INSTANA
an IBM Company

Observability for Developers

What is Observability

Table of Contents

Introduction

Developers are faced with a growing challenge: How do we troubleshoot software that may be comprised of many disparate services running in a variety of languages and platforms. How can we notice critical changes, see into our black box services, and discern the true causes of errors?

Not too long ago, debugging a program usually meant one thing: browsing error logs. This approach was fine for small teams running simple programs in a small number of instances.

But things are always changing. Our software architecture paradigms evolved have from monoliths to microservices.
Our responsibilities have changed with the emergence of DevOps, which represents a shift in the way developers take responsibility for our programs after delivery.

Observable: noticeable, visible, discernable

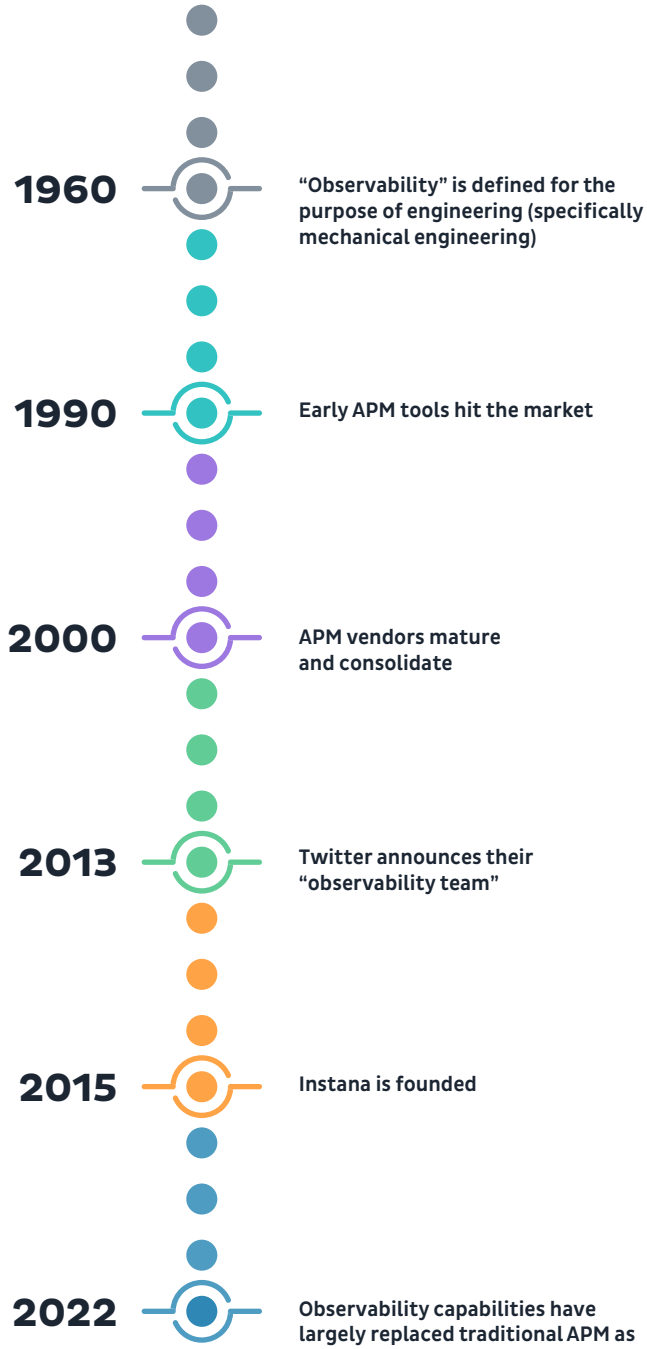
The Origins of Observability

The term “observability” was first defined for engineering purposes in 1960 in R.E. Kalman’s paper / ./ **On the general theory of control systems** As it pertained to mechanical engineering, the term observability was defined as the ability to understand the inner state of a system by measuring its outputs.

Fast-forward fifty years to 2013. Developers and software professionals are monitoring their systems using tedious instrumentation tools — if they are monitoring at all. But the shift towards distributed systems is already underway. **Twitter announces** that they are creating a new “observability team” to centralize and standardize the collection of telemetry data across Twitter’s “hundreds” of services.

