. We use the hour to:

- Demo everything that was built in the previous week to keep the team informed and connected.
- Review the objectives and sprint goals we established the previous week and agree on whether we achieved them.
- Define our objectives for our next sprints. At Atlassian, a sprint objective isn't the same thing as a ticket. A sprint objective is a unit of work that you have to be able to demo to the team, or ship to production at the end of the sprint.

After the meeting, we break out. With our new objectives in hand, our developers can go through all the issues in our backlog and pick out the ones that will help us achieve the sprint objectives we took on during the meeting.



The screenshot shows a HipChat chat window. At the top, a notification from Twitter is displayed for user Eric Wood (@ejwood79) with the message "hey @hipchat, any news about deeper JIRA integration? issue links!". Below this, Tanguy Crusson (7:50 PM) says "Hey guys check this out. Nice one, this would prevent a lot of context switching. I'll create an issue for it." Patrick Streule (1:57 PM) replies "ah, nice". A JIRA notification card follows, titled "Tanguy Crusson created HC-30363: Preview JIRA issues on link share". The card details the issue: "Users should get a preview of JIRA issues when they mention an issue key, or paste a link to a JIRA issue, in a HipChat room". It lists the type as "Improvement", priority as "Major", and status as "OPEN". Finally, Julien Hoarau (Apr-4 2:31 PM) responds "works for me". A sidebar on the right lists participants: Norman Atashbar, Patrick Streule, Phillip Piper [Atlassian], Tanguy Crusson, Conor MacNeill [Atlas...], Michael Oates, Andrew Wakeling, Dugald Morrow [Atlas...], Edwin Ho, Julien Hoarau, Mohammad Syahrul [...], Oleg Karpov [Atlassian], Ramiro Berrelleza, and Tom Davies.

The end result is complete buy-in from the team. Everyone is fully involved in defining our goals, how we are going to achieve them, and how we are dividing the work.

Spike early and often

You're probably familiar with the term "spike" in agile development. A spike is a short effort to gather information, validate ideas, identify early obstacles, and guesstimate the size of initiatives. Instead of building a shippable product, we focus on end-to-end prototyping, to arm us with the knowledge we need to get the job done right.

At the end of each spike, we have a better idea of the size and technical obstacles we will encounter for each initiative, and we categorize them: Extra Small, Small, Medium, Large, Extra Large, or Godzilla.

We regularly rotate between normal sprints and spikes, and hold regular "innovation weeks" that result in really amazing prototypes and insights around project scope and approach. Most teams at Atlassian hold innovation weeks, too, and they love to write about them.

Keep even the biggest changes small

Instead of shipping big things infrequently, ship small changes very often. It makes it very easy to roll back a particular change if we need to, or even better: fix and roll forward, and it helps us iterate very fast.

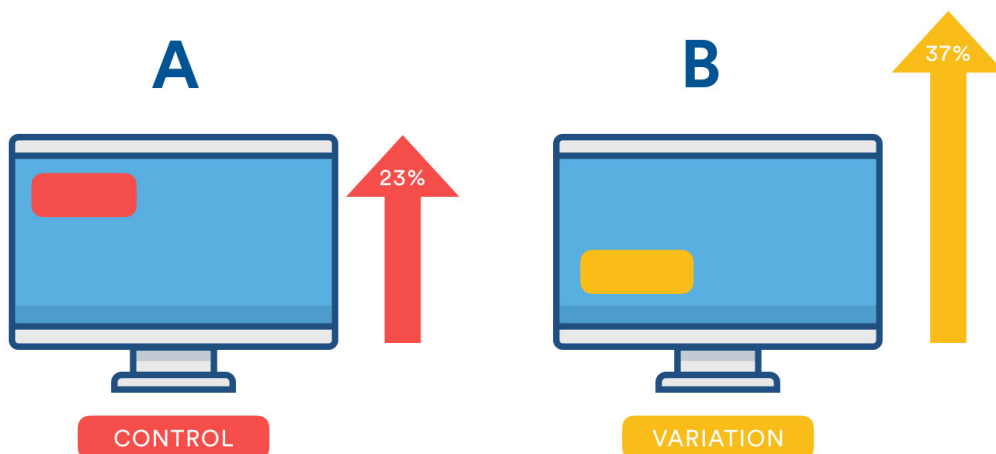
For really big changes—like highly anticipated new features, for example—we still take a “start small” approach, setting “step by step” goals and running frequent A/B tests and experiments to see what our users like best.

“ Instead of shipping big things infrequently, ship small changes very often.

