New Relic.

White Paper

Reducing MTTR the Right Way

Best practices for fast incident resolution

Introduction

MTTR—or mean time to resolution—is one of the most widely used metrics in the systems reliability toolbox. Paradoxically, it's also one of the most misunderstood metrics; many developers and operations teams lack a clear vision for how to define MTTR, how to use it, and how to improve it in a consistent and sustainable way.

As modern organizations increasingly rely on software to run their businesses, these disconnects around MTTR aren't just inconvenient; they threaten the bottom line by potentially disrupting the increasingly important digital customer experience, not to mention adding significant cost, risk, and complexity to the software development process.

The key to avoiding these problems is to adopt a progressive approach to defining and applying MTTR—one that combines comprehensive instrumentation and monitoring; a robust and reliable incident-response process; and a team that understands how and why to use MTTR to maximize application availability and performance. To help you do that, New Relic has collected 10 best practices for reducing MTTR the right way—all in the context of building a healthy incident-response strategy. We'll also explain how the New Relic platform supports a DevOps team's MTTR-reduction goals in a number of very important ways.

The many meanings of MTTR

The challenges around MTTR begin with how to define the term. Its original, Industrial Age meaning addressed machine maintenance: mean time to repair referred to the time required to fix a piece of machinery, a device, or a facility's physical plant.

In today's software-defined world, MTTR commonly stands for "mean time to restore" or "mean time to resolution." While the distinction may seem trivial, these definitions focus on different outcomes and reflect two very different approaches to dealing with software performance issues:

- **Mean time to restore** (sometimes called mean time to recovery) is the digital equivalent of mean time to repair: the time required to get an application back into production following a performance issue or downtime incident.
- **Mean time to resolution**, on the other hand, focuses more on the big picture: It addresses the time required to fix a problem and to implement subsequent "clean ups" or proactive steps designed to keep the problem from recurring. A team must address both of these tasks before it can declare an issue resolved.

This second, more comprehensive, definition of MTTR demands the ability to find and fix the underlying causes of performance issues. It deliberately avoids quick-fix solutions that may solve the immediate problem—but that fail to prevent the problem from happening again.

MTTR is a statistic—not a silver bullet

Remember that MTTR is a mean, or average, statistical value. Learning how to use it effectively means also learning its limitations:

MTTR is more useful as a "turnkey" metric when incidents (and resolutions) tend to be similar. If you often deal with outlier incidents that vary widely in terms of the time required to resolve them, MTTR on its own may present a distorted view of your true response capabilities.

Similarly, it's easy to forget that MTTR (perhaps surprisingly) doesn't take time into account. An MTTR calculation, for example, can't capture the critical difference between incidents that impact your systems and applications during peak times versus those that happen during quiet times.

Neither of these points negates the value of MTTR, but it's important to think about the downside of relying too much on a single metric.

Two best practices for using MTTR can go a long way towards minimizing these risks:

1. **Use MTTR in conjunction with other metrics to tell a more complete, accurate, and nuanced story of application and infrastructure performance.** An "error budget" metric is one way to accomplish this. An error budget may specify, for example, that one minute during peak time is equal to one hour during non-peak time. Used alongside MTTR, an error-budget metric helps you understand the true cost and impact of downtime, and therefore understand the value of gains or losses in your MTTR trends.
2. **Validate your efforts to reduce MTTR by focusing on resolving incidents and maximizing availability.** As noted, resolving an application failure means getting to the bottom of a problem—finding the responsible service or infrastructure component, and coming up with a durable solution. That's often significantly different than a quick fix designed to address the immediate issue, but not to uncover the true source of an incident or all of its contributing factors.

Thinking in terms of availability helps to keep the focus on finding durable, long-term solutions to application performance problems. Availability puts a premium on preventing recurring issues and on thinking about how an issue impacts your digital customer experience—even if this approach yields solutions that are more costly and take longer to implement than a short-term fix.

Incident response: your key to a winning MTTR strategy

Once you understand the ground rules for using MTTR wisely, the next step is to learn how modern DevOps teams improve their incident resolution times—and how they lock in those improvements once they achieve them?

