Una infrastruttura per la raccolta di dati di telemetria in ambienti distribuiti: OpenTelemetry

Michele Laddaga
michele.laddaga@studio.unibo.it

May 2022

Questo progetto ha lo scopo di analizzare e testare la struttura e le principali feature fornite dall'infrastruttura OpenTelemetry (https://opentelemetry.io). OpenTelemetry è uno standard aperto per i dati di telemetria, definisce inoltre la tecnologia per raccogliere ed esportare i dati dalle applicazioni distribuite per poterli monitorare e analizzare.

Indice

1 Obiettivo

Il progetto si focalizzerà sull'analisi delle caratteristiche dello standard OpenTelemetry (OTel) verificando inizialmente le soluzioni proposte atte alla risoluzione delle problematiche di gestione di dati di telemetria in ambienti distributi andandole in seguito a testare in un ambiente di prova. Dopo una iniziale introduzione alla telemetria si andranno ad analizzare le caratteristiche e gli strumenti offerti da OpenTelemetry con una breve parentesi su alcuni sistemi back-end di visualizzazione dei dati raccolti. Si terminerà con un esempio pratico in .NET di raccolta di dati telemetrici attraverso OpenTelemetry.

2 Introduzione alla telemetria

La telemetria è un processo automatizzato di comunicazione attraverso il quale vengono effettuate misurazione e vengono collezionati dati remoti [3].

In ambito software si tratta di recuperare dati relativi a utilizzo e performance di applicazioni e relativi componenti come ad esempio:

- misurazioni dei tempi di avvio e di esecuzione di applicativi
- frequenza e tempistiche di utilizzo di determinate feature
- tracciamento di anomalie/crash applicativi
- statistiche di utilizzo

Questa tipologia di raccolta dati può rivelarsi essenziale nello sviluppo software avendo accesso a dati provenienti da una grande varietà di endpoint, varietà difficilmente ottenibile internamente. Si rivela poi strategica per identificare eventuali funzionalità apprezzate dal cliente su cui focalizzarsi o al contrario scarsamente utilizzate e quindi da rimuovere o porre in secondo piano.

3 Telemetria nei sistemi distribuiti

3.1 Cos'è un log?

Il termine viene utilizzato per indicare [1]:

- la registrazione sequenziale e cronologica delle operazioni effettuate, da un utente, un amministratore o automatizzate, man mano che vengono eseguite dal sistema o applicazione;
- il file o insieme di file su cui tali registrazioni sono memorizzate ed eventualmente accedute in fase di analisi dei dati, detto anche registro eventi.

Il logging tradizionale generalmente mette a disposizione l'intero stack trace di esecuzione che può aiutare a risolvere eventuali errori applicativi.

Il problema di questi log è il grande numero di informazioni che si vengono a creare con una conseguente inefficienza di eventuali sistemi di ricerca di dati.

Archiviare e analizzare dati di log sono operazioni dispendiose quindi diventa essenziale gestire solamente ciò che può aiutare ad identificare le problemariche.

I livelli di log (Error, Warning, Info, Debug e Trace) permettono di categorizzare i messaggi aiutandone la comprensione e il monitoraggio. Potendo variare il livello di logging a runtime senza modificare il codice sorgente si possono ottenere dettagli diversi in base alle necessità del momento.

Un altro aspetto molto importante è la standardizzazione del messaggio per poterlo analizzare più semplicemente, Questo si ottiene memorizzando i log in un formato strutturato che può essere parserizzato più semplicemente del semplice formato testuale (utilizzando ad esempio il JSON).

3.2 Monitoraggio e analisi di un applicativo distribuito

Come visto nelle sezioni precedenti i log forniscono un contesto di una singola applicazione, come vedremo a breve il trace distribuito invece aiuta a tracciare le richieste mentre si spostano attraverso varie applicazioni interdipendenti.

Entrambi aiutano nelle operazioni di debugging e analisi dei problemi e permettono non solo di monitorare l'intero sistema in tempo reale ma anche di andare indietro nel tempo per identificare eventuali problemi e disservizi.

I Log catturano lo stato dell'applicazione e sono il più elementare strumento di monitoraggio.