To test, we divide our users into cohorts. For example, cohort A might see one version of a HipChat feature, and cohort B might see a slightly different version. We look at the usage data to see which version of the feature is performing

best against the goals we defined during planning—and we keep iterating and testing until we get to the best version of that feature.

A tool we use during these testing phases is Launch Darkly, which lets us release new features to small segments of users, gather feedback, and then gradually increase the audience size until we’ve fully deployed. We often start with just 5% of users running the new feature—and then slowly increase by 10 or 15 percent increments after each feedback and revision cycle.

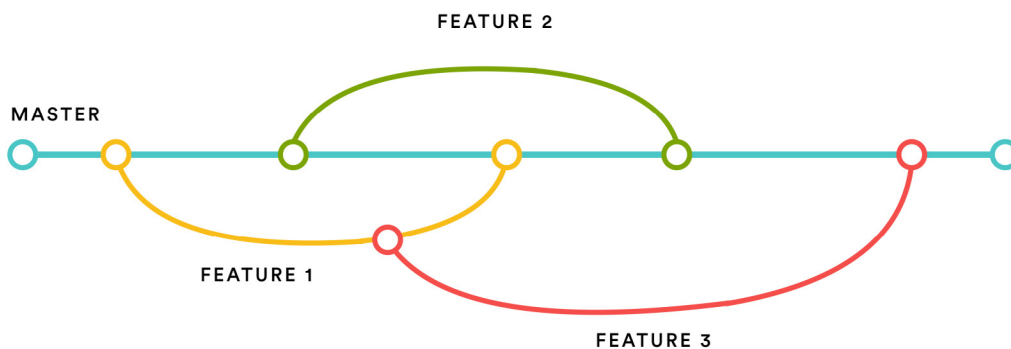


Git + Bitbucket + Bamboo = automated awesome

We're heavy users of Git and Bitbucket, using feature branches to make continuous integration far more effective. Any feature however small, translates into a feature branch, which is automatically tested via our Bamboo builds.

After we test a feature branch, we create a pull request to merge it back to the master branch, and we select a minimum of two reviewers from our team to review and verify the code. Once you get a green build and 2 approvals, you're good to go.

Since our master branch is what gets shipped to production, we require that the master be “green”—no known bugs, issues, or errors—at all times. If a build goes “red,” that means all hands on deck, and the entire team has to drop everything to fix the build.



Encourage accountability

We follow one of the main DevOps principles. We're big on “you build it, you ship it, you run it”, meaning the team that is responsible for writing a feature also becomes the team responsible for deploying it and providing ongoing maintenance once it's live.

But isn't that going to introduce a lot of issues in production? In fact it's quite the contrary: It encourages every developer to build the very best version of something, and gives each of us a vested interest in its ongoing success.

What this leads to is 100+ developers being able to ship to production at any point in time. This is made possible with the right process and especially the right tools. We use Chef and Puppet for automa-



tion, and developed a number of Chat Apps (HipChat add-ons) to help us coordinate this process.

Finally, accountability for us also means keeping our users informed of what's going on. Occasionally, bad stuff happens, and glitches have the potential to impact all of our users. We love StatusPage.io for keeping everyone up to date on the status of all of our services.

We're big on “you build it, you ship it, you run it”

We're big on “you build it, you ship it, you run it”, meaning the team that is responsible for writing a feature also becomes the team responsible for deploying it, and providing ongoing maintenance once it's live.

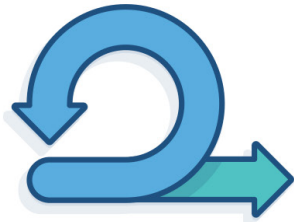


Continuous Delivery for Infrastructure



Michael Knight
Build Engineer,
Atlassian

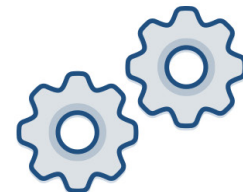
It's not just development teams that can use DevOps practices. You can apply the same practices to your hardware and configuration work, too. At Atlassian, we've built a team of a dozen employees (called Build Engineers) that are dedicated to helping our developers code faster, by giving them the best hardware and infrastructure services possible. We oversee our continuous integration service (Bamboo), our artifact storage and retrieval service (Sonatype Nexus), and all the hardware, server configurations, applications, and services that glue them together and provide a smooth experience to our dev teams.



Stable & fast CI



**Artifact storage &
retrieval**



Associated tooling

Let's take a deeper dive into the technology and processes we depend on, and my top tips for running a Build Engineering team more efficiently and effectively.

