Palantir

APOLLO DEMO DAY

April 27, 2022

→ English Transcript

This transcript is provided as a courtesy and is intended to be viewed, and is subject to, the accompanying oral presentation and related materials, including any legal disclaimers.

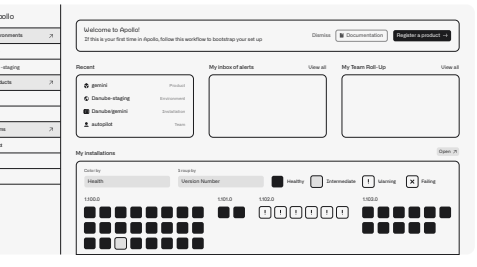
Index

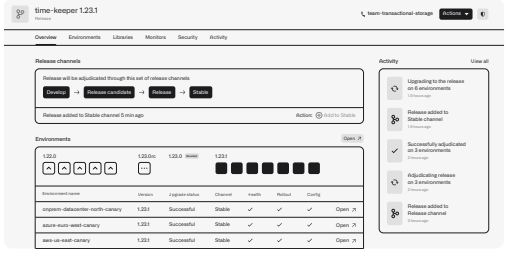
Why We Built Apollo 03
→ Shyam Sankar, Chief Operating Officer

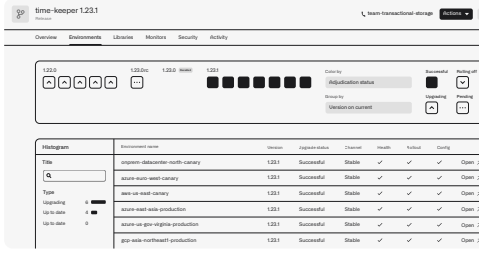
The Core Principles that Unlock Autonomous Software Deployment 04 — 07
→ Greg DeArment, Chief Architect & Head of Apollo Platform

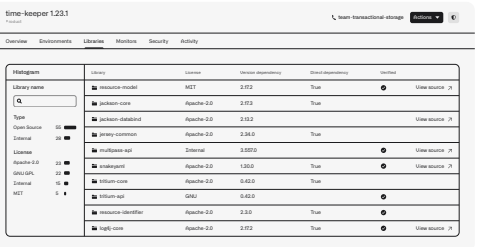
Apollo in Action: Software Demo 08 — 12
→ Sean Hacker, Forward Deployed Engineer

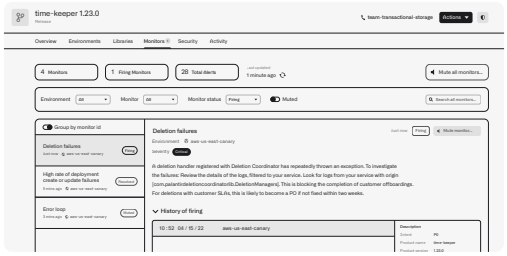
Disclaimer 13 — 15

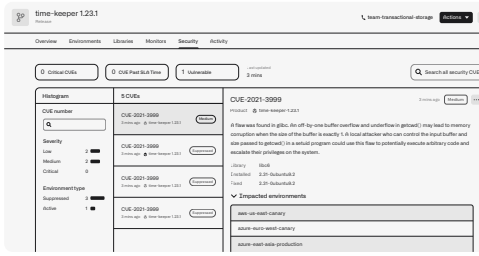












Why We Built Apollo

Shyam Sankar
→ Chief Operating Officer

At Palantir we always talk about working on the world's hardest problems. And by hardest, we don't mean most intricate or most cerebral or most technically complex. We mean the most existential. We have been building Apollo for six years because we believed software delivery was one of those existential problems. And frankly, I'm not sure anyone else really did. Then SolarWinds happened, then Log4j. And it is painfully clear that the software delivery and management—i.e., the software supply chain—it is an existential issue for modern enterprises.

We can attest to the utility and efficacy of Apollo by simple virtue of the fact that we use it to deploy our own software. Apollo manages, deploys, and maintains Foundry and Gotham worldwide, supporting mission critical work across some of the most challenging remote environments imaginable. From satellites to submarines, high side networks to Humvees, and, of course, every flavor of cloud and on prem, Apollo manages software across more than 250 distinct customer environments. It is what enables us to deliver and meet our moment.