Observability Today

Since 2013, our services have only become more granular and our job responsibilities more cross-functional. Microservices are giving way to serverless, and DevOps has led to Site Reliability Engineering. Observability is as important as ever, and the scope of the challenge has grown exponentially.



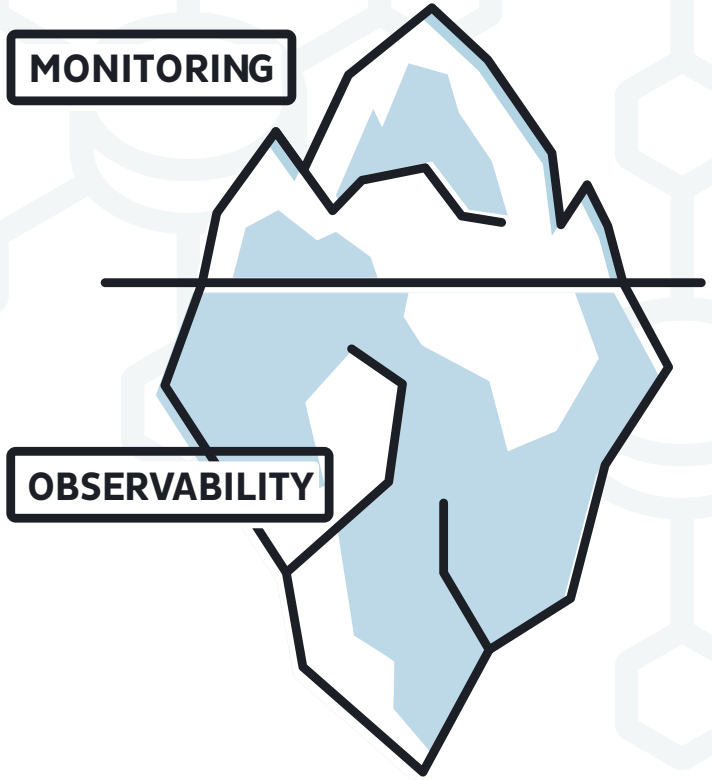
How is Observability Different from Monitoring?

We can't know where we're going if we don't know where we've been. Application Performance Monitoring is an important set of capabilities that can coexist with modern observability.

The problem with traditional monitoring is that it focuses on measuring predefined aspects of known components. With distributed architectures, the components of our application are always changing, sometimes by the second. Traditional APM tools can't keep up with this dynamic environment.

The worst time to discover you're missing some crucial datum is when you're in the middle of triaging a production outage.

That is where observability comes in. With observability, instead of predetermining what to measure, we build the capability to see everything that's happening between and even inside our services. This allows us to answer questions we couldn't anticipate and confront unknown unknowns.



Types of Telemetry

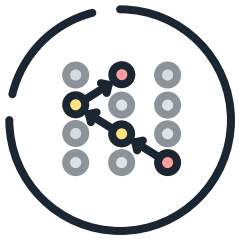
Monitoring and observability share the same fundamental datatypes: logs, traces, metrics, and EUM (End-User Monitoring) beacons.

Logs

Fundamentally, all logs are simply events — and these structured events form the foundation of observability telemetry signals. Other types of telemetry data including traces and metrics can often be derived from logs. Whether it's request, error, or debug logs, finding the right log message is often the first step to troubleshooting
an issue.

But there is the challenge — how do we find the right log message amongst the mountains of logs generated by a distributed application with multiple instances of each service? We need annotated, structured logs from all of our services to be aggregated and indexed.

We could do this ourselves, as we have in the past, or we could use an observability platform to automatically gather, annotate, and index our log messages for us.