Typically, DevOps teams need three core capabilities:

1. Proactive anomaly detection, quick incident response and resolution, and workflows that accelerate the remediation process
2. A unified source of multi-dimensional insights into application and infrastructure health and performance that provides context about the entire software system
3. A commitment to move the needle on long-term service reliability—increasing the odds that when a problem gets fixed, it stays fixed

Technology plays an essential role in achieving these capabilities, but technology alone isn't enough to win this battle. You need a strategy that combines technology, rock-solid resolution processes, and your team members' individual talents and skill sets.

Your **incident response process** is the place where all of these elements converge. Incident response covers every point in a chain of events that begins with the discovery of an application or infrastructure performance issue, and that ends with learning as much as possible about how to prevent issues from happening again. It covers every aspect of a solid strategy for reducing MTTR, and it ensures that your team can continue to improve even as your business and applications scale.

The keys to creating a best-in-class incident response process

Talking about incident response begins with a question: What defines an incident? The answer is essential, since the incident-response clock starts to tick the moment an incident is discovered.

New Relic defines an incident as an application or infrastructure performance issue that meets three criteria:

1. **It's high stakes to the business.** Incidents impact customers, directly or indirectly.
2. **It's urgent.** Incidents must be resolved immediately. During an incident, teams are solving a problem under time pressure, and they have to function differently than their day-to-day engineering activities.
3. **It involves collaboration.** Significant incidents typically require collaboration between multiple people, often from several parts of an organization. Coordinating everyone during a high-stakes, high-pressure event requires a particular kind of management response—and it can incur costs that should be accounted for in the teams' budget.

Based on how New Relic deals with incidents, these 10 best practices are designed to help teams reduce MTTR by helping you step up your incident response game:

1. Create a robust incident-management action plan

At the most basic level, teams need a clear escalation policy that explains what to do if something breaks: whom to call, how to document what's happening, and how to set things in motion to solve the problem.

Most organizations choose from three common incident-management strategies:

1. **Ad hoc.** Younger, smaller companies typically use this approach. When an incident occurs, the team figures out who knows that technology or system best and assigns a resource to fix it. There's not a lot of structure—and that's exactly the point.
2. **Rigid.** This is the traditional information technology systems management (ITSM) approach often used by larger, more conventionally organized enterprises. IT is generally in charge of incident management in this kind of structure. Change management concerns are paramount, and teams must follow very strict procedures and protocols. In this case, structure isn't a burden—it's a benefit.
3. **Fluid.** This is the approach that many modern companies—especially companies that have undergone digital transformations—take. Responses are shaped to the specific nature of individual incidents, and they involve significant cross-functional collaboration and training to solve problems more efficiently. This approach is based on Lean principles of constant experimentation and learning, so response processes evolve continuously over time.

The fluid approach is typically favored by modern software companies, unless there's a specific reason to use either the rigid or ad hoc models. A fluid incident response model allows companies to marshal the right resources and to call upon team members with the right skills to address situations in which it's often hard to know at first exactly what is happening—or what capabilities might be necessary to solve the problem.

As teams learn more about a problem, a fluid approach also makes it easier to come up with creative solutions to challenging new problems.

2. Define roles in your incident-management command structure

At New Relic, we designate an incident commander to play a key role at every step of the incident-response process, acting as a centralized point of leadership during incidents and helping teams make essential trade-offs during the response process. The incident commander is responsible for directing both the engineering and communication responses (the latter involves engaging with customers, both to gather information and to pass along updates about the incident and our response to it). The incident commander makes sure that the right people are aware of the issue.

Some companies assign semi-permanent incident commander roles, while others rotate staff members into the role. At New Relic, any engineer can be an incident commander, since the first person to respond to a customer-impacting alert will typically assume that role.

Each incident may also require a **technical lead** and a **communications lead,** each of whom reports to the incident commander. The technical lead, not the incident commander, typically dictates the specific technical response to a given incident. In some cases, you may need more than one technical lead, depending on how many systems are impacted. The technical lead should be an expert on the system(s) involved in an incident, allow them to make informed decisions and to assess possible solutions, so they can speed resolution and optimize the team's MTTR performance.

