

On the Radar: Zipkin distributed tracing system helps troubleshoot latency problems in production

An open source technology project for the resolution of latency issues in microservices architecture

Publication Date: 07 Jun 2016 | Product code: IT0022-000695

Michael Azoff



Summary

Catalyst

Distributed systems are a particular challenge in performance monitoring, more so with enterprise applications based on microservices and containers. This is because the scale can increase and they are also highly dynamic and transient. Zipkin is an open source project that addresses this challenge with a distributed tracing system for monitoring latency issues. It has been designed with a number of features that make it ideal for microservices. Finding and eliminating abnormal delays to events is crucial in mission-critical production systems. Zipkin gathers the timing data needed to troubleshoot these latency problems and manages both the collection and visualization of the data, enabling operations staff to rapidly pinpoint and fix problems.

Key messages

- Zipkin is designed for live production systems by having low overheads (order of bytes) for its agents, which inhabit message headers.
- The tool has a plug-in architecture that allows tracers for different computer language runtimes to be connected.
- Zipkin is easy to install and run using a Docker image and in-memory server.
- It has a strong community of users and stakeholders and a high adoption rate using well established technology.
- Zipkin follows the Google Dapper paper, which was a well thought out design for a tracing system.
- Zipkin is still evolving and there are many areas that can and are being enhanced, such as the Data Format and a rewritten drop-in-compatible server built on Spring Boot. There are also improvements planned for search and the UI.

Ovum view

There are many comprehensive application performance monitoring solutions on the market, but for some tasks dedicated solutions can provide the right balance that can match or improve on existing full-fledged products. The problem of tracing in distributed systems, which are often of large scale as arises in microservices architecture, is one such challenge. Due to the dynamic and even ephemeral nature of these systems, monitoring in live production is essential, but this brings with it the concern from operations of overheads and footprint sizes. Zipkin overcomes these issues by only passing forward information in the headers of messages and data packets that the forward node needs to know in order to trace the parent-child relationship of message passing. At each node the detailed tracing information required for analysis, as configured by the tool administrator, is dumped into a database so that it is out of the process under investigation. The amount of data collected needs to be managed carefully so that it does not overwhelm the user of the solution. The levers the user can control are typically the range of details required to be sampled, the sampling rate, and the extent of the network being examined. A typical sample setting would be to examine 10% of all traffic.

Zipkin is a powerful tool to help those with complex microservice architectures understand and troubleshoot latency problems in production. By separating the process of sending trace identifiers from details, it adds only a low overhead, which means Zipkin is safe to "leave on" in production.

Zipkin's distributed tracing assists in pinpointing latency issues faster and with less human effort or the need for detailed knowledge about the architecture. Not all operations are relevant or on the critical path but those that are and have delays are critical to the production performance.

Recommendations for enterprises

Why put the Zipkin distributed tracing system on your radar?

Zipkin's wide adoption is the result of a fine pedigree from Google's research and Twitter's implementation in addition to the urgent need to trace latency issues in complex microservices-based production systems. Any organization using or contemplating the use of such systems should therefore be familiar with Zipkin and its associated technologies.

Zipkin provides near real-time information, where "near" can mean minutes or less. It is not designed for long-term storage and analysis, although the user could easily send data into a data warehouse. Data gathered is typically stored at most for a few days, and its main use is in helping with problems in production as they occur.

Highlights

Technology

A "trace" is an end-to-end latency graph composed of spans. A span is an individual operation that took place and contains timestamped events and tags. A trace takes spans that are linked and collects data. In addition to facilitating inspection of traces and spans, Zipkin provides a visual graph of the connections that helps to provide a high-level view of the system under inspection.

Zipkin is a tracing system and the individual runtime tracers need to be plugged in. Popular tracers used in the community include:

- Finagle for Scala
- Brave for Java
- HTrace for C and Java
- ZipkinTracer for Ruby
- ZipkinTracerModul for .NET
- Pyramid_zipkin for Python
- Open Tracing API for Go, Python, Java, and JavaScript
- Spigo (Simulate Protocol Interactions in Go) for Go

Enterprises using Spring Boot should integrate with Zipkin using Spring Cloud Sleuth. Spring Cloud Sleuth automatically instruments Boot apps, so that common components are traced appropriately. It is maintained by distributed tracing and Spring Boot experts, who keep it current and supportable.

Distributed tracing systems collect end-to-end latency graphs (traces) in near real time. In practice the user looks to compare similar interactions in the system happening at the same time, less for historical comparisons. This is due to the dynamic nature of the system under observation, which is the norm for microservices and container-based applications. The user will therefore compare similar traces taking place concurrently to understand why certain requests take longer than others.

Applications are instrumented to report timing data to Zipkin via the tracers listed above. The Zipkin UI also presents a dependency diagram showing how many traced requests went through each application. If you are troubleshooting latency problems or errors, you can filter or sort all traces based on the application, length of trace, annotation, or timestamp. Once you select a trace, you can see the percentage of the total trace time each span takes, which allows you to identify the problem application.

Tracing information is collected on each host application using the instrumented libraries and sent to Zipkin. When the host application makes a request to another application, it passes a few tracing identifiers along with the request to Zipkin so it can later tie the data together into spans.

Tracers typically:

- Aggregate spans into trace trees.
- Provide query and visualization for latency analysis.
- Have short retention policies (usually days).
- Collect timing data and transport it over HTTP or Kafka.
- Include collectors that store spans in MySQL or Cassandra (other databases are possible).

