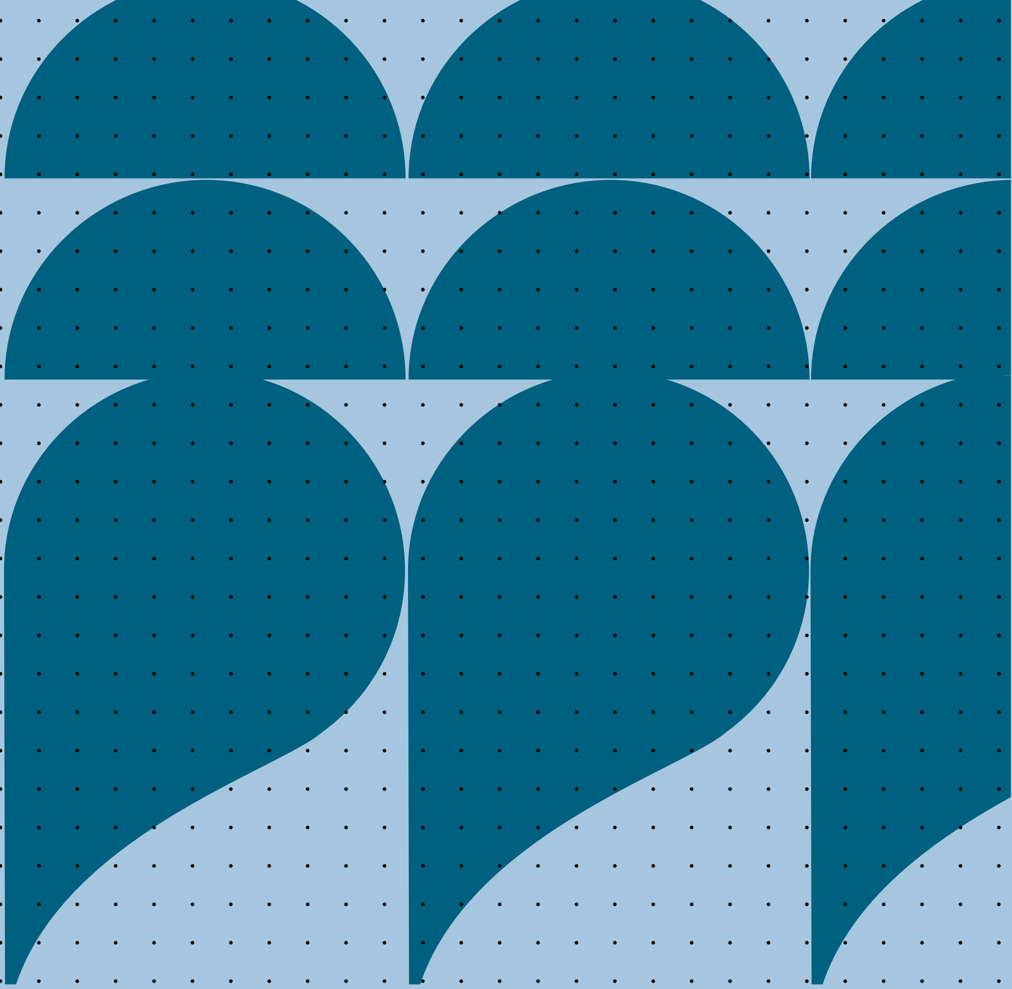
PagerDuty

Automation for Incident Remediation



Introduction

Automation has become a key competency for modern IT teams. The proliferation of platforms, tools, and contexts for developing, testing, and running applications creates a near infinite number of toolset combinations. Each tool requires expertise to use well, and that knowledge needs to be preserved, evaluated, and learned from over time. This process is difficult if all the processes are manual. While teams have become comfortable with the automation of certain steps in their workflows (like building software or performing some testing), teams are often slow to apply much automation to the operational part of the software delivery lifecycle.

Manual processes combined with complex operational environments have a number of negative potential consequences for technical teams:

- Fast-moving Agile development practices outpace documentation
- Mistakes are easy to make in manual processes
- Manual toil contributes to burnout and employee disengagement

Automating manual processes helps teams avoid these drawbacks, helping mitigate the risk, cost, and negative team health impacts of this type of work.