This fragmentation of delivery environments is fundamentally changing the game for software providers. Apollo solves this. What Apollo does is it enables engineers to write code once that works for all environments. It enables the same piece of software to meet the unique needs of each environment by allowing the developer to encode its own preconditions in expected behavior of the environment, whether it's on prem, in the cloud, or at the edge, so that developers can focus on creating better products and avoid all of that human toil of getting those products where they need to be. Apollo has been the backbone of how we have delivered our software over the last six years, and it will be over the next 60 years. I am thrilled to share the full power of the platform with you today.

The Core Principles that Unlock Autonomous Software Deployment

Greg DeArment,
→ Chief Architect &
Head of Apollo Platform

You know, microservices and service oriented architectures have become so popular because the promise of faster innovation through distributed work. But in order for you to actually realize these benefits, the teams need to be able to actually evolve at different speeds. Teams should be able to ship new features and capabilities as soon as they're ready, instead of having to be slowed down and wait for a weekly, monthly, or quarterly release schedule that is determined by the slowest mover or your ability to, like, QA the platform. In a service-oriented architecture, they're are just going to be API dependencies between the different services, but you want those to be able to ship new releases at different rates.

So from the very beginning, Apollo is built to address these intrinsic problems that come with distributed systems by allowing the software engineer teams to encode the preconditions and the expectations of their software next to the code itself. Apollo then takes these embedded constraints and ensures they aren't violated the act of upgrading and makes it possible to safely automate rollback of a given service without causing impact to the availability of the entire platform. So at compile time, storage systems are able to declare the set of schemas that they are able to read from and write to, specific migration steps they support, and a way for the current version of their schemas to be reported at runtime. This makes it possible for Apollo to safely upgrade storage services through schema migrations and rollback when necessary, with safety rails that prevent data corruption.

The second principle, a software developer shouldn't need an in-depth understanding of the environment or infrastructure system that their software will be run in and in order to build capabilities that power world class software. When the world was building and shipping software monoliths and deploying into one or two virtual machines or servers, the hardest deployment problem that developers faced was trying to understand Linux. But today, the infrastructure that software is being built on is far more complicated, and it's only becoming more so. With that being said, the infrastructure systems available today make it possible to provide the non-negotiable capabilities of modern software. Things like data and encryption and network security, software that is highly available and can be updated without user downtime, and the ability to deploy across multiple cloud providers and to the edge.

The Core Principles that Unlock Autonomous Software Deployment

Greg DeArment,
→ Chief Architect &
Head of Apollo Platform

[CONT.]

These are all increasingly crucial for business critical software. But for a developer to build and deploy a modern Java or Python application that needs be highly available — maybe it needs some configuration, it need certificates to serve HTTPS traffic and expose through a front door proxy — they will need to understand, configure, and maintain six or seven different Kubernetes objects, in addition to the cloud infrastructure components that expose that, like network load balancers. That just isn't a good use of most developers' time. So Apollo provides an SDK the developers can use to build modern software applications. Apollo's SDK allows the developer to declare the properties of their software like, you know, maybe I need a local persistent volume or I need three replicas of my service and quorum for my service is having any replica be available. Apollo can then generate and manage the appropriate Kubernetes pod controllers based on those declared properties. A certificate in the corresponding public and private keys are generated and provided to the service, so the service can configure TLS encryption for HTTPS traffic without needing to understand how PKI material is created or managed by the platform. And the developer is able to declare that their service produces an API endpoint and request that endpoint get exposed over reverse proxy at a certain path, all without needing to understand the complexities of the way a Kubernetes cluster ingress system is set up or how ingress objects work. All of these capabilities are possible without making a single line of code change.

The third principle is making it possible to deploy software where it is needed by enabling software platforms to be repeatable, reproducible, and portable across cloud providers and infrastructure types. For example, many companies may find the idea of delivering their software into a new environment daunting, as their platform is nearly impossible to set up from scratch, much less operate multiple installations of at once. Others have deployment pipelines that are highly tailored to a single environment, where introducing a new production environment would fundamentally break their existing deployment process. Market access just isn't an obtainable goal for many, because technology and tooling used to deliver the software has been designed for the single SaaS environment paradigm. And as the world has evolved, their tooling has not. So Apollo takes a very different approach than other tools in the DevOps or platform engineering space. Apollo starts by modeling software you want to run across your environments and enterprise into a single software catalog. This allows all parties to share a single view of

The Core Principles that Unlock Autonomous Software Deployment

Greg DeArment,
→ Chief Architect &
Head of Apollo Platform

[CONT.]