The communications lead usually hails from the customer service team. This person understands the likely impact on customers and shares these insights with the incident commander. At the same time, as information flows in the opposite direction, the communications lead decides the best way to keep customers informed of the efforts to resolve the incident.

3. Train the entire team on different roles and functions

To maximize the benefits of a fluid model, it makes sense to invest in cross-training for engineers so they can assume multiple incident-response roles and functions. There's always a need for specialists on specific systems and technologies, but relying too heavily on a handful of specialists is a recipe for burnout—and staff turnover. Other members of the team should build enough expertise to address most issues, allowing your specialists to focus on the most difficult and urgent incidents. Comprehensive "runbooks" (see best practice #7) can be a great resource for gathering and transferring specialized technical knowledge within your engineering team.

Cross-training and knowledge transfer also helps you to avoid one of the most dangerous incident-response risks: situations in which one person is the only source of knowledge for a particular system or technology. If that person goes on vacation or abruptly leaves the organization, critical systems can turn into black boxes that nobody on the team has the skills or knowledge to fix.

Look around your engineering organizations, assess your dependencies on specific individuals, and put redundancies in place to eliminate these knowledge bottlenecks—just as you do with your systems resources.

4. Monitor, monitor, monitor

It's a truism that you can't fix something if you don't know it's broken. Getting proper visibility into your applications and infrastructure will make or break any incident response process.

Consider an example of a troubleshooting process without monitoring data: a server hosting a critical database or application goes down, and the only "data" available to diagnose the problem is the lack of a power light on the front of the server. The response team is forced to diagnose and solve the problem with a heavy dose of guesswork—likely leading to a long and costly repair process, and almost certainly an unacceptably long MTTR.

Compare this scenario with one where real-time monitoring data flows from the application, server and related infrastructure—giving the team an accurate read on server load, memory and storage usage, response times, and other metrics. The team can formulate a theory about what is causing a problem and how to fix it using hard facts rather than guesswork.

In addition, teams can use monitoring data to assess the impact of a solution as it's being applied, and to move quickly from diagnosing to resolving an incident. This is a powerful one-two combination, making monitoring perhaps the single most important way to promote an efficient and effective incident-resolution process and reduce MTTR.

5. Leverage AIOps capabilities to detect, diagnose, and resolve incidents faster

A new set of technologies has emerged in the past few years that enables on-call teams to harness AI and machine learning (ML) capabilities so they can prevent more incidents and respond to them faster. Gartner coined the term "AIOps" (Artificial Intelligence for IT Operations) to describe this category. AIOps uses AI and machine learning to analyze data generated by software systems in order to predict possible problems, determine the root causes, and drive automation to fix them.

AIOps complements your monitoring practices by providing an intelligent feed of incident information alongside your telemetry data. When you use that information to analyze and take action on that data, you'll be better prepared for troubleshooting and incident resolution. High-performing DevOps and SRE teams use AIOps capabilities to respond to incidents quickly and increase their mean time between failures.

AIOps can help in four main ways:

1. **Proactive detection of anomalies** before an issue hits production or impacts customer experience or SLOs
2. **Noise reduction** to help teams prioritize alerts and focus on the issues that matter most, by correlating related incidents and enriching them with metadata and context
3. **Intelligent alerting and escalation** to automatically route incidents to the individuals or teams best equipped to respond
4. **Automated incident remediation**, which includes workflows to resolve the incident when it occurs and reduce MTTR

6. Carefully calibrate your alerting tools

With all the monitoring tools available today, it's possible to have too much information about your systems, which can make it difficult to develop a clear plan for how to use the data. This is where programmatic alerting becomes essential.

A practical first step is to set alerts in the form of thresholds for service level indicators (SLIs). These are simple metrics or thresholds you can track with automated monitoring tools, and which indicate when a serious problem might be happening or is about to happen. You might say, "If throughput drops below this threshold, that indicates that something is wrong somewhere in the system." Or "If latency spikes for more than this amount of time, then we need to look into it." It's basically ways to quantify whether your system is healthy or not.

