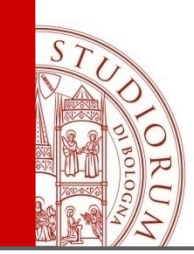
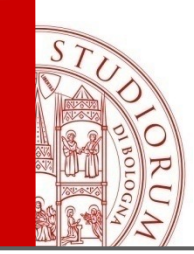


Una infrastruttura per la raccolta di dati di telemetria in ambienti distribuiti: OpenTelemetry

Michele Laddaga
michele.laddaga@studio.unibo.it
May 2022



Cosa vedremo

- Breve introduzione alla **telemetria**
- Analisi delle problematiche di gestione di dati di telemetria in **sistemi distributi**
- Analisi delle caratteristiche dello standard **OpenTelemetry**
- Strumenti e Back-end di analisi
- Esempio pratico in .NET Core
- Conclusioni

Introduzione alla Telemetria

La telemetria è un **processo automatizzato** di comunicazione attraverso il quale vengono effettuate **misurazioni** e vengono collezionati dati remoti

In ambito **software** si tratta di recuperare dati relativi all'utilizzo del prodotto e alle performance di applicazioni e relativi componenti. Di seguito alcuni esempi:

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

Essenziale nell sviluppo software e nell'identificazionee risoluzione dei problemi

Telemetria nei sistemi distribuiti

- Logging:
  - monolitico vs centrale vs distribuito
- Tracing:
  - tracciare le richieste mentre si spostano attraverso varie applicazioni interdipendenti
  - Span, traceID, tag, misurazioni
- Metrics:
  - andamento del sistema nel tempo

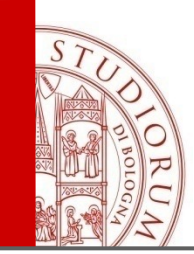
Perchè OpenTelemetry?

OpenTracing (progetto della Clound Native Computing Foundation - CNCF) e **Open-Census** (Google Open Source) standard di riferimento open source per il tracing distribuito fino al 2019 quando si sono uniti formando **OpenTelemetry**.

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

5 punti chiave, la telemetria dovrebbe*:*

- essere **semplice**, soprattutto per l'utilizzatore finale
- essere **universale:** protocolli e convenzioni
- essere **indipendente dal produttore**
- strumenti con **basse inter-dipendenze**, utilizzo solo gli strumenti che mi servono
- essere **built-in**, come avviene per i commenti nei codici di programmazione



Perchè OpenTelemetry?

Principi ingegneristici alla base di OpenTelemetry

- Compatibilità
  - Conforme agli standard, indipendente dai produttori (vendor-neutral), coerenti e costanti rispetto ai vari linguaggie componenti.
- Stabilità e retro-compatibilità
  - poiché molte librerie hanno dipendenze sulle API di OpenTelemetry
- Resilienza al cambiamento
  - continuare a collezionare dati anche quando l'applicazione non si comporta come previsto
- Performace
  - non è accettabile il presentarsi di interferenze inaspettate con l'applicazione host

OpenTelemetry API e SDK