everything that's known about a piece of software from what is declared by the developers—things like, you know, cross service dependencies, support and schema versions, resource requests, and so forth—to information that is augmented by external systems like virus scanners, or vulnerability scanners, or the security scanners that you're already using across your enterprise. Different software platforms can then be defined as the composition of the individual services and products that exist in the Apollo catalog. Apollo then models each environment that you want to deploy and manage software into and declare a set of entities unique installations of a given software product from the catalog. Release channels are used to represent the different stages software progresses through as it rolls out across your environments. Release channels allow you to model non-linear rollout processes. For business models more like Palantir's, it requires you to manage a large number of environments and deploy your software to where your customers are, you may choose to use release channels to model confidence of new releases, where you first deploy your software to canary environments before rolling it out to more mission critical environments that are more risk averse. This approach to software rollout avoids the most painful failure modes of traditional pipeline driven continuous deployment models. Where pipelines tend to be specific to a single or small set of services and evolve in silos along team boundaries, release channels provide a consistent approach for your production management story across teams, while still allowing each dev team to control the inputs to the system that dictate how their software gets rolled out.

And the final principle, effectively managing your software production, extends beyond the simple act of deploying a piece of software and depends on the active collaboration of software developers, operators, security and compliance teams. If we've learned anything over the last few years, software supply chain security and related problems are not going away. In the same way that we don't need every engineer understanding the details of the infrastructure that their software runs on, we also don't need them spending time becoming experts in how software security works. It should be an integrated part of the production management story, and it should come for free. With Apollo, the software catalog becomes a central gate that governs what software can and cannot be deployed into production, based on the policies determined by the security and compliance teams. Without impeding on the natural software development process, it integrates with your existing DevOps

The Core Principles that Unlock Autonomous Software Deployment

Greg DeArment,
→ Chief Architect &
Head of Apollo Platform

[CONT.]

toolchain, like your continuous integration system, your artifact and container registries and your vulnerability and security scanners. Your security teams can define vulnerability policies that enable Apollo to automatically recall releases that violate those policies, or notify the security team where there are releases that are out of compliance for prompt annual review.

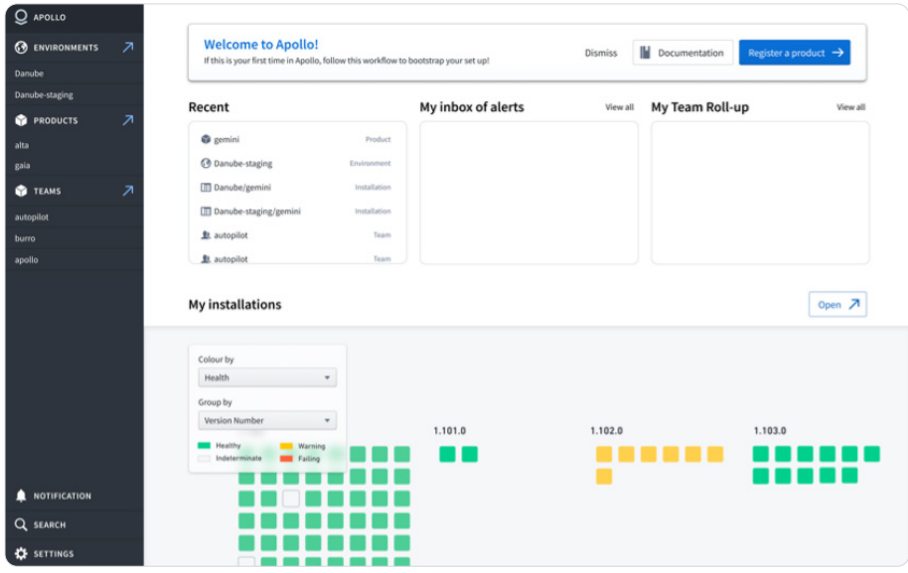
Post software deploy, Apollo will continuously check the health of each of the software installations if the development team opted into the health system. This allows developers to encode SLOs and other criteria that their software should meet in order for it to be considered healthy. Additionally, they can also declare how their software should be monitored within the software artifact itself, making it possible to define the expected behavior without having to modify the code. Apollo aggregates this information into a single pane of glass and can automate the rollback and other active and passive remediation actions when the SLOs are violated, health checks reported issues, or with the embedded monitors fire.

