INSTANA

Instana Release Notes - Build #114

4. September 2016

Features

PHP Tracing

We proudly present PHP Tracing by Instana! As always we focus to deliver low impact, low effort and powerful tracing.

The configuration to activate PHP tracing is located at
<agent_install_dir>/etc/instana/configuration.yaml
The current version of the Instana Agent contains an example of the required configuration settings - just download it from the Management Portal.

You just need to uncomment this section. If you do not have this section in your configuration.yaml, please add the following snippet (indention is 2 spaces!):

```
com.instana.plugin.php:
  tracing:
    enabled: true
```

By default we do not activate the PHP tracing at the moment. We will activate PHP by default soon

The PHP sensor is linked to a PHP SAPI sensor. This means, tracing will only start when the Instana Agent detects a supported SAPI sensor (currently PHP-FPM).
The PHP sensor will gather information about the SAPIs configuration and download an appropriate Instana extension based on your version of PHP and your host's architecture.

Supported PHP versions on 64 bit architectures

- 5.3
- 5.4
- 5.5
- 5.6
- 7.0

Required PHP extensions

- JSON

Currently unsupported

- Automatic setup for php running in containers (work in progress)

Once the Instana Tracing extension is downloaded, it will place n zzz_instana.ini file either into your additional ini directory or add it to the ini file used by the PHP SAPI.

If your SAPI uses preforked workers you need to restart PHP.
This is all you need to do.

You find the these detailed installation instructions also at
http://docs.instana.com/articles/instana-agent-configuration-php.html

PHP Tracing currently supports:

Framework/Library	**Supports**
PHP	Entry
file_get_contents	Exit
fastcgi_finish_request	Event
PDO	Exit
MongoDB	Exit
Predis	Exit
cURL	Exit
Memcache	Exit
Session	Exit

Custom Java Instrumentation SDK

Instana automatically instruments well known frameworks for calls coming into a monitored JVM, which we call Entry (also known as Server in OpenTracing), and calls leaving a monitored JVM, which we call Exit (also known as Client). Usually calls pass instrumented code twice: at the start of a request, and at the end. The time elapsed between those two passings is the duration this call took. This duration, along with additional specific metadata is collected in so called Spans.

For the case where Instana does not (yet) know how to instrument a framework, or for monitoring the requests of a very custom application, the SDK can be used.

There is little code needed to enhance monitoring of a custom application. The most common use cases are:

- Adding a new Entry
- Adding a new Exit
- Collecting additional Metadata

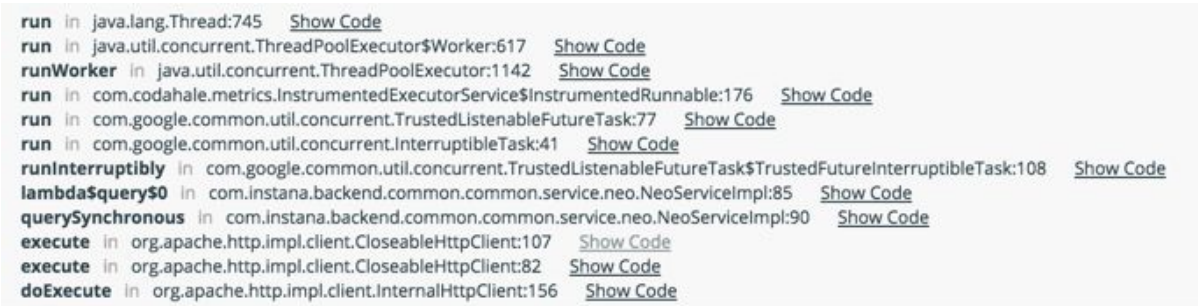
By design, the SDK will remain inactive when no Instana agent is monitoring the JVM process. It will also return to inactivity when the Instana agent is stopped. Therefore it is safe to keep it in your application code even when deploying to systems not yet monitored by Instana.