For example, even if team members don't know all the intricacies of a downstream customer-facing database, they can monitor thresholds and learn that a problem may be about to occur. When a system reaches a SLI threshold, it can ping engineers to address the potential incident before customer calls and tweets start pouring in. Just make sure to tune your alert thresholds properly to avoid alert fatigue—no one likes getting woken up in the middle of the night unless it's absolutely necessary.

Also, choose an alerting tool that offers muting rules, so you can take more control over your alerts and suppress notifications during times of known system disruptions, such as maintenance windows, deployments, and during testing.

7. Create runbooks

As you develop incident response procedures and establish monitoring and alerting practices, be sure to document everything. Write everything down and use these notes to create "runbooks"—documentation that tells on-call responders exactly what to do when a specific problem occurs.

Use runbooks to collect your team's "tribal knowledge" about a given incident-response scenario in one document. In addition to helping you reduce MTTR, runbooks are useful for training new team members, and they're especially helpful when important members of the team leave the organization.

Keep in mind that a runbook won't cover every scenario or provide a "recipe" to fix every problem. There are simply too many variables and too many unique events to cover every possibility. The idea is to use a runbook as a starting point—saving time and energy when dealing with known issues, and allowing the team to focus on the most challenging and unique aspects of a problem.

8. Follow up on incidents to understand how and why they occurred

Whether you call it a postmortem, an incident retrospective, or a day-after analysis, part of reducing MTTR involves a strong incident follow-up procedure. This is when you investigate what happened, figure out how it happened, identify the triggering event and likely causes, and strategize ways to prevent the problem from cropping up again.

In addition to holding blameless retrospectives, you may want to institute a don't-repeat-incidents (DRI) process. The DRI model involves stopping new work on a service involved in an incident until you fix or mitigate the causes. It reinforces the commitment to resolving issues rather than accepting short-term fixes, helping teams remain fully accountable for closing the loop on the incident resolution process. It reminds everyone that quality is an imperative—not an option.

9. Practice failure through chaos engineering

Chaos engineering is the practice of attempting to randomly inject problems into your systems—in a highly controlled way, of course—so you can test how resilient your system is. You can use chaos engineering to address such critical questions as:

- Did the system fail in the way you expected?
- Were you able to fix it promptly?
- What did the monitoring data look like?
- How long did it take for the service to be available again?

Chaos engineering can benefit your organization in multiple ways. First, teams learn where they may have opportunities to make systems more available and more resilient. Chaos engineering exercises can also double as incident-response dress rehearsals—allowing you to test your processes, escalations, policies, monitoring and alerting, and other elements. By removing the friction from actual incident-response situations, such rehearsals can have a direct impact on your MTTR performance.

10. Focus on the correct fix—not just the fastest one

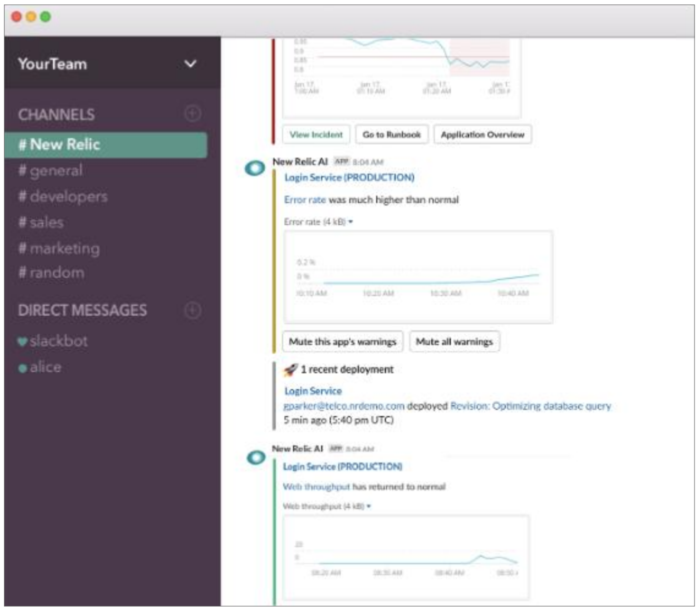
It can be difficult in the heat of the moment, but resist the urge to take shortcuts to achieve quick and easy, but often illusory, MTTR reductions. MTTR is an average that incorporates all your incident response times. Today's "quick and dirty" fix could contribute to a major, and completely avoidable, issue in the future. Tackle the underlying causes of a performance issue now, and you'll help ensure that it doesn't come back to haunt you.

