aws marketplace

EBOOK

Cloud-native Observability: Actionable intelligence for complex systems

How to architect observability for modern applications and infrastructure using metrics, events, logs, and traces (MELT) data.

aws marketplace

CLOUD-NATIVE OBSERVABILITY:
ACTIONABLE INTELLIGENCE FOR COMPLEX SYSTEMS

Table of contents

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

End-to-end monitoring and insights power reliability and innovation

Implementing microservices and cloud-native architectures has been a goal for many organizations who hope to increase speed and agility. But gaining these capabilities introduces additional complexity, as cloud-native applications are decoupled and distributed. Systems become increasingly challenging to observe, and when issues occur it's often difficult to identify their root cause. DevOps teams can also find it difficult to monitor components outside of their control.

In this ebook, we'll offer guidance for architecting end-to-end Observability for cloud-native applications and infrastructure using telemetry data. You'll take away practical methods for establishing baselines and uncovering the why versus the what when things go wrong.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Modernization is a business priority today

To delight customers and win new business, organizations need to build reliable, scalable, and secure applications. That means adopting new technologies, practices, and consuming services as APIs.

As an application development professional, your goal is to deliver business value fast. Modern applications help achieve this goal by separating and decoupling the monolith into smaller functional services—or **microservices**—that focus on one thing and do it well. Each microservice often has its own data store and can be deployed and scaled independently. They represent the real world, where service boundaries equal business boundaries.

This has forced organizations to evolve by giving engineering teams the autonomy to architect, develop, deploy, and maintain each microservice. With this approach, you end up with the ability to make decisions very quickly because your decisions only impact individual services. After all, innovation requires change. You can learn faster by making lots of little changes to drive incremental innovation, rather than waiting to take one giant leap.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Increasing the speed of innovation

Modern applications were born out of a necessity to deliver smaller features faster to customers. While this directly addresses only the application architecture aspect, it requires other teams to build and execute in a similar manner to be successful. In order to continuously deliver features, there is a need for all cross-functional teams to operate as a single team—a strategy referred to as One Team.

Each type of change will need its own fully automated delivery pipeline—for example, application, infrastructure, configurations, feature flags and OS patching will either need their own pipeline or need to be part of the Continuous Delivery (CD) pipeline. And capabilities like test automation and security testing need to be integrated into the pipeline so there is a high degree of confidence for changes that flow through the pipeline are ready to be deployed into production.

Key aspects of modern applications:

- Use independently scalable microservices such as serverless and containers
- Connect through APIs
- Deliver updates continuously
- Adapt quickly to change
- Scale globally
- Are fault tolerant
- Carefully manage state and persistence
- Have security built in

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What is cloud-native? Why does it matter?

Cloud-native is an evolving term. The vast amount of software that's being built today needs a place to run and all the components and processes required to build an application need to fit together and work cohesively as a system.

The Cloud Native Computing Foundation (CNCF) definition states:

Cloud-native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds.

This definition has to broadly apply to everyone, but not everyone has the same capabilities. This is known as the lowest common denominator problem. It is where you try and appeal to a broader group and their capabilities, but in doing so you also need to limit the capabilities that can be leveraged.

Amazon Web Services (AWS) goes many steps further by providing a broad set of capabilities that belong to a family called serverless. Serverless technologies are more than just AWS Lambda—these services remove the heavy lifting associated with running, managing, and maintaining servers. This lets you focus on core business logic and quickly adding value.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws marketplace

Observability for cloud-native development

As we cover the different capabilities your organization needs to acquire to go fully cloud-native, it's useful to view each one as a step in a journey.

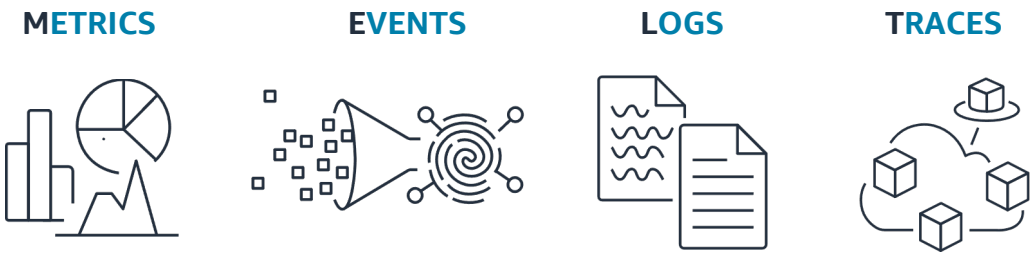
The map below is a model for how organizations typically evolve their cloud-native understanding. As your organization or team moves from stage to stage, you are gaining capabilities that make releasing new features and functionality faster, better, and cheaper. In the following sections, we'll be focusing on the capability of Observability.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

But first: What is Observability?

The concept and terminology of Observability has only recently been applied to information technology (IT) and cloud computing. The term originated in the discipline of control systems engineering, where Observability was defined as a measurement of how well a system's internal states could be inferred from its external outputs. A system is observable if its current state can be determined in a finite time period using only the outputs of the system. For such a system, all of its behaviors and activities can be evaluated based on its outputs.

However, the term has increasingly been applied to improving the performance of distributed IT systems. In this context, Observability uses four types of telemetry data: **Metrics, events, logs, and traces (MELT)**.



Analyzing MELT data provides deep visibility into distributed systems and allows teams to get to the root cause of issues and improve their system's performance.

When you instrument each component of your system to collect MELT data, you can form a fundamental working knowledge of connections—the relationships and dependencies within your system—as well as the system's detailed performance and health. This is what it means to practice Observability.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

More about MELT

Let's look at the individual types of data that make up MELT.

METRICS

Metrics are numeric measurements. They can include:

- A numeric status at a moment in time, such as CPU percent used
- Aggregated measurements, such as a count of events over a one-minute timeframe, or a rate of events-per-minute
- The types of metric aggregation are diverse—for example, average, total, minimum, maximum, sum-of-squares—but all metrics generally share the following traits:

- **They have a name** • **They have a timestamp** • **They have ≥1 numeric values**

EVENTS

Events can be user actions—such as clicking a mouse button or pressing a key—or system events such as new user created. In other words, a discrete action happening at a moment in time. AWS provides two event-focused services:

1. **Amazon CloudWatch Events** delivers a near real-time stream of system events that describe operational changes in AWS resources.
2. **Amazon EventBridge** is a serverless event bus that ingests data from your own apps, SaaS apps, and AWS services and routes that data to targets. For example, when an Amazon Elastic Container Service (Amazon ECS) task changes from running to stopped, you can have a rule that sends the event to an AWS Lambda function.

LOGS

Logs are the original data type. In their most fundamental form, logs are essentially lines of text a system or application produced when certain code blocks were executed.

Developers rely heavily on logs to troubleshoot their code and to retroactively verify and interrogate the code's execution. In fact, logs are incredibly valuable for troubleshooting databases, caches, load balancers, or older proprietary systems that aren't friendly to in-process instrumentation.

TRACES

Traces—more precisely called *distributed traces*—are samples of causal chains of events or transactions between different components in a microservices ecosystem. And like events and logs, traces are discrete and irregular in occurrence.

In distributed systems, traces are stitched together to form special events called spans. Spans help you track a causal chain through a microservices ecosystem for a single transaction. To accomplish this, each service passes correlation identifiers known as trace context to each other—this trace context is used to add attributes to the span.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What should you be observing?

Now that we've learned about MELT, let's talk about what you should be observing. There are obvious choices such as data and metrics from traditional monitoring—infrastructure metrics like CPU, memory, and disk usage. But you should expand this practice to areas that can provide context on user experience and insights that help you streamline troubleshooting. If a system errors, you should collect all associated data, such as runtime variable values, stack traces, and any other information that will help you identify the root cause faster.

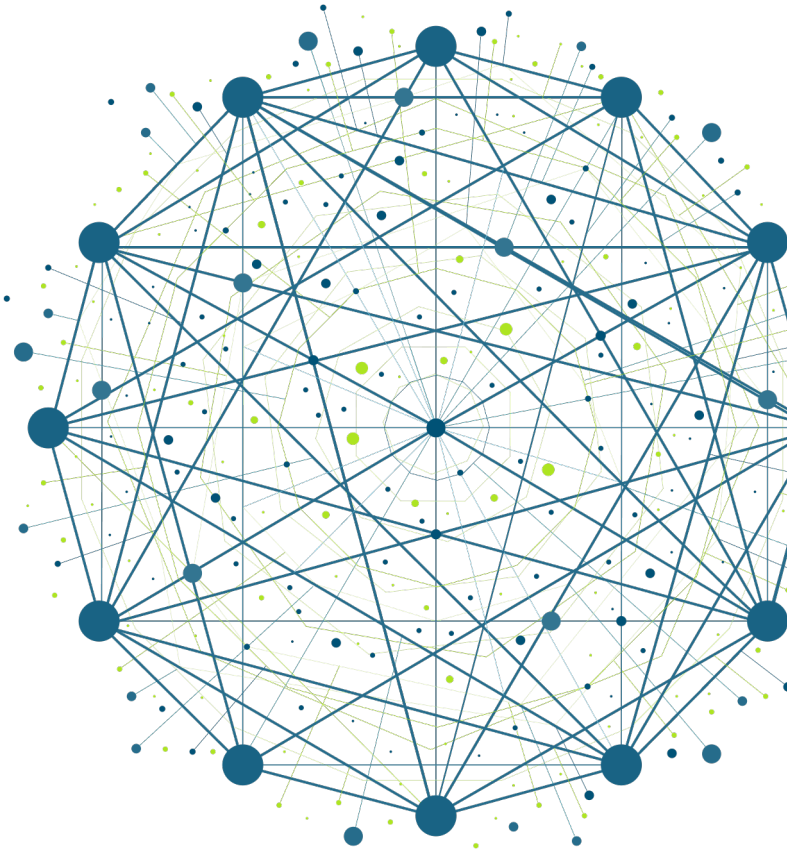
Besides standard compute information, also bring in telemetry from edge cases such as API gateways, border routers, caches, and third-party services for authentication, payments, and location. Cloud-native systems are highly distributed, so you're going to need data from any and all components that are part of the user experience.

Imagine that this image is a service, and all the different dots are components that make up the service. If a user reports an issue, how would you pinpoint in this diagram where the issue occurred?

Why Observability?

Now that we have explored the what, it's time to move into the why. Systems no longer consist of a single server in a single environment. With cloud-native, systems are composed of multiple applications and services running together to solve business problems. And the evolution of applications have made them highly distributed, which addresses scalability and resiliency, but comes at the cost of added complexity.

Observability becomes incredibly important in distributed systems because you need information from several different applications to gain a complete picture.



© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Anti-patterns: Where poor Observability hurts you

There can be too much of a good thing—and in this case, that is the misuse of Observability.

