

## Instana Release Notes - Build #113

15. August 2016

Dear Instana User,

With this release we proudly present to you Application Performance Monitoring by Instana. There will be some more blog posts about the details and ideas around it and I do not want to make these notes longer than necessary: so just describing the features and asking you to check [instana.com](http://instana.com) and our newsletter for updates! And please feel free to contact me via Slack or email at [michael.krumm@instana.com](mailto:michael.krumm@instana.com). We have a lot more cooking! Stay tuned.

Best regards on behalf of the Instana Team,  
Michael

## Features

### Logical View

Our Agent is tracing Java applications for a while now and we support an increasing amount of frameworks and libraries. Based on the data collected by Traces and the data collected by Component Sensors we discover your application landscape, the Services implementing it and monitor their health.


Until now we just showed the Traces themselves in the Trace View. With this release we introduce the Logical View showing your application flow in real time including the architecture of your services and their communication:




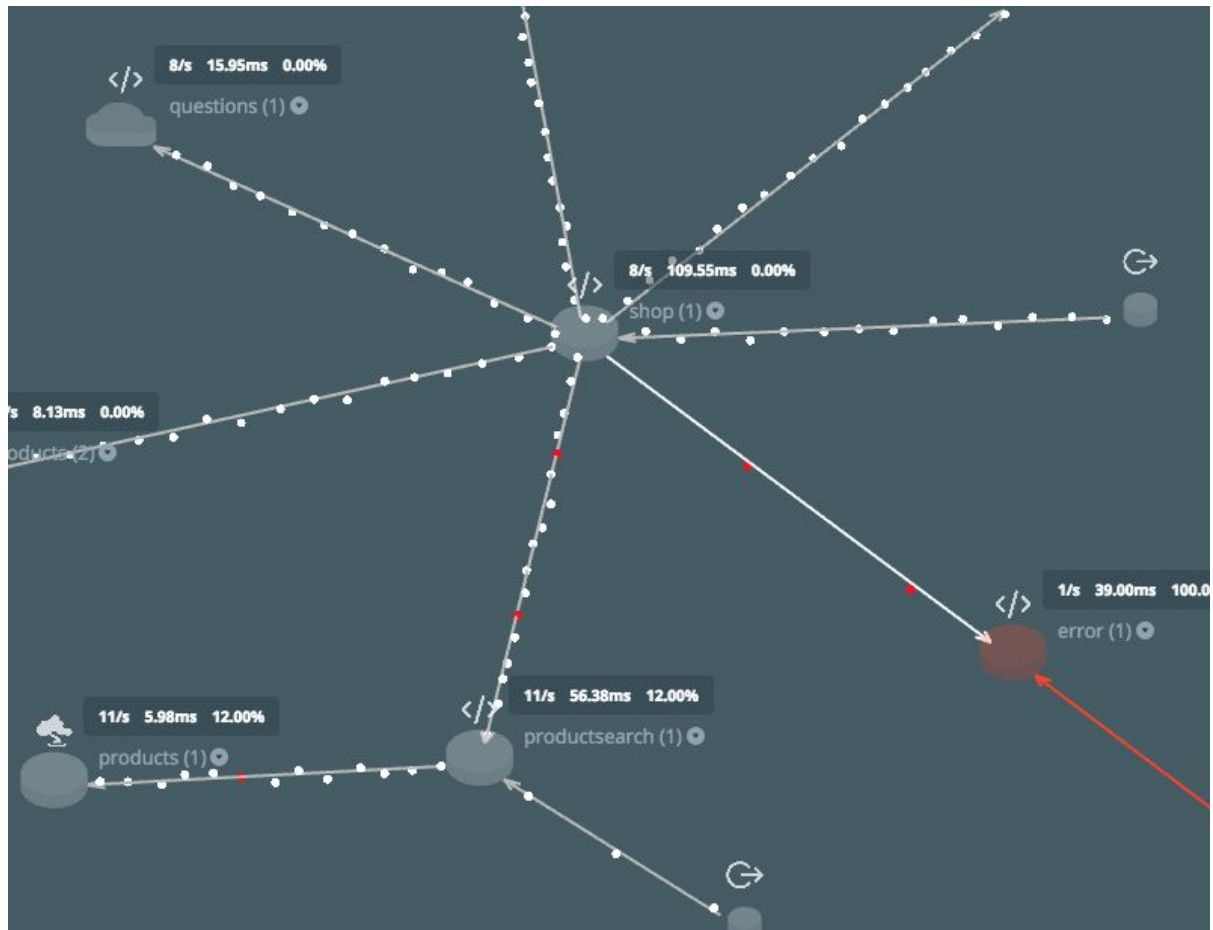
# INSTANA

The picture above shows a subset of our demo application. All you see above is discovered and continuously updated in real time with our standard data granularity of 1 second.

