Your Builds, Unmasked:

How to Drive Observability Across Your Dev Team

INCREDIBUILD

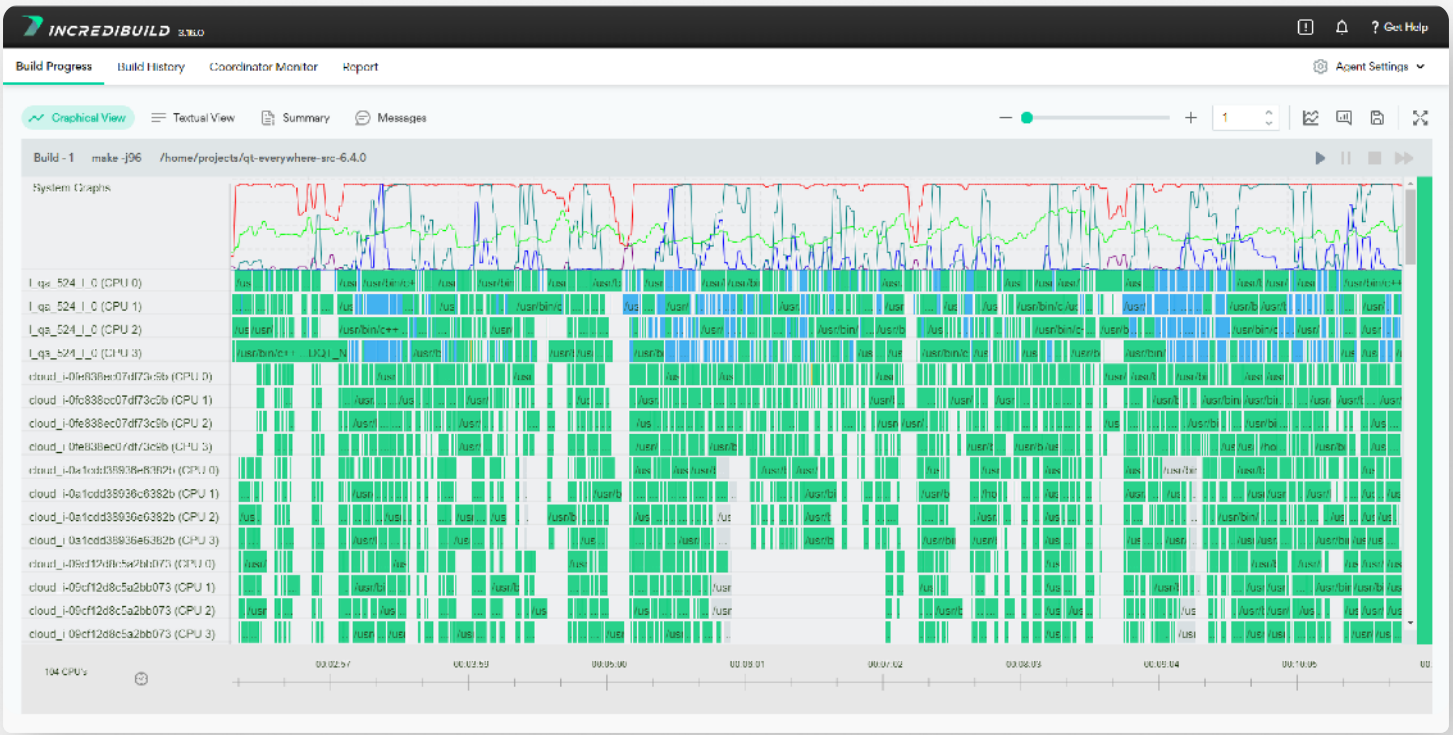
Introduction

Your latest build is off your desk and ready to go out into the world.
It's tempting to think that the hard part is over. Your work might not be "done," but you've certainly dealt with the "most important" parts of the process — designing, building, testing, making adjustments, and testing again until your creation is ready to see daylight.