The diagram below illustrates some anti-patterns of Observability. Each anti-pattern represents a common way that development teams undermine themselves with weak capabilities for Observability. For example:

Environment inconsistency

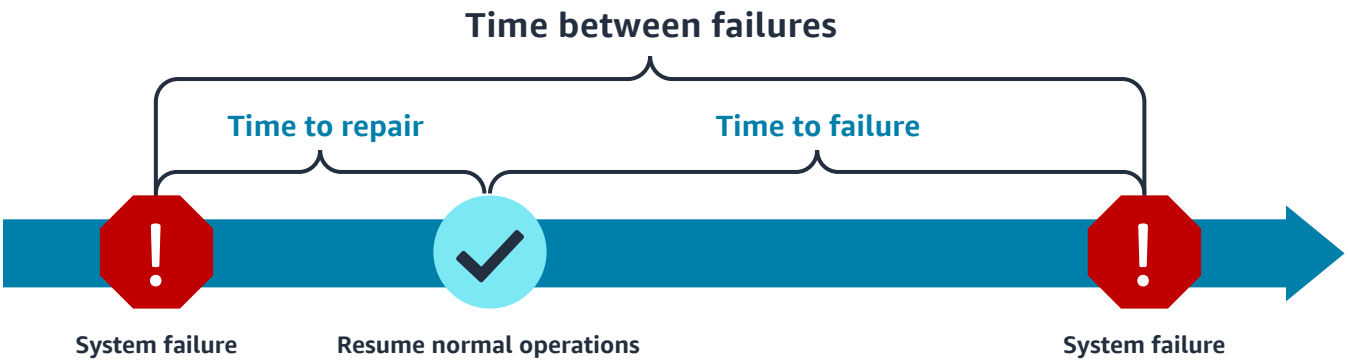
Development, test, stage, and production environments need to be consistent in terms of the resources and the processes used to observe those environments.

Unnecessary alerts

If engineers receive too many unactionable alerts, they may eventually suffer from alert fatigue and may simply ignore all alerts, even those they should be looking into.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Issues that result from poor Observability



Slower mean time to resolution (MTTR)

Not having good Observability capabilities can be disadvantageous in two important ways:

1. With business services composed of several microservices, not having full visibility into your own system and how it interacts with upstream and downstream systems can leave you with blind spots.
2. MTTR will suffer as you find yourself digging through logs and calling on engineers from other teams to help identify the cause of failure.

Poorly performing systems that you can't see

Without Observability, you have very little visibility into how systems are actually performing. You may have monitoring data that shows CPU and memory utilization, but do you have accurate information regarding calls that are blocking or where your biggest source of wait time for a method call is?

Without a platform to raise this type of system visibility, you may not know where to invest your engineering efforts to give you the biggest return on your time spent. Observability can go even further and provide visibility into the features and functionalities that end users are utilizing. This will help eliminate some technical debt by focusing attention on code areas that are heavily used by customers and removing code with no usage.

Higher costs

Poor Observability ends up costing you more in resources. Companies often over-provision resources preparing for peak traffic that may never arrive or is inconsistent. Good Observability helps with cost savings by letting you scale up or down based on traffic patterns.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Who needs Observability?

We just covered challenges related to poor Observability into systems. Now let's cover who needs Observability. The short answer is everyone. But, practically, the personas presented here are the ones that will see an immediate benefit.

QA/Test Engineer

- Wants to know how tests impact the system
- Wants contextual data about bugs
- Wants to know about test coverage
- Wants to understand how the new features are functioning
- Wants visibility into the overall health of the system as tests execute various areas of the codebase

Operations Engineer

- Wants to know if systems are meeting customers' demands
- Wants to reduce signal-to-noise ratio in alerts
- Wants to correlate data across distributed systems and get to the root cause faster
- Wants to know if systems are configured with best practices
- Wants to know if all systems are patched
- Wants visibility into thresholds and patterns; what is normal versus an anomaly

Developer

- Wants to know how software can be optimized
- Wants to understand the performance impact of changes
- Wants the ability to troubleshoot issues
- Wants to know how the pipeline and systems are performing
- Wants to be able to trace transactions through the entire system
- Wants contextual information when failures occur

Security Engineer

- Wants to know if systems are secure
- Wants to know what software has been deployed and where
- Wants to know patch versions
- Wants to ensure compliance with corporate standards and best practices
- Wants to make sure no new vulnerabilities are being introduced with applications as they are being developed
- Wants real-time analysis of security alerts generated by applications and network hardware

Leadership

- Wants visibility into business KPIs
- Wants to know about issues and risks
- Wants assurances that systems are performant and cost-optimized
- Wants visualizations and reports that reduce signal to noise ratio

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws marketplace

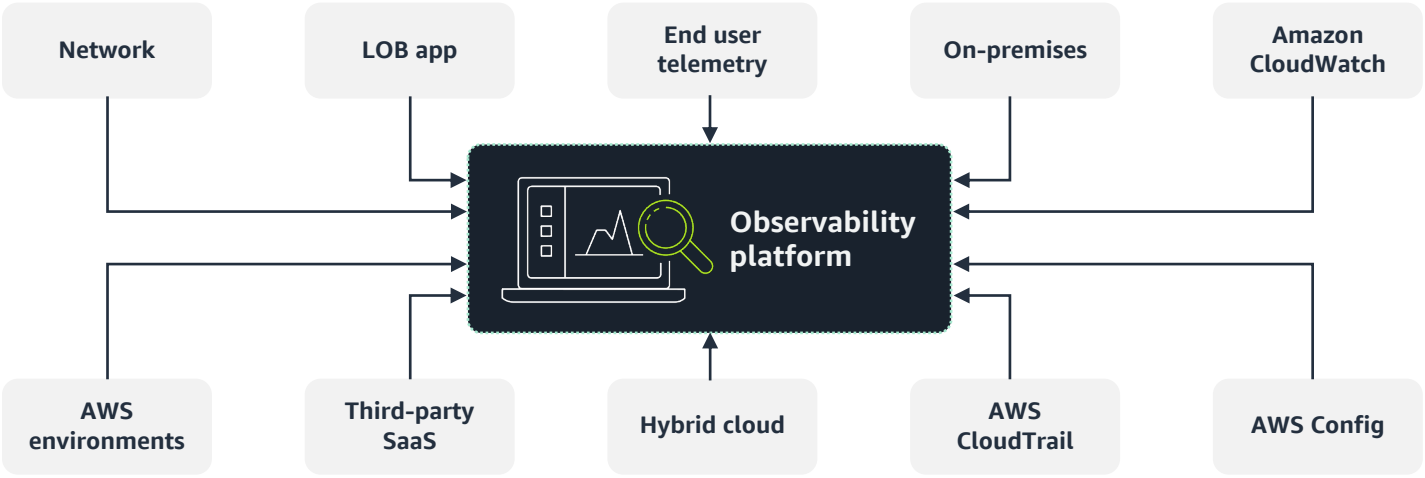
Where to start?

From a high level, we now know what Observability is, and we also know why we need Observability. Now let's talk about how to start building your Observability capabilities.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Data sources

To make your journey more manageable, start with an application or service you want to observe. Armed with different personas and their needs, work backwards from there and determine what data sources will get you the information you need. In this diagram are just a few examples of common data sources to think about. After you compile a list of the data sources, you'll be in a better position to determine which Observability platform will best serve your needs.



Unified logging and standardization

Once you have a good understanding of your data sources, you'll want to think about how to get that data into a common format that can be shared across different systems. It's not uncommon that systems don't standardize on how logs are written or what metrics are collected and how they are aggregated. This makes it impossible to trace transactions across services and difficult to compile analytic and forensic information. Because of this, it's important to standardize on the way MELT data is created, published, and shared.

PROBLEM: Log data in various formats

- Cross-service tracing impossible
- Complexity for monitoring, forensics, analytics

SOLUTION: Standardize the log data model

- Annotate log records with distributed tracing states
- Adopt OpenTracing (http://opentracing.io/)
- Provide SDK that supports major languages
- Integrate with vendor application performance monitoring (APM) products

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What is OpenTelemetry?

OpenTelemetry is a CNCF incubating project, formed through a merger of the OpenTracing and OpenCensus projects. To help standardize Observability, the OpenTelemetry project provides a set of APIs, SDKs, and agents to enable developers to instrument their application once and send correlated traces and metrics to a destination of their choice. It supports SDKs in 11 languages and auto-instrumentation agents in Java. JavaScript, Python, .NET, and more are actively being developed.

- **An Observability framework for cloud-native software**
- **You use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) for analysis in order to understand your software's performance and behavior**
- **Supports 11 languages and growing**

AWS Distro for OpenTelemetry (ADOT)

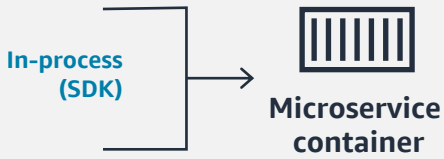
ADOT is an open-source distribution downstream of OpenTelemetry. With ADOT, you can collect metadata from your AWS resources and managed services to correlate application performance data with underlying infrastructure data, reducing the mean time to problem resolution.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

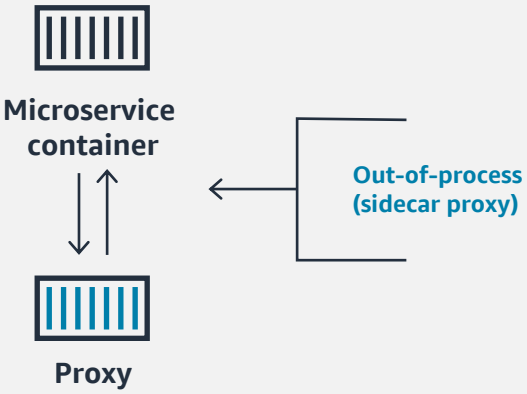
Observability for containers

To add Observability to containers, customers are using one of two options:

Option 1



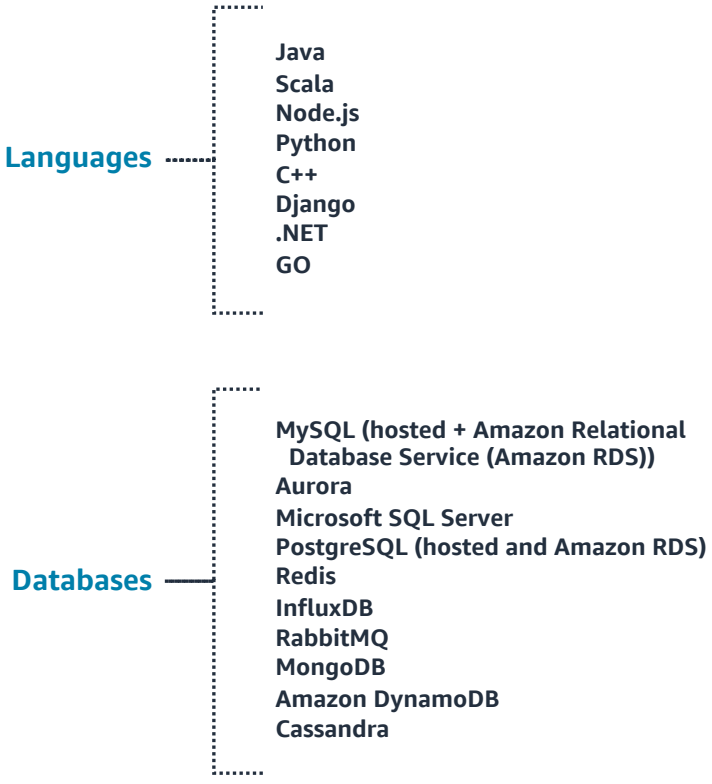
Option 2



Option 1 is to use an in-process SDK and add code that is embedded with the application. This method is also referred to as agentless.