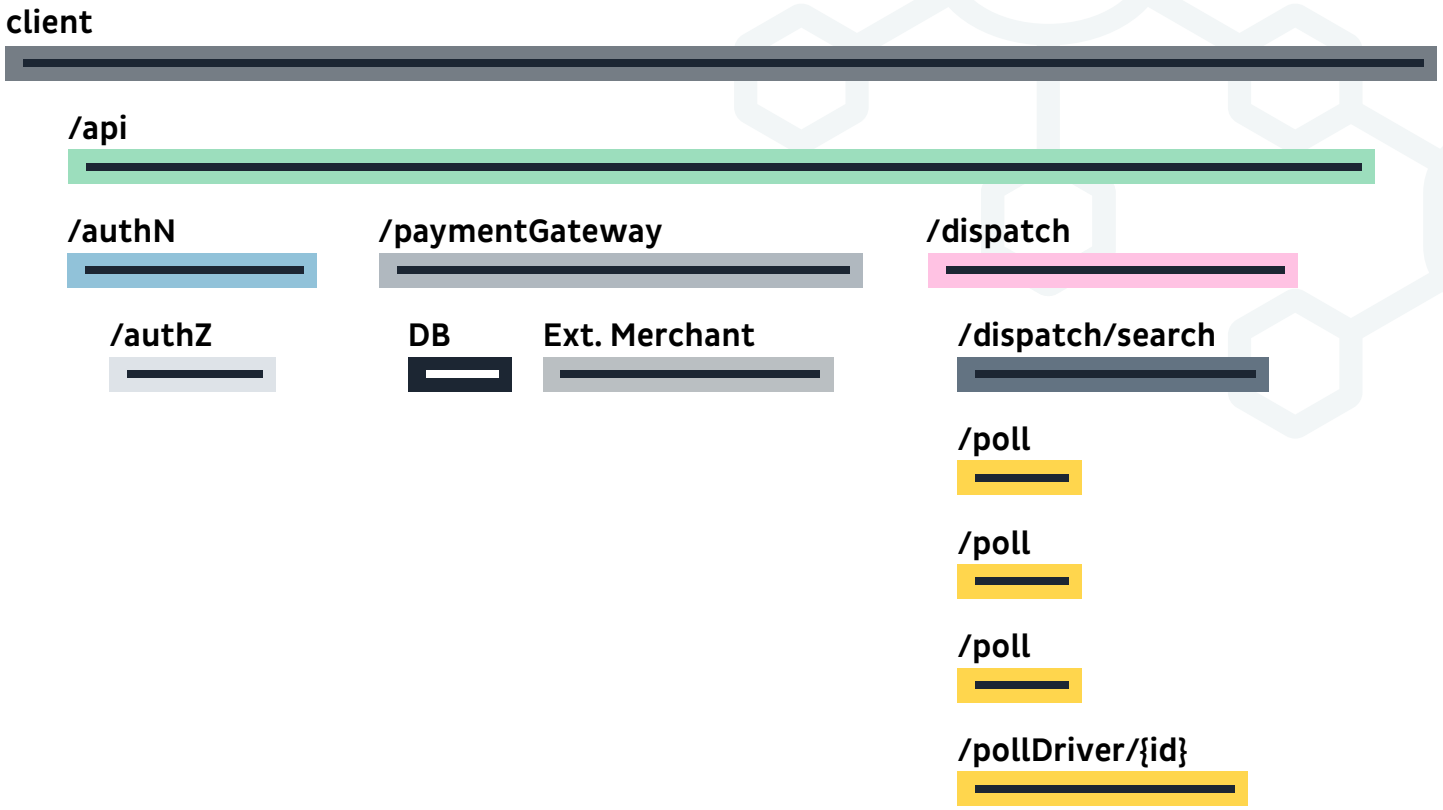


Traces

Distributed traces are a necessary tool for understanding request flows in modern applications. We can think of traces as request-scoped logs. They allow us to correlate events from downstream services with the end-user request that triggered the event.

Creating traces is fairly straightforward. With each incoming external request, a new span is created with a unique ID. That ID is sent to downstream services, which include it as a parent_id in their own spans.

A backend of some kind is then required in order to make use of these spans and combine them into searchable traces. Jaeger and Zipkin are the two most popular open source choices, with Jaeger being somewhat newer.



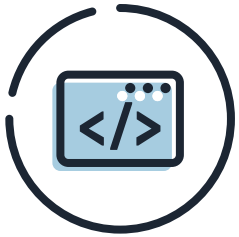
Metrics

Logs and traces are great tools for monitoring our applications. When it comes to monitoring the infrastructure that our applications run on, metrics are crucial.

The defining characteristic of metrics is that they are aggregatable. Because metrics are represented as numbers, we can set thresholds and optimal bounds with the hope of catching issues before they happen.

Metrics can be gathered in a “pull” method where a metrics service scrapes monitored services at a set interval, or a in a push method where the monitored services send requests as their data changes. Prometheus is the most popular open source metrics service and it uses the pull method of metrics gathering.