The New Relic platform: Tools that elevate your incident response game

The 10 best practices above can help you adopt and internalize an approach to MTTR based on the principles of incident response and availability. And the New Relic observability platform can be a key to successfully adopting this approach.

The New Relic platform offers monitoring, AIOps, alerting, incident diagnosis, and other capabilities that contribute directly to faster, smarter, more efficient incident response; driving significant improvements to MTTR and other performance metrics.

Our comprehensive AIOps capabilities, which we call New Relic AI, empower your team with intelligence and automation to find, troubleshoot, and resolve problems faster. New Relic AI enhances the detection process, automatically surfacing anomalies across multiple tools in your stack and suggesting actions to monitor similar conditions in the future. Best of all, New Relic AI can deliver anomaly information to you via Slack, enabling you to quickly and collaboratively assess potential problems.

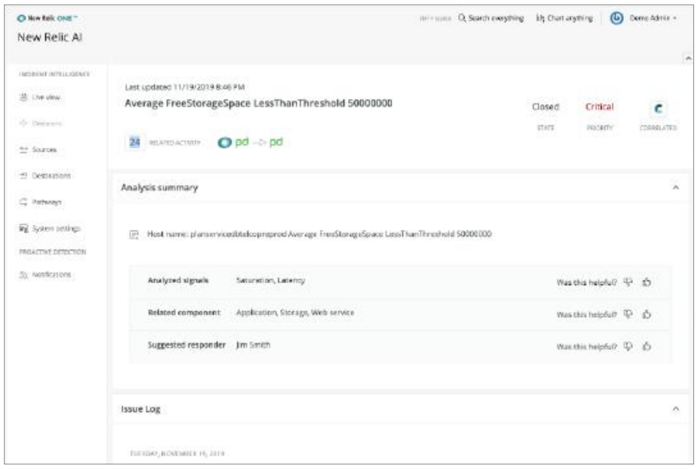


New Relic AI APP 8:04 AM
Login Service (PRODUCTION)
Error rate was much higher than normal
Error rate (4 kB)

1 recent deployment
Login Service
gparker@telco.nrdemo.com deployed Revision: Optimizing database query
5 min ago (5:40 pm UTC)

New Relic AI APP 8:04 AM
Login Service (PRODUCTION)
Web throughput has returned to normal
Web throughput (4 kB)

But we don't just stop at detecting incidents. New Relic AI uses a baseline of industry-standard knowledge, and then learns from your data and your team's feedback to intelligently suppress alerts you don't care about. New Relic AI also correlates related incidents, enriching those incidents with valuable metadata and context to help you diagnose issues faster. You'll also get useful context about your existing issues, including their classification based on the "Four Golden Signals" (latency, traffic, errors, and saturation) and correlated issues from across your environment. Finally, New Relic AI can even suggests responders and individuals on your team best equipped to handle a specific incident, so you can reduce the number of noisy alerts sent to the wrong people.



Last updated 11/19/2019 8:46 PM

Average FreeStorageSpace LessThanThreshold 50000000

Closed	Critical	
STATE	PRIORITY	CORRELATED

24 RELATED ACTIVITY pd → pd

Analysis summary

Host name: planservicedbtelcoprepod Average FreeStorageSpace LessThanThreshold 50000000

Analyzed signals	Saturation, Latency	Was this helpful?
Related component	Application, Storage, Web service	Was this helpful?
Suggested responder	Jim Smith	Was this helpful?