Gather feedback from developers

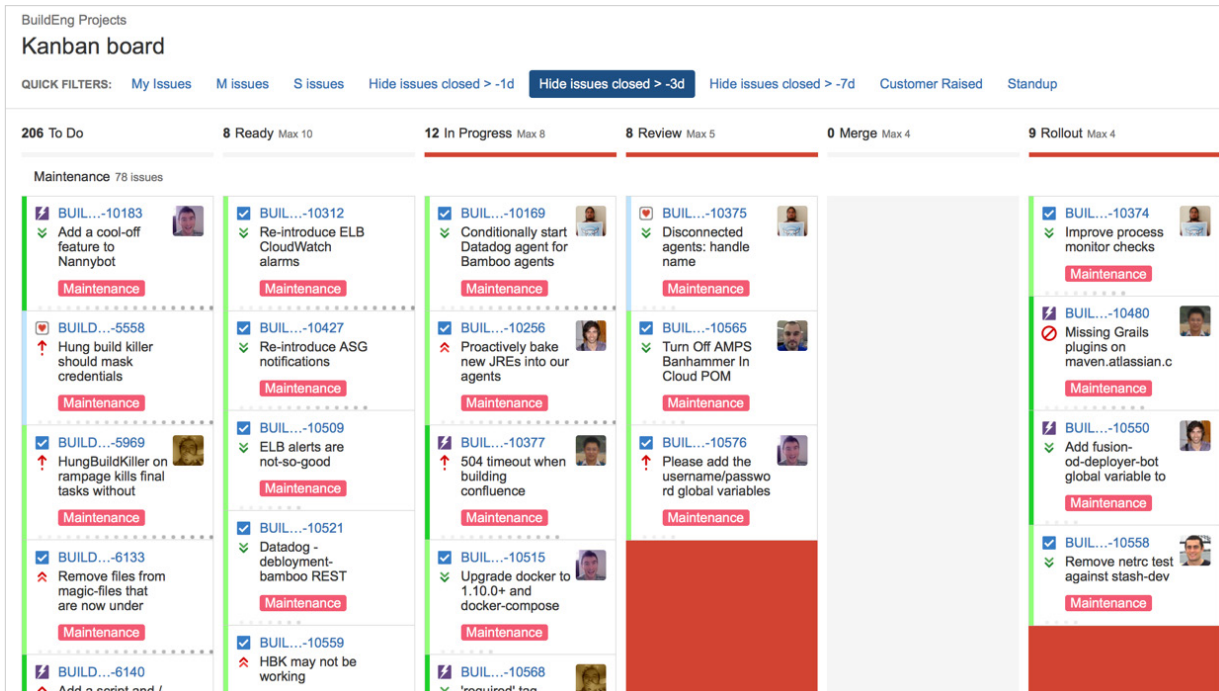
Our customers are Atlassian's developers. We used JIRA Service Desk to create our own engineering service desk, and that's how they contact us to submit requests and provide us with feedback.

“Walk the board” during standups

Each morning, we have standups just like most software dev teams, where we go through all the issues in flight using our Kanban board in JIRA Software. Each issue is categorized as:



We set a maximum threshold for the number of issues that can be in each status column. Below, you’ll see a few columns that have “gone red” because we’ve exceeded our defined thresholds. This helps us determine in our standup that we need to finish the work in that column before we pick up anything new.



BuildEng Projects
Kanban board

QUICK FILTERS: My Issues | M issues | S issues | Hide issues closed > -1d | **Hide issues closed > -3d** | Hide issues closed > -7d | Customer Raised | Standup