Apollo in Action: Software Demo

Sean Hacker,
→ Forward Deployed Engineer

Let's jump into a demo to understand how each of these groups leverages Apollo over the life cycle of a release. Let's imagine I'm a developer, who leverages Apollo to continually deploy my software across all of my environments. I've just cut a release of the time-keeper product on version 1.23.0. Immediately. I can see that my release has been added to the appropriate release channel in accordance with my team's deployment policies for all tagged software releases. And as a result, Apollo has begun automatically upgrading my three environments that subscribe to that release channel for updates to the time-keeper service.

• Palantir Apollo Control Center



Let me show you around my product's release page, where I'm able to understand everything about this software release from a single location. From here, I can see that my current release is being adjudicated against my defined canary environments and is set to automatically mark itself as stable, once that rollout is complete and all installations remain healthy throughout the declared soak period. As you can see, my team has chosen canary environments that span a couple of different cloud providers, as well as an on premise data center. It looks like one of my monitors is firing, which probably means that my upgrade failed in one or more of my canary environments. When this happens, the first thing I do is dig into my Releasesrelease's Monitors tab to better understand what notifications are firing and what that will mean for my release. Monitors allow me to encode service level objectives for my product into health

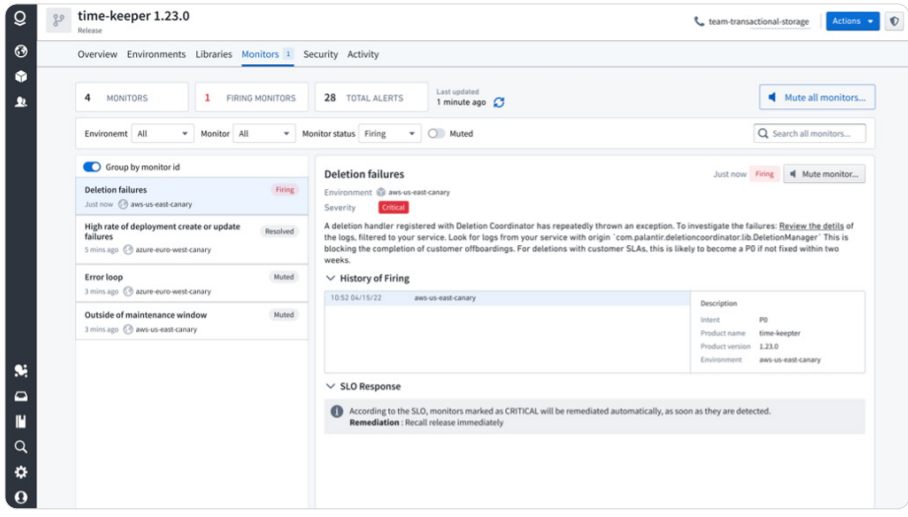
Apollo in Action: Software Demo

Sean Hacker,
→ Forward Deployed Engineer

[CONT.]

checks that Apollo can action. As we can see here, it looks like one of the critical monitors that I encoded within my product has failed. A critical monitor firing is typically a sign that something has gone seriously wrong with my release, and the release is not stable enough to be rolled out across the fleet. In this instance, I was the one who caught the monitor firing, but it's oftentimes a member of one of our operational teams that's responsible for managing these environments. Because I've been able to encode my SLOs within the monitor and have Apollo automatically track this for me, my colleagues on the ops team are just as capable as I am in triaging alerts and actioning things appropriately to ensure that my software is delivering value reliably for customers in production.

• Palantir Apollo Monitoring



As we were talking, Apollo has automatically recalled my release, not only ensuring that time-keeper 1.23.0 won't be installed on any new environments, but also kicking off an automated downgrade for those canary environments that had already upgraded. Navigating back to the overview page for my 1.23.0 release, I can see that Apollo is now prominently informing me that this release has been recalled, hethe reason why it was recalled, as well as the strategy that Apollo took to get back to a known good version. Having all of this information in a single place provides crucial historical context for me and my team. As you can see, the activity panel has been updating as Apollo has undertaken automatic action to detect my firing monitor, recall the release, and begin the rollback to the last known good version of my product. Apollo's ability to proactively track and audit all actions within my managed environments allows the products I am developing to run in environments that may require strict security and compliance controls.