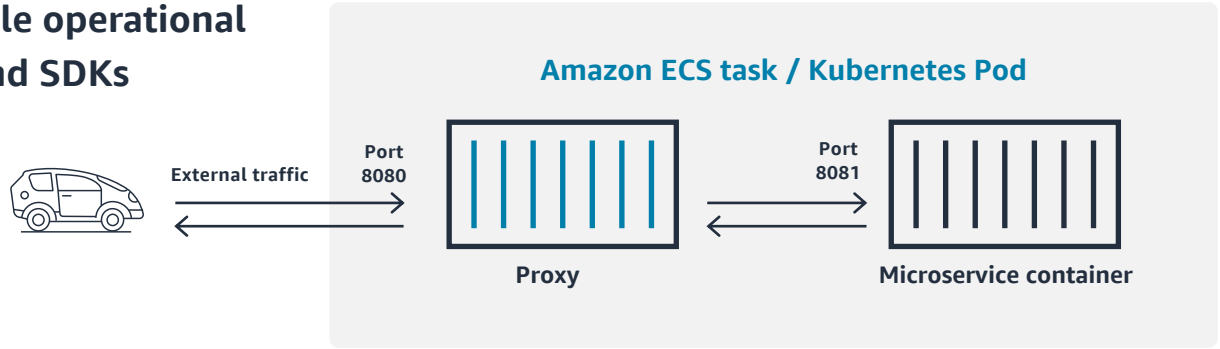
- **SDK maintenance**
- **Application code changes**
- **Retrofitting**
- **Unknown dependencies**

This option tends to be more heavyweight and requires application code changes and retrofitting. It's language-dependent, so you would need to ensure the language you choose is supported. Plus, the resources used need to support this style of instrumentation.



© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Decouple operational logic and SDKs



Option 2 is to use an out-of-process method also known as a sidecar. A sidecar runs as a separate proxy container that manages all the communication outside of the microservice container. The sidecar method allows for decoupling of the operational logic and the SDKs.

Benefits of the sidecar pattern is that it provides a straightforward way to implement logging, tracing, and metrics without having to retrofit code or apply an SDK. They are language-independent and can be used as a method to capture telemetry data for applications and services where you might not have access to instrument the programming code.

But sidecars can be difficult to scale, which is why we recommend AWS App Mesh for service-level communications. AWS App Mesh is a service mesh that provides a dedicated infrastructure layer for your applications. It allows you to transparently add capabilities like Observability, traffic management, and security, without adding them to your own code.

Observability for AWS Lambda

AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers. Observability for AWS Lambda is instrumented via ADOT.

ADOT is provided as an AWS Lambda layer, which is an archive containing additional code such as libraries, dependencies, or even custom runtimes. Using layers can make it faster to deploy applications.

ADOT Lambda layers provide a plug-and-play user experience by automatically instrumenting a Lambda function. In this method, OpenTelemetry is packaged together with an out-of-the-box configuration for AWS Lambda in an easy-to-set-up layer. Users can enable and disable OpenTelemetry for their AWS Lambda function without changing code.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The four golden signals

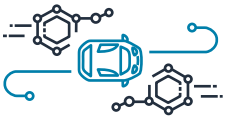
We've talked about instrumentation of containers and AWS Lambda. Now let's transition to what signals in information we need to include in our Observability solution.

The four golden signals are key metrics used to monitor the health of a service and the underlying systems. We will explain what they are, and how they can help you improve service performance.

Latency

Latency is the time it takes a system to respond to a request. Both successful and failed requests have latency and it's vital to differentiate between the latency of both. For example, an HTTP 500 error triggered because of a connection loss to the database might be served very quickly.

However, since HTTP 500 is an error indicating a failed request, factoring it into the overall latency will lead to misleading calculations. Alternatively, a slow error can be worse as it factors in even more latency. Therefore, instead of filtering out errors altogether, keep track of the error latency. Define a target for a good latency rate and monitor the latency of successful requests against failed ones to track the system's health.



Traffic

Traffic is the measure of how much your service is in demand among users. How this is determined varies depending on the type of business you have. For a web service, traffic measurement are generally described in HTTP requests per second, while in a storage system, traffic might be described in transactions per second or retrievals per second.

By monitoring user interaction and traffic, engineers can usually figure out users' experience with the service and how it's affected by shifts in the service's demand.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Errors

Error is the rate of requests that fail in any of the following ways:

Explicitly For example, HTTP 500 internal server error

Implicitly For example, HTTP 200 success response coupled with inaccurate content

By policy For example, as your response time is set to one second, any request that takes over one second is considered an error

Engineers can monitor all errors across the system and at individual service levels to define which errors are critical and which are less severe. By identifying that, they determine the health of their system from the user's perspective and can take rapid action to fix frequent errors.

Saturation

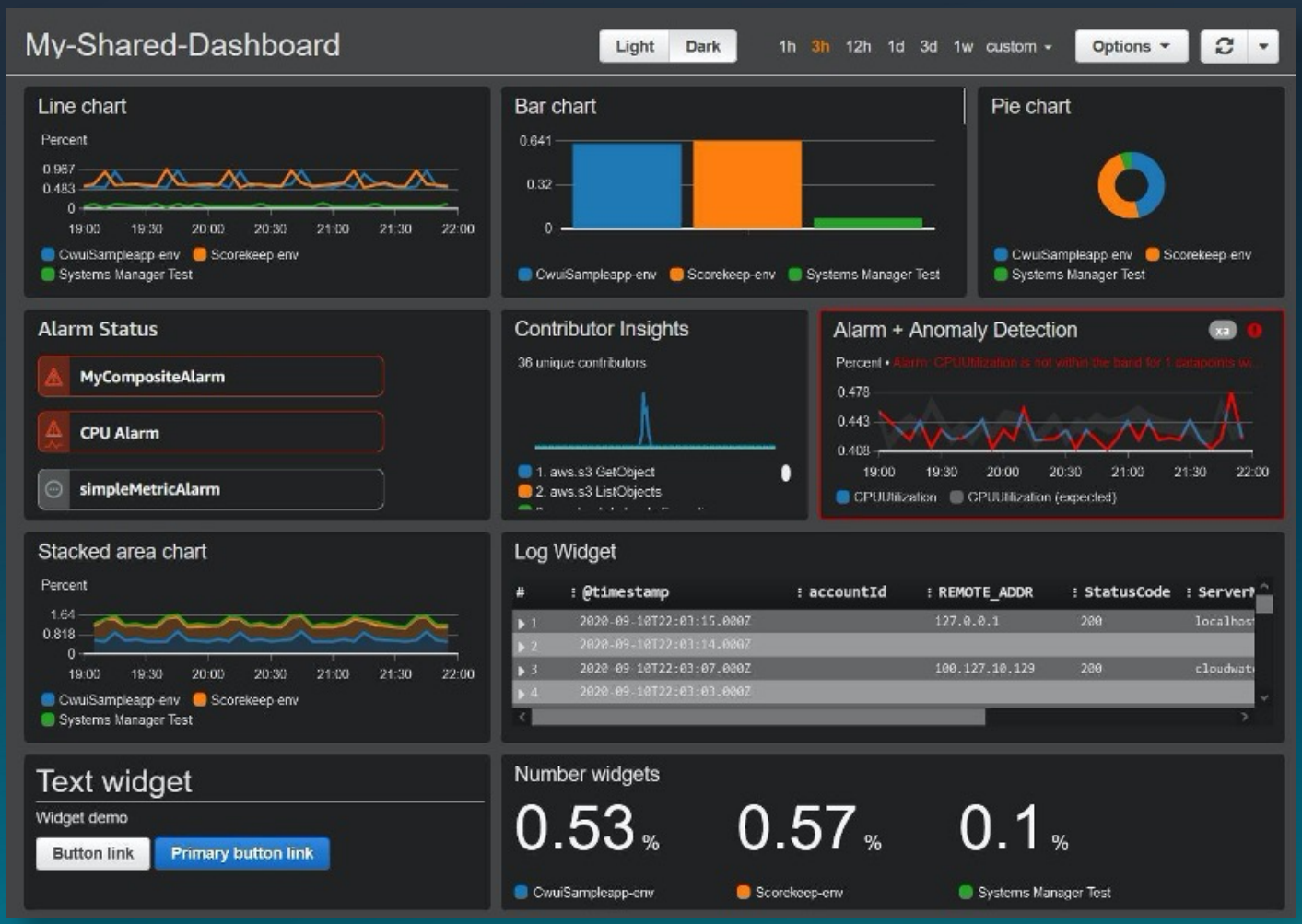
Saturation refers to the overall capacity of the service or how "full" the service is at a given time. It signifies how much memory or CPU resources your system is utilizing. Many systems start underperforming before they reach 100 percent utilization. Therefore, setting a utilization target is critical as it will help ensure service performance and availability for users.

An increase in latency is often a leading indicator of saturation. Measuring your 99th percentile response time over a small time period can provide an early indicator of saturation. For example, a 99th percentile latency of 60 milliseconds indicates that there's a 60-millisecond delay for every one in 100 requests.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Dashboards and visualizations

This is a great spot to briefly touch on the concept of dashboards. A simple definition: A dashboard is a visual display of all of your data. While it can be used in all kinds of different ways, its primary intention is to present information—such as KPIs—at a glance. A good practice for dashboards is to ensure these can be defined using Configuration as Code (CaC).



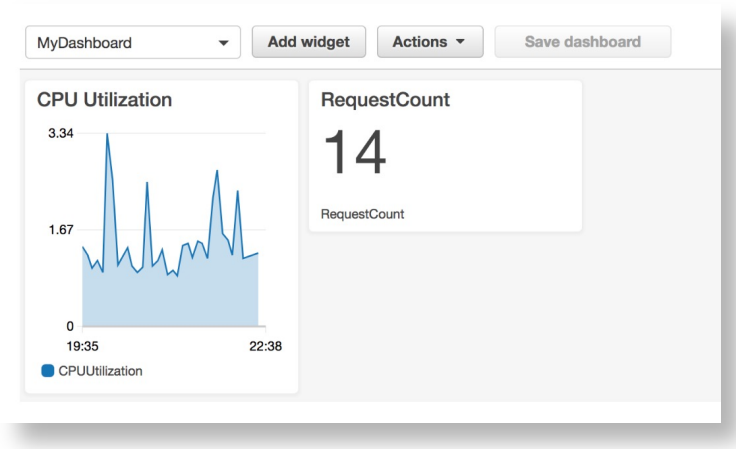
© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Codifying dashboards

In the below diagram, you'll see AWS CloudFormation—an Infrastructure as Code (IaC) service that allows you to easily model, provision, and manage AWS and third-party resources—on the left-hand side of the screen, which produces the dashboard on the right. Defining configuration in AWS CloudFormation or Terraform lets you codify dashboards for consistency across environments.