At this stage, it's easy to overlook one of the final steps of the dev and DevOps cycle: integrating build observability tools.

With dev teams under such intense pressure to deliver bigger, better applications faster than ever, it's not surprising that many dev teams focus their attention - and their budget - elsewhere.
But build observability tools are becoming increasingly necessary for dev teams.

While IT observability - in other words, the ability to monitor the performance of applications throughout the development lifecycle - is important, build observability is arguably one of the most crucial types of observability.



"Incredibuild's Build Monitor in action"

Why?
Because it allows DevOps teams and individual developers to:

 Optimize development lifecycles

 Spot system and code issues faster

 Fix problems more rapidly

 Use past mistakes to improve future builds - and avoid making the same mistake twice

It's no wonder that Gartner predicts 70% of all organizations who build an effective observability practice will "achieve shorter latency for decision making, enabling competitive advantage for target business or IT processes."

So what's stopping dev teams from investing in build observability tools?

The real problem is that there is no one-size-fits-all way to improve the observability of your builds. The perfect solution will depend on your team's maturity; the size, type, and complexity of your codebases and builds; your budget; and even the regulatory environment in which you are working.

To truly reap the benefits of observability, you'll need a strategy. A strategy that is perfectly tailored to your current needs, but that can also evolve as your team matures and your codebases or builds become more complex.

In this guide, we'll break down how to go about creating an effective observability strategy that's tailored to your people, your processes, and your tools.

So you can choose an observability tool that's as impactful and carefully honed as the rest of your DevOps processes — and reap the benefits.

What's the difference between IT observability and build observability?

IT observability refers to your ability to monitor, analyze, understand, and optimize your entire software development lifecycle - from planning to modeling, testing, and deployment.

Build observability is just one part of general IT observability. Instead of helping you monitor your entire system, build observability allows you to examine your builds and understand how they are performing once they are in production — both in real time, and over longer periods.

It covers a wide range of insights:

How long did your build take?

Why did your build fail? Who made the changes that broke it, and what exactly did they do?

How often does your build fail?

What is the status of your ongoing tasks?

How long does each task take to complete?

Where are there errors in your build?

Of course, every team knows it's important to keep a close eye on the performance of your builds once they're in production.

But while dev teams have already automated and optimized almost every other aspect of the dev cycle, build observability has fallen by the wayside.

Most teams still rely on manual legwork, combing through build logs by eye in order to identify errors or opportunities for improvement.

Why?

Because of another major difference between IT observability and build observability:

There are far fewer build observability tools available on the market. And, even when teams can find tools that promise to fulfill their needs, they still often struggle to understand which tools are right for them.

As a result, teams have no visual, accessible way to monitor builds. Which means they're constantly working from an incomplete picture - and failing to see any of the real benefits of observability.

What is build observability?

The “big-picture” answer to this question is clear: Build observability allows your team to truly understand how your build is performing, and take steps to make it better.

But let’s break that down a little further.
Many of the benefits of build observability stem from one thing: data visualization.

Build observability tools can analyze data from your builds at volumes that humans could never hope to manage — and, most importantly, make that information digestible and actionable.

Bringing these insights to the surface by visualizing your data in the form of reports, graphs, charts, and alerts is the heart of build observability.

These new levels of insight mean that build observability delivers eight broad benefits:

- **It helps you understand your CI and DevOps’ behavior.** How many builds are you running? How many of those builds are succeeding? How much resource does each build use?

- **It highlights trends and anomalies in your builds.** How long does the average build take? Are there large fluctuations in build times across different projects and tasks? Are there some occasions or types of projects where performance is significantly better or worse?

- **It helps you spot bottlenecks faster.** Build observability tools can visualize the point in your builds where processes collide — where one process has to stop running until another one is complete. Once you can see these bottlenecks clearly, it’s much easier to avoid them; you can strategically plan your pipelines to minimize downtime and ensure your builds are always flowing as fast and as freely as they can.