206 To Do	8 Ready Max 10	12 In Progress Max 8	8 Review Max 5	0 Merge Max 4	9 Rollout Max 4
<p>Maintenance 78 issues</p> <ul style="list-style-type: none"> BUIL...-10183: Add a cool-off feature to Nannybot (Maintenance) BUILD...-5558: Hung build killer should mask credentials (Maintenance) BUILD...-5969: HungBuildKiller on rampage kills final tasks without (Maintenance) BUILD...-6133: Remove files from magic-files that are now under (Maintenance) BUILD...-6140: Add a script and / 	<ul style="list-style-type: none"> BUIL...-10312: Re-introduce ELB CloudWatch alarms (Maintenance) BUIL...-10427: Re-introduce ASG notifications (Maintenance) BUIL...-10509: ELB alerts are not-so-good (Maintenance) BUIL...-10521: Datadog - deployment-bamboo REST (Maintenance) BUIL...-10559: HBK may not be working 	<ul style="list-style-type: none"> BUIL...-10169: Conditionally start Datadog agent for Bamboo agents (Maintenance) BUIL...-10256: Proactively bake new JREs into our agents (Maintenance) BUIL...-10377: 504 timeout when building confluence (Maintenance) BUIL...-10515: Upgrade docker to 1.10.0+ and docker-compose (Maintenance) BUIL...-10568: 'required' tag 	<ul style="list-style-type: none"> BUIL...-10375: Disconnected agents: handle name (Maintenance) BUIL...-10565: Turn Off AMPS Banhammer In Cloud POM (Maintenance) BUIL...-10576: Please add the username/password global variables (Maintenance) 		<ul style="list-style-type: none"> BUIL...-10374: Improve process monitor checks (Maintenance) BUIL...-10480: Missing Grails plugins on maven.atlassian.c (Maintenance) BUIL...-10550: Add fusion-od-deployer-bot global variable to (Maintenance) BUIL...-10558: Remove netrc test against stash-dev (Maintenance)



Pull requests: swarms, approvals and keeping things green

We create branches for any hardware or configuration change, no matter how small, exactly the same way that our software development colleagues do. Every single pull request is linked to a JIRA issue, and we manage the pull requests in Bitbucket, requiring two approvals from our colleagues (plus a green feature branch build) to move forward.

Our team also has a HipChat room where we wrote a bot to keep track of all our pull requests. It shows all open pull requests, and how close they are to being merged. We leave it up to the team to swarm over the pull requests and jump in and provide feedback for the ones they feel most qualified to review. Everyone pitches in and works really well to move us through the pipeline faster and knock out our in-process work.

So HipChat, JIRA Software, JIRA Service Desk, and Bitbucket are a big part of our day-to-day operations.

Build Engineering - Private
Happy Easter! 🐰

- BUILDENG-10169_elastic: increased max tries because datadog agent i... · 0d7e79c57d2
- BUILDENG-10169_elastic: removing bash sleep from process check · 0c8054a3e1f
- Merge branch 'master' of ssh://stash.atlassian.com:7997/bui... · 73d37d92ebb
- Merge pull request #1716 in BUILDENG/buildeng-puppet from BUILDENG-... · ba36c3041af
- Merge pull request #1719 in BUILDENG/buildeng-puppet from noissue-... · 8a917bfc233

[See more](#)

Milos Kleint · Mar-24 14:52
open prs

Open PRS · Mar-24 14:52
Waiting for approvals:

- [bamboo-isolated-docker] #43: BUILDENG-10449 custom scaler
Milos Kleint, updated 2 days ago, 1 approval(s)
- [bamboo-sox-plugin] #17: BUILDENG-10560 add triggers for builds
Thomas Tsang, updated a few seconds ago, 0 approval(s) ✓
- [bamboo-sox-plugin] #16: BUILDENG-10444: integrate BB Server SOX verification REST endpoint
Chandler Zhang, updated 15 hours ago, 1 approval(s) ✓
- [buildeng-puppet] #1724: BUILDENG-10572: add global variable for micros team
Chandler Zhang, updated 15 minutes ago, 0 approval(s)
- [buildeng-terraform] #464: noissue: fix up ECS agent names + place them under server jurisdiction
Oswyn Brent, updated a day ago, 0 approval(s) ✓

Open PRS · Mar-24 14:52
Waiting for a green build:

- [buildeng-puppet] #1723: BUILDENG-10573: Added micros token and jwt private key for export validation engine
Thomas Tsang, updated 2 hours ago, 2 approval(s)

Open PRS · Mar-24 14:52
Ready to be merged:

- [buildeng-terraform] #465: BUILDENG-10375: Changed jira_software_bamboo name to be consistent with all other bamboo service names.
Bradly Swart, updated 2 days ago, 2 approval(s) ✓ ✓

People

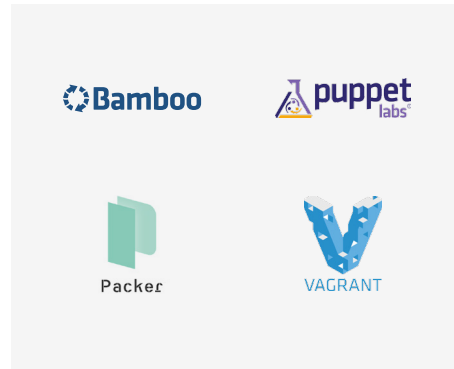
- BuildengBot
- Mike Knight
- Bradly Swart [Atlassian]
- Chandler Zhang
- Colin Hebert
- Lachlan Dally
- Martin Meinhold
- Oswyn Brent
- Otto Jongerius
- Peter Hardy
- Thomas Tsang [Atlassian]
- Tim Cinel
- Milos Kleint
- Peter Leschew