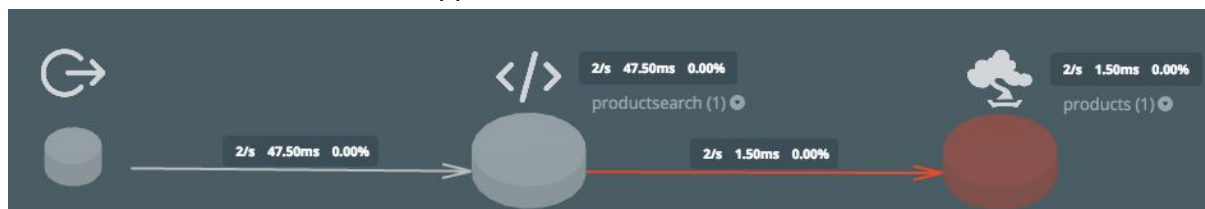
You can click and drag the Services to rearrange the map.  
There are new map controls on the bottom right of the map.

A click on  triggers an automatic layout of the map.

A click on  turns particles on. They show the traffic and error rate visualized as particles to give a quick visual impression of the flow between the services:



Let's have a closer look into the application:



In this part of the monitored application is shown that calls are done to the “productsearch” service with a rate of 2 calls per second and a average response time of 47.50 ms with an error rate of 0%.



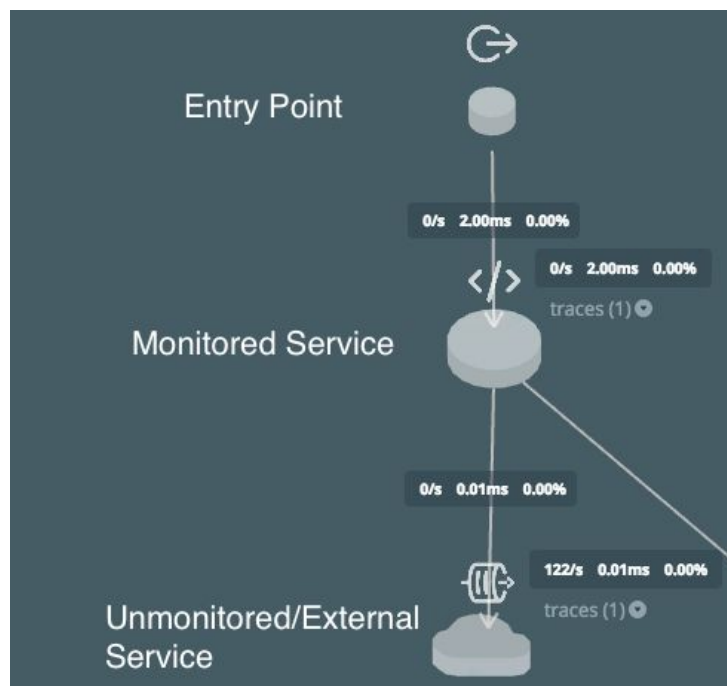
# INSTANA

We focus on 3 Key Performance Indicators (KPIs) that are important for understanding the performance of a service:

1. **Traffic** - measured in calls per second. "How much Traffic is the Service handling?"
2. **Latency** - measured in ms. "How fast is the Service handling requests?"
3. **Error Rate** - measured in % of calls. "How is the error rate of the Service?"

The Service "productsearch" itself calls then an Elasticsearch Index called "products" with 2 calls/s, an average response time of 1.50 ms and an error rate of 0%.

Instana discovers three different types of Services plus Connections with this release:



The **Entry Point** shows that Traces and therefore Calls are started at the called Service. It shows the "User" or other unmonitored upstream Services.