The negative impacts of manual processes follow applications into production environments and have outsized impact when something goes wrong. Incorrect documentation, copy-and-paste errors in manual processes, and repeated steps create risk when all systems are running as expected, but can have catastrophic impacts during an incident. Applying automation to incident response contributes to the overall consistency, predictability, and reliability of the response process.

If your team is not yet applying automation to your incident response processes, we hope this guide will help you think about and experiment with automating for incident response.

Automation and Its Use Cases in IT

With the spread of DevOps as a framework for overhauling IT workloads, as well as the increasing popularity of public clouds, automation of various work has become key to successful digital transformation. Cloud platforms offer opportunities for flexibility and developer autonomy, along with complexity, scale, and speed that is difficult to maintain with manual processes. As more and more technology teams find themselves at the intersection of traditional manual practices and modern API-driven practices, it is necessary to not only build the skills needed for automation, but also to embrace the mindset that automation is a benefit to the overall health of the systems.

Automation for IT processes has many of the same goals as automation in any other industry. We strive to deliver consistent, reliable results, no matter who the practitioner is, or which machine or service is involved. Automating repetitive, low-value tasks reduces toil and allows people to perform higher-value, more rewarding work.

Many teams will find that a variety of tools will help them alleviate the issues related to incorrect documentation, mistakes, and toil that plague teams using manual processes. As systems have increased in number and complexity, the need for automation has also increased. Safe, reliable automation gives IT teams a place to abstract their processes so they don't have to be memorized or exposed to the risk of manual work. Manual workflows like cutting-and-pasting carry heavy risks for mistakes; outdated documentation can slow down task completion or cause errors on its own; and the toil of repetitive manual tasks has negative impacts on employee productivity, satisfaction, and engagement.

Automation can be applied to many facets of a technical project, from building and testing software, to creating the runtime environments, to responding to error conditions and incidents. Various stages of the software development lifecycle include tasks that are candidates for automation, as well as tasks that are done repeatedly that require little or no input from humans, and have consistent end states. Creating and maintaining automation, whether in the form of scripts or libraries or other components, comes with a cost, so we want to find the tasks that will have the most impact on team resources when they are performed by the automation.

When we perform tasks or otherwise make changes to any part of a system, there are some goals we should keep in mind. In *Architecting for Scale*, Lee Atchison has categorized these goals as the following:

- The procedure should be testable
- The procedure should be flexible and implemented for future improvements
- The procedure should be reviewable by someone else as a check
- The procedure should be put under version control
- The procedure should be applicable to related resources, and "one-off" changes should be discouraged, and it should be applied to all related resources in the same way
- The procedure should be repeatable and auditable

While we can potentially satisfy these goals with manual processes, automation codifies the tasks into repeatable processes and ensures they are performed the same way each time they are needed.

If we keep these goals in mind when we are thinking about every change that will be made to our systems, we establish practices that prioritize reliability of our systems in all phases of the lifecycle, including when we are responding to incidents. Let's look at some ways your team is already using automation.

Build and Deploy

Building and packaging software applications is inherently a form of automation. Different types of languages and runtimes have different requirements, but the tasks are well defined and repetitive. Whether an application requires compilation and linking, or the organizing of external dependencies, the developer can employ automation to perform these tasks reliably. Application build can be run as part of a pipeline of processes, where the entire workflow is created and stored for repeated use, ensuring that steps aren't forgotten or run incorrectly.

For technical and organizational reasons, automated deployments are not as ubiquitous as automated builds, for technical and organizational reasons. For example:

- Organizations utilizing rigid change management programs are reluctant to authorize automated deployments of new builds to production, though they may deploy into shared development environments or a staging environment for testing and integration purposes.
- Even in loosely coupled environments, services may have some dependency on changes to data schemas that require more manual intervention, such as a full backup to be taken before the update.
- Automated deployment often comes at the end of a long modernization journey, after teams have successfully changed a number of other development practices, including preferring small, incremental changes and employing methods like feature flagging and dark launching.

Automated build and deployment are often combined into a set of practices known as *Continuous Integration/ Continuous Deployment* or CI/CD. The challenges presented by automated deployments have expanded the definition of CI/CD to also be defined as Continuous Delivery. Delivery places the software in a repository for later use, while deployment installs the software into a specific environment. Tools that perform CI/CD tasks help teams create pipelines or workflows to build, test, package, and deploy or deliver software.