Il tracing fornisce benefici quando si hanno richieste che attraversano più sistemi fornendo informazioni sui tempi delle richieste, entità coinvolte e latenza introdotta ad ogni interazione.

Ad ogni modo il solo tracing non può fornire la radice del problema alla base di un errore di servizio o una latenza, per quello è necessario analizzare i log. Il tracing aiuta a identificare dove l'anomalia si trova, il log fornisce dettagli aggiuntivi sull'eventuale problema.

Comparato al logging il trace aggiunge più complessità nello sviluppo e risulta anche più dispendioso in termini di risorse.

In base alla tipologia di sistema va valutato se abilitare il tracing: in una struttura a microservizi ad esempio diventa essenziale assieme all'utilizzo di log e metriche per il debug di errori e diagnostica di problematiche produttive.

3.3 Logging centralizzato e distribuito

In questo contesto per logging centralizzato si intende l'aggregazione dei dati provenienti dai singoli applicativi in una posizione centralizzata per un accesso ed una analisi più semplice.

Nei sistemi monolitici le operazioni si strutturano all'interno della stessa macchina: in sistemi distribuiti si potrebbe utilizzare un simile approccio identificando ogni parte come un singolo monolite. Il problema di questo approccio è appunto che vengono raccolti

solamente dati dei singoli applicativi permettendo quindi di risolvere solo problematiche del singolo processo.

Il **logging centralizzato** *colleziona ed aggrega i log* dai vari sistemi *in un solo punto centralizzato* dove vengono indicizzati in un database. In questo modo i dati di log possono essere ricercati, filtrati e raggruppati nel relativo software di gestione dei log per stato, host, livello, origine e timestamp.

Questa gestione fornisce molti vantaggi permettendo di avere tutte le informazioni rilevanti in un solo punto e riducendo quindi le risorse e il tempo necessario alla loro analisi. Inoltre se un server o un container terminano in maniera imprevista anche i loro log non saranno più a disposizione. Centralizzandoli questi restano su un repository prevenendone la perdita.

Poichè questi dati non vengono memorizzati come semplice testo ma attraverso una specifica formattazione (come accennato in precedenza) si ottiene la possibilità di effettuare ricerce più sofisticate e definite fornendo una prospettiva più chiara delle performance del sistema come unica entità. Il **logging distribuito** al contrario lascia i file di log decentralizzati e questo può portare vari vantaggi in base alle dimensioni del sistema:

- spedire i log in un unico posto può consumare parecchia banda e in base alla rete e al numero e frequenza dei log generati si può creare una situazione problematica per il servizio stesso.
- alcuni sistemi di storage dei log inoltre sono più affidabili quando sono più vicini al device che genera il log stesso.
- questa soluzione diventa preferibile per sistemi su larga scala rendendo la centralizzazione dei log più problematica e meno efficiente in termini di costi.

3.4 Cos'è un Trace distribuito?

Il Trace distribuito è un componente chiave di analisi in sistemi interconnessi e come accennato in precedenza si focalizza sul monitoraggio delle performance e sull'identificazione delle problematiche.

Per trace (distribuito) si intende un'insieme di Span: questi sono l'unita di trace più piccola. Possono essere una richiesta HTTP, una chiamata ad un database, l'esecuzione di un messaggio preso da una coda o altre operazioni di interconnessione.

Un Trace permette di vedere come una richiesta viene processata attraverso vari servizi che interagiscono fra loro.

Quando parte una request, ad esempio una HTTP request, viene creato ed assegnato un Trace ID univoco alla richiesta stessa. Man mano che la richiesta si muove nel sistema, ad ogni operazione effettuata su di essa, le viene assegnato un secondo identificativo, chiamato SPAN, il quale sarà composto dall'ID della prima request iniziale unito ad un suo ID univoco e ad un identificativo dell'operazione che ha generato la richiesta (chiamato "Parent SPAN"). Ogni SPAN è un singolo passo nel percorso che effettua la Request ed è codificato con importanti informazioni relative al processo del microservizio che sta eseguendo l'operazione, come ad esempio:

- il nome del servizio e l'indirizzo del processo che sta gestendo la richiesta,
- misurazioni, i Log ed eventi che forniscono un contesto sulle attività del processo,
- Tag per cercare e filtrare le richieste ID di sessione, database host, metodo HTTP e altri identificativi,
- Stack trace dettagliati e messaggi di errori nel caso di eventi relativi a problematiche.