- **It pinpoints how you can improve.** Slicing and analyzing the data in different ways helps you figure out why and how things go wrong, so you can fix the root cause of the problem. On a macro scale, observability allows you to pinpoint when build times start increasing, so you can identify commonalities and find the root cause. Or diving into the detail allows you to gather more information about specific friction points. With these insights, you can find new efficiencies and fix small issues that add up to bigger frustrations and delays.

- **It improves your ability to debug your builds.** With better observability, you can identify problems that keep cropping up on certain types of builds, or across your entire dev process.

- **It helps you identify and analyze errors faster.** The human eye might be able to spot an error in your build manually — if you're very lucky. But doing so takes time and effort that you could better spend elsewhere. Build observability tools use data visualization to make those errors impossible to miss — whether that's by showing anomalous activity in a different color on a chart, or by actively grabbing your attention with a pop-up dialog box that tells you exactly what's wrong. Real-time monitoring also allows you to spot bugs faster and earlier in the build process — which means you can fix them before they break your entire build.

- **When something goes wrong, it helps you solve who "broke the build."** By quickly pinpointing the cause of errors and issues, you can focus on resolving the issues and making sure the relevant team members understand how to avoid making the same mistake in the future.

- **It makes reporting significantly easier.** By automatically organizing data from your builds into neat, digestible charts and reports, build observability tools make it much easier for you to review your builds from a high-level, strategic point of view. It also puts data right at your fingertips whenever you need to justify or explain a change to your build processes — to both management and your people.

Section One:
How much does your business need to know?

Every dev team needs some form of observability in place. It's the only way to deliver the highest-quality builds and ensure that they keep running reliably well into the future.

But not every project or team needs the same level of observability.

To create a truly effective strategy, you'll need to tailor your observability to suit a few different factors.

First of all, your observability strategy will change significantly depending on the size of your organization and how your teams are set up. In simple terms, the bigger your organization, the more important build visibility becomes.

Let's take a look at how the size of your organization should shape your strategy.

You are	Number of Devs	Number of daily full builds	Broken build resolution time (avg.)	You need
A small team	1-15	1-50	20min	High level insights
A large team	15-100	50-500	200min	Granular day-to-day data
An enterprise team	100+	500+	24hrs	Macro-level insights + data sharing

If you are: A small team with fewer projects, simpler builds, and/or smaller codebases.

You need: High-level insights

If your team is only made up of a few devs, and your codebases are relatively small and simple, build observability might not seem like a priority. But it can still deliver significant benefits that you shouldn't overlook.

At this stage, it's probably relatively easy to uncover the sources of any issues by just manually reviewing logs. If you need to figure out who made a particular change to the code, and why, you'll probably only need to speak to a few team members to get to the bottom of it.

Which means an intensive, highly granular observability strategy probably isn't worth your resources. If you invest too heavily in building up your observability, you might find yourself drowning in unused functions, unnecessary reports, and unhelpful insights.

But that doesn't mean you should throw observability completely out of the window. You can still benefit from having some kind of basic observability strategy in place to help you spot high-level trends - the kind that are hard to pick out manually.

If errors are growing more frequent, or if your builds are taking longer than usual, or if a certain type of project creates the same problems every time, you'll want to know about it. A basic tool can create high-level reports on things like your compile times, which can help you understand how much dev time is wasted - and how you can use it better.

If you are: A large team tackling frequent, complex projects
You need: Granular day-to-day data

As your team grows and your projects get more advanced, your ability to keep an eye on your builds shrinks rapidly.

You're building more frequently. There are more team members, which means more hands on your code. Your hardware and infrastructure have multiplied to keep up with growing demand.

All of which means two things: more opportunities for errors to creep in, and a much smaller chance that you'll catch those errors manually.