CloudFormation snippet

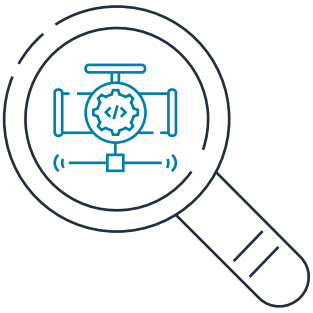
```
BasicDashboard:
Type: AWS::CloudWatch::Dashboard
  Properties:
    DashboardName: MyDashboard
    DashboardBody:
      ...
"CPUUtilization","InstanceId"…
```



Visualizations

Visuals are persuasive and convey important information quickly. Use visualizations to highlight trends, patterns, and anomalies. In this example, the eye is drawn to the orange dot on the left-hand portion of the map, which allows an operator to quickly focus on an area that needs attention.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Visibility into the pipeline

Observability of your Continuous Integration/ Continuous Delivery (CI/CD) pipeline and your application stack is critical, as it gives you important early insights into bottlenecks in your pipeline, application performance issues, runtime application errors, test results, system errors, security vulnerabilities, and many more. Issues when caught at earlier stages of development are easier and cheaper to fix. With visibility into the pipeline, you get the following benefits:

Know the performance of pipelines—you can't improve what you can't measure

Identify bottlenecks like long-running test, out of memory errors, and long-running database (DB) queries

Visibility into compile time and runtime errors

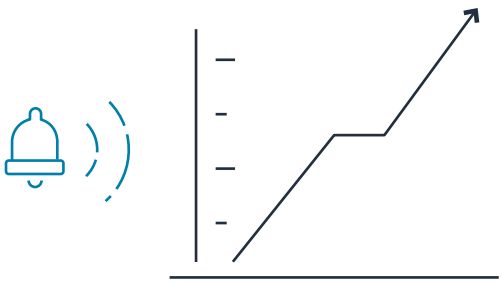
Value stream mapping—with visibility into your pipeline, you can identify inefficiencies in the system, bottlenecks, track stories in the pipeline, team capacity, and many more

Alerts

The reality is that no matter how good a dashboard or a visualization is, no engineer is going to spend all their time staring at it. This is why you need alerts. Alerts draw your attention to something you need to look at —but with the caveat that alerts need to be meaningful and actionable, otherwise engineers will get alert fatigue and may ignore them (one of our anti-patterns for Observability).

Alert on significant events

Alerts don't always have to be about negative impacts. Alerts should also include business milestones such as meeting revenue targets or launch of a new promotion. Setting up these types of alerts adds important contextual information that can help your organization make better business decisions.



© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Observability of third-party services

With modern applications being hybrid and distributed, it is important to get visibility into third-party services such as email, payments, location services, delivery services, and others to get a complete picture as you are debugging your application.

A good example would be debugging a failed order. Was it due to a third-party payment gateway error? An issue with availability? An order confirmation email failure? Without proper visibility into third-party systems, it's almost impossible to debug and get to the root cause of an issue.

Remediation

Ultimately, Observability is supposed to help you remediate faster, and auto-remediation is the ultimate capability for SRE/Ops engineers. A simple example of auto-remediation is auto-scaling. For example, when an Amazon Elastic Compute Cloud (Amazon EC2) health check fails, if the Amazon EC2 instance is part of an auto-scaling group, the auto-scaling replaces the instance without impacting end-user experience.

AIOps and auto-remediation

No cloud-native Observability solution is complete without artificial intelligence for IT operations (AIOps). No human can realistically be expected to make sense of the massive amounts of data that streaming in—the sheer volume and disparate types of information makes it impossible.

This is where AIOps comes in.

AIOps takes all that data and automatically correlates it for you to help improve the signal-to-noise ratio, surfacing those data points and alerts to which you really need to pay attention. AIOps is a relatively new area of Observability that is beginning to expand into auto-remediation to not only detect issues, but to also automate the remediation of those issues.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Key takeaways for Observability

To sum up what we have discussed so far:

The components of MELT

—metrics, events, logs, and traces—and how these make up the data types for Observability

The importance of Observability

to improve mean time to resolution, improve system performance, and optimize for cost

The needs of different personas

and how Observability helps provide specific information for each role

The importance of standardization

using tools such as AWS Distro for OpenTelemetry (ADOT) and auto-instrumentation for containers and AWS Lambda functions

The four golden signals

and how to use these to inform service performance improvements

Leverage advanced capabilities in AIOps

and how to use these to inform service performance improvements

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws marketplace

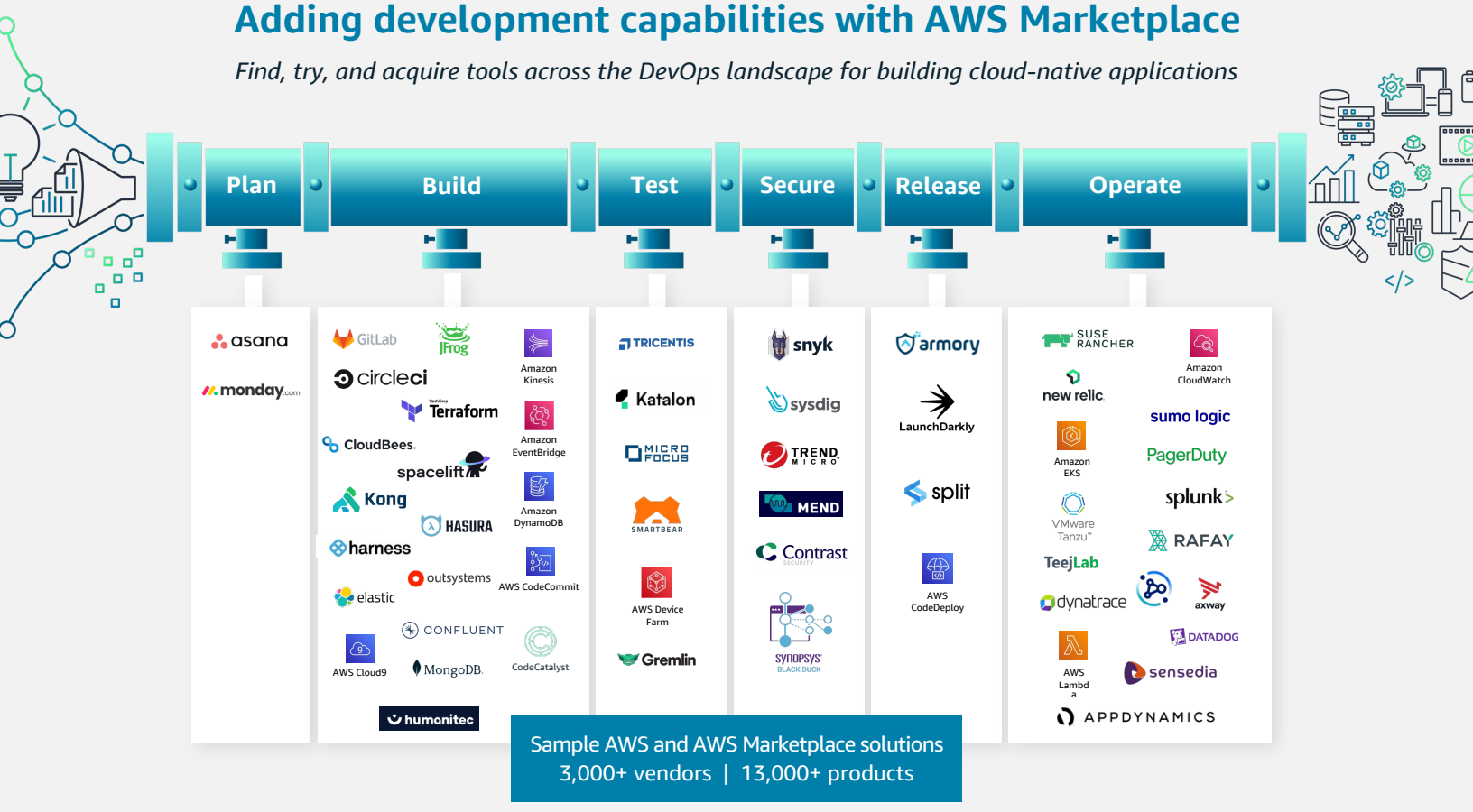
Tools for building cloud-native

In this section, we'll look at some of the best-fit tools you could use to achieve the outcomes discussed in the previous section. At AWS, we've long been believers in enabling builders to use the right tool for the job—and when you build with AWS, you're provided with choice. You can build using the native services AWS provides or use AWS Marketplace to acquire third-party software offered by AWS Partners to take away the heavy lifting and allow your development teams to focus on delivering value to customers.

Let's take a deeper look at two key components at this stage of your cloud-native journey: an artifact repository to store and use approved software versions, consistent environments in which to effectively experiment, build, and test, and observability throughout the entire process.

Adding development capabilities with AWS Marketplace

Find, try, and acquire tools across the DevOps landscape for building cloud-native applications



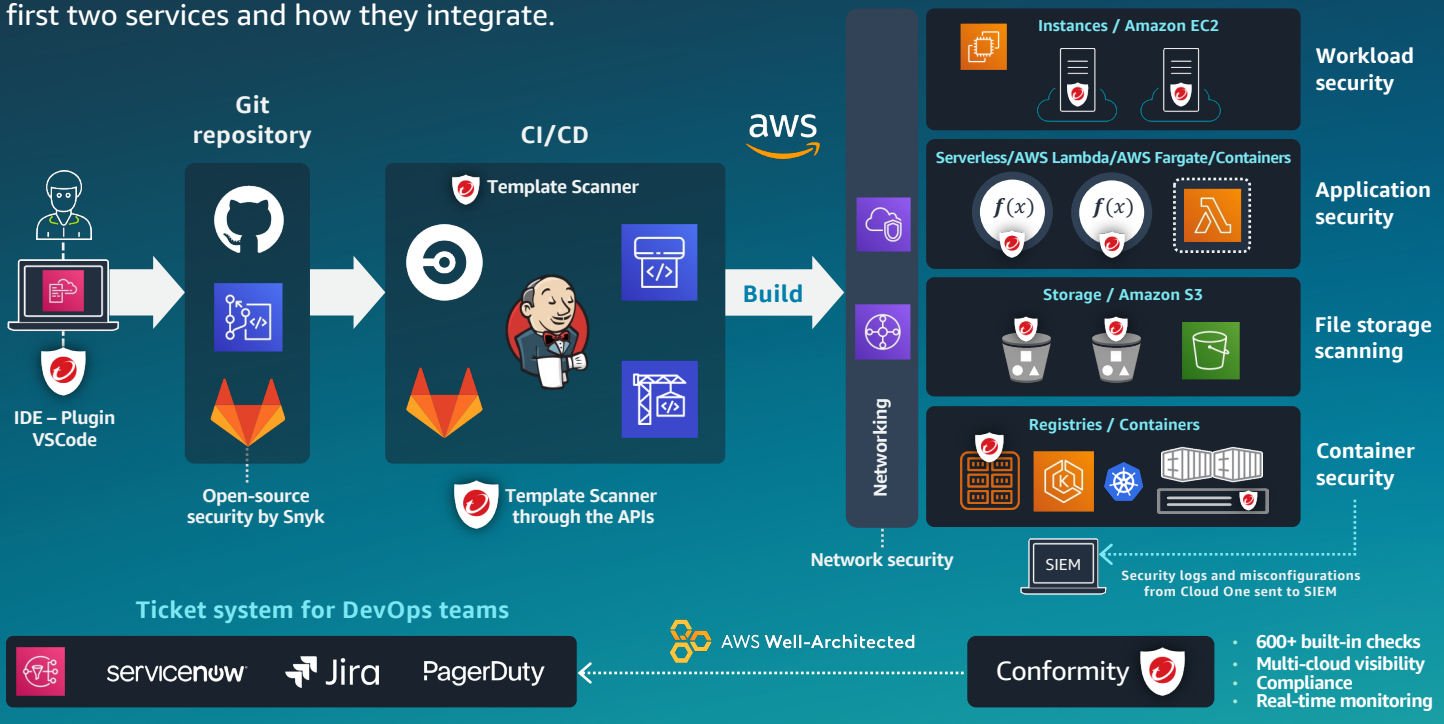
Sample AWS and AWS Marketplace solutions
3,000+ vendors | 13,000+ products

AWS Marketplace is a cloud marketplace that makes it easy to find, try, and acquire the tools you need to build cloud-native. More than 13,000 products from over 3,000 Independent Software Vendors are listed in AWS Marketplace—many of which you can try for free and, if you decide to use, will be billed through your AWS account.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws marketplace

For our example architecture, we'll use **Trend Micro** for Security Observability, **Sumo Logic** for end-to-end Observability of the application, and **ADOT** to help with standardization. Let's take a high-level look at the first two services and how they integrate.