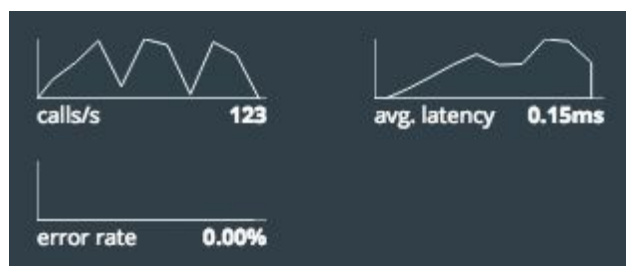
The **Monitored Services** are shown with a Circle and are Services that are directly monitored and instrumented with an Instana Agent.

The **Unmonitored/External Services** (with the Cloud Symbol) are called by a monitored Service but there is no Instana Agent running on that Service or the recognition failed. These will

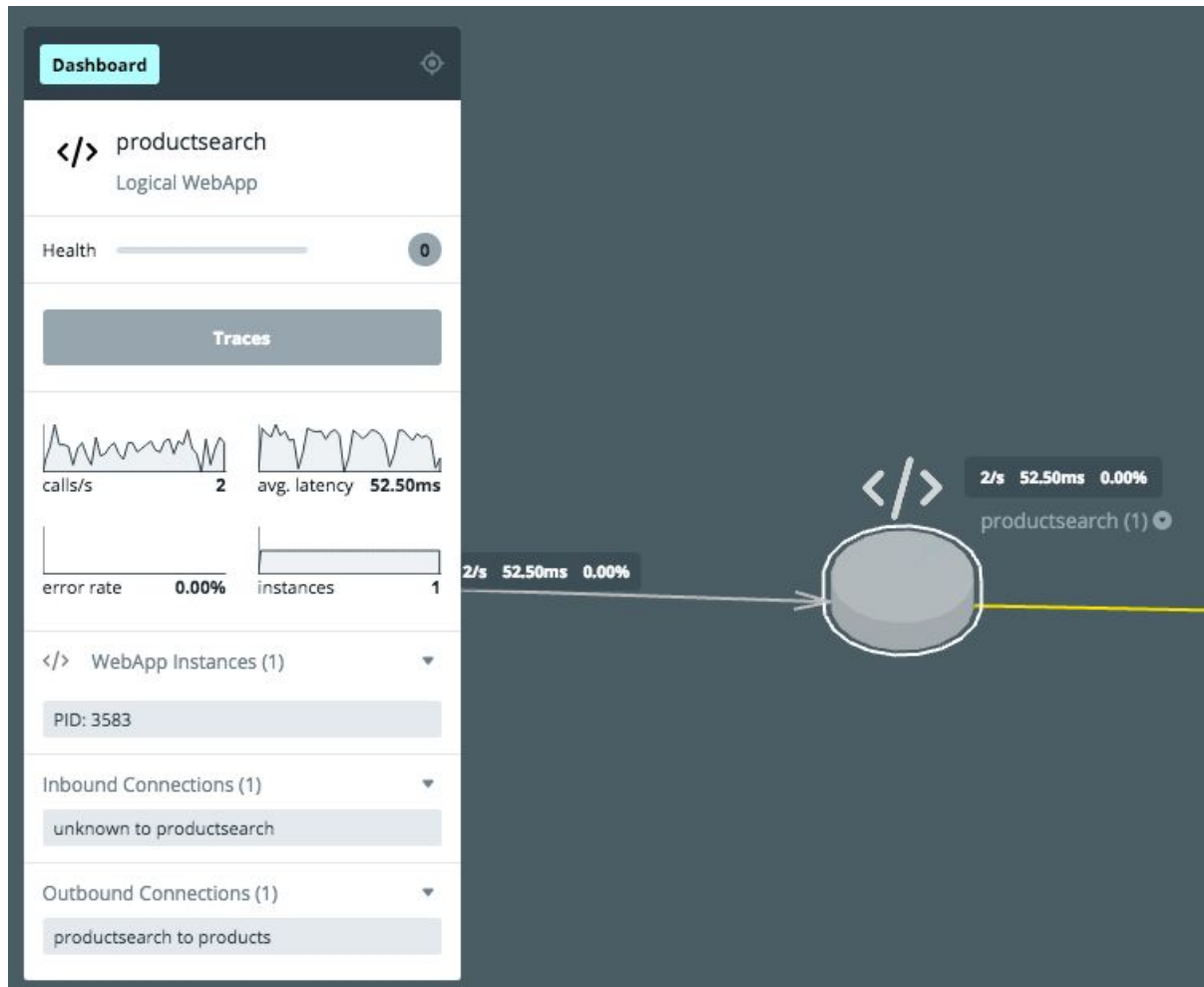
usually be calls to external Services like e.g. a 3rd party service, or an internal service not yet being instrumented with an Instana agent.