Favorite Pipeline Tools

You might be wondering what tools to use for handling software, configuration, and hardware deployments. Here are a few of our favorites:

Software Pipeline

Just like our software development team, we use Bamboo on the infrastructure side, to manage and run our build plans and deployments. We use Bamboo to manage Puppet, where we write new modules to install and configure components on our servers, like a model to install the SSH keys from everyone on our team.



Vagrant lets us spin up test servers easily, which we apply Puppet configurations to for testing purposes. Puppet and Vagrant integrate really well, and the combination makes it really easy to test new AWS server configurations automatically.

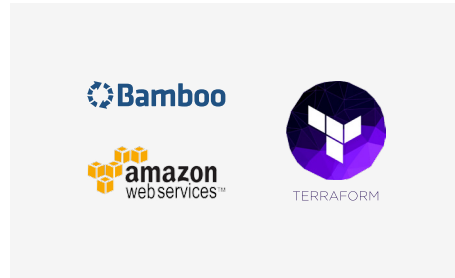
Cucumber is great for testing, too. We use it to confirm that our agents are installed properly, and that the changes we have made haven't broken anything.

Once we're finished testing a configuration or change, we deploy our new Puppet tree out to production, and HipChat will automatically post a notification to the issue assignee to verify that the change is working in production, and to also close the issue in JIRA.

As always, Bamboo shows the status of the build, and the details of each release, like which environments it's been deployed to, and which JIRA issues are addressed in each build and release.

Hardware pipeline

Bamboo manages everything in our hardware pipeline as well, from start to finish. Since we make heavy use of Amazon Web Services (AWS), we use Terraform to manage our hardware infrastructure. We love it because it allows us to use software best practices and workflows to make changes to our hardware.



For example: Changes we request to our hardware infrastructure through Terraform have to be verified through pull requests, and deployed through a continuous delivery pipeline—the same process our software developers have to follow for their work. This keeps us consistent about how we manage quality across the board.

Here's a quick example of what Terraform code looks like, just in case you're curious:

```
resource "aws_instance" "nat" {
  ami = "${var.aws_nat_ami}"
  availability_zone = "us-east-1b"
  instance_type = "m1.small"
  key_name = "${var.aws_key_name}"
  security_groups = ["${aws_security_group.nat.id}"]
  subnet_id = "${aws_subnet.us-east-1b-public.id}"
  associate_public_ip_address = true
  source_dest_check = false
}

resource "aws_subnet" "us-east-1b-public" {
  vpc_id = "${aws_vpc.nat-vpc.id}"
  cidr_block = "10.0.0.0/24"
  availability_zone = "us-east-1b"
}
```

Here, we're basically setting up a new NAT server on AWS. We use code to set all the parameters, like subnet, etc. We can feed an entire hardware configuration into Terraform, and it will figure out all the API calls it needs to make to AWS to change our server topography from its current state to what is specified by the code. Then, we

can ask Terraform to execute the plan and make those changes. It's magical.

We track all of these releases with Bamboo, just like we do our software. Bamboo deploys each Terraform release into our staging environment first, and then our production environment once we're ready. Bamboo is also used to see which releases have been deployed across what environments.

Three core concepts to remember

Nothing changed the game more for our team than the idea of “infrastructure as code.” It's allowed us to adopt software development's best practices, but apply them to hardware and configuration management, and it's greatly improved the stability of our platform. Doubling the number of servers dedicated to running Bamboo at Atlassian was pretty much the same amount of work as just adding one would have been in a less efficient model.

Our team follows three basic principles that pretty much any engineering team can adopt:

1. Automate everything

It's critical that our builds work. If we don't test them thoroughly, we can't be confident they will work. Automated testing helps prevent regressions, gives us confidence in our changes, and makes continuous delivery possible for us.

We automate notifications, too, and just about anything we can to reduce human error and make sure we don't miss important tasks.

Finally, with more automation, we can keep our team smaller. That means less communications overhead, and more speed—which is exactly our team's charter.

2. Stay focused on continuous delivery

Stable hardware and reliable configurations are critical to making sure our developers can get their work done. So we follow continuous delivery best practices, just like they do:

- **OUR CODE IS ALWAYS RELEASABLE**

Our master is always “green” and stable, so it can be released at any time.