You might already be using a solution like Prometheus or Splunk to get some insight into your builds. But, when your team reaches this stage, you'll need a more advanced observability strategy - and a tool that is more specifically designed for build observability and monitoring - in order to maintain service quality for users without putting the brakes on your growth. That means you'll need to consider how you can gather insights into your:

- **Build capacity** – Are you using your build resource effectively? Would taking on more, bigger, and more complex projects put you at risk of overflowing your capacity, damaging your uptime and the user experience? Or do you have plenty of room to grow?
- **Utilization rates** – Are you maximizing available CPU capacity? Can you harvest idle CPUs to speed up your builds? Or are you already running at full utilization - which could mean your builds are at risk of crashing completely?
- **Build trends** – Now that you're probably handling more projects, it's harder for the human eye to spot recurring trends and patterns in your builds. Do the same delays keep cropping up in different builds? Do certain team members need support to work more effectively and build sturdier applications? You'll need a good observability tool to bring these trends to light, so you can act on them.

If you are: A very large team with separate dev and DevOps departments

You need: Macro-level insights and information sharing

There are many reasons why organizations might split their dev and DevOps functions into different departments — not least because it allows each function to receive a great deal more attention and expertise than they would otherwise expect.

But doing so does make keeping track of build performance across both functions more challenging. With some hard work, you might just about be able to pull together an overview of each team's build performance.

But "seeing" your builds and actually getting meaningful, actionable insights from them are two different things;

your teams might be good at sharing data, but that data won't help anyone if neither team knows what to do with it.

At this level, it's almost impossible to give both teams the insights they need without an effective build observability tool — one that can see the data, uncover the insights you need, and communicate them quickly to everyone.

You'll need to ensure you're gathering and sharing insights that suit both teams' priorities:

- Dev teams tend to focus less on the detail of how builds work and the resources it takes to complete them. Instead, they're interested in whether the builds themselves are working. Or, if they're not working, why they might be failing — and how team members can protect the user experience by finding errors fast and preventing them from happening in the future.
- DevOps teams are much more about the behind-the-scenes machinery that makes builds possible. Your build observability strategy will need to provide them with the information they need to optimize your infrastructure, ensure your devs have the CPU capacity they need to work uninterrupted, and keep build times down as far as possible.

Section Two:
Choosing the right build observability tool

Now that you understand the principles and functionalities that should underpin your strategy, you can start thinking about which build observability tool will be best for your team.

Build observability tools can be broadly divided into three categories: DIY, open source, and proprietary or commercial tools.
Let's take a look at each of those types of tools in turn.

DIY build observability tools

Best for: Small teams with relatively simple observability needs
If you only need enough observability to spot when your builds are slower than usual, or to identify errors that keep cropping up, you might struggle to justify the time and budget for a build observability tool.
Even free open-source tools take time and effort to set up and maintain, after all - time that your team would probably prefer to spend on the actual dev work.

But you can still take advantage of the benefits of build observability, if you have the skills to create your own, simplified build observability tool.

Many small teams find that they can get a significant productivity, uptime, and efficiency boost from developing tools that can perform simple observability tasks.

Often, this takes the form of a script that can scan through your logs to find errors. These scripts can also output surface-level trend data on things like total build time, build frequency, and build rate.

If you don't want to build your own, some of these DIY tools are available online - this is a good example of one that helps users track how much time they spend idly waiting for builds to complete.

Open-source build observability tools

Best for: Larger teams that are just starting to explore build observability

When it comes to build observability, open-source tools represent a kind of halfway point between DIY tools and the more sophisticated paid tools on the market.

That makes them perfect for two types of teams: those that are scaling, but can't yet justify the budget for a paid observability tool; and teams that want to try out observability tools, but aren't sure exactly which metrics they need to monitor.

For both of these situations, open-source tools give dev teams the chance to try out a more advanced observability tool without making a major monetary commitment.

