Zipkin Filtering Proxy Design

By Andrey Falko, Principal Engineer, Salesforce

Disclaimer

This document is currently light on implementation details because it is seeking buy-in from the Zipkin community on the overall strategy first. Ideally, this document will evolve to include an iterative step-by-step plan.

Problems we are solving

Service Abuse

We run distributed systems at scale and Zipkin helps our developers build better software. As Zipkin gets more utilized and as more non-homengous services get deployed by global teams we are finding that we need to protect our Zipkin service from abuse. Zipkin gets abused by our developers in three major ways:

1. Developers underestimate their sampling rates. This increases network and storage cost.

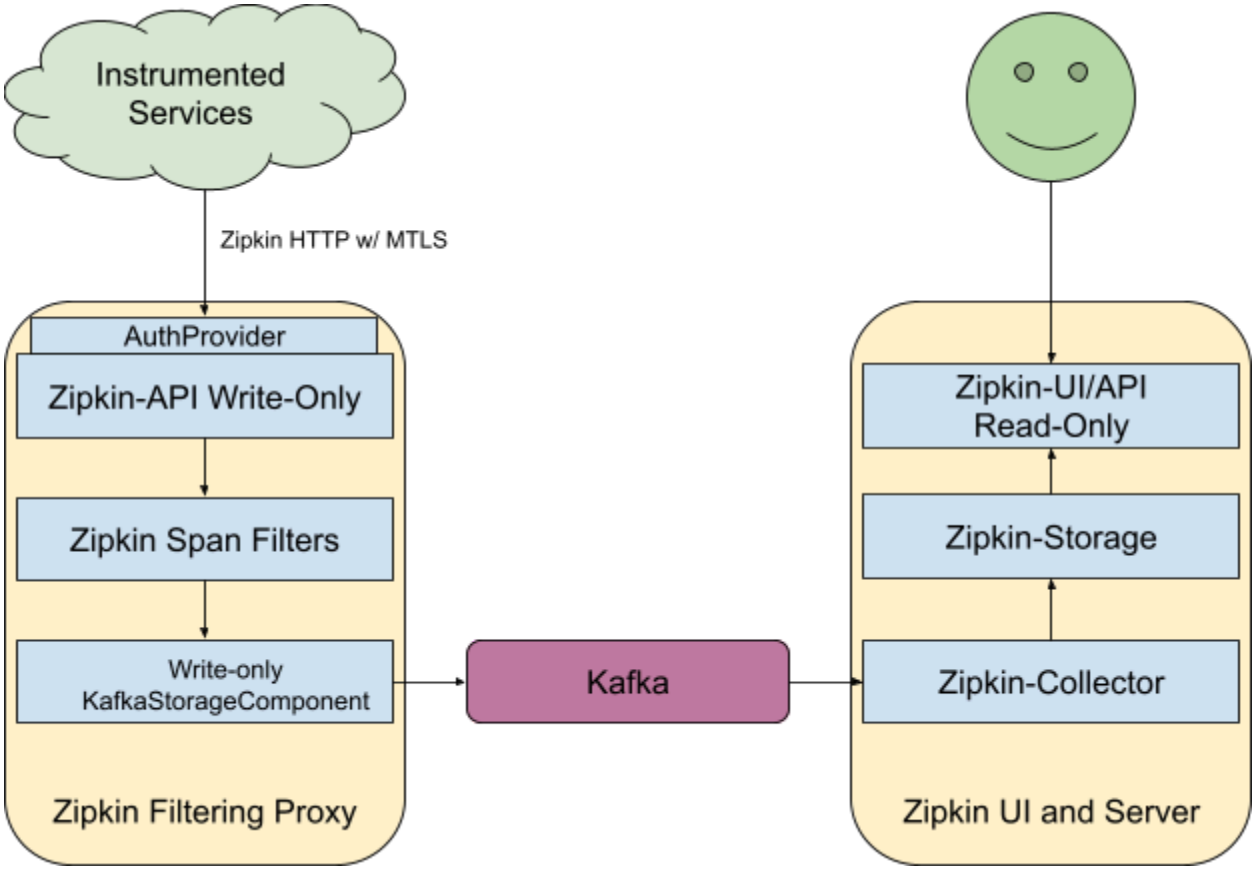
2. Developers use tags and annotations that other teams are already using. This reduces productivity because developers need to make hand-shake agreements to stay away from certain tags and annotations.
3. Developers unintentionally put GDPR data into the free form tags and annotations fields. This is an administrative burden because data needs to be identified and then deleted in a timely manner.