- **WE RELEASE FREQUENTLY**

This reduces risk, since there are only small changes from release to release, and we can revert easily as needed.

- **WE FOCUS ON FAST VALUE DELIVERY**

Since our users are Atlassian developers, we want them happy. Continuous delivery ensures we get improvements and fixes out to them as quickly as possible.

“As a result, we’re able to perform 10x more builds, without adding a single person to our engineering team.”

3. Embrace infrastructure as code

Simply put, this just means that we execute code to automatically configure servers, apps, and more instead of manually configuring them via other less efficient methods like in-tool configuration screens and wizards.

We can literally use code to hammer out commands like “give me N servers configured with apps X, Y, and Z”, and then use review and approval workflows to reduce human error significantly.

As a result, we’re able to perform 10x more builds, without adding a single person to our engineering team. We can deploy with far higher confidence, and more independence.

With more automation, we can keep our team smaller. That means less communications overhead, and more speed.



Handling incidents at Atlassian



Nick Wright
Head of Service
Operations,
Atlassian

But what about when things aren't working as planned—like when a feature rolls out that isn't performing optimally? That's where our Service Operations team comes in. Our job is to make it easier to spot and fix incidents, and prevent them from happening again in the future.

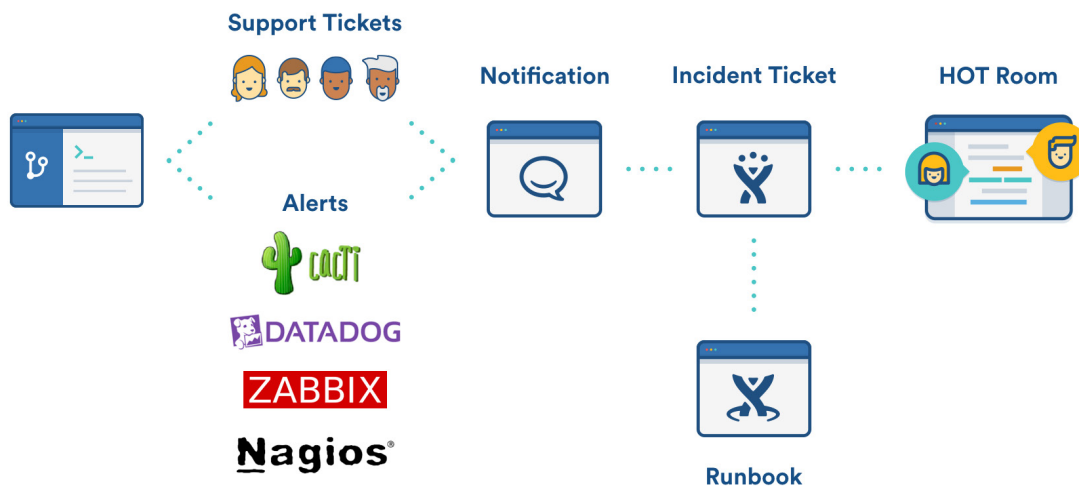
We use ITIL as the basic framework for our service management practice. It gives us a standard set of terminology and processes that make it easier to communicate and work together. More specifically, ITIL provides a strong foundation for how to classify incidents, define severity, and perform and track investigations into root cause and more.

Let's take a look at how Atlassian handles incidents when the poop (or anything else, really) does eventually hit the fan.