TREND MICRO

[Try it in AWS Marketplace ›]

[Start a hands-on lab ›]

Trend Micro Cloud One can protect web applications and APIs regardless of what programming environment you are running on, what cloud strategy is used, or what architecture was employed. Furthermore, Trend Micro designed the product to handle a very broad array of threat vectors such as malicious content, OWASP top 10, bot attacks, as well as business logic events. Trend Micro even embedded a scripting engine to extend the platform to cover advanced use cases.

sumo logic

[Try it in AWS Marketplace ›]

[Start a hands-on lab ›]

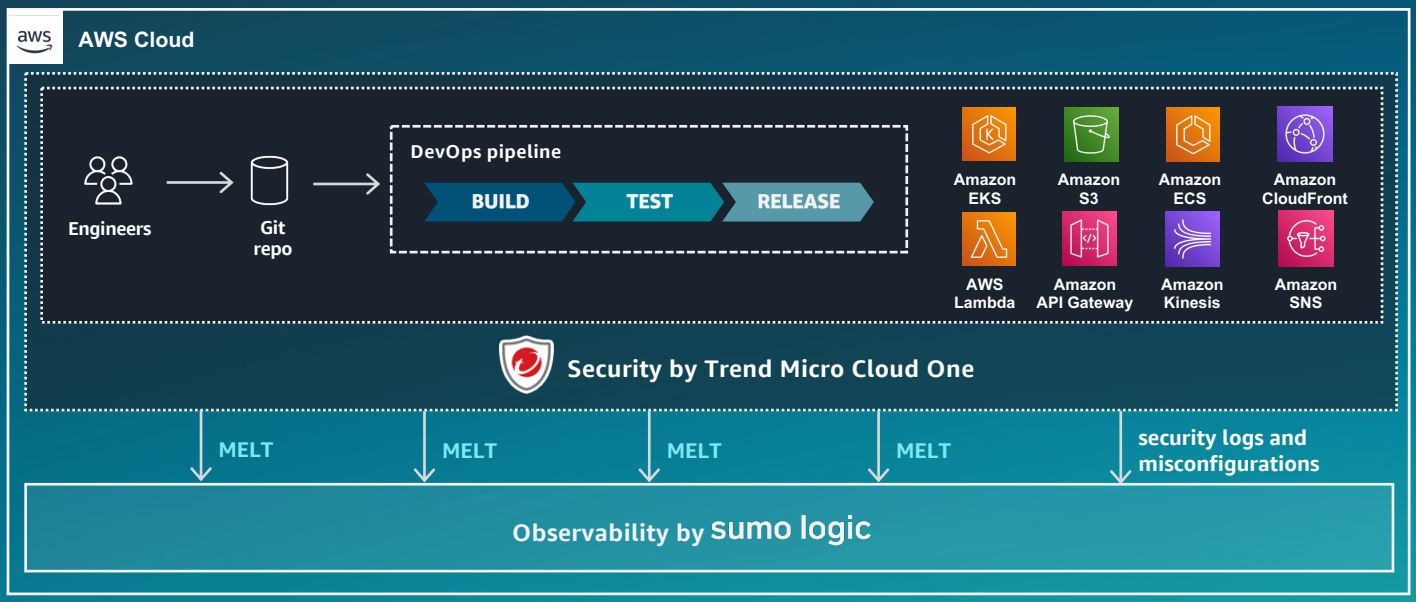
Sumo Logic unifies visibility across key AWS services such as Amazon Elastic Kubernetes Service (Amazon EKS), AWS Lambda, Amazon Elastic Container Service (Amazon ECS), Amazon Relational Database Service (Amazon RDS), Amazon ElastiCache, Amazon API Gateway, and Amazon DynamoDB. With Sumo Logic, you can easily navigate from overview dashboards into account, AWS Region, Availability Zone, or service-specific views. Intuitive navigation ensures teams can quickly resolve issues, minimize downtime, and improve performance.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws marketplace

Observability architecture

Below, you'll find a high-level architecture using Trend Micro and Sumo Logic products that enable teams and organizations to accelerate their cloud-native capabilities with Observability.



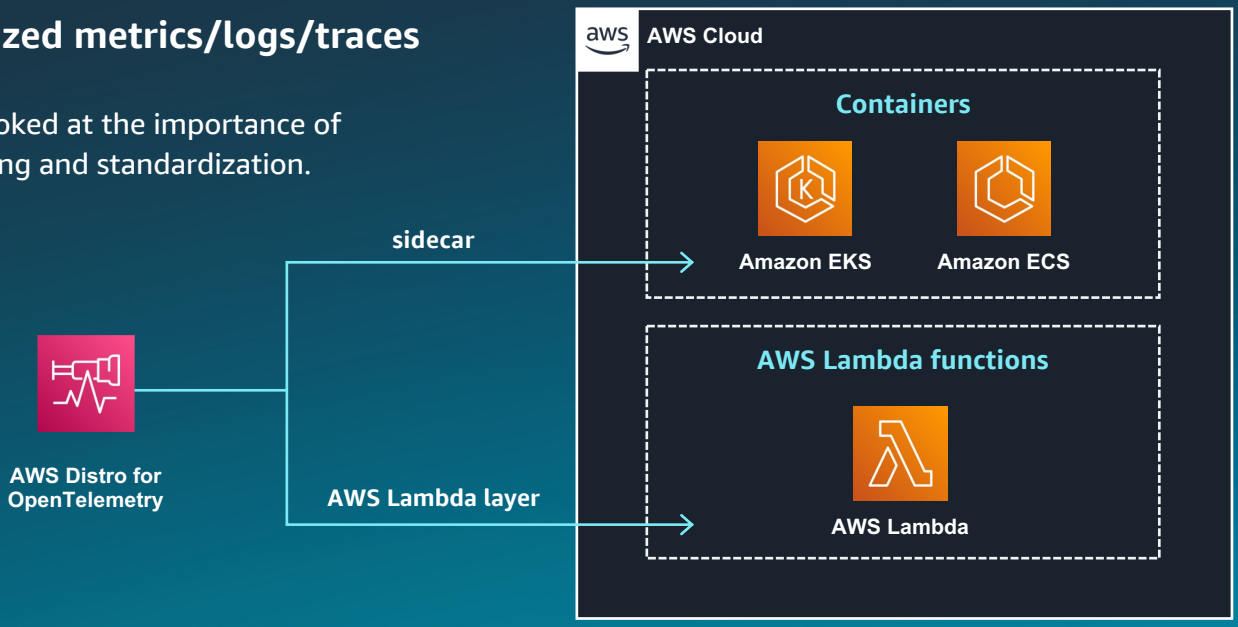
On the top left, engineers develop software by composing applications and storing artifacts in a Git repository. The applications go into a DevOps CI/CD pipeline where they are released to different environments. In each environment, the application is tested and promoted to the next environment where the resources and application are deployed to production.

Trend Micro provides security at each of these different stages. An integrated development environment (IDE) plugin surfaces information and findings while a developer still has context. The source repository is continuously scanned, and issues are surfaced with instructions for the developer to resolve the issue and associated pull-request for one-click mitigation.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Standardized metrics/logs/traces

Earlier we looked at the importance of unified logging and standardization.



In this diagram, you start with ADOT—for containers, you use the sidecar pattern and for AWS Lambda you use AWS Lambda layers. As a reminder, OpenTelemetry is a collection of tools, APIs, and SDKs. You can use it in your solution to instrument, generate, collect, and export telemetry data such as metrics, logs, and traces to help you analyze your software's performance and runtime behavior.

OpenTelemetry application instrumentation

Instrumenting an application for traces, metrics, and logs is typically a straightforward process: Install the SDK, bootstrap the instrumentation, and launch the application with the OpenTelemetry wrapper.

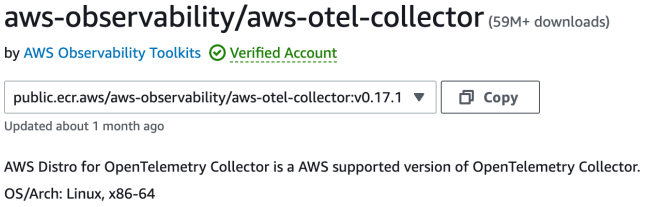
```
# Install required packages for instrumentation and to support
tracing with AWS X-Ray
$ pip install opentelemetry-distro[otlp]>=0.24b0 \
              opentelemetry-sdk-extension-aws~=2.0 \
              opentelemetry-propagator-aws-xray~=1.0
# Automatically install supported Instrumentors for the
application's dependencies
$ opentelemetry-bootstrap --action=install
$ OTEL_PROPAGATORS=xray \
OTEL_PYTHON_ID_GENERATOR=xray \
opentelemetry-instrument python3 ./path/to/your/app.py
```

The example diagram uses auto-instrumentation for Python. There are auto-instrumentation libraries for Java, Node.js, Ruby, .NET, and Python. Plus, there are SDKs for all major languages, frameworks, and application servers. A few examples include Spring, Express, and Flask. This example focuses on traces, but for metrics and logs the process is similar: Import the appropriate SDK, set some environment variables, and call metric or logging methods for OpenTelemetry.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

OpenTelemetry for Amazon EKS

After instrumenting the application, you will need a collector and exporter. Luckily this is also fairly straightforward when using ADOT.

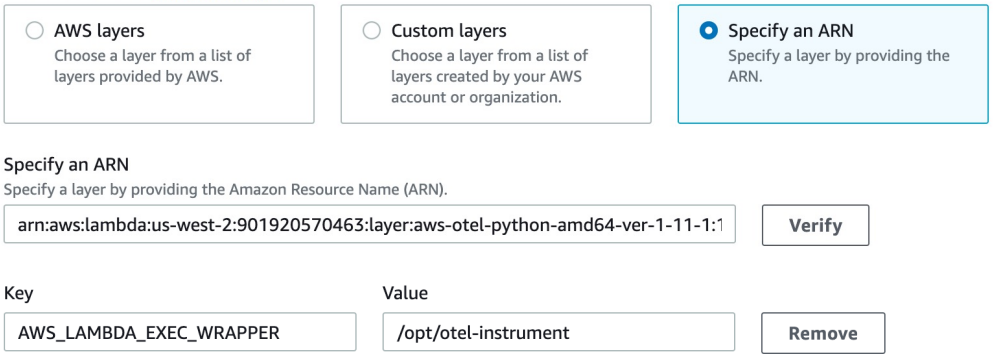


Amazon EKS has a feature for add-ons: Software that provides operational capabilities in an Amazon EKS cluster that make it easy for users to operate production-grade clusters in a stable and secure manner. ADOT support for Amazon EKS is done through an add-on.

You would launch this container and associated configuration as a sidecar on Amazon EKS. Then any logs, metrics, and traces would be configured to send data to the collector, which would take care of routing it to the configured destination.