The whole SDK contained in this repository is provided with an MIT License to allow any use and conflict with strict open source licensing requirements.
Please find more details at https://github.com/instana/instana-java-sdk

We will publish a more detailed Blog Post about this topic soon at https://www.instana.com/blog/

Java Code View

You can now review what Java Code runs in your deployed Java Applications by a simple click in the UI of Instana!
When you open up a Java Stack Trace in the Trace View you can click on “View Code”. Then the Agent will on demand download the selected class and Instana will show the Code of it. We neither download all your application code, nor will we store any code you view.



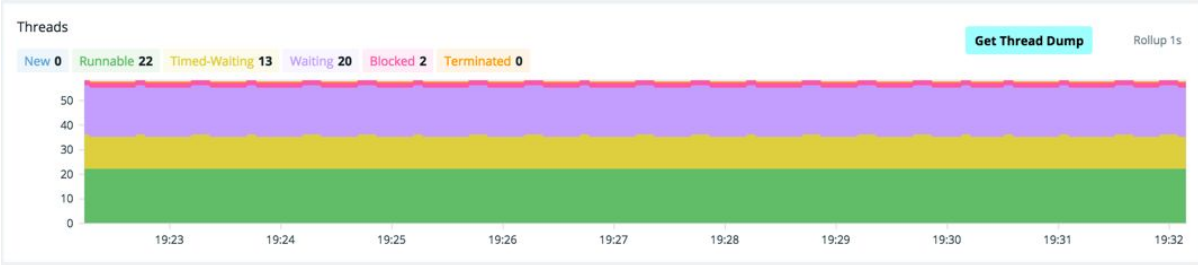
File: org.apache.http.impl.client.CloseableHttpClient

```
package org.apache.http.impl.client;

import org.apache.http.annotation.*;
import org.apache.commons.logging.*;
import org.apache.http.protocol.*;
import java.io.*;
import org.apache.http.client.methods.*;
import org.apache.http.client.utils.*;
import java.net.*;
import org.apache.http.client.*;
import org.apache.http.util.*;
import org.apache.http.*;

@ThreadSafe
public abstract class CloseableHttpClient implements HttpClient, Closeable
{
    private final Log log;

    public CloseableHttpClient() {
/* 58*/     this.log = LogFactory.getLog(this.getClass());
    }

    protected abstract CloseableHttpResponse doExecute(final HttpHost p0, final HttpRequest p1, final HttpContext p2) throws IOException, ClientProtocolException;

    public CloseableHttpResponse execute(final HttpHost target, final HttpRequest request, final HttpContext context) throws IOException, ClientProtocolException {
/* 71*/     return this.doExecute(target, request, context);
    }

    public CloseableHttpResponse execute(final HttpUriRequest request, final HttpContext context) throws IOException, ClientProtocolException {
/* 81*/     Args.notNull((Object)request, "HTTP request");
/* 82*/     return this.doExecute(determineTarget(request), (HttpRequest)request, context);
    }

    private static HttpHost determineTarget(final HttpUriRequest request) throws ClientProtocolException {
/* 88*/     HttpHost target = null;
/* 90*/     final URI requestURI = request.getURI();
/* 91*/     if (requestURI.isAbsolute()) {
/* 92*/         target = URIUtils.extractHost(requestURI);
/* 93*/         if (target == null) {
/* 94*/             throw new ClientProtocolException(new StringBuilder().append("URI does not specify a valid host name: ").append((Object)requestURI).toString());
             }
         }
```

Note that we show the runtime code, which not necessarily represents the actual source code used to compile the class.

Java Thread Dumps

You can fetch Java Thread Dumps on demand directly from the JVM Dashboard



The Agent will gather the data after the click and the UI will present it as soon it is available