We will discuss specific metrics further the next chapter: SLO Methodology.



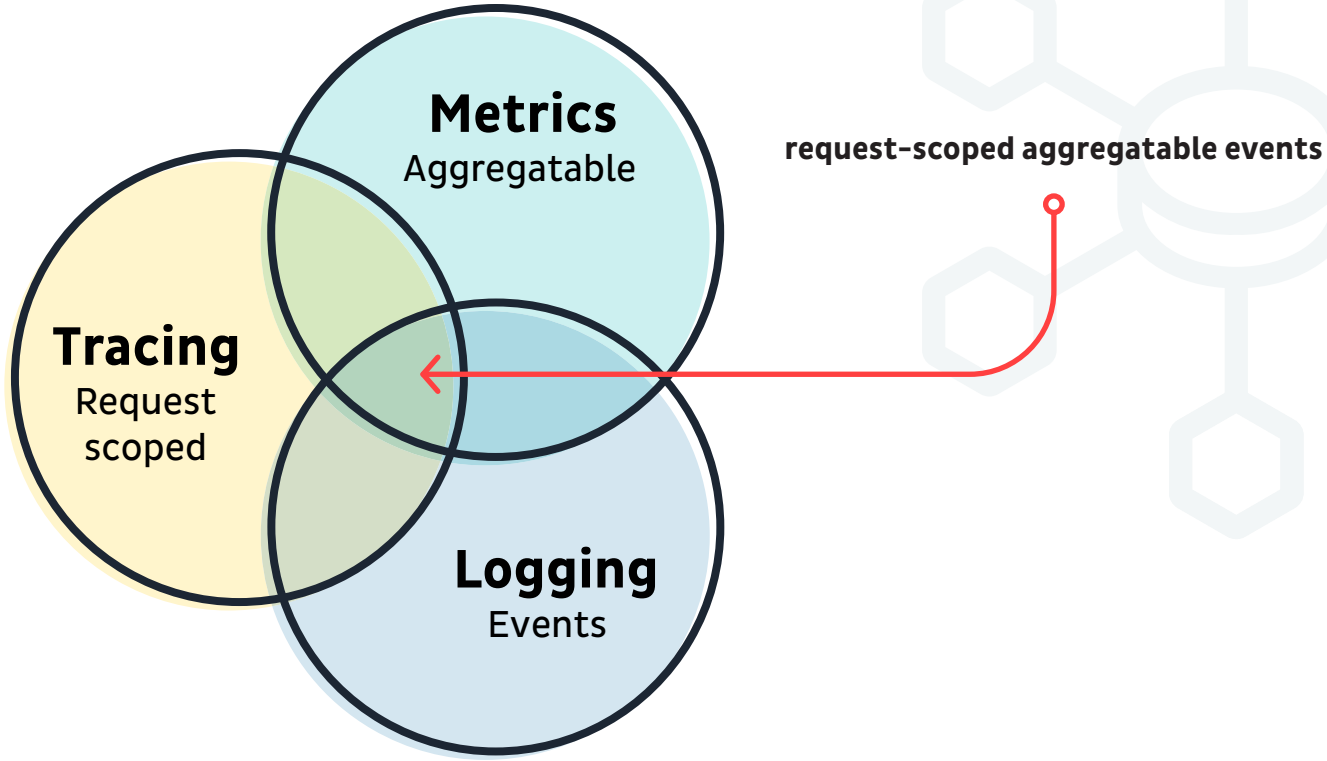
EUM/End User Monitoring (or Real User Monitoring) Beacons

Most of our end-users are not calling our backend APIs directly. They are usually using a web or mobile app interface of some kind. End User Monitoring allows us to understand the true user experience of our applications including issues with the interface or networking.

By placing an agent directly in our web or mobile applications, we can gather telemetry beacons from our users. Combined with the metrics, traces and logs from our backends, this data can give us a full picture of the user experience of our applications.

Putting it all together

While these signals are sometimes useful in isolation, the answers we need are found when all of this data can be correlated and searched in a meaningful way. **Peter Bourgon** elegantly laid out the crucial sweet spot overlapping between metrics, traces, and logs: Request-scoped, aggregable events.



Observability Standards & Open Source

In 2019 the OpenTracing and OpenCensus projects merged and became OpenTelemetry. This open source project under the CNCF is quickly establishing open standards for telemetry that are widely adopted by both open-source tools and vendor observability platforms like Instana.