Issue Log

TUESDAY, NOVEMBER 19, 2019

Tools that keep your response team fresh, focused, and efficient

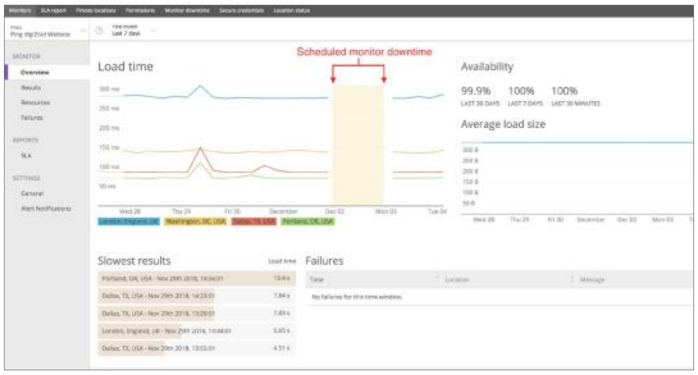
Your ability to alert the right people—quickly and efficiently, using accurate and actionable performance insights—can make or break your incident response strategy. New Relic's full-stack, programmatic alerting puts these capabilities, and more, at team members' fingertips.

By defining alert conditions based on the results of custom New Relic Query Language (NRQL) queries, for example, your team can evolve alerts tied to specific, high-load system calls. Performance issues at these points can provide leading indicators of a problem even before it impacts production applications—giving your team the opportunity to find and fix problems before they lead to downtime, lost revenue, and customer complaints.

New Relic Alerts also helps to prevent alert fatigue—a growing problem for incident-response teams operating in microservices environments. Flexible alert policies and notification-channel options give teams greater control over the flow of alert-incident data, while minimizing "noise" due to redundant alert conditions.

Tools that assess end-to-end system performance

In addition, operations teams can use New Relic Synthetics to close a critical blind spot for many DevOps teams: monitoring and understanding end-to-end system behavior. Synthetics gives organizations a range of options to measure endpoint performance—from sending a simple ping command to in-depth monitors that run scripts to simulate complex scenarios. Synthetics also supports the use of containerized private minions to monitor internal sites and expand geographic coverage—raising the bar on security, cloud-readiness, and flexibility.



Availability		
99.9%	100%	100%
LAST 30 DAYS	LAST 7 DAYS	LAST 30 MINUTES

Slowest results

	Load time
Portland, OR, USA - Nov 29th 2018, 14:34:01	13.4 s
Dallas, TX, USA - Nov 29th 2018, 14:23:01	7.84 s
Dallas, TX, USA - Nov 29th 2018, 13:28:01	7.49 s
London, England, UK - Nov 29th 2018, 14:44:01	5.65 s
Dallas, TX, USA - Nov 29th 2018, 13:55:01	4.51 s

Failures

Time	Location	Message
No failures for this time window.		

Tools that add user experience insights

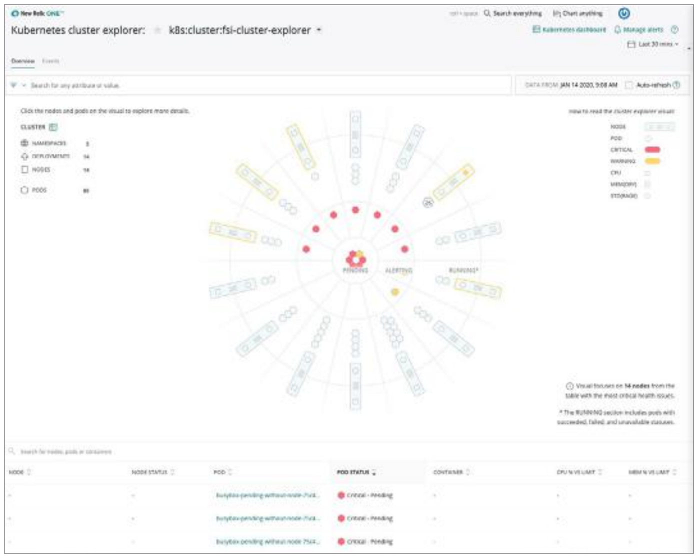
In many cases, a fast and successful incident resolution requires the ability to adopt a user's point of view—whether to understand how an incident impacts user experience or to assess the impact of user interactions. New Relic Browser achieves this goal by offering deep visibility and insight into how users are interacting with an application or website. Browser goes far beyond page-load timing to address the entire life cycle of a page—from individual session performance and AJAX requests to JavaScript errors and monitoring of single-page application architectures.