Automated deployment isn't possible for all software. Software that wasn't designed for automated installs or upgrades might require the acceptance of a license agreement with no option to pre-populate metadata, settings file, or registry entry. It might still require the installer to make selections manually without the aid of a pre-populated inventory or other aid. These installs fail the goals of repeatability and are incredibly difficult to reproduce effectively, making them harder to repair or rebuild when needed.

Test

Automated testing is a topic all its own. It's beyond our scope here, but it's part of the application lifecycle so we've included it for completeness.

Various programming languages and frameworks have specific testing paradigms and tooling, but overall, manual testing is a tedious process. With the increasing complexity and volume of software components used in many organizations, manual testing is simply no longer possible or time efficient.

Automated testing can start from the first time a developer saves a file in their text editor, when the editor itself initiates a syntax checker or linter. As a change proceeds through the build process, more tests can be performed, from unit tests to integration tests, depending on the features or services impacted by the change. There's no chance that a test step will be forgotten if they are all automated.

Provisioning

The building and maintenance of environments on full systems, virtualized systems, and containers is another area that has been highly automated. For new projects, no one will ever put a warm coat on, walk into a highly air-conditioned data center, roll a "crash cart" with keyboard and mouse up to a rack of servers, and install the OS on them from a CD or DVD, clicking through options and package selections manually, like we did 10 or 15 years ago.

Building infrastructure is susceptible to the challenges we outlined in the introduction. Documentation of the process will rapidly become out of date, mistakes are easy to make, and the repeated tasks required to build out a large environment are tiresome. Automation for the base layers beneath your applications is key to maintaining large-scale production environments, but it is also important for the maintenance of smaller environments. Automating the provisioning of infrastructure ensures that:

- All the like systems in the environment are identical, even if they aren't built at the same time
- All the systems in the environment will have all the components required in the correct versions
- The systems can be rebuilt to the correct specifications in case of disaster or catastrophic loss

When services run in cloud or Infrastructure as a Service environments, the APIs provided by the vendor also give teams a strong starting point for automated management of services and systems. Tools like HashiCorp's Terraform, combined with configuration management tools like Chef, Puppet, or Ansible, provide an automated path from first provisioning through preparation for services to actively managing and maintaining services. This path for making planned changes is fairly well understood, and also has a set of testing tools like Test Kitchen and ansible-test to help achieve our goals for testing changes before they are applied to our environments.

Automation for Incident Remediation

Responding to incidents and alerts presents opportunities for automation in some environments as well. While we find automation in the creation steps of technical products, it is less common in the longer term maintenance of the systems. Automation improves the health of our running systems and can help us better manage the issues that arise. When considering automation for incident remediation, keep in mind that in complex systems, failure is inevitable. Our automation goals at this stage aren't to prevent failures, but to swiftly deal with failure when it happens and optimize for it as much as possible.

Creating monitoring and health-checks for production systems is fairly ubiquitous since organizations that don't count IT as a customer product space still heavily rely on IT services to be functional and performative. As long as all services are up and running, everything is fine. When something goes wrong, what happens next could be chaos or it could be a well-managed practice of contacting responders and remediating issues. Automation can be employed from the first blip or hiccup, including how the correct team is contacted, how they are able to respond, and whether there is additional infrastructure to support troubleshooting and remediation. More efficiency and the intentional use of automation in even these early phases of remediation reduce the time it takes to acknowledge and repair issues that arise.

Making changes safely is particularly important when a team is attempting to fix an incident, but unplanned changes made during troubleshooting are often made manually. When a service is unavailable or not performing in some way, making a mistake because of a manual process can delay the resolution. It could even make matters worse. If an incident responder makes a copy-and-paste error, or skips a step in a runbook, or executes a command in the incorrect terminal, any number of unpredictable things could happen. So we look to employ automation in our remediation processes.

We want automated remediation for many of the same reasons we want automation in general—as systems increase in number and complexity, the amount of information needed to run and maintain them effectively also increases. The decision to auto-remediate alarms from certain inputs might consist of several points:

- How often the alarm triggers. We reduce the noisiest alarms for the greatest gain.
- How often the alarm is non-impacting to end users at first instance. Early warnings like disk usage can be dealt with by automation.
- If the first step in a manual remediation is always the same for the alarm. If a human typically restarts a service to see if that fixed the issue, the automation should do that step.