Attraverso degli strumenti di analisi di tracing distribuito, quali Zipkin o Jaeger ad esempio (i quali verrano in seguito approfonditi), è possibile correlare poi i dati degli Span e formattarli in varie visualizzazioni disponibili attraverso delle web interface.

Immagine di esempio di richieta con Jaeger o Zipkin

3.5 Best practice per il log di sistemi distribuiti

Come visto fin'ora il logging di sistemi distribuiti è guidato da una serie di **consuetudini** atte a gestire la natura modulare e potenzialmente eterogenea di questi sistemi. L'obbiettivo è quello di portare coerenza al sistema per una gestione più efficiente e accurata del debugging e più genericamente della risolzione dei problemi.

Riassumendo quanto detto nelle sezioni precedenti queste consuetudini si traducono nei seguenti punti:

- **Correlare le richieste**: ogni servizio in un sistema distribuito interagisce con gli altri per gestire una richiesta. Marcando la richiesta iniziale con un ID univoco diventa possibile tracciarla facilmente attraverso tutto il sistema, identificando potenziali errori e rivelando dove questi nascano permettendo di comprendere se dipendono ad esempio dalla richiesta in analisi o dalla precedente/successiva. Lo sviluppatore potrà cercare tale ID univoco nel software di ricerca dei log utilizzato per recuperare i log da tutti i vari servizi per l'analisi.
- **Informazioni di Logging da includere**: maggiore è il numero di log e maggiore sarà il contesto a disposizione per comprendere il problema. Il nome del serizio che ha generato il messaggio di log, l'ID di correlazione, l'indirizzo IP della parte server e quello della parte client che ha effettuato la richiesta, data e orario dell'invio del messaggio sono solo alcune delle informazioni da considerare di includere.
- **Come strutturare i dati di log**: uno dei vantaggi di poter operare in maniera distribuita è la possibilità di utilizzare diverse tecnologie per i vari servizi: tuttavia il risultante numero di *formati di log* spesso crea l'analisi di queste informazioni molto più complessa. Strutturare i dati in un *formato standard* come può essere il JSON porterà a semplificarne il parsing permettendo di ricercarli tramite un varietà di campi diversi da una postazione centrale.
- **Log centralizzati o distribuiti**: spesso è preferibile optare per un logging centralizzato per prevenire la perdità di dati in caso di problemi dei vari server e

permettendone un'analisi più veloce. Tuttavia va valutata la situazione caso per caso come definito in precedenza.

4 Perchè OpenTelemetry?

OpenTracing (progetto della Clound Native Computing Foundation - CNCF) e OpenCensun (Google Open Source) sono stati i due standard di riferimento nell'ambito di progetti open source per il tracing distribuito fino a che a maggio del 2019 non si sono uniti in un progetto CNCF prendendo il nome di OpenTelemetry.

OpenTelemetry's Mission: to enable effective observability by making high-quality, portable telemetry ubiquitous. [2]

OpenTelemetry si pone come obiettivo di fornire agli sviluppatori uno strumento per la telemetria pronto all'uso e di alta qualità evitando di dover reingegnerizzare tutto internamente con evidenti costi in ambito di tempo e risorse.

L'idea è stata quella di fornire un set, standardizzato e indipendente dai fornitori di software, di SDK, API e strumenti di recupero, trasformazione e invio di dati ad un software back-end di analisi (a sua volta open-source o proprietario).

A tale scopo sono stati fissati 5 punti chiave:

1. La telemetria dovrebbe essere **semplice**, soprattutto per l'utilizzatore finale. Questo significa che OpenTelemetry deve avere un time-to-value veloce, impostare dei default ragionevole e allo stesso tempo permettere personalizzazioni, il tutto affiancato da un'ottima documentazione fornendo agli sviluppatori un'esperienza di livello superiore

2. La telemetria dovrebbe essere **universale**, i protocolli e le convenzioni utilizzate dovrebbero essere unificate tra i vari linguaggi e tipi di segnali (tracing, logging, metrica, ...). Questo significa che OpenTelemetry mira a fornire soluzioni tecniche che funzionano in maniera consistente sia localmente che globalmente.