OpenTelemetry for AWS Lambda

For AWS Lambda, AWS provides a managed layer for ADOT Python, which is a plug-and-play user experience by automatically instrumenting an AWS Lambda function.



To enable auto-instrumentation for an AWS Lambda function:

1. Open the AWS Lambda function you intend to instrument in the AWS console
2. In the Layers in Designer section, choose Add a layer
3. Under specify an ARN, paste the layer ARN, and then choose Add
4. Add the environment variable for the LAMBDA EXECTION WRAPPER and set it to /opt/otel-instrument

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Security guardrails with Trend Micro Cloud One

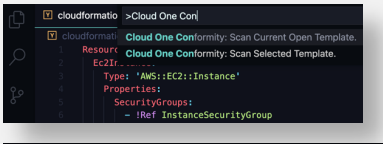
In the solution architecture we've been reviewing, Trend Micro Cloud One's security guardrails start with the developer and are applied at every step of the software delivery lifecycle. At the workstation, an IDE plugin provides fast feedback for developers. In the code repository, Trend Micro is continuously scanning on check-ins, such as pull-requests, as well as any time new vulnerabilities are added to the list of threats.

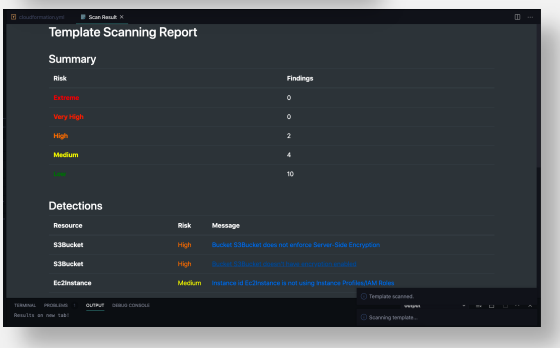
Within CI/CD, common vulnerabilities and exposures are checked in the application, container configuration, and images. Additionally, best practices are also checked as part of Configuration as Code (CaC) scanning when using Micro Trend CloudFormation, CDK, or Terraform.

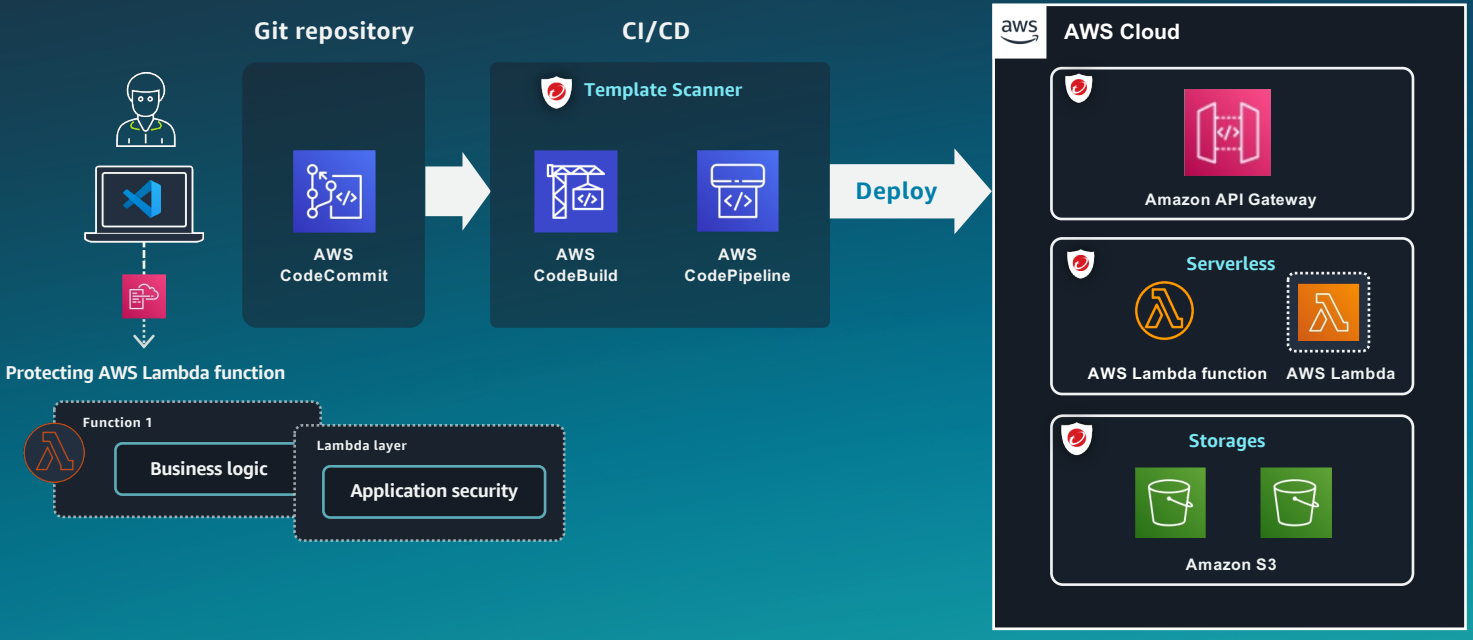
Even after applications are deployed, Trend Micro continuously checks environments against AWS best practices. This ensures that drift and risks outside the CI/CD pipeline are flagged and mitigated.

IDE security plugin

Here is an example of a plugin that has been initiated, performed a manual scan of a template, and issued an associated report. Notice in the report that risks are ranked by risk level. The resource flagged has additional information and details on how to mitigate or correct the risk.







© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

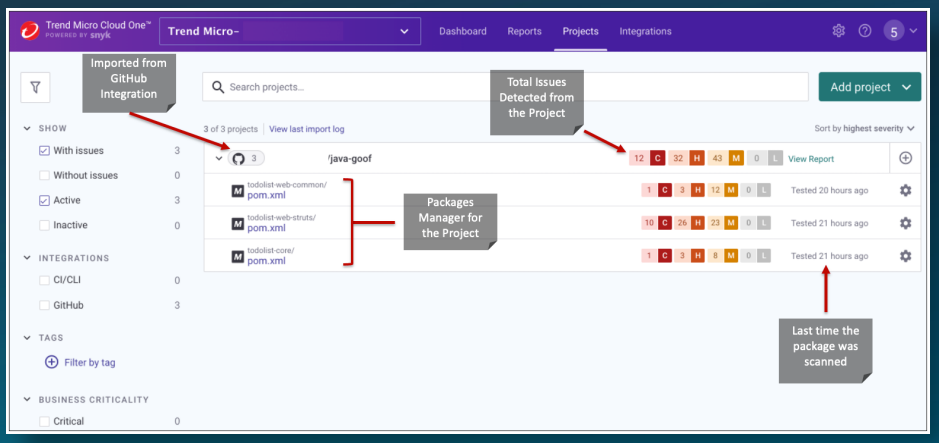
aws marketplace

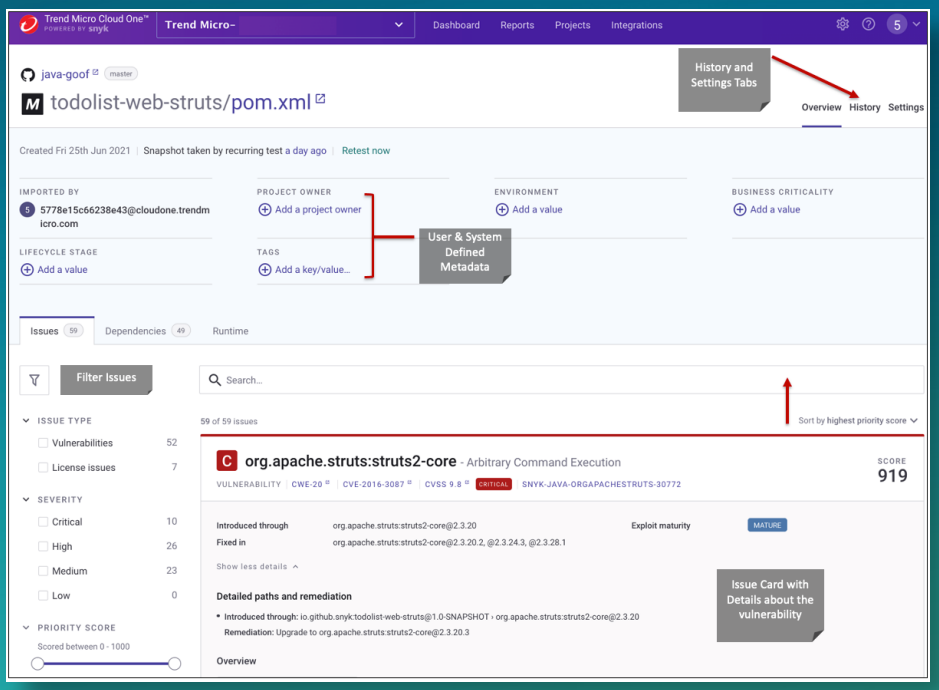
Open-source vulnerabilities

Trend Micro also integrates with your Git repository to provide Observability into threats that have been committed to code.

In this example, Trend Micro Cloud One automatically detects the language and package manager for the project. Risk levels are applied and guidance for correcting the issue is provided for each risk identified.

Details about the risk are on the top screen.



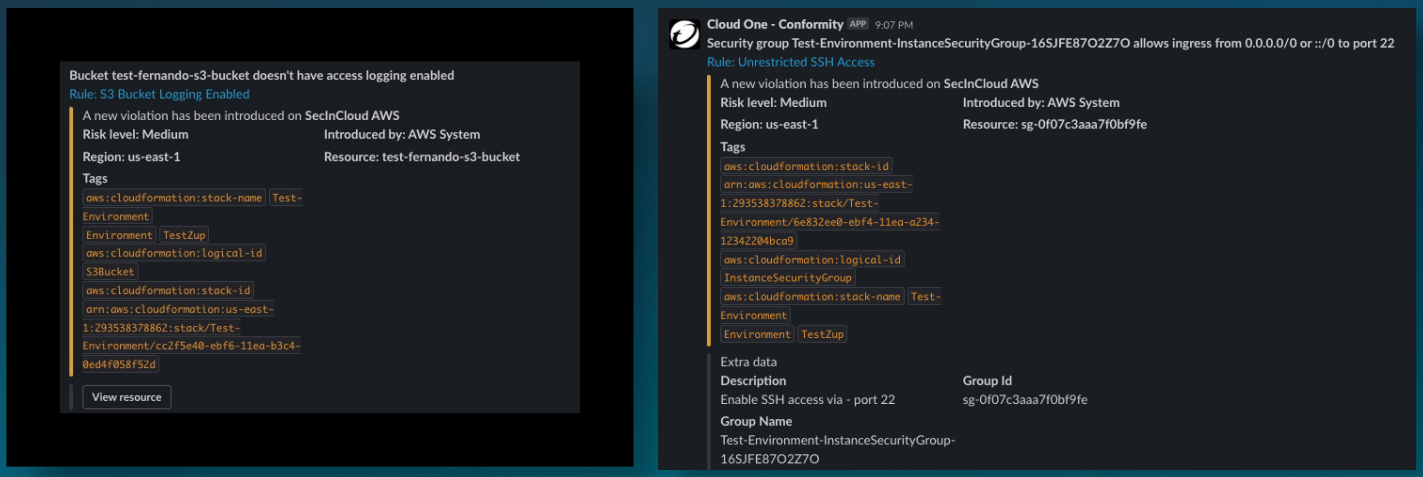


Proactive fixes

As a developer, you know that getting a list of a few dozen risks doesn't really help much if you don't have the time to address them. This is why Trend Micro issues a pull-request with the fix already implemented in code, so you only need to check the pull request, review the change, and merge.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