The icons above the symbols show the different types (Entry, Application, Queue in the above screen) as specifically as possible.

The metrics shown are for the corresponding Services and Connections. If you hover over them you will be presented with Sparkcharts to get a fast understanding of the recent history of the metrics:



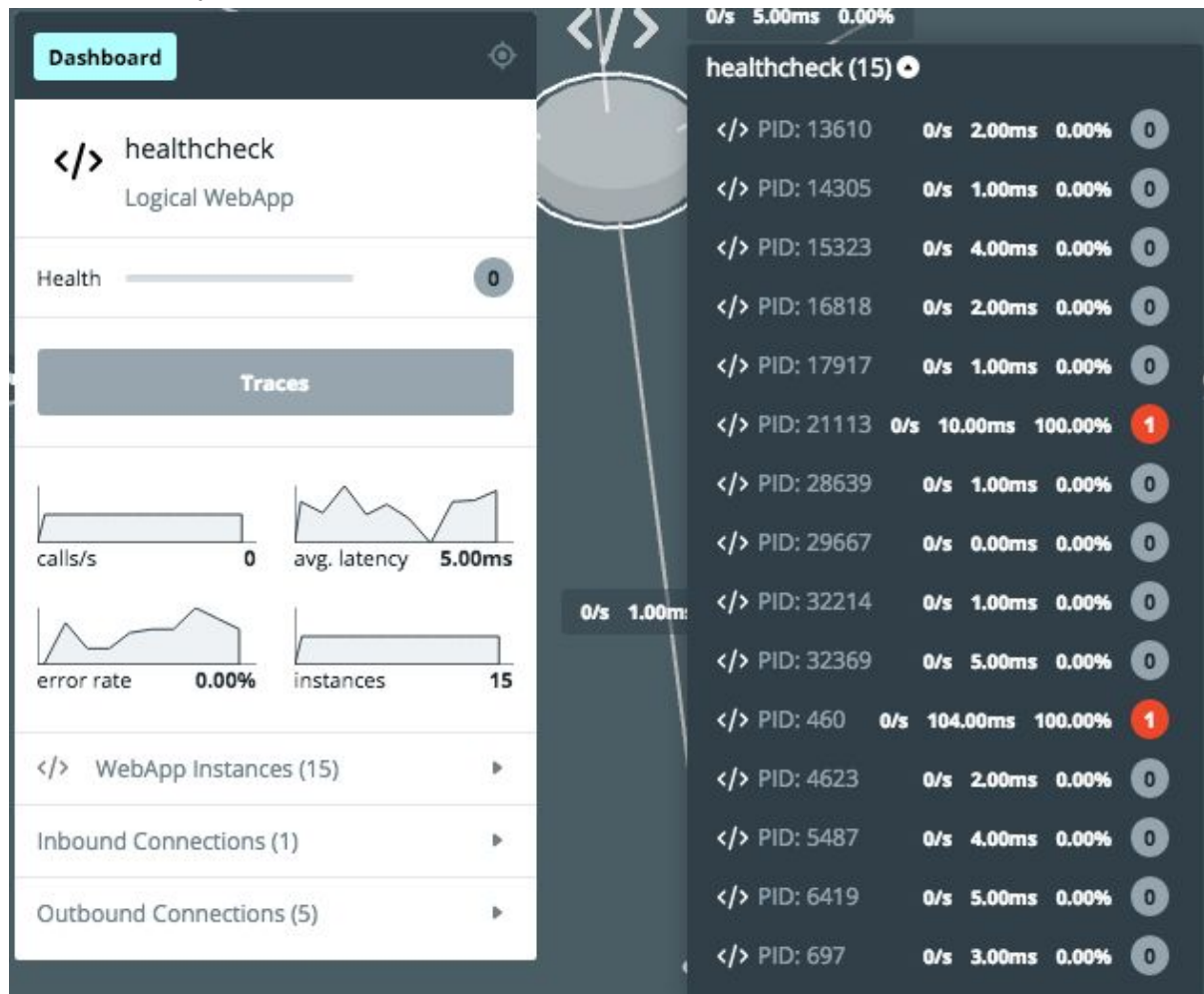
You can click on Services and Connections to pop the Sidebar and get more detailed overview:



Simply click on the “Traces” button to jump into the Traces that started at the discovered Service: you will get into the filtered Trace View and will be able to dig into every single trace.