Business Context

Zipkin helps analyze a current architecture view in production, not what the graphs should be, but the actual descriptive data showing the reality. It is important, for example, to evoke the IPs or the shards that were in use at the time of an issue because these dynamically change and cannot be easily reproduced.

With distributed computing, the context has to be more specific than a service because there could be 100 shards in the service doing the same type of work. Zipkin will therefore often capture more metadata, such as “who is my peer and who's actually talking to me at the time?” and “what cluster do I belong to?” This is often a combination of analysis configuration and discovery service or other configuration, but with the combination of configuration and logs. Organizations need to get to the business context of latency, such as, for example, “is this one-and-a-half seconds abnormal?” Zipkin uses various policies to look at events and flag up abnormalities.

When organizations start to use tracing they often find that they are able to trim application latencies just by removing dependencies that should not have been there in the first place. Because of the complexities of modern architectures it is not possible to draw them on paper, and it is very easy to get into a position where there are unnecessary dependencies and indirections. Latency troubleshooting with Zipkin provides insight into how the services are being used, and stakeholders can examine a graph and question why, for example, a service call is replicating to such an extent.

Background

Zipkin's design is based on the Google Dapper paper, "Dapper, a Large-Scale Distributed Systems Tracing Infrastructure" by Sigelman et al, and was originally created by Twitter in 2012 to help pinpoint latency problems related to "fail whales". Last year Zipkin became community-driven when the OpenZipkin project started, led by employees of Pivotal, SoundCloud, Finn (a well-known Norwegian classified site), Lookout, Prezi, and The Last Pickle. Twitter remains active in Zipkin, with tracing built into its Finagle RPC platform.

Current position

Zipkin collaborates closely with the OpenTracing initiative, which aims to clean up and de-risk distributed tracing instrumentation with interfaces that decouple instrumentation from vendor-specific dependencies and terminology. This enables applications to switch products with less effort. Zipkin also works closely with other open groups including Apache HTrace, which centers on tracing data systems such as Hadoop.

Since 2014 OpenZipkin has gained momentum with active contributors including Uber, Yelp, and Line (messenger app in Taiwan and Japan). This has resulted in new areas of functionality including .NET, Spark, and Elasticsearch integration, as well better user support and documentation.

The expectation is that as an open source project Zipkin will become the standard reference for this type of application. The closest thing to a standard is currently the OpenTracing project, and although the scope is narrow it has been helpful because it has done quite a lot to catalog concepts and clean up some of the value propositions of being consistent about tracing and how this applies to systems. It could possibly be in 2016 that a tracing system standard is agreed and it's definitely much closer now than it has ever been. One of the reasons for this is that through the working group, people are agreeing to common vocabulary and working with each other, even though they have diverse interests.

Those currently involved in enhancing Zipkin are the operators of distributed systems including Yelp, Uber, and Twitter that have large distributed systems that require systems like this to exist. For whatever reason, they don't tend to use commercial software and so they participate directly.

The individuals involved are small in number, with Twitter having only one person on tracing. Groupon has a maximum of four, Uber might have two or three, with Pivotal having one or two.

The use base for Zipkin is much larger and Ovum believes it will grow dramatically as microservice architectures become mainstream.

Data sheet

Key facts

Table 1: Data sheet: Zipkin

Product name	Zipkin	Product classification	Tracing system
Version number	n/a	Release date	2014
Industries covered	All	Geographies covered	Global
Relevant company sizes	Mainly large	Licensing options	Open source

Number of employees	Open source project	Routes to market	Direct and through partners
---------------------	---------------------	------------------	-----------------------------

Source: Zipkin/Pivotal

Appendix

On the Radar

On the Radar is a series of research notes about vendors bringing innovative ideas, products, or business models to their markets. Although On the Radar vendors may not be ready for prime time, they bear watching for their potential impact on markets and could be suitable for certain enterprise and public sector IT organizations.

Authors

Martin Gandar, Associate Senior Analyst

Michael Azoff, Principal Analyst, Infrastructure Solutions

michael.azoff@ovum.com

Ovum Consulting

We hope that this analysis will help you make informed and imaginative business decisions. If you have further requirements, Ovum's consulting team may be able to help you. For more information about Ovum's consulting capabilities, please contact us directly at consulting@ovum.com.

Copyright notice and disclaimer

The contents of this product are protected by international copyright laws, database rights and other intellectual property rights. The owner of these rights is Informa Telecoms and Media Limited, our affiliates or other third party licensors. All product and company names and logos contained within or appearing on this product are the trademarks, service marks or trading names of their respective owners, including Informa Telecoms and Media Limited. This product may not be copied, reproduced, distributed or transmitted in any form or by any means without the prior permission of Informa Telecoms and Media Limited.

Whilst reasonable efforts have been made to ensure that the information and content of this product was correct as at the date of first publication, neither Informa Telecoms and Media Limited nor any person engaged or employed by Informa Telecoms and Media Limited accepts any liability for any errors, omissions or other inaccuracies. Readers should independently verify any facts and figures as no liability can be accepted in this regard – readers assume full responsibility and risk accordingly for their use of such information and content.

Any views and/or opinions expressed in this product by individual authors or contributors are their personal views and/or opinions and do not necessarily reflect the views and/or opinions of Informa Telecoms and Media Limited.

CONTACT US

www.ovum.com

analystsupport@ovum.com

INTERNATIONAL OFFICES

Beijing

Dubai

Hong Kong

Hyderabad

Johannesburg

London

Melbourne

New York

San Francisco

Sao Paulo

Tokyo