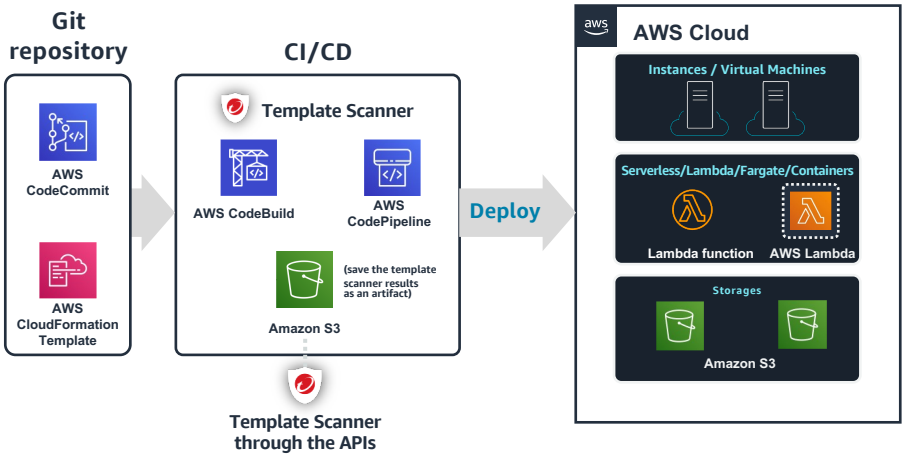
Trend Micro Cloud One built-in security checks



Here is an example of a notification sent by the Trend Micro Cloud One platform for resources that failed the best practice guardrail checks. On the left is a notification that the configuration for an Amazon Simple Storage Service (Amazon S3) bucket is missing access logging.

The right-hand example shows a security group has allowed connections to SSH port 22 from anywhere. The notice raises an alert to an engineer to correct the risk. Trend Micro has over 600 built-in checks.

Fast feedback to development team

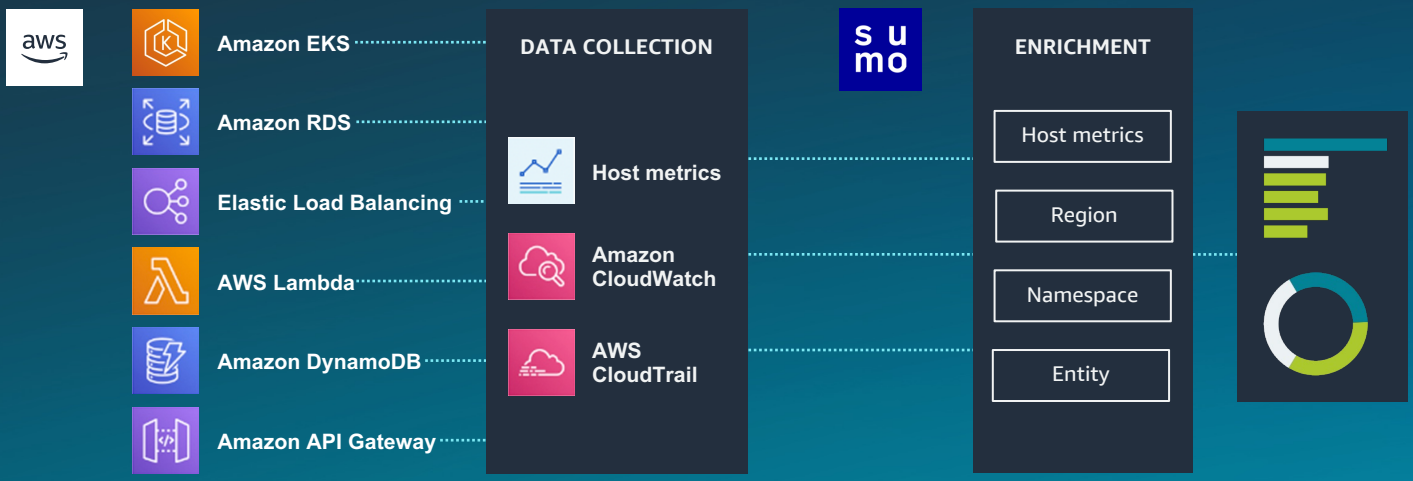


The Trend Micro Cloud One platform also includes checks in the CI/CD pipeline. Checks in the pipeline provide fast feedback to developers about issues and how to address those issues. The CI/CD checks can also test container templates, container images, AWS CloudFormation templates, Terraform templates, open-source vulnerabilities, and more.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Sumo Logic quick setup

Now that we've looked at security Observability and how it's incorporated into the solution, let's turn our attention to end-to-end application Observability.



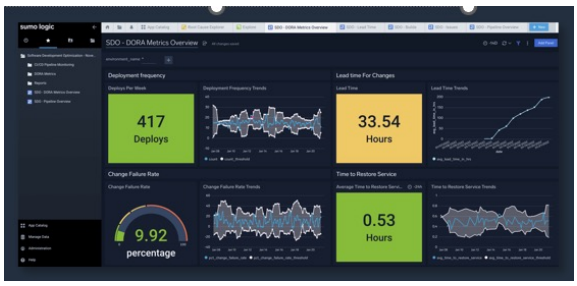
Setting up Sumo Logic to quickly provide insights is done through an AWS CloudFormation template, which is a simple form that sets up the collection of data from an AWS account. Data from Amazon CloudWatch and AWS CloudTrail are streamed to Sumo Logic and tagged with metadata about where on AWS the data originated. Because the setup is done through AWS CloudFormation, collection of newly provisioned AWS resources is automated to save your team valuable time.

You might be wondering why the example solution has Sumo Logic even though we have Amazon CloudWatch. The reason is that applications in cloud-native are composed of many different services from both AWS and AWS Partners. To get an end-to-end picture you need to bring in telemetry data from all different sources—not just AWS. As an example, cloud-native applications that take credit card payments, tend to use a third-party gateways.

If you are troubleshooting or working to improve performance, you'll want to have visibility into those different services, whether they are on AWS or third-party services. Sumo Logic enables you to centralize and correlate all the different data sources into one tool.

Software development optimization

Out of the box, Sumo Logic provides several dashboards to get you started. On this screen is an example of the DevOps research and assessment (DORA) dashboard. The research has identified four key metrics, which are deployment frequency, lead time for changes, mean time to recover, and change failure rate. These metrics provide teams with a way to benchmark their performance in comparison to other companies.

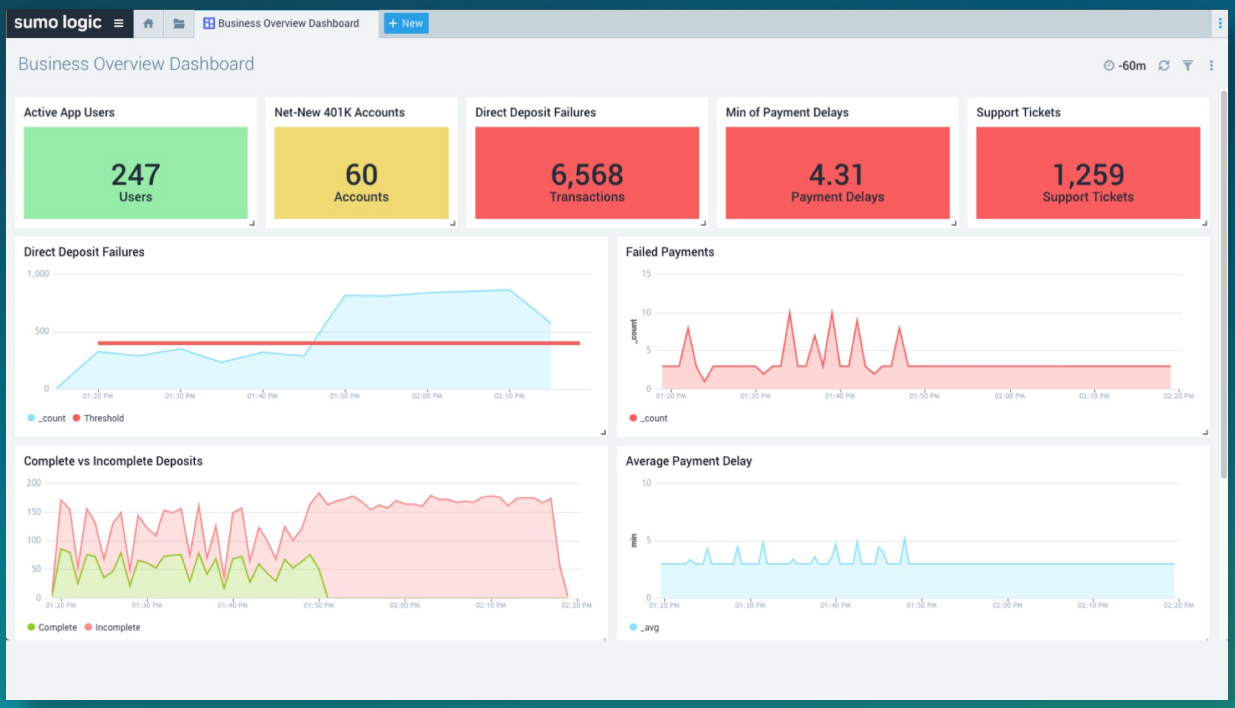


© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws marketplace

Observability for developers

Sumo Logic's Observability platform for developers reduces downtime and solves customer-impacting issues faster by pulling all of your application data—including logs, metrics, and traces—from across the entire development lifecycle. Sumo Logic's integrated analytics platform seamlessly correlates system issues.



The example on this screen shows several KPIs with color-coded visuals to draw attention to areas that merit investigation.

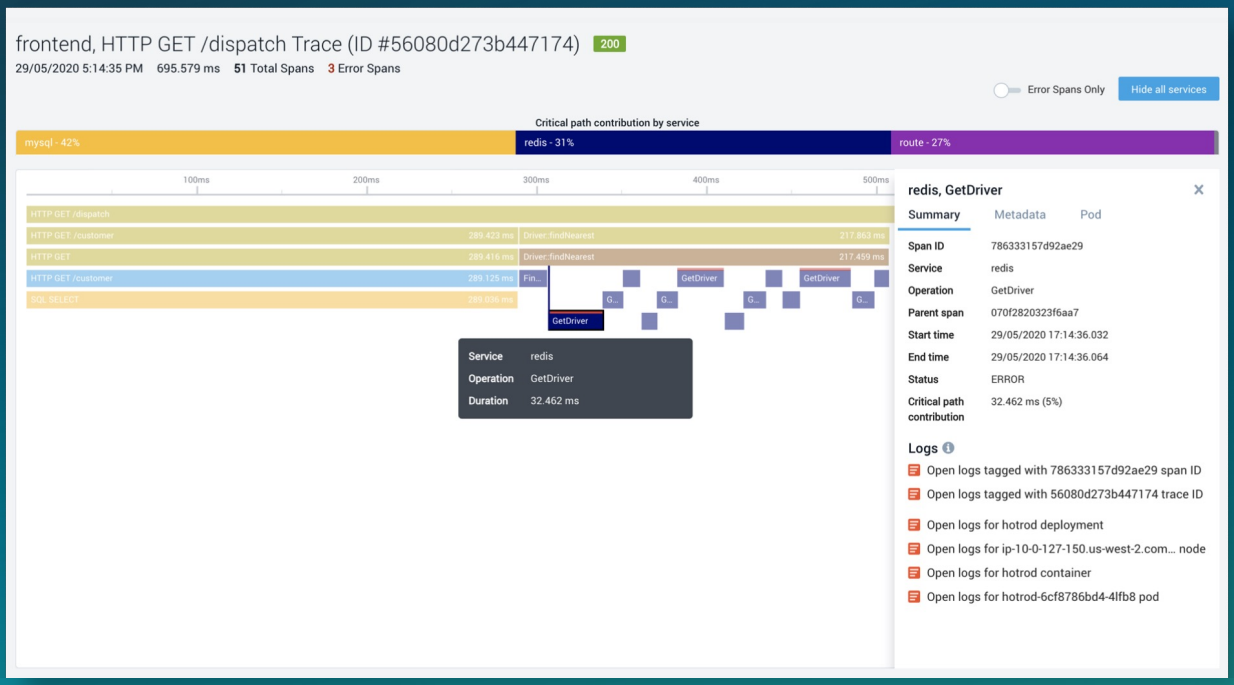
The main point?