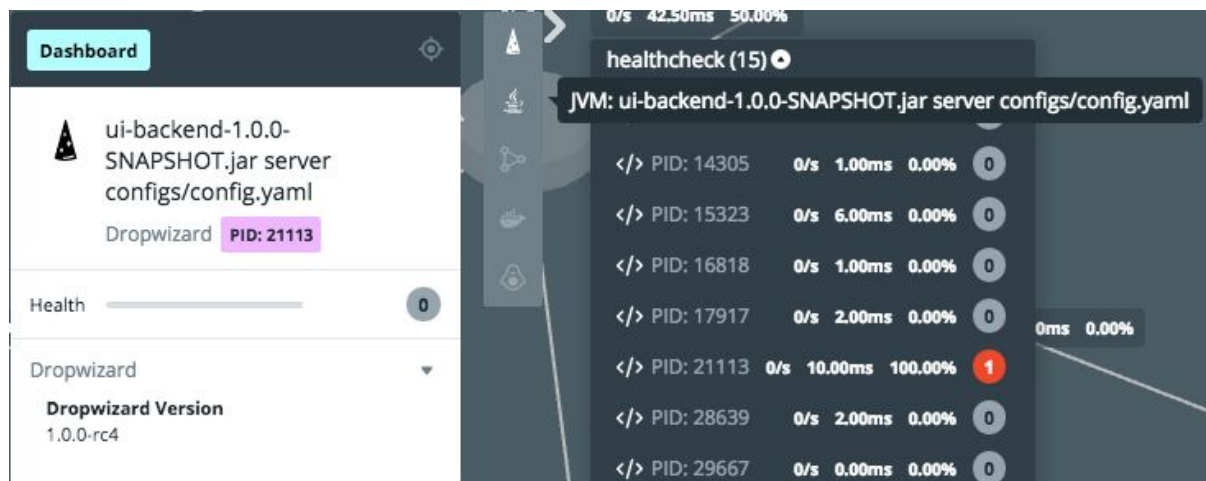
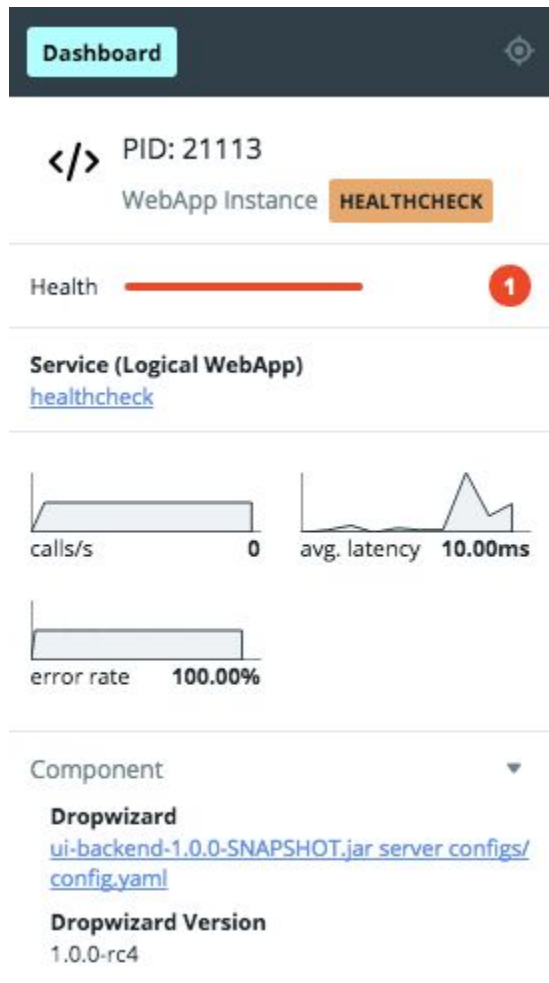
Please also note the “Instances” metric. Instana automatically tracks how many Instances implement a Service. Here is an example of a Service called Healthcheck that is actually implemented by 15 Instances (in this case JVMs):



We derive the mentioned KPIs and health check for each Instance, as you can see from the above screenshot. If you click on an Instance you will see the sidebar and more Information around its KPIs and the “Physical Component” that is in this case the JVM that is the underlying Runtime for the Service.