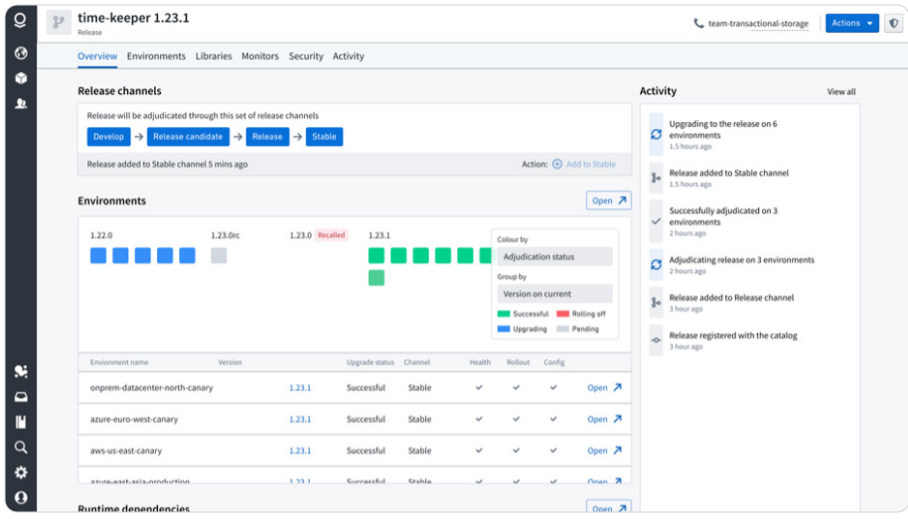
Apollo in Action: Software Demo

Sean Hacker,
→ Forward Deployed Engineer

[CONT.]

As we can see from the environments panel, Apollo has just completed the rollback of my canary environments off of my bad 1.23.0 version. Since Apollo was able to automatically begin the process of rolling out my release across canary environments, catch an issue with my release before it hit critical production environments, and then immediately recall and roll back that release, I now have plenty of time to debug what went wrong and focus on improving my product. I've gone ahead and fixed that issue and cut a new release of time-keeper 1.23.1. This release has successfully upgraded across all three of my canary environments and has not caused any monitors to fire for my defined soak period. Apollo is therefore automatically adjudicated the 1.23.1 release and marked it as stable. Within my team, this is the release channel we use to signify a release is ready for production, and Apollo will have automatically begun the process of rolling it out across the fleet.

• Palantir Apollo Software Catalog



In addition to release channels, you've already seen me leverage Apollo's environments view to quickly understand the state of my releases across environments. Environments in Apollo are a critical concept and can take many forms. Some of the environments Apollo deploys my software into are traditional development or test environments that are a staple for many developer teams. Other environments are true production environments, but allow me to deploy the same version of my software across different cloud regions cloud providers—for example, Amazon's US East region and Azure's Europe West Region—and some of the environments are production environments that have been set up on a per customer basis, maybe inside of their VPC or data center.