Thread dump for JVM: issue-tracker-1.0.0-SNAPSHOT.jar server configs/config.yaml

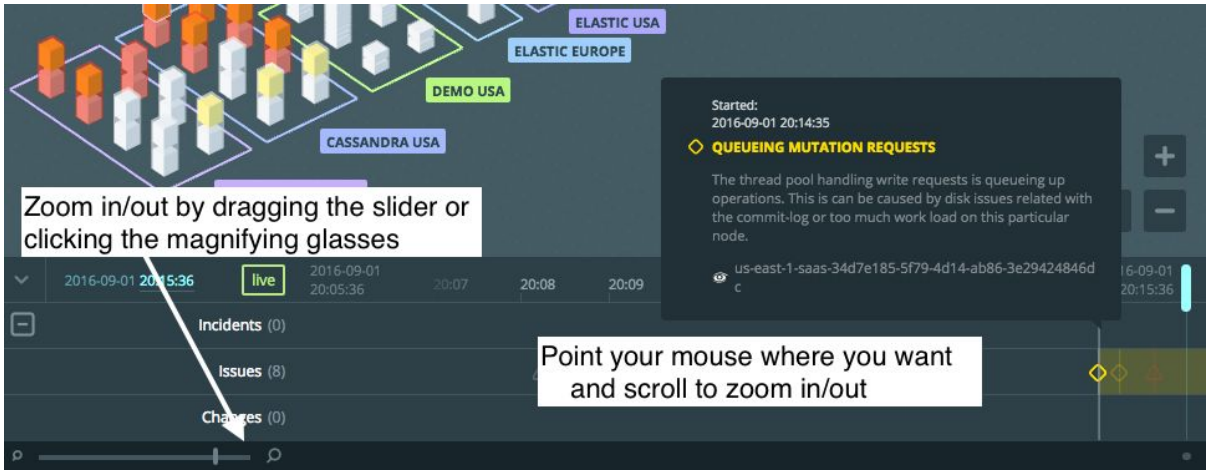
```
Reference Handler id=2 state=WAITING
    - waiting on <0x00c2712d> (a java.lang.ref.Reference$Lock)
    - locked <0x00c2712d> (a java.lang.ref.Reference$Lock)
    at java.lang.Object.wait(Native Method)
    at java.lang.Object.wait(Object.java:502)
    at java.lang.ref.Reference.tryHandlePending(Reference.java:191)
    at java.lang.ref.Reference$ReferenceHandler.run(Reference.java:153)

Finalizer id=3 state=WAITING
    - waiting on <0x15b8de81> (a java.lang.ref.ReferenceQueue$Lock)
    - locked <0x15b8de81> (a java.lang.ref.ReferenceQueue$Lock)
    at java.lang.Object.wait(Native Method)
    at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:143)
    at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:164)
    at java.lang.ref.Finalizer$FinalizerThread.run(Finalizer.java:209)

Signal Dispatcher id=5 state=RUNNABLE

log-message-cache-reset-0 id=12 state=TIMED_WAITING
    - waiting on <0x43bb8a31> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject)
    - locked <0x43bb8a31> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject)
    at sun.misc.Unsafe.park(Native Method)
```


Charts Rework

The Charts experienced a full rework in order to further scale and improve the quality. Also when you deselect a metric now the charts resize, so that you can investigate the metrics deeper.