Teams may find that their alarms have a consistent set of solutions that can be automated. Creating this automation, via any number of tools, removes these alerts from the immediate attention of the team, and lessens the potential for what we refer to as "alert fatigue."

Alert fatigue occurs in a number of industries where workers are exposed to alerts and alarms on a regular basis to the point where the alerts lose meaning. Large numbers of alarms, or alarms with high frequency, can cause responders to become desensitized over time. As responders become desensitized, their response times become longer and the potential for mistakes increases when there is an important alarm. We see this in IT when a preponderance of low-urgency alerts are passed to responders in real time, 24 hours a day, instead of being added to a work queue, delayed to working hours, or otherwise managed.

IT teams can deploy automation to combat the contributing factors to alert fatigue. While a dashboard may seem like a good idea since it will help eliminate the cacophony of beeps, chimes, chirps, and buzzes from alerts, a screen full of red status reports or flashing issues can be difficult to make use of in a timely manner. If using a dashboard, teams that categorize their alarms by severity and urgency can also categorize them as targets for future automation. When everything else has been mitigated by automated processes, the team will have more capacity to deal with the alerts that do need human attention.

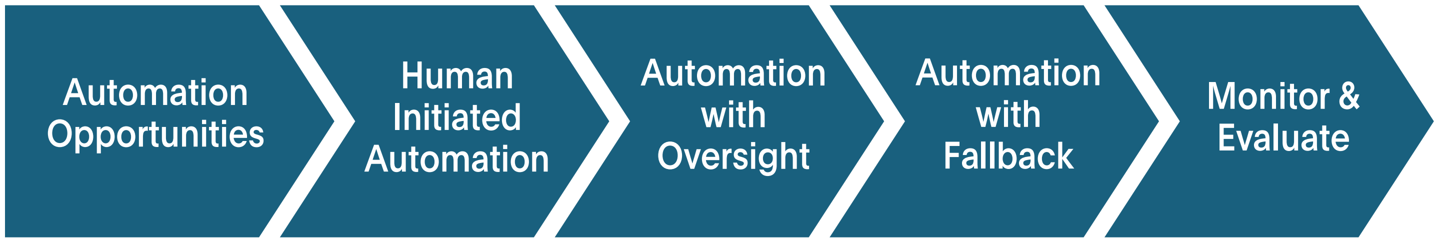
We also want to use automation when the solution should be faster than a human could be expected to perform the actions. This might include production activities like autoscaling when a service is under heavy load or prohibiting IP addresses that are repeatedly attempting a bad request. Depending on your use of IaaS platforms, you might already be making use of some of these functions that are built into the service.

Machines are faster than humans at some tasks, and they don't mind work that is boring and repetitive. As we build automation, we focus on the tasks with the most toil; i.e., those that require humans to do a lot of work, but work that is relatively low value. Those are the tasks that can be completed by automated processes. Automation will help a team respond to incidents in a predictable and defined way.

Your team may already be using documentation or guides like runbooks that prescribe the steps to take to remediate an issue. Where those runbooks can be performed by automation, fewer distractions and alerts will go to the human responders. Particularly for remediation tasks that are low value, like restarting services or clearing disk space, this work is better allocated to automation. The automation can then also be applied to multiple sets of similar systems.

Getting Started With Automated Remediation

We can take a multistage approach for implementing automated remediation in our production systems:



Our automation should be reliable and consistent, so as we build out our automation, we'll also want to keep these goals in mind:

- The procedure should be testable
- The procedure should be flexible and implemented for future improvements
- The procedure should be reviewable by someone else as a check
- The procedure should be put under version control
- The procedure should be applicable to related resources, and "one-off" changes should be discouraged, and it should be applied to all related resources in the same way
- The procedure should be repeatable and auditable

Start By Reducing Noise

Before even thinking about automating processes, take a long, hard look at the alerts being generated by your systems.

- Are there unactionable alerts?
- Are there alerts that are overly complex?
- Are there alerts that should be fixed in engineering?