Browser also helps responders to understand the role that geography plays during an incident: filtering performance metrics and Apdex scores by global region or state, for example; and maintaining URL-segment whitelists and domain-specific blocking or monitoring.

Tools that combat complexity and simplify troubleshooting

Finally, New Relic helps organizations to deal with the growing complexity of modern, distributed microservices environments. Complexity is the price we pay for reaping the benefits of microservices, but it's also a major barrier to creating a fast and efficient incident resolution process.

The New Relic Kubernetes cluster explorer is a prime example of how New Relic helps to give a team clarity and visibility into highly complex systems—even at massive scale. Kubernetes cluster explorer provides a multi-dimensional representation of a Kubernetes cluster that lets you zoom into your namespaces, deployments, nodes, pods, containers, and applications. The cluster explorer lets you retrieve the data and metadata of these elements easily, and to understand how they are related with the help of highly intuitive visualization tools.


New Relic ONE
Search everything
Chart anything
Kubernetes cluster explorer: k8s:cluster:fsi-cluster-explorer
Kubernetes dashboard
Manage alerts
Last 30 mins
Overview
Events
Search for any attribute or value
DATA FROM JAN 14 2020, 9:08 AM
Auto-refresh
Click the nodes and pods on the visual to explore more details.
CLUSTER
NAMESPACES
DEPLOYMENTS
NODES
PODS
How to read the cluster explorer visual
NODE
POD
CRITICAL
WARNING
CPU
MEMORY
STORAGE
PENDING
ALERTING
RUNNING*
Visual focuses on 14 nodes from the table with the most critical health issues.
* The RUNNING section includes pods with succeeded, failed, and unavailable statuses.
Search for nodes, pods or containers
NODE
NODE STATUS
POD
POD STATUS
CONTAINER
CPU % VS LIMIT
MEM % VS LIMIT
Critical · Pending

Also, by moving effortlessly between high-level and highly detailed views, Kubernetes cluster explorer gives every stakeholder in the process a single, shared point of reference for troubleshooting and understanding the health of a cluster. This can accelerate your resolution process by getting everybody on the same page, and eliminating needless finger-pointing and miscommunication.

The distributed tracing capabilities in New Relic APM also help to combat complexity—in this case, the problems that arise tracing the cause of latency and other performance issues in distributed application architectures. Distributed tracing allows a team to trace the path of a request as it travels across a complex system; it reveals the latency of components along that path; and it shows which component in the path is creating a bottleneck.

Distributed tracing also leverages the intelligence built into the New Relic platform—using tools like anomalous span detection, trace charts, and custom queries of distributed trace data to help you isolate, diagnose and troubleshoot problems quickly and with confidence.

Taking the right approach to MTTR can be a complex and challenging task—that's the reality of working with modern application architectures. But with all of these capabilities (and many others), the New Relic platform is an essential tool to help you implement a faster, smoother, and more reliable incident-resolution process.

Conclusion: Remember the formula for long-term success with MTTR

Finally, keep in mind that MTTR is important, but hardly the only metric for measuring incident response. Sure, you want to minimize time to resolution, but don't spend countless people-hours trying to optimize resolution time and focusing too much on short-term results.

Instead, put tools like New Relic in place that feed you a continuous stream of real-time data, coordinated with carefully calibrated alert policies, and then use those tools to support a robust incident-management process. That's the best formula for systematically and efficiently resolving incidents. It's also the best way to continuously improve your efforts to reduce MTTR —delivering long-term, sustainable gains in application availability.

Deliver more perfect software

Try New Relic One today and start building better, more resilient software experiences. Learn More