However, there are some downsides to open-source tools: First, open-source tools tend to have limited capabilities. These tools might be able to provide some insights, but they tend to lack the advanced analytics, dashboards, and ongoing monitoring capabilities that you would get from a paid tool.
You should also bear in mind that, with open-source tools, you'll need to do a lot of the legwork. Getting open-source tools up and running tends to take a lot more time and effort than investing in a more sophisticated solution. Paid tools are generally designed to get up and running fast, and the best ones come with support from your provider that helps you maximize the impact of your investment. You might be saving your money by using an open-source tool, but you'll need to give up more time and effort in exchange.

A few examples of open-source build observability tools are:

Grafana

Grafana Labs' interactive data visualization platform, known for its reliability and thriving community of users.

Prometheus

Billed as the "leading open-source monitoring solution," Prometheus uses time-series data to generate alerts related to the performance of your builds, and also offers advanced data visualization.

openreach

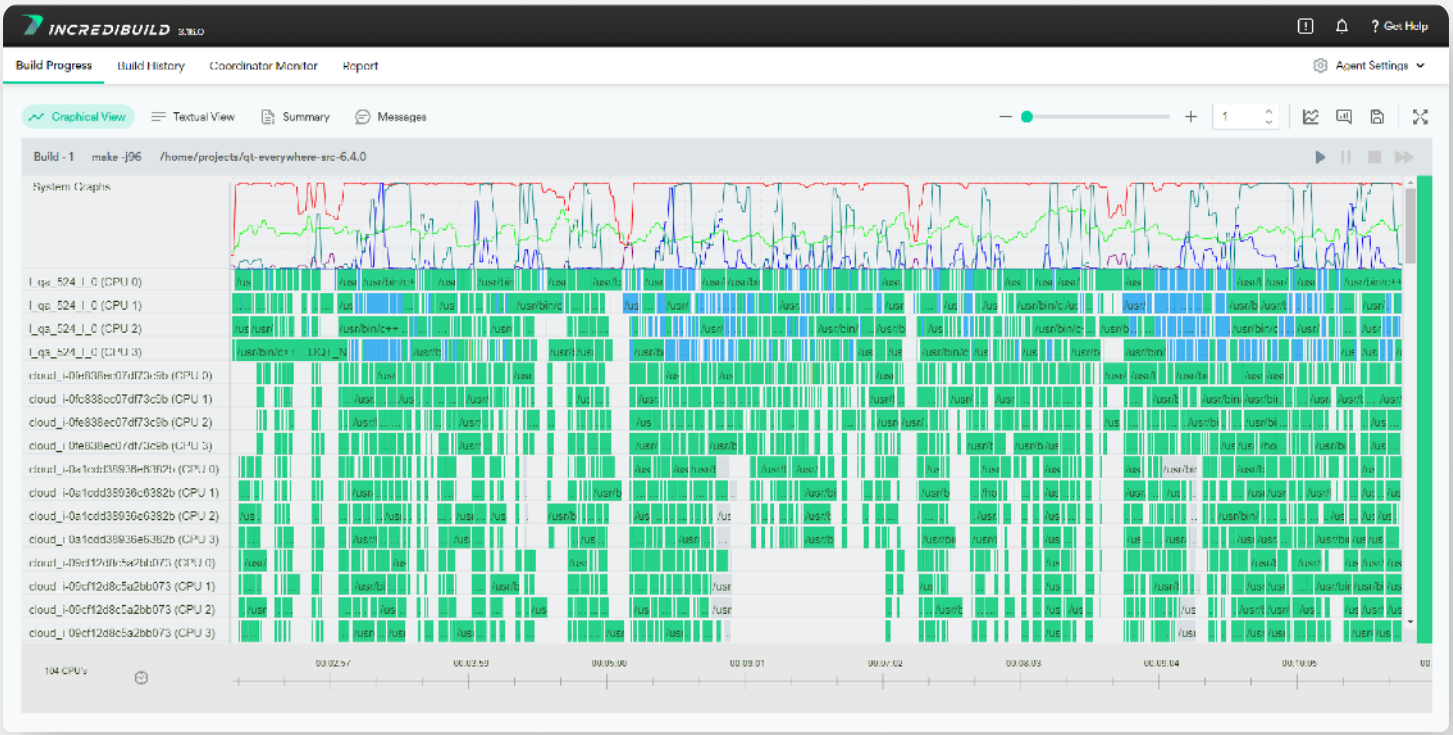
A "distributed search and analytics engine" that allows you to perform full-text searches of your data. OpenSearch comes with a collection of plugins and applications referred to as "Observability — allowing you to explore, discover, and query your build data in detail.