Round 1 of automating incident response is to ensure the alerts that are coming through are useful and can be fixed in the production environment. You can find more guidance on creating useful alerts in our Ops Guide on Incident Response. Good alerts will contain an appropriate amount of useful information about the impacted system. They'll be rated in line with their impact on users. And they'll be something that can be remediated in production under normal conditions. If your systems generate alerts that can't be fixed via changes to the production environment, send them back to engineering. For example, when moving to a distributed services model in a cloud, you might see a need to increase the timeouts for requests to remote services. This is an expected performance tradeoff for the architectural change, and the timeout for those types of requests often needs to be increased.

Identify Initial Automation Candidates

Once you have the alerts cleaned up, take a look at the data you have on the number of alerts that fire, when they occur, and what their impacts are. This will give you candidates that will have the most impact on your responders when you create automated remediation. Create a list of potential alerts that can be automatically remediated based on their volume or simplicity. Potential candidates for your first round of automation might be alerts for single subsystem issues, like disk space warnings, or alerts that already have a manual runbook that can be automated.

The Automate, Observe, Refine Cycle

Your first automated remediation targets should be well defined and well contained. Part of building up trust in your automation tools will come from creating cumulative successes, so start with a small collection of alerts to automate. Keeping the first set all within a single team of responders will help with training and communication.

Use your data set to determine the performance of your automation efforts. Is the team seeing a reduction in alerts? Are incidents still getting resolved in a timely and correct manner? Has there been any negative impact to the customer? Have you reduced the amount of toil the team is required to do on a daily basis to support the services?

You might want your team to implement automation in phases, allowing the automation to run but still alerting a human responder as a check. Even before that, you can build trust in the automation by alerting a team member and having that person initiate the automation process. Over time, this ensures that the automation runs as expected, but also gives the team background knowledge of what the automation is intended to do. Working with unfamiliar automation can have negative impacts for responder teams who aren't sure what behaviors might have triggered the automation and what side effects are of the automation itself.

Some teams might also want a place for the automation to report a status for later tracking and trend determination. For example, your automation might be clearing unused files out of a cache directory to clear disk space, but if this starts happening more and more often, your team will want to engage and find the underlying cause. The automation can only do so much.

This is a good point to report your efforts to other teams to highlight what you've learned about the process and how the automation is making the operation of systems and services better.

Maintaining Automation

The addition of an automation component to your production incident response will require tracking for updates when the services they work on are updated. Downstream activities might be impacted by changes to things like service names or command options. Remediation automation components, if they meet the goals mentioned above, will be testable and checked into version control. They travel the software development lifecycle with the services that they support, either in their own repository or in the project repository. Make sure you have a plan for how they are updated, tested, and released when new versions of your services are deployed.

Challenges of Automation

Automation has quite a history, starting in the early days of the Industrial Revolution. When weavers and textile workers were faced with the realities of new machinery, they worried that their hard-won skills and craftsmanship would no longer be needed, and some banded together to destroy machines. We still use their name, Luddites, to refer to people who are reluctant to use or are hesitant around technology. For more than 200 years, processes and human work have become increasingly automated in many industries, from electrical power generating, to aerospace, to shipping, to IT.

Making use of automation creates its own set of challenges. Researchers in human behavior, human-computer interaction, neuropsychology and other intersecting fields have been looking at the effects of automation on workers for decades.

While not all the findings in these areas are applicable to IT, there are a few that we might find interesting; in particular, a set of "ironies" presented by Lisanne Bainbridge in a 1983 paper entitled *Ironies of Automation*. This paper is short, only a few pages, but it has had a significant impact on the field of automation research. Automation research often focuses on industries in which failures create spectacular or catastrophic results, such as aviation, power generation, or healthcare. The downstream learnings can help us work with the automation in our systems as well.

The Ironies, as presented by Bainbridge, focus on the impact automation has on the behavior and readiness of human operators. Two in particular are related to how the operator feels about their engagement with their work. In a heavily automated system, a human operator may not feel ownership of the performance of the system. Their detachment from many of the activities the system performs also contributes to a question of the value of their work. An operator watching automated processes for feedback can find that work incredibly boring, but the level of responsibility on that operator if they are needed by an anomalous situation creates an odd dichotomy. This also contributes to a problematic balance between boredom and vigilance that is still being studied, especially in the field of autonomous vehicles.

