Distributed Tracing

Nicola Mössner
2268685m

17. January 2020

1 Background

When reflecting on the Software Development Life Cycle (SDLC) of different projects I have personally been involved in, observed, or read about, it becomes apparent to me that the last stage which is often described as the 'Maintenance stage' forms a major part of the whole software development process.

Maintenance involves various disciplines, such as:

- Correcting issues/bugs in the code.
- Adapting the behaviour of the application to meet changing and evolving requirements.
- Enhancing the performance of the software to improve user experience and satisfaction.
- Updating technologies, platforms, frameworks and infrastructure to keep up with the latest standards and innovations, and making necessary changes to the software accordingly.
- Monitoring and analysing the behaviour and performance of the software.

It is in fact widely known and agreed on that the maintenance cost far exceeds the development costs for most software projects. Maintaining and improving an existing software application/solutin consumes a vast amount of resources and this is no different at my current employer, ResDiary.

ResDiary provides its customers with a reservation and table management system for the hospitality sector. The nature of this industry demands highly available and well performing software solutions. Lacking in those areas can result in loss of profit for our customers. For this reason we are constantly monitoring and enhancing the stability and performance of the services we provide.

To improve the process of pinpointing potential areas of improvement, we are eager to introduce *Distributed Tracing* to our appliactions.

> Distributed tracing helps pinpoint where failures occur and what causes poor performance. [**?**]

With this in mind, we expect this method to assist us in identifying issues in our software that we can address either with an immediate quick fix, or a with a complete overhaul of a particular software component, sql query, etc.

2 Problem Description

Occasioanl long request times,

With the growing trend of microservices, orchestrated into large-scale applications, it becomes more challenging to trace errors in the code, to spot bottle-necks and slow parts of the application, to find buggy code or to simply find areas of improvement. For example, when a request to a web application takes longer than expected, it is difficult to determine if this was caused by part of the application code, a database query, perhaps some code in another web-app that the request gets routed to or a call to a third party application/library.

At ResDiary we currently have no distinct approach to investigate the cause for slow or failing requests. At the moment we resort to log messages and graphs depicting the health and/or uptime of components and infrastructure. This approach can be very broad and slow as it takes time to pin down the exact nature of an issue and may involve some guesses and investigation into a wrong direction at first.

Tracing an incoming request from start to finish, and timing how much time it takes within the different components and perhaps even functions and methods of the whole application would shed some light onto those existing issues and provide a clear and straight forward way of determining areas of improvment.

Another aspect where we would hope to see benefits from such a tool, is an improved ...(response/reaction/fixing) time for our on-call developers who deal with alerts relating to our web-applications and databases. Allwoing them to use a 'Distributed Tracing System' to track for example slow or failing requests from users would give them an insight into where the problem occurs and allow them to fix it faster.

- How do we trace errors at the moment (Grafana, Stackify, Kibana)?

- Experiences at ResDiary: Query tuning (try to improve response time of a request, only after the enhancements were made, deployed and observed over time in production, we can say for sure if the improvements to the query had an impact. This is a slow process, through tracing we could say for sure beforehand if the query was the issue, the thing that took too long.)

3 Objectives

The choice of an appropriate *Tracing Framework* is part of the Work Plan (see section 4), hence for now I will refer to it as *Tracing Framework.* Possible choices are *OpenTracing* and *.NET Core 3.0 Diagnostics.*

In the same manner, I will refer to the eventual implementation choice of the *Distributed Tracing System* such as *Jaeger* or *ZipKin* simply as *Distributed Tracing System.*

1. The *Tracing Framework* is implemented in our *.NET Core* applications.

2. Delivered a *helm* chart for the *Distributed Tracing System* that can be deployed locally and to Azure Kubernetes Services (AKS).

3. The *Distributed Tracing System* is running in the ResDiary AKS staging cluster.

4. The *Distributed Tracing System* is running in the ResDiary AKS production cluster.

5. Traces gathered through the *Tracing Framework* are visualised meaningfully in the *Distributed Tracing System.*

6. Delivered an experience report from the DevOps team and on-call developers at ResDiary, discussing some of the following concerns:

- How has this new approach affected the process of finding issues in general?
- How has it affected the time to get to the bottom of (and resolve) an on-call support incidents?
- Does this additional tool increase your confidence to track and resolve issues while on-call or in your everyday work?
- Does the data gathered and depicted in the *Distributed Tracing System* provide useful indications for areas of improvement.

4 Work Plan

Decide on *Tracing Framework* (16.10.2020)
Investigate the differences, pros and cons between frameworks (OpenTracing, etc.) and choose the most appropriate one.
Outcome: Brief statement justifying the decision.
Assignees: Nicola Mössner*

Decide on *Distributed Tracing System* (16.10.2020)
Investigate the differences, pros and cons between systems (Zipkin, LighStep, Jaeger, etc.) and choose the most appropriate one.
Outcome: Brief statement justifying the decision.
Assignees: Nicola Mössner*

Instrument applications with Tracing Framework (30.10.2020)
The existing .NET Core applications need to be instrumented with the tracing framework for it to be able to trace requests.
Outcome:
Assignees: Nicola Mössner*

Deploy Distributed Tracing System locally (06.11.2020)
Configure and deploy a helm chart for the Distributed Tracing System on a local Kubernetes cluster.
Outcome: All resources are created as expected and Pods are running in local Kubernetes cluster.
Assignees: Nicola Mössner*

Tracing Framework communicating with Distributed Tracing System locally (27.11.2020)
...
Outcome:
Assignees: Nicola Mössner*

Deploy Distributed Tracing System to staging environment (01.01.2021)
Configure and deploy a helm chart for the Distributed Tracing System on a local Kubernetes cluster.
Outcome: All resources are created as expected and Pods are running in ResDiary AKS staging cluster.
Assignees: Nicola Mössner*

Tracing Framework communicating with Distributed Tracing System in staging environment (15.01.2021)
...
Outcome:
Assignees: Nicola Mössner*

Deploy Distributed Tracing System to production environment (28.01.2021)
Configure and deploy a helm chart for the Distributed Tracing System on a local Kubernetes cluster.
Outcome: All resources are created as expected and Pods are running in ResDiary AKS production cluster.
Assignees: Nicola Mössner*

Tracing Framework communicating with Distributed Tracing System in production environment (12.02.2021)
...
Outcome:
Assignees: Nicola Mössner*

Visualise meaningful traces in Distributed Tracing System (26.02.2021)
...
Outcome:
Assignees: Nicola Mössner*

Finish experience report of developers using the tool (26.03.2021)
...
Outcome:
Assignees: Nicola Mössner*

* Signifies the lead responsible for ensuring the work is completed.