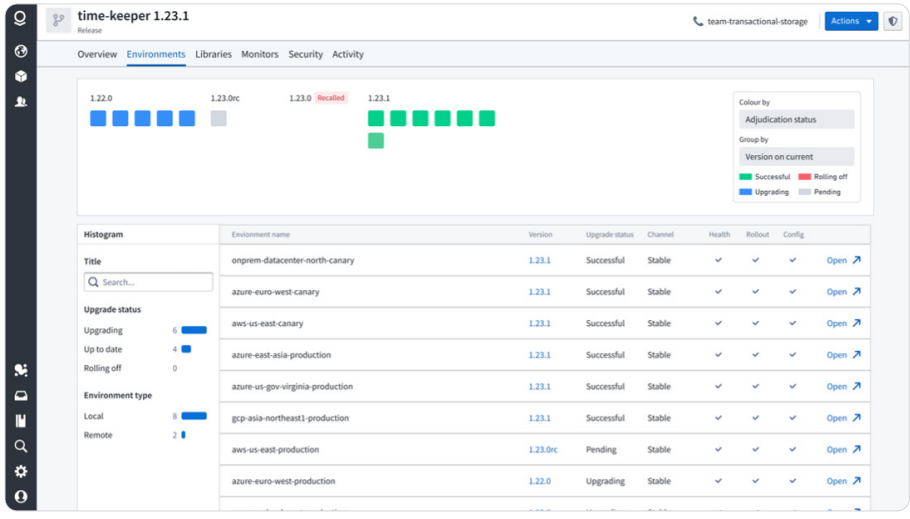
Apollo in Action: Software Demo

Sean Hacker,
→ Forward Deployed Engineer

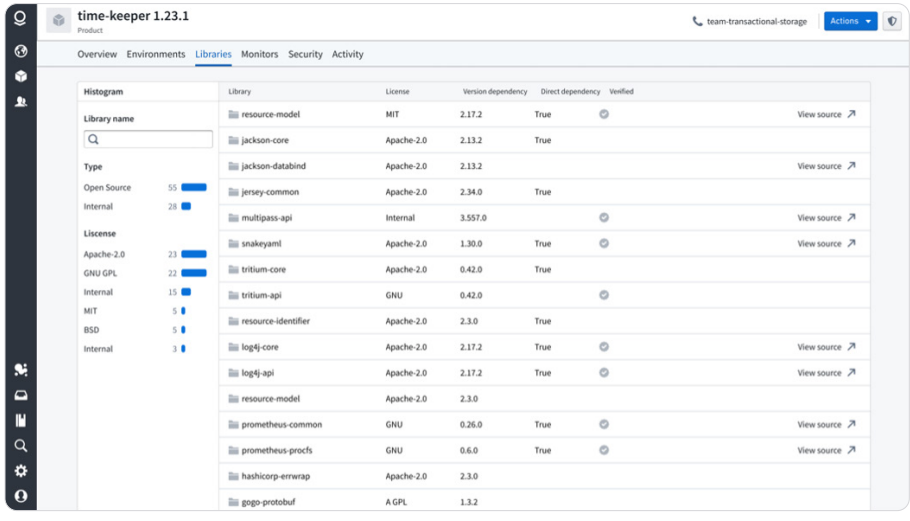
[CONT.]

By encoding all of this information within my release in the Apollo catalog, I'm able to allow Apollo to manage this complexity for me. This gives me time back and allows me to ship software to every environment where it's needed reliably and efficiently.

• Palantir Apollo Delivery



• Palantir Apollo Software Supply Chain



As we all know, in the wake of Log4j and SolarWinds, it's increasingly crucial for organizations to holistically understand their software supply chain. Apollo provides me with an easy to use software bill of materials for my release, giving me visibility into all my software dependencies. I'm able to quickly filter down and understand which versions of Log4j my release is leveraging. In this instance, we can see I'm using the safe 2.17.2 Version, so there's no action necessary. Apollo gives me 360-degree

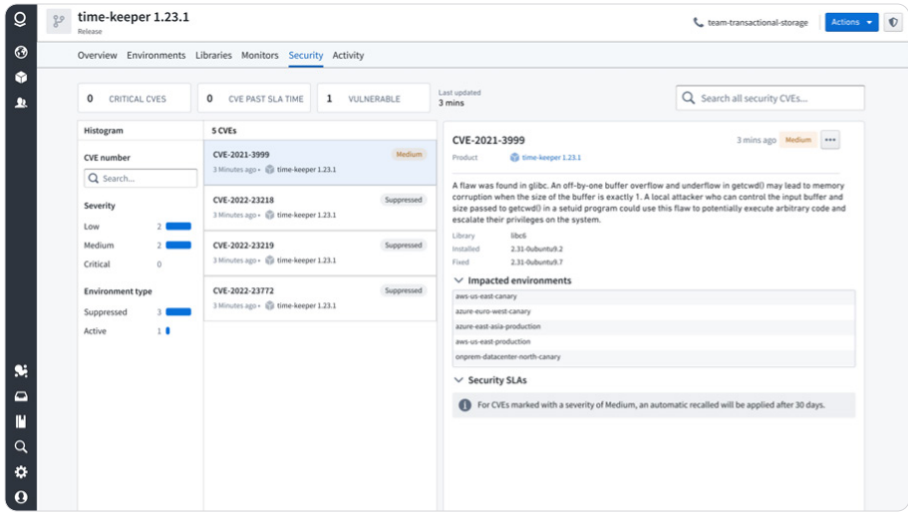
Apollo in Action: Software Demo

Sean Hacker,
→ Forward Deployed Engineer

[CONT.]

transparency into all software I have in production regardless of the environment. In addition to allowing me to understand the supply chain for my software, Apollo also provides visibility into the security of my releases. As a developer, it's incredibly valuable to me to have this information alongside all of the other information we just looked at, so that I can truly understand my software from a single pane of glass. Apollo will continue scanning my release as time goes on, ensuring that we catch any new vulnerabilities that are discovered in the future and can automatically recall this release or notify me that a vulnerability needs my attention.