OpenTelemetry

Otherwise known as OTel, this observability framework is designed to help teams instrument, generate, collect, and export telemetry data, providing a standardized format for collecting and transferring your data that makes it easier to analyze.

All of these tools can provide valuable insights. But remember that they're all built to serve very specific functions; none of them will be able to provide all of the specific build observability functions you'll need to really optimize your builds.

Commercial or proprietary build observability tools



Incredibuild's Build Monitor in action

Best for: Teams of any size with clear build observability needs

If you're making build observability a priority, you'll need a tool that does it all: detailed data gathering, real-time reporting, high-level insights, and granular detail.

To get all of this, you'll likely need to invest in a commercial build observability tool. The good news is that, with the right tool, this investment quickly pays off in productivity, uptime, and efficiency. Although these tools are generally designed with larger dev and DevOps departments in mind, they can be useful for teams of any size.

The beauty of these systems is that they come with all the essential tools you need for build monitoring included - from data collection to real-time monitoring, analysis, and resolution

And, because they're specifically built to support build observability for teams of any size, they'll give you the complete picture you need to truly optimize your builds. They'll check your codebase to ensure that every part of it is complete and accessible; they'll flag issues as soon as they arise, so you can begin addressing them right away; and they'll make sure that every build finishes as fast as possible.

Shine a light on your builds - your way

Many teams have managed to scrape by without build observability tools until now - particularly smaller teams with simpler builds, who find it easier to manually monitor their performance.

But approaching build observability this way turns an important function into an enormous headache. Whether you're manually combing through your logs or mixing and matching open-source tools with limited capabilities, build observability is probably taking up far too much of your time and energy - and still not giving you the insights you really need.

And, without the right tool on your side, monitoring your builds is only going to take up more your time.

Builds are becoming more complex. Increasing reliance on the cloud, container orchestration, and serverless build models mean we're deploying thousands of instances at once - making it harder than ever to track down bugs. Which means the risk of bugs and performance issues slipping through the cracks is greater than ever.

And, at the same time, the tolerance for those bugs and performance issues has never been lower.

In short, almost every team needs some form of build observability built into their DevOps processes.

Not just because it can prevent disaster in the form of crashing applications, wasted time, and frustrated users.

But also because it can make build observability easy and effective. Cutting out the manual work so that you can unlock new levels of efficiency. Giving you the understanding you need to optimize every minute spent building your applications.