Observability means getting the full picture of how the business is doing by correlating all the different streams of data.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Transaction tracing

This is an example of the Sumo Logic Tracing interface, which enables teams to monitor and troubleshoot transaction execution and performance across a distributed application environment.



Tracing data is fully integrated with logs, metrics, and metadata in order to provide a seamless end-to-end experience during the process of managing and responding to production incidents, and to reduce downtime by streamlining root cause analysis. Sumo Logic Tracing supports the OpenTelemetry standards and is included in ADOT to collect distributed tracing data.

Unified Observability?

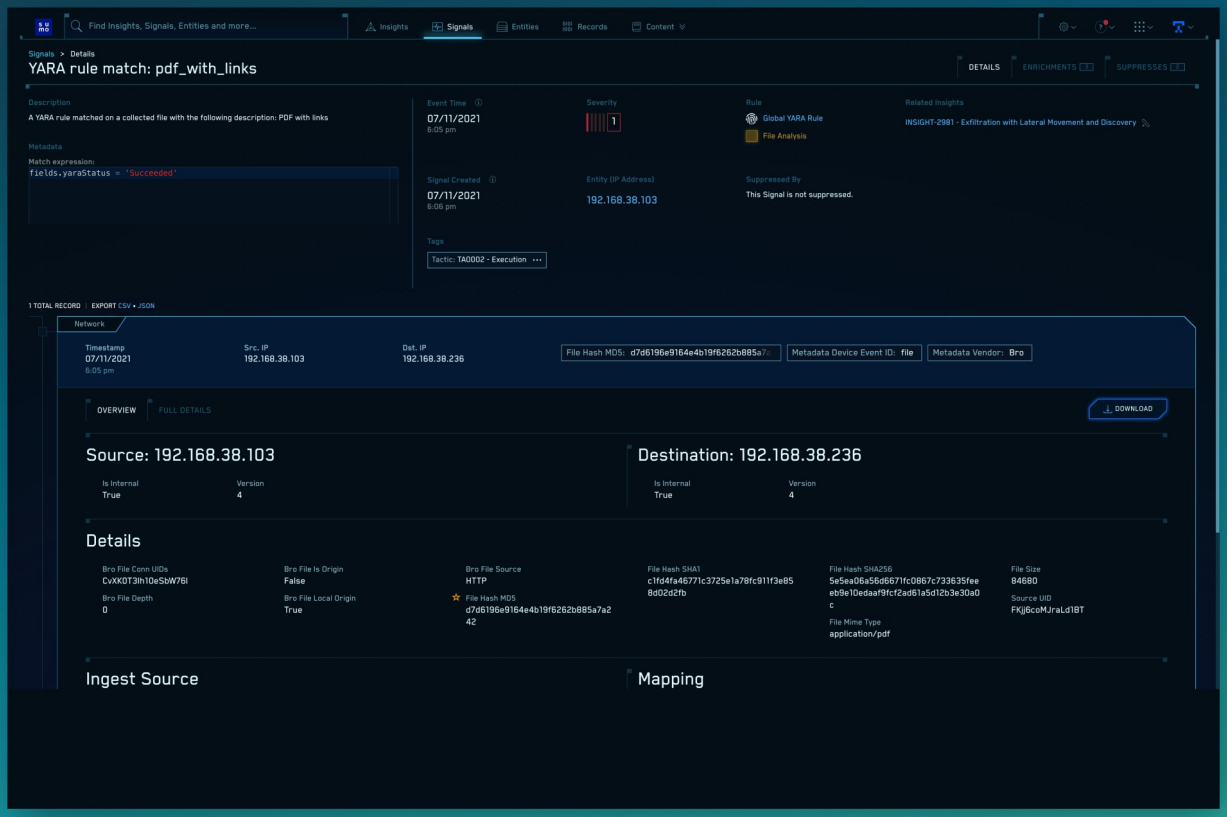
Sumo Logic Observability combines logs, metrics, and trace datasets into a single platform and leverages an entity model that enables users to correlate signals between logs, metrics, and traces as they go from an alert to root cause. These entities are discovered automatically from the metadata across logs, metrics, and traces generated by the application and its infrastructure.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws marketplace

Sumo Logic security Information and event management (SIEM)

To tie the Trend Micro Cloud One and Sumo Logic services together, the SIEM becomes the central hub into which all things security-related are reported. The Trend Micro security events are sent to the Sumo Logic SIEM to provide a single-pane-of-glass experience.



Threat investigation dashboard

The cloud SIEM ingests and analyzes security telemetry and event logs, but also reassembles network traffic flows into rich protocol-level network sessions, extracted files, and security information. Analysts can see raw network traffic details, related connections, and protocol activity, and gain visibility into East/West network traffic. Sumo Logic's deep library of cloud API integrations can pull security telemetry directly from sources such as VMware Carbon Black, Okta, and AWS GuardDuty.

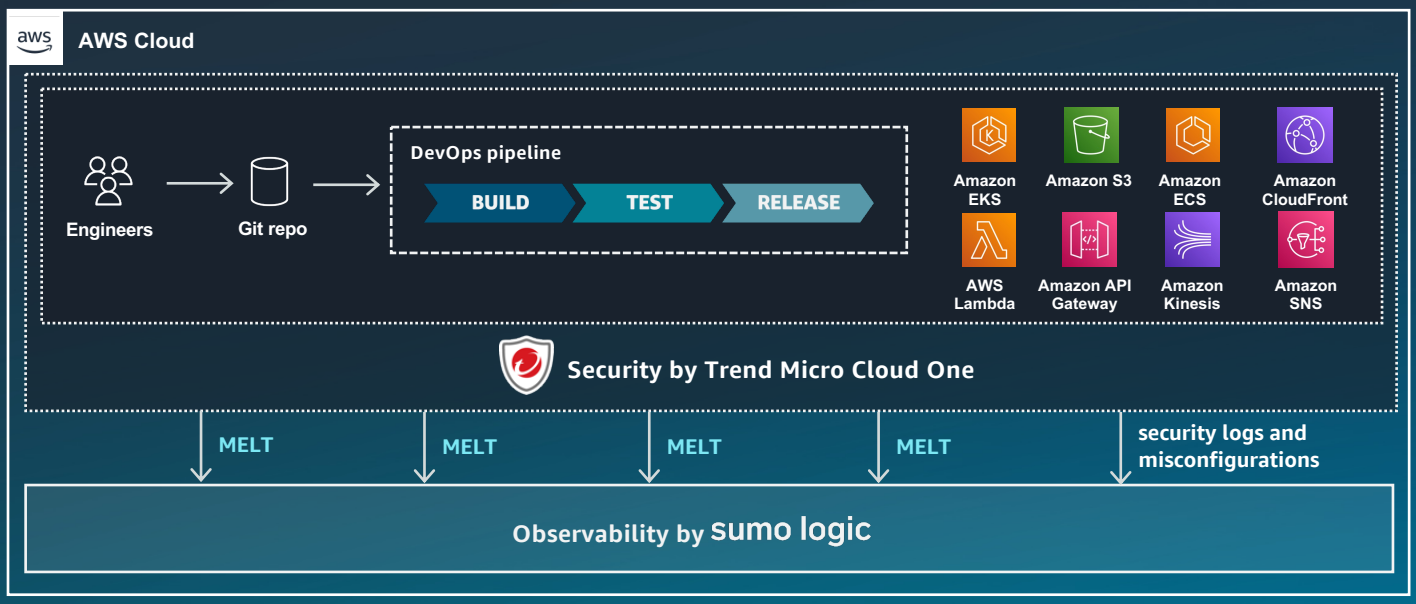
© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Recap: Observability solution architecture

Let's review the example Observability solution we have created.

We started with the AWS Cloud and ADOT for instrumentation, standardization, and collection of MELT data. Then we added Trend Micro for Security Observability and setting guardrails and checks all the way from the developer, through the CI/CD pipeline, and into the deployed resources in the AWS account.

Lastly, we talked about end-to-end Observability on the Sumo Logic platform—how Sumo Logic brings all the different sources of telemetry data into a single unified experience. Regardless of whether that data is from AWS, on-premises, or third-party services, Sumo Logic brings it all together into a single pane of glass for any role in your organization.



© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws marketplace

Continue your journey with AWS Marketplace

Trend Micro Cloud One and Sumo Logic can be used to build a well-engineered approach to Observability for your AWS applications. Establishing these capabilities can provide a strong foundation for continuing to advance through your cloud native journey. You can get started with these tools and others across the DevOps landscape today in AWS Marketplace.

To get started, visit: https://aws.amazon.com/marketplace/solutions/devops

AWS Marketplace

Third-party research has found that customers using AWS Marketplace are experiencing an average time savings of 49 percent when needing to find, buy, and deploy a third-party solution. And some of the highest-rated benefits of using AWS Marketplace are identified as:

Time to value

Cloud readiness of the solution

Return on Investment

Part of the reason for this is that AWS Marketplace is supported by a team of solution architects, security experts, product specialists, and other experts to help you connect with the software and resources you need to succeed with your applications running on AWS.

Over 13,000 products from 3,000+ vendors:

GitLab · snyk · bridgecrew BY PRISMA CLOUD · circleci · HashiCorp · LaunchDarkly

Buy through AWS Billing using flexible purchasing options:

- Free trial
- Pay-as-you-go
- Hourly | Monthly | Annual | Multi-Year
- Bring your own license (BYOL)
- Seller private offers
- Channel Partner private offers

Deploy with multiple deployment options:

- AWS Control Tower
- AWS Service Catalog
- AWS CloudFormation (Infrastructure as Code)
- Software as a Service (SaaS)
- Amazon Machine Image (AMI)
- Amazon Elastic Container Service (ECS)
- Amazon Elastic Kubernetes Service (EKS)

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws marketplace

Get started today

Visit aws.amazon.com/marketplace to find, try and buy software with flexible pricing and multiple deployment options to support your use case.

https://aws.amazon.com/marketplace/solutions/devops

Authors:

James Bland
Global Tech Lead for DevOps, AWS

Aditya Muppavarapu
Global Segment Leader for DevOps, AWS

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.