• Palantir Apollo Security



In the time that I've been walking through how my team leverages Apollo, it's been hard at work upgrading my production environments to my 1.23.1 time-keeper release. We can see that the rollout is now complete, and Apollo has upgraded my software across all of the environments on which it's deployed. The world that we're deploying software into today is much more dynamic than it was even a few short years ago, and that dynamism is only increasing. Apollo was built to manage software at scale and pace, delivering further, faster and more efficiently than ever before.

Disclaimer

This presentation and the accompanying oral commentary include discussion of Palantir products, features and capabilities, including recent updates to our products, as well as potential product direction. They are intended for information purposes only and shall not be deemed to be incorporated into any contract or agreement and do not constitute a guarantee or warranty of any kind. They are not a commitment to deliver any material, code, or functionality, and should not be relied upon in making procurement, purchasing or investment decisions. The development, release, and timing of any features, capability, or functionality mentioned herein remains at our sole discretion.

This presentation and the accompanying oral commentary contain "forward-looking" statements within the meaning of the federal securities laws, and these statements involve substantial risks and uncertainties. All statements other than statements of historical fact could be deemed forward-looking, including, but not limited to, expectations of future operating results or financial performance, market size and growth opportunities, plans for future operations, competitive position, technological capabilities, and strategic relationships, as well as assumptions relating to the foregoing. Forward-looking statements are inherently subject to risks and uncertainties, some of which cannot be predicted or quantified. In some cases, you can identify forward-looking statements by terminology such as "guidance," "expect," "anticipate," "should," "believe," "hope," "target," "project," "plan," "goals," "estimate," "potential," "predict," "may," "will," "might," "could," "intend," "shall," and variations of these terms or the negative of these terms and similar expressions. You should not put undue reliance on any forward-looking statements. Forward-looking statements should not be read as a guarantee of future performance or results and will not necessarily be accurate indications of the times at, or by, which such performance or results will be achieved, if at all.

Disclaimer

Forward-looking statements are subject to a number of risks and uncertainties, many of which involve factors or circumstances that are beyond our control. Our actual results could differ materially from those stated or implied in forward-looking statements due to a number of factors, including but not limited to risks detailed in our filings with the Securities and Exchange Commission (the "SEC"), including in our quarterly report on Form 10-Q for the quarter ended September 30, 2020 and other filings and reports that we may file from time to time with the SEC. You can locate these reports on our investor relations website (https://investors.palantir.com/financials/sec-filings/) or on the SEC website (https://www.sec.gov). If the risks or uncertainti-es ever materialize or the assumptions prove incorrect, our results may differ materially from those expressed or implied by such forward-looking statements. Except as required by law, we assume no obligation and do not intend to update these forward-looking statements or to conform these statements to actual results or to changes in our expectations.

This presentation contains statistical data, esti-mates and forecasts that are based on independent industry publications or other publicly available information, as well as other information based on our internal sources. This information involves many assumptions and limitations, and you are cautioned not to give undue weight to these estimates. We have not independently verified the accuracy or completeness of the data contained in these industry publications and other publicly available information. Accordingly, we make no representations as to the accuracy or completeness of that data nor do we undertake to update such data after the date of this presentation. All data shown in product demonstrations is notional or publicly available and any resemblance to actual persons, entities or events is purely coincidental and should not be inferred. Certain visualizations and capabilities shown in product demonstrations may rely on or reflect third party data sources that are not included as part of Palantir's standard product offering.

Disclaimer

This presentation also contains links to publicly available websites, data, or other information. We have not independently verified the accuracy or completeness of such websites, data, or information and accordingly we make no representations as to their accuracy or completeness nor do we undertake to update such data or information after the date of this presentation. The inclusion of external links does not constitute endorsement by Palantir of the linked websites or the data or information contained therein.

By attending or receiving this presentation you acknowledge that you will be solely responsible for your own assessment of the market and our market position and that you will conduct your own analysis and be solely responsible for forming your own view of the potential future performance of our business.

Unless otherwise noted, all product, feature, or service names, logos, and trademarks, including without limitation Palantir and the Palantir logo are the intellectual property of Palantir and / or its affiliates in the United States and/or other jurisdictions. Any non-Palantir logos or trademarks included herein are the property of the owners thereof and are used for reference purposes only. Such use should not be construed as an endorsement of Palantir or the platforms and products of Palantir.