# INSTANA



This is the bridge for the “Physical Perspective” that Instana offers since the beginning. Instana discovers and knows all dependencies and relations - from the Service down to the Host and all layers in between and you can navigate through all of them.

As usual you can investigate all Services and Service Instances in more details through Dashboards:



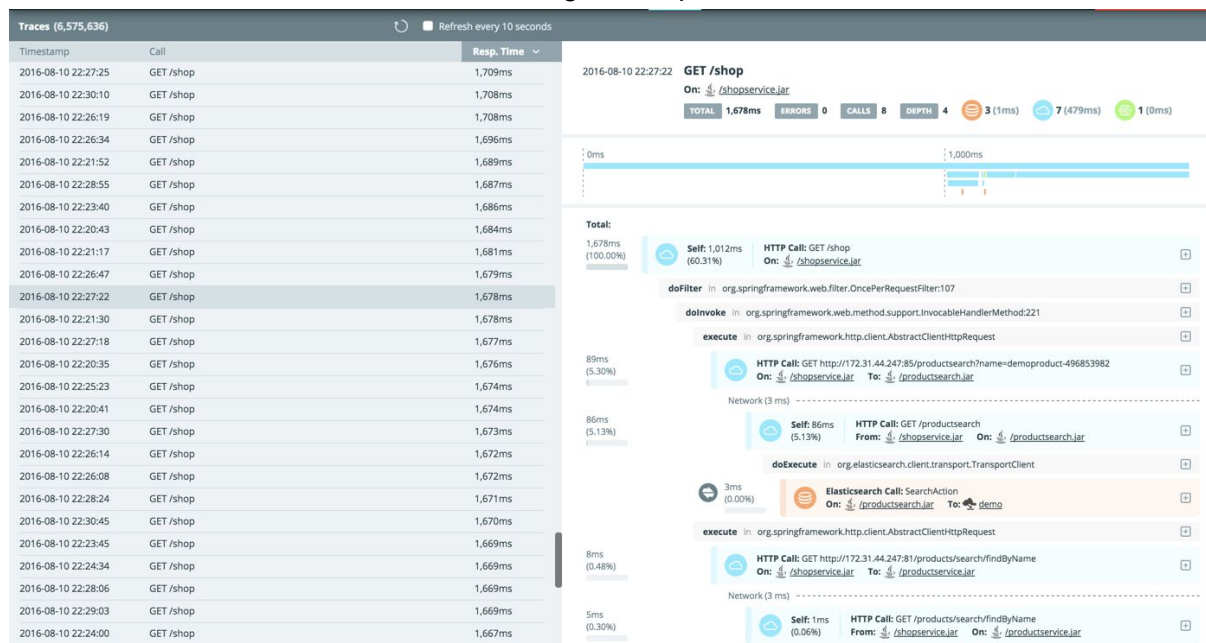




The Latency Overview shows the distribution of the latency metric to give you an understanding of the variance. Please keep in mind that you can adjust the roll ups with the timeline by zooming out. This can be very helpful when working with such graphs.

## Trace View

We got a lot of feedback regarding the Trace View and worked hard to improved it. We decided to rework the Trace View from the ground up and here is the result

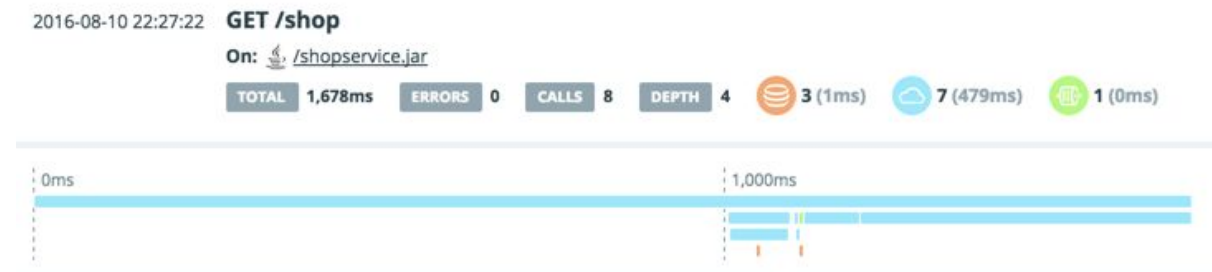


On the left you can sort by timestamp and response time. When you select a Trace the details are presented on the right.