One place that might impact your team's overall |performance after deploying automation at scale is the potential degradation of skill of the operator. Bainbridge discusses this a bit, but a follow-up paper by Barry Strauch in 2017, *Ironies of Automation, Still Unresolved After All These Years*, investigates in more depth, especially in relation to aviation. Operators stepping into an incident in the middle of some automated action may not have the knowledge required to solve the problem or may not have a full accounting of what the automation has already done. In the worst cases, the operators may have never had sufficient training to repair a system that has had some kind of automation failure because they trained only with the automation in place.

When we look to apply this foundational idea to IT automation, we might find that operators become rusty over time when they aren't constantly exposed to the system. We ask the folks responding to an incident to be **more** knowledgeable about the system than they were before the automation—they will only be stepping in when the incident is more complex than the automation can take care of. At PagerDuty, senior staff may be well prepared for this kind of lifecycle, but it presents new challenges when training junior staff.

Catastrophic failures attributed to automation, as they get reported in the press, also make our automation tasks more difficult from an organizational standpoint. When folks express reluctance or trepidation around automation, it's often down to several common concerns:

- Fear of the automation creating a larger problem
- Fear of the automation just being wrong or doing nothing
- Fear of losing their job

Any unplanned work on a system might cause more errors or have no positive effect on an alert, whether a human is responding or the system has an automated response. While no effect is better than increasing the scope of a problem, it also delays the time to recover from the incident.

PagerDuty

Part of keeping the automation “trustworthy” is maintaining a narrow functional scope. This contains the potential *blast radius* if anything does go wrong. The automation we build should focus on one particular function and should report back if something doesn’t work as expected.

Think about our earlier example of installing a software package. There are a couple of checks we’d want to employ inside the automation, such as not continuing if the package can’t be accessed. We also want to keep the focus narrow; the installation of the package doesn’t necessarily mean we also want the service started immediately on all platforms, so we might leave that out of the automation component. Similarly for tasks like clearing disk space. Our human responders might first log into the system and double check that the “usual” culprit is responsible for the issue before taking any action. Our automation should also run these commands and act accordingly.

Job loss or recategorization is a real issue in organizations applying automation to many parts of their workflow. As with increased automation in many industries, the types of jobs that are available will change. If your team chooses to use more cloud-based infrastructure via automation and APIs, you’ll have less need for someone on your team to spend their day performing tasks like imaging machines or building virtual hosts. These are repetitive tasks more [aptly] performed by automation; we want to save the human time for solving problems. So part of your automation journey will likely require you to spend some time and attention on helping some of your staff retrain and learn new tasks.

It’s unlikely, as you automate, that you’ll have less work to do. Once organizations start reaping the benefits of faster time to market and higher quality output, their ability to innovate also increases. More innovation creates more projects for your technical teams to work on as well.

Conclusion

Deploying automation helps your team reduce toil and creates space for innovation that would otherwise be used to respond to unplanned work.

Our automation needs to fit snugly into the application lifecycle; in some cases it might be completely bespoke, in other cases it might be something off the shelf. But it should be treated in the same way we treat the development of the application, undergoing testing and employing good practices. When we approach the operational life of our applications as part of the features of the applications, creating and maintaining automation components becomes part of the application development process itself.

References and Further Reading

Allspaw, John. *Taking Human Performance Seriously in Software*. Monitorama Conference, 2019. https://vimeo.com/341144396

Bainbridge, Lisanne. "Ironies of Automation". *Automatica*, Vol. 19 No. 6 pp 775-779, 1983. International Federation of Automated Control.

Forsgren, Nicole, PhD. Gene Kim. Jez Humble. Accelerate: *The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. Portland: IT Revolution Press, 2018.

Gene Kim, Jez Humble, Patrick Debois, John Willis. *The DevOps Handbook*. Portland: IT Revolution Press, 2016.

Perrow, Charles. Normal Accidents: Living with High-Risk Technologies. Princeton, NJ, Princeton University Press, 1999.

Strauch, Barry. "Ironies of Automation: Still Unresolved After All These Years". *IEEE Transactions on Human-Machine Systems*. August 2017. Pp 1-15.

PagerDuty