1. Someone (or something) reports the incident

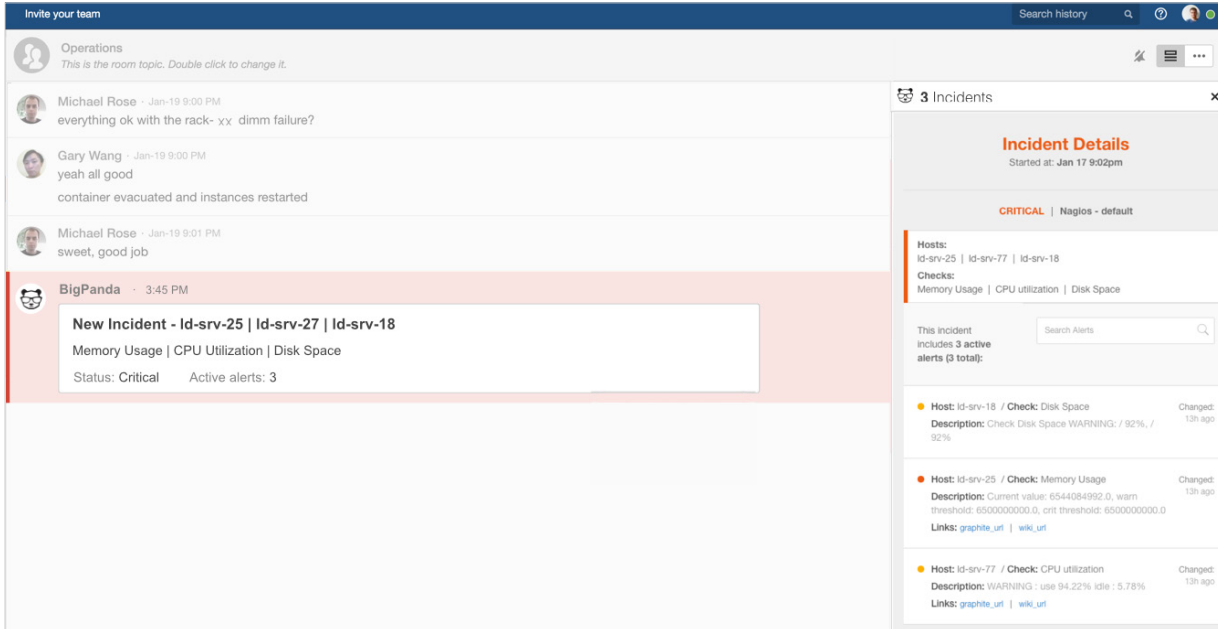
We learn about system outages and other potential performance glitches in two ways:

- Our users raise incidents using JIRA Service Desk
- Our monitoring systems (like Cacti, DataDog, Zabbix, and Nagios) send us a notification



2. We aggregate the alerts into HipChat

We aggregate all of our incident alerts into a single stream in a HipChat room, so our teams get directly informed that there is a problem. This can sometimes generate noise, so we turn to tools like BigPanda to help out. BigPanda correlates massive amounts of IT alerts and events, and helps group them together, saving us a ton of time.



The screenshot displays a HipChat interface for an 'Operations' room. A message from 'BigPanda' at 3:45 PM reports a 'New Incident' involving servers Id-srv-25, Id-srv-27, and Id-srv-18, with critical alerts for Memory Usage, CPU Utilization, and Disk Space. A detailed side panel on the right provides further information, including the incident start time (Jan 17 9:02pm), its critical status, and a list of three active alerts with their respective descriptions and thresholds.

3. We create an incident ticket

Occasionally, a team may know the outage was caused by a change they just made, and they can quickly disable that change. But more often than not, we need to pull a team together to troubleshoot and resolve something. The first step is to raise an incident ticket in JIRA Service Desk.

To create a ticket, we enter a few details, like a short name and description of the vent, and then categorize each incident by the impact it could have on a service, the number of users impacted, and how urgently it should be handled.

4. We notify our users

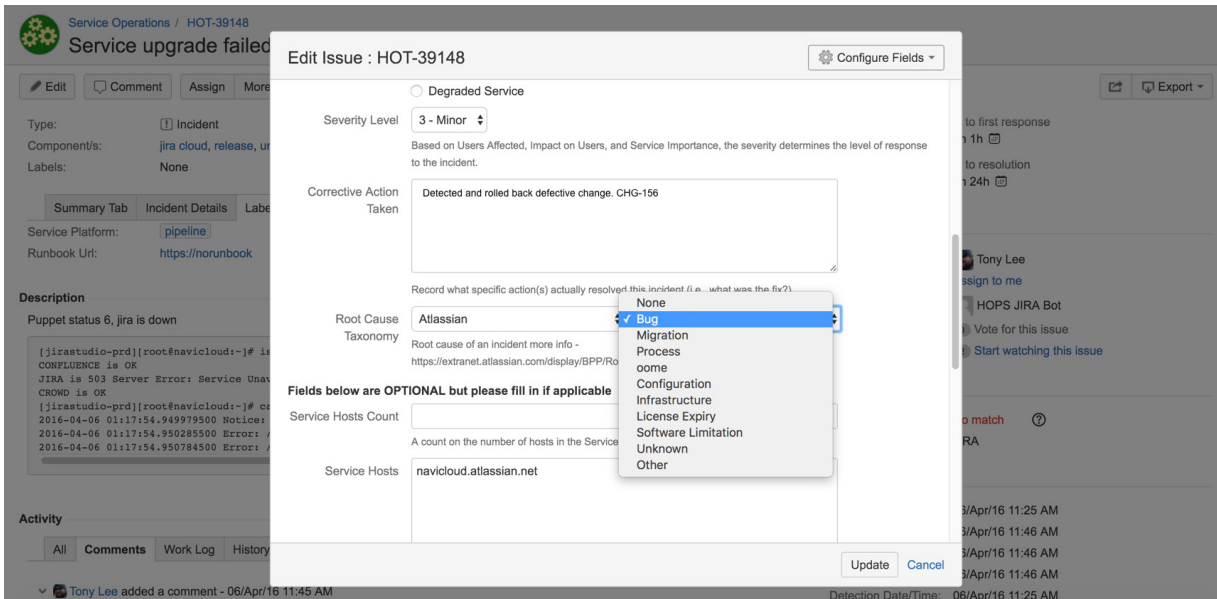
We use StatusPage.io to communicate with internal and external stakeholders, and push updates with incident status at regular intervals.