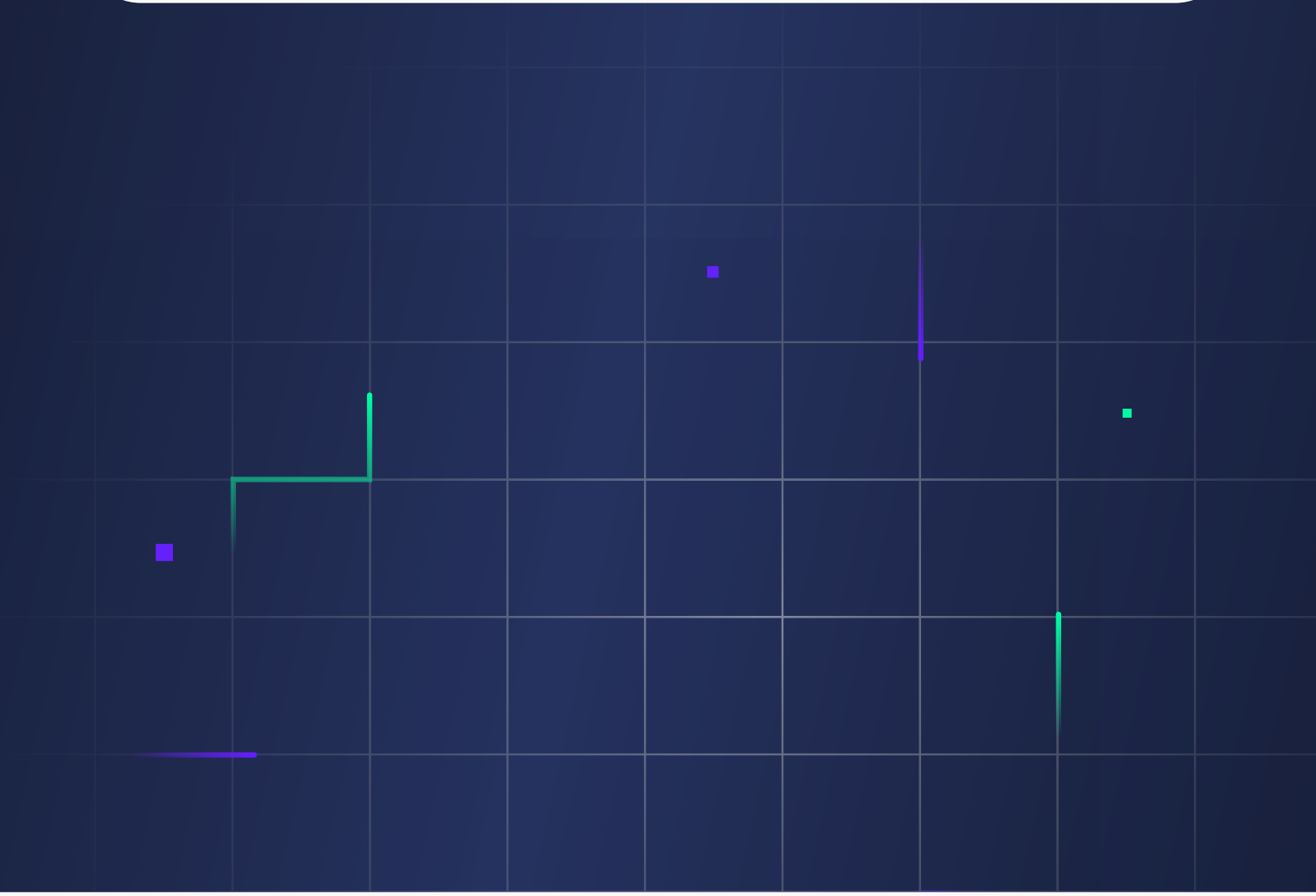
The good news is that - as we've shown above - there's no one right way to improve your observability. From DIY solutions for small, agile teams, to powerful commercial tools that give you ultimate oversight of your applications, there's something for everyone.

The key is finding the strategy and the tools that work for you - and that give you exactly the amount of insight you need to build your best applications.

About Incredibuild

Incredibuild is the leading platform for Development Acceleration.
Our Virtualized Distributed Processing™ and patented Build Cache technologies help you build faster and lower development costs on-prem and in the cloud so your dev teams can iterate more, fix bugs faster, and build great products.
Incredibuld also helps you get more out of your current investments in hardware and optimize cloud resources to get more bang for your buck. Incredibuild doesn't require changes to your existing tools or processes, it's a lightweight, code-free solution that most teams can get running in hours. Incredibuild is designed to accelerate distributed dev teams, even those working from home and with low bandwidth - right from the first sprint.

Find out more about how Incredibuild works: Incredibuild.com



INCREDIBUILD