# INSTANA

The top shows a summary of the Trace including an icicle graph of the whole trace. The different call types have symbols and are color coded (in this example Database - orange, HTTP - blue, Queue - green). If you click on a Span in the graph you jump to the details of it.



In the detail view shown below you can expand each detail in parallel to allow easier investigation.



The Spans (colored) show details per type. In addition we show the stack traces per Span (grey).

The total time is shown on the left and in the Spans the self time of a Span.

## Incidents based on Service Health

With our ability to derive KPIs (traffic, latency, error rate) of Services we apply machine learning to understand health of Services and their Connections. In other words: We detect various situations in which Services/Applications are not behaving as they should or did. Because of this we are now triggering Incidents only if there is a Service impact noticed. The

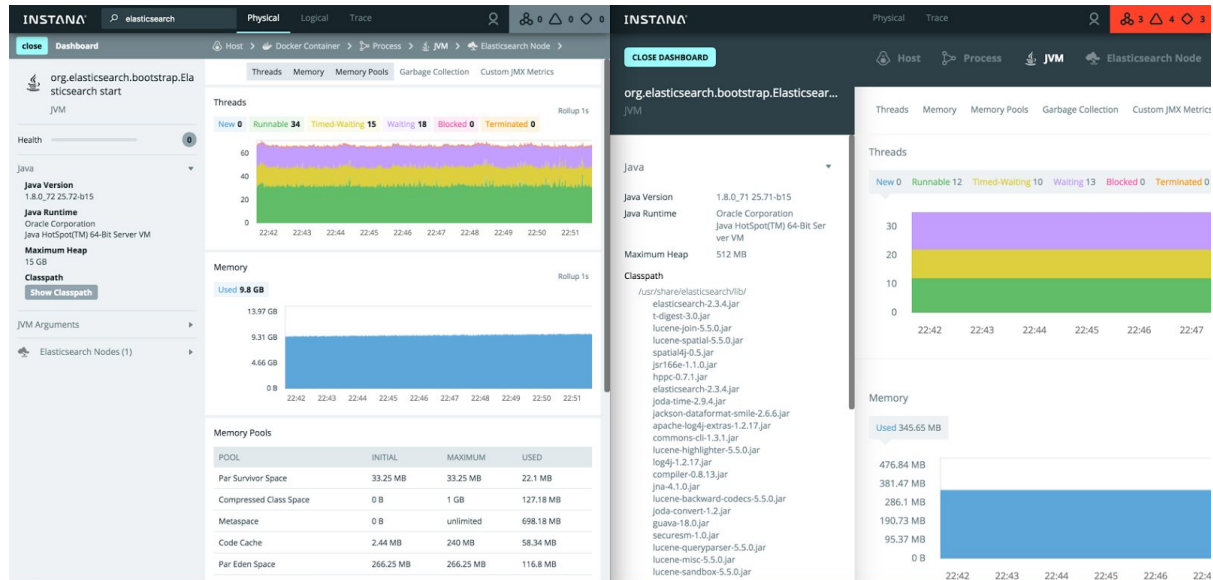




moment this is done the Graph is traversed to group all relevant Issues and other Events into the Incident. This greatly reduces noise and make Incidents a lot more meaningful.

## Dashboard Rework

Dashboards are a core element to help you work with Instana and investigate situations in more detail. We did rework the Dashboards with the focus on making them easier to read while at the same time showing you more data at one glance (on the left the new...).



## Search in Logical and Trace

Please enter in the Logical Perspective the name of the Service you are looking for. You can enter an asterisk (\*) at the end of the search term as a wildcard.

In the Trace View you can search for the request (e.g. "GET /shop") by entering any part without the "/" sign. Please also leverage the wildcard \*.