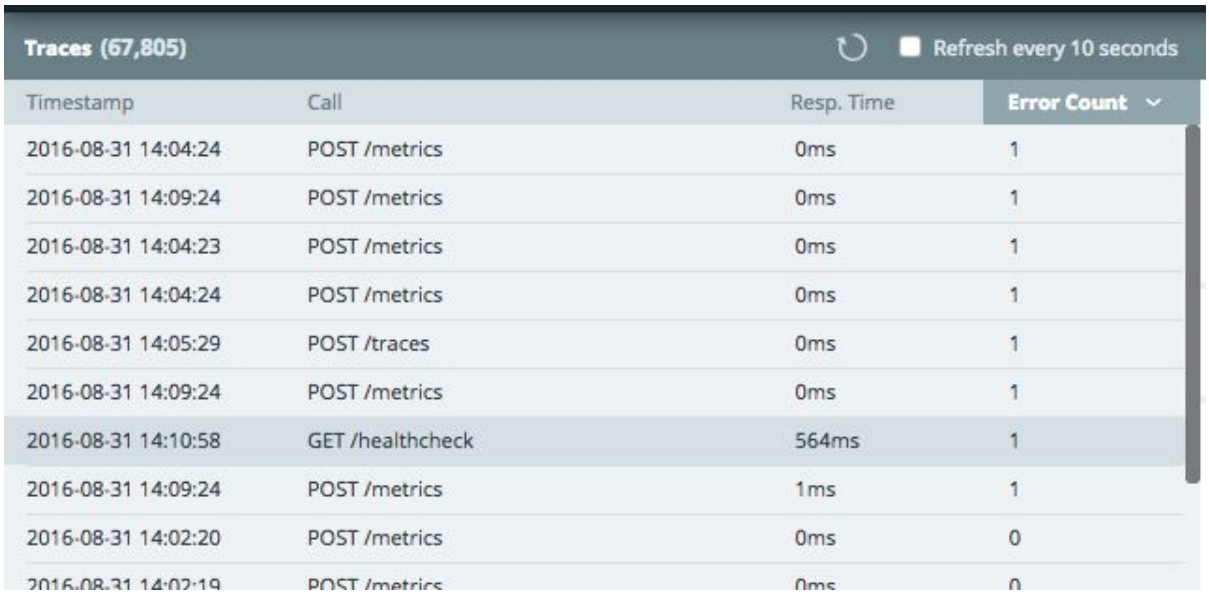
Timeline Drill down to 1 Minute

The Timeline is now able to drill down to 1 Minute Granularity so that you can investigate the Events and Metrics in more detail.



Error Sorting in Trace View

You are now able to sort the Traces by error count



License Overview for SaaS

In the Management Portal under Usage SaaS Customers can review their licenses.

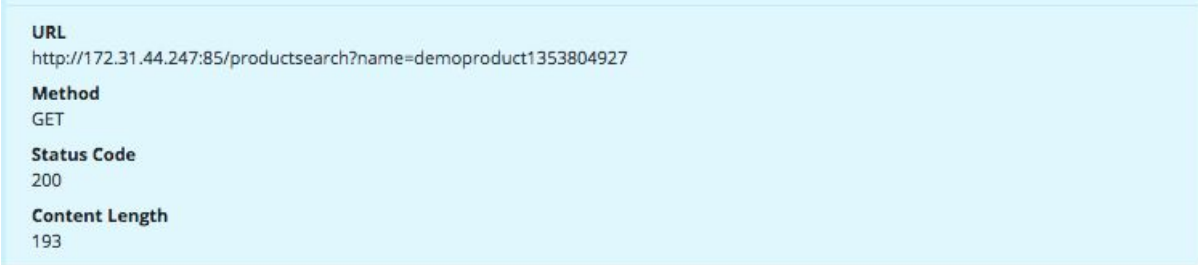
License Activation and View for On Premise

On Premise Customers have another Tab “License” in the Management Portal in which Licenses can be actived and renewed.

Please find more details at
http://docs.instana.com/articles/instana-onpremise-license-activation.html

Response Size tracking in Traces

Response Sizes are important to understand performance. Instana now gets response sizes for requests and displays them in the Spans.



Newly supported technologies

- **okhttp3 tracing**
- **Resteasy tracing**

Improvements

- Showing metrics on logical map earlier
- Improve Service Instance Naming
  We now better resolve the Service Instances by traversing the Graph and taking the communication into account
- Events show the exact moment in time they started
- Trace Drill down on Services shows count of available Traces
- Apache Modules overview cleaned up and added local search
- If nothing is selected pressing “F” on the keyboard leads to center and zoom the map to fit the screen
- 3D is way faster and consumes less resources
- Metrics, Tags and Table View on the Physical View are now available directly on the map
- Detecting rejected connections in Redis
- Showing Redis max memory configuration
- Not attaching to jstat java process
- MySQL wait states are now handled nicer
- Tuned Sudden Drop Detection
- Reduced overhead of rabbitMQ monitoring
- Support for memcached authentication and better representation of auth errors
- Improved stability of the Instana Agent Windows service (InstanaPCP.exe)
- Optimized performance of sending large amounts of data
- Trace instrumentation is now faster and uses less memory

Fixed Issues

- Sorting of Traces was inconsistent
- Memcached Sensor did not connect in rare scenarios
- Search in Trace View does not support forward slashes (/).
- Postgresql JDBC driver is now instrumented correctly
- Under high load traces could have appeared twice. This has been resolved.

Known Issues

Please feedback via Slack or Mail to 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!