At the core of the OpenTelemetry project is OTLP: open specifications for observability data. The OTLP specifications are stable for all three core signals: metrics, traces, and logs. A specification for Real User Monitoring has been proposed but is not yet available.

The OpenTelemetry community has created a large suite of excellent SDKs, APIs, and tools around the OTLP standard:

Instrumentation SDKs

Instrumentation SDKs are available and stable for most programming languages. Automatic Instrumentation (like Instana AutoTraceTM) is even available for some runtimes.

OpenTelemetry Collector

The collector service acts as an agent on your hosts or nodes. It ingests telemetry signals from the other processes on the host and then transforms the data (for example, sampling traces to reduce costs) before sending it to your choice of observability backends.

Kubernetes Operator

An OpenTelemetry Operator for Kubernetes is available that can provide auto-instrumentation for compatible workloads running in a Kubernetes cluster.

Open Source Observability Backends

The OpenTelemetry project does not provide a backend for storing and analyzing your monitoring data, but a number of open source tools are compatible with the OTLP signals.

Logging

Popular open source backends for aggregating logs include: Loki, Graylog, and Logstash with ElasticSearch.

Tracing

Zipkin and Jaeger are the most common tracing backends in use today. Zipkin has been around for a while and it's trace format is supported by the OpenTelemetry Collector. Jaeger is capable of ingesting traces in many formats including Zipkin's format and OTLP.

Metrics

Prometheus is by far the most popular open backend for metrics aggregation. It can support a wide range of dashboard, alerting, and other tools that depend upon the metrics data.

SLO Methodology

By now you have probably been hearing a lot about SLOs and SLIs — Service Level Objectives and Service Level Indicators.

SLO methodology represents a new way to think about software performance and health that goes hand in hand with observability. Observability as a concept comes from control theory, which focuses on the optimal — not perfect! — operation of machines. SLO methodology teaches us that when we strive for perfection we often achieve worse outcomes than when we set realistic targets.

It all starts with Service Level Indicator — an SLI is any metric or statistic that can be converted into a percentage. In the context of running software, our SLIs are things like the % of requests that were served successfully or the % of requests that had acceptable latency.

An SLO is a target bound for an SLI. SLOs are often expressed as a certain "number of nines" — for example, 4-nines would indicate that an SLI meets the target 99.99% of the time.

Very importantly, our SLOs should never be "100%" — this is unrealistic in all situations. We have to leave ourselves an error budget — some extra wiggle room for planned and unplanned outages. In fact, one team at **Google** found that they could increase overall system reliability by artificially causing downtime for their service in order to prevent other teams from expecting it to be 100% reliable.

Service Level Agreements or SLAs may appear to be similar to SLOs, but they are used for very different purposes. An SLA is part of a business contract and it specifies what happens when the target is violated — usually this involves financial compensation of some kind.

SLAs are not interesting to developers until they are violated. SLOs are extremely useful to developers because we can use them to understand the overall reliability of our applications and to determine if it is safe to invest in new features.

Observability in a Serverless World

When they announced their observability team in 2013, Twitter referred to their “hundreds of services.” This almost seems cute to anybody working with serverless systems today.

> **Conventional APM methods such as language-specific tracing and metrics collection are of limited use for gaining visibility into serverless performance and availability. — [Monitoring Serverless Successfully]**

As the number and granularity of our services increases, so does the need for automation. Maintaining accurate and complete code-level instrumentation of every serverless function would be an exercise in insanity. Instead, we need to think about observability as part of the underlying service mesh in a similar way to how we handle authentication.

Beyond Open Source

With the open-source tools from the previous chapter, we are able to gather a lot of telemetry data from our services and begin to put it together in a useful way. However, it is difficult to say that we have achieved observability.

“Observability” demands that we can answer questions about unknown unknowns. Our telemetry needs to adapt and change as our services do. So far, no open-source tool can provide that level of automation. But vendor solutions such as Instana can.

Observability Solutions are Adaptable & Dynamic

Our applications are likely comprised of dozens or hundreds or even thousands of services using different languages and technologies. Maintaining consistent manual instrumentation across our entire application’s surface area would be a Sisyphean task.