5. We create a dedicated chat room and swarm to resolve the incident

Within the incident ticket in JIRA Service Desk, we use the “create a room” feature to move the conversation to a dedicated HipChat room and pull in the right team to solve the problem at hand. The team discusses what went wrong, and agrees on an approach for troubleshooting and fixing it.

6. We resolve and categorize the root cause

ITIL recommends that we categorize each issue (bug, license expiry, infrastructure or configuration issue, etc.) once we’ve identified the root cause and taken corrective action. We also document the correction actions we took as well, and can use all of this information to run detailed reports highlighting our most common incident types and more. This helps us to take a more preventative approach to incident and problem management.



The screenshot displays the 'Edit Issue' form for incident HOT-39148. The form includes the following fields and options:

- Severity Level:** 3 - Minor
- Corrective Action Taken:** Detected and rolled back defective change. CHG-156
- Root Cause:** Atlassian
- Root Cause Taxonomy:** A dropdown menu is open, showing options: None, Bug (selected), Migration, Process, oome, Configuration, Infrastructure, License Expiry, Software Limitation, Unknown, and Other.
- Service Hosts Count:** (Optional field)
- Service Hosts:** navicloud.atlassian.net

The background shows the incident details page with a description of a puppet status error: "Puppet status 6, jira is down" and a log entry: "[jirastudio-prd][root@navicloud:~]# ... JIRA is 503 Server Error: Service Unavailable ...".

7. Finally, we conduct a post-mortem and document what went wrong

Possibly the most critical step to resolving an incident is learning from it. At Atlassian, we have a couple of different options for tracking the post-incident review activities: JIRA or Confluence. Confluence lets us configure templates for a standard incident report layout, and it’s easy to get started quickly. JIRA, on the other hand,

lets us build structured workflows that guide teams through the post-incident review process, and allow us to track each post-mortem review through to completion.

We've used both successfully. More important than the technology you use in the post-mortem process is making sure that you are able to develop a good understanding of the root cause of your outage. Use that to take the right set of actions to prevent the same outage from occurring again.

Our top recommendations:

- **CAPTURE THE DATA WHILE IT'S FRESH IN YOUR MIND**

We use a JIRA workflow we developed to walk our team members through the entire incident report process, complete with target timeframes for each step.

- **MAKE SURE YOU DOCUMENT EVERYTHING IN YOUR KNOWLEDGE BASE**

We write all our incident reports in Confluence (and link to them from JIRA), so we can refer back to them for future similar incidents and ensure we keep getting smarter (and sharing the knowledge) along the way.

- **AUDIT YOUR RESULTS REGULARLY**

We run reports in JIRA to make sure our team is doing a good job of resolving incidents and of documenting the results.

By introducing better workflow and diagnosis tools and following a standardized approach to incident and problem management, we've reduced our mean-time-to-diagnosis from 113 minutes to just 23 minutes—and we're committed to cutting it even more.

It's Time for Your DevOps Story

In this ebook, we've given you a quick glimpse at how Atlassian does DevOps behind our own walls. We've profiled how our software development teams use continuous development practices, and how our Build Engineering team follows those very same guidelines to manage our hardware and configuration infrastructure with tremendous efficiency. We've looked at the Atlassian and third party tools both teams use to increase our throughput and quality, and we even looked at how standardized frameworks like ITIL help us to resolve incidents faster and more efficiently when issues inevitably arise at Atlassian.

But what about your story? We'd love to hear the different ways that DevOps is powering your business. The more unique, the better.

Submit your story

[Submit](#)

Explore our tools for DevOps

[Learn more](#)