Instrumentation Library Compatibility

Our services are globally distributed and we use Kafka to stream traces to a central Zipkin cluster. Many instrumentation libraries in use for Zipkin support publishing directly to Kafka. However some libraries, such as those used in Envoy do not have that capability. It takes time and effort to contribute to these libraries. New libraries that are created have to repeat the same work as well, increasing development cost of new instrumentations.

Proposed Changes To Zipkin

In a nutshell, we'd like to run zipkin-server as a proxy that pushes messages to a Kafka bus that is processed by zipkin-collector downstream. We'd like to configure this proxy to track quotas per-service and drop spans that violate GDPR and other admin-established constraints. What follows is a set of changes we'd like to contribute to the Zipkin project. We don't want to fork it internally and would love to give something useful back to the community. Here is an illustration of the changes and one of the architectures our changes would support:



Zipkin administrators would be free to skip the Kafka bus and use the Authentication Interface and filter interface. Likewise, they would be able to define their own filter implementations without using the Authentication provider. We have our own goals and our own idea of minimum viable set of changes we plan to contribute.

Zipkin-Server Authentication Interface

In order to do quotas and other filtering we need to determine the origin of the service that is sending. We also need to prevent other services from spoofing other services. For our minimum viable set of changes, we plan to only support a MTLS implementation. The key output that the interface defines in a unique service name. This name will be used by the filter interface below. This work may or may not provide an interface that can solve issue 782.

Zipkin-Server SpanFilter Interface

Before the accept call to any storage component, we would run through a user-defined list of filters. By default, no filters would be defined and Zipkin would work for admins the way it works today. New filters would live in repositories outside of Zipkin and would be added via a combination of configuration and classpath injection. Autodiscovery through classpath can be added later.

The filter interface would need to support returning either a “drop” or “changed” result. In other words, the filters could change fields that they pass on to the storage system or drop spans all together. See below for details on that.

ServiceQuotaExceededFilter

This filter would take in static configuration that would have a mapping of how many spans are allowed per unique service per time period. The minimum viable set of changes for us is the ability to set higher quotas for some services and lower quotas for other services.

If a service exceeds quota, spans get dropped and the sending service is sent 429 errors back with a message saying quota exceeded. When the time window ends, spans are allowed back in.

We are open to other ideas. Filters with better quota management implementations would be welcome.

ServiceTagMappingFilter

This filter would take in a static configuration where services claim tag and annotation identifiers. If another service comes in and tries to use a reserved tag, it will get a 409 (conflict) error from our server. An alternative scheme can be developed whereby annotations and tags are namespaced, the drawback of that would be an increase in storage requirements.

Another way to implement this filter is to remove conflicting tags, but not drop the message. In that case, the regular 202 http message is sent back to the caller.

GDPRFilters

These filters depend on the company and the nature of the data processed by the apps of that company. There might be cases where the implementer might want to enrich spans or replace the data within the spans. These would likely never have any use if they are open sourced. They are not in the minimum viable change path.

Write-only KafkaStorageComponent

We want to preserve all of the code that is involved in Zipkin’s span handler. Implementing the accept call in new storage component will allow us to send raw spans to our Kafka Bus. Zipkin-collector can pick those spans up on the other end and write them to a normal storage backend that can be used by Zipkin UI.

Abandoned Ideas

Alternative Passthrough Solution

We can build a completely decoupled server, separate from the zipkin repo. Rather than decoding and peeking at the spans send to the system, we send raw data received to the kafka bus. We would use KafkaSender to send these spans or write our own producer. If we use KafkaSender we will need to add a new interface method that takes a byte array rather than a List of byte arrays to this supplant the existing method.

Drawbacks

- Users wouldn't get feedback if their traces are dropped. We would only be able to send a non-200 message back to the sender when kafka refuses to accept more traffic. Kafka can only set a global quota on the Kafka topic. Users that are sending too many spans might cause spans sent by other users to be dropped in this case.