By relying on automations such as dynamic service discovery and automatic instrumentation, we can be certain that we have a complete understanding of our systems before incidents occur — because while we’re in the middle of triaging a production outage is not the time to discover that we are missing a key piece of the puzzle.

The Automatic Correlation Benefit

A major benefit of an Enterprise Observability solution is the ability to correlate machine, infrastructure, and application / services metrics and traces. Distributed traces deliver the understanding of the request’s flow, while metrics provide the necessary performance points.

Correlating manually, on the other hand, is quite cumbersome. The main reason people hate having multiple dashboards for different services is that it’s almost impossible to match any data and gather the overall context of the issue.

The biggest benefit of automatic correlation is the immediate insight. When looking at an issue or incident, the vendor solution does all the detective work, providing important pieces of information as contextual evidence, and leads us right to the area of interest.

For the best customer outcomes we can combine automatic detection with automatic remediation and **achieve true software health**.

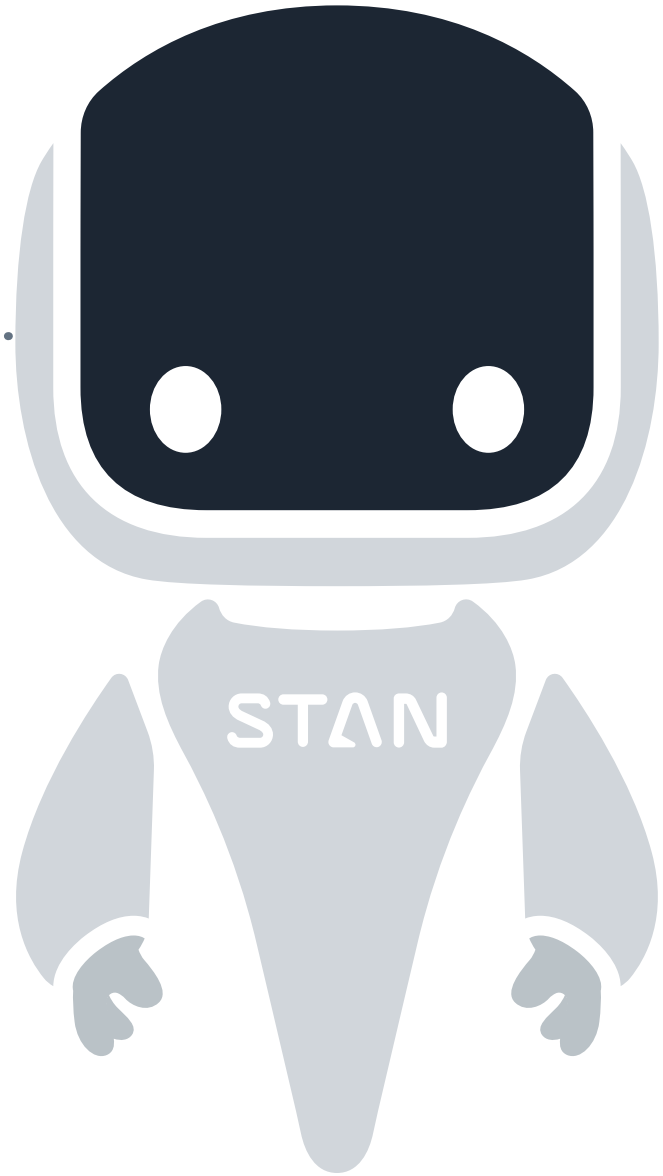
About Instana

Instana, an IBM Company, provides a real-time, automated Enterprise Observability Platform that includes application performance monitoring capabilities to businesses operating complex, modern, cloud-native applications no matter where they reside—on premises or in public and private clouds, including mobile devices or IBM Z® mainframe computers. Users can control modern hybrid applications with Instana's precise metrics, full end to end traces for all transactions and AI-powered contextual dependencies discovery inside hybrid applications.

Instana helps System's Reliability Engineers improve the reliability and resiliency of cloud-native applications by preventing issues from turning into incidents and by providing fast remediation times when incidents occur. Instana also provides visibility into development pipelines to help enable closed-loop DevOps automation with actionable feedback for optimizing application performance, enabling innovation, mitigating risk, and managing cloud technology expenditures.

For more information, visit https://instana.com.

Start Your Trial Today



Stan
Your Intelligent
DevOps Assistant

INSTANA

an IBM Company