In addition to this you can search for specific response times of traces by entering the one of the following expressions (you can always use =, <, >) with the time in ms:

duration>100

time=100

latency<100

## Extended Tracing support

We are currently supporting distributed Tracing for Java and Scala and instrument standard libraries and frameworks to assure stability and lowest impact (please read more about our approach to safely instrument in the Blog post [How Instana Safely Instruments Applications for Monitoring](#)).

We are quickly adding support for libraries in close cooperation with you - our users and customers. Please contact us in case something that you need is missing.

Instrumentation is done by supporting certain technologies (frameworks, libraries, clients). A new Trace is created when an execution hits at least one instrumented method as Entry Point and is not called through an already instrumented call (aka has a Parent Span ID tagged to the payload and will be correlated to another trace in the backend).



If, within a Trace, a method that is instrumented as an Exit Point is hit, a Span is created for it and - if possible/it makes sense - the ID for the trace is added to the call in order to enable correlation.

Framework/Library	Supports
Apache HttpClient	Exit
Cassandra	Exit
EJB (Glassfish, JBoss, OpenEJB)	Entry and Exit
Elasticsearch	Exit
HTTP UrlConnection	Exit
JDBC	Exit
Jedis	Exit
Jersey	Entry
JMS	Entry and Exit
Kafka	Exit
Lettuce (3&4)	Exit
Mongo	Exit
Play2	Entry
RabbitMQ	Entry and Exit
Servlet	Entry
Spring-Batch	Entry
Spring-Web	Entry
Vertex	Entry and Exit

## Extended Async Tracing support

We support

- Akka
- Executor Pools
- Hystrix
- Fork Join
- Vert.x

## Trace Error Tracking

We capture HTTP 5xx Status codes as Errors as well as Errors logged by the following loggers

- Java Util Logging
- SLF4J
- Log4j
- Log4j2



## Newly supported technologies

- **ActiveMQ**
  - Configuration
  - Broker and Memory Metrics
  - Health Monitoring for Performance and Configuration
- **JBoss Data Grid**
  - Configuration
  - Latency, Throughput, Hit Ratio, Pools and a lot more
  - Health Monitoring for Performance and Configuration
- **Memcached**
  - Configuration
  - Commands, Read/Writes, Hits/Misses/Hit Ratio, Used Bytes, Connections
  - Health Monitoring for Performance and Configuration
- **Solr**
  - Configuration
  - Core Monitoring: Requests, Cache Statistics, Insertions/Evictions



## Improvements

- Links to Events sent by Notifications directly open the time the Event has been detected.
- Improved compatibility with DNS names generated by Microsoft Azure.
- Support for Docker 1.12 was added.
- Added support to read (Custom JMX) MBeans from multiple MBean servers.
- Tags are read from configuration.yaml at runtime.
- Support using container and bridge network addresses to monitor containerized components.
- Reduced agent overhead for systems running many network connections
- Tomcat Dashboard now handles connectors which are using an executor pool correctly and displays also executor configuration
- Links in the UI are now real links and can be opened in new tabs and are correctly recognized as links when right-clicking them
- UI maintains connection to the backend dynamically
- UI shows clearer when there is a connection issue
- Reduced system load when the UI is open, but not visible to the user
- Disk Saturation Prediction tuned
- Missing Configuration of Sensors shown in Dashboards
- Usage Overview has consolidated forecast overview
- Node.js sensor log output will now be automatically forwarded to the Instana agent for centralized log management
- Added zoom buttons to the map

## Fixed Issues

- When searching without search expression for strings containing - (hyphen) search now returns results
- Fixed Agent Memory/File Handle Leak. If Agent has not been restarted since August 3rd, please restart at next convenience to resolve this issue.
- Fixed reading memory data on Proxmox OpenVZ
- MySQL Sensor will not correctly present currently active connections.
- Fixed situation where a monitored JVM would leave temporary agent files behind.
- Fixed higher than normal cpu usage monitoring rabbitmq servers with very large amount of queues configured.

## Known Issues

- Search in Trace View does not support forward slashes (/).

**Please feedback via Slack or Mail to [michael.krumm@instana.com](mailto:michael.krumm@instana.com) around the new features, the improvements and especially your ideas and other points you have. We have a lot cooking so stay tuned - more to come soon!**