3. La telemetria dovrebbe essere **indipendente dal produttore**, per decenni soluzioni di monitoraggio proprietarie sono state le scelte più optate per la telemetria. Sfortunatamente la mancanza di uno stadard o di API comuni hanno portato i clienti a rimanere bloccati alle soluzioni proprietarie scelte inizialmente inibendo l'innovazione accoppiando in maniera bloccante la raccolta di dati telemetrici con il loro storage e la loro analisi. OpenTelemetry ha quindi la missione di fornire un unico campo di lavoro per tutti i provider per l'osservabilità dei sistemi, evitare blocchi proprietari di qualsiasi fornitore e interoperare con qualsiasi progetto open source nell'ecosistema della telemetria.

4. La telemetria deve fornire strumenti con **basse inter-dipendenze**, l'utilizzatore finale deve poter scegliere lo strumento che vuole senza dover essere costretto ad aderire all'intero progetto. Per ottenere questo obiettivo l'*architettura software di OpenTelemetry è disaccoppiata dove possibile.* Come corollario questo significa anche che OpenTelemetry non vuole definire un "vincitore" quando si parla di

determinato progetti o tecnologie: dove possibile preferisce fornire all'utente finale la *possibilità di scelta*.

5. La telemetria dovrebbe essere **built-in**, storicamente la telemetria è stata qualcosa che gli svilupattori integravano manualmente o attraverso agenti in post compilazione. OpenTelemetry crede che una telemetria di alta qualità possa essere *integrata* all'interno dell'intero stack software, così come avviene oggi per i commenti.

Mentre la struttura e i dettagli tecnici di OpenTelemetry potranno cambiare nel tempo questi 5 punti rimarranno invariati fino al raggiungimento della Mission guardando a loro per orientarsi nel futuro.

4.1 I principi ingegneristici alla base di OpenTelemetry

Alla base degli sviluppi di questo progetto si possono identificare quattro valori fondamentali dell'ingegneria di sistemi distribuiti:

1. **Compatibilità** : visto il numero di componenti in interesse e piattaforme supportate seguire specifiche e garantire interoperabilità diventa di vitale importanza per l'utilizzatore. OpenTelemetry si sforza di essere conforme agli standard, indipendente dai produttori (vendor-neutral), coerenti e costanti rispetto ai vari linguaggi e componenti.

2. **Stabilità** : poichè molte librerie hanno dipendenze sulle API di OpenTelemetry, la stabilità e retrocompatibilità di quest'ultime diventa essenziale per gli utenti finali. Di conseguenza non vengono introdotti nuovi concept a meno di non avere la certezza che siano necessari ad una buona parte degli utilizzatori di OpenTelemetry.

3. **Resilienza al cambiamento** : OpenTelemetry è progettato per lavorare e continuare a raccogliere dati telemetrici anche quando l'applicazione non si comporta come previsto.

4. **Performace** : l'utente non deve scegliere tra telemetria di qualità e un'applicazione performante. Le alte prestazioni sono un requisito fondamentale e non è accettabile il presentarsi di interferenze inaspettate con l'applicazione host.

5 OpenTelemetry API e SDK

Le specifiche OTel descrivono i requisiti e le aspettative cross-language per tutte le implementazioni attraverso:

- API: definiscono i tipi di dato e le operazioni per la generazione e correlazioni dei dati di tracing, metriche e logging.

- SDK: definiscono i requisiti per le implementazioni specifiche per i vari linguaggi delle API. Sono inoltre qui definiti i concetti di configurazione, data processing e esportazione.
- Dati: definiscono il protocollo OpenTelemetry (OTPL) e le convenzioni semantiche indipendente dai fornitori di software per cui un back-end di telemetria può fornire supporto.

6 Raccolta dati, trasformazione ed esportazione

7 Sistemi back-end di analisi dati OpenTelemetry

8 Un esempio pratico

9 Conclusioni

Riferimenti bibliografici

[1] *Definizione di log*. URL: https://it.wikipedia.org/wiki/Log.

[2] *Open Telemetry GitHub*. URL: https://github.com/open-telemetry/community/blob/main/mission-vision-values.md#otel-mission-vision-and-values.

[3] *Telemetry communications*. URL: https://www.britannica.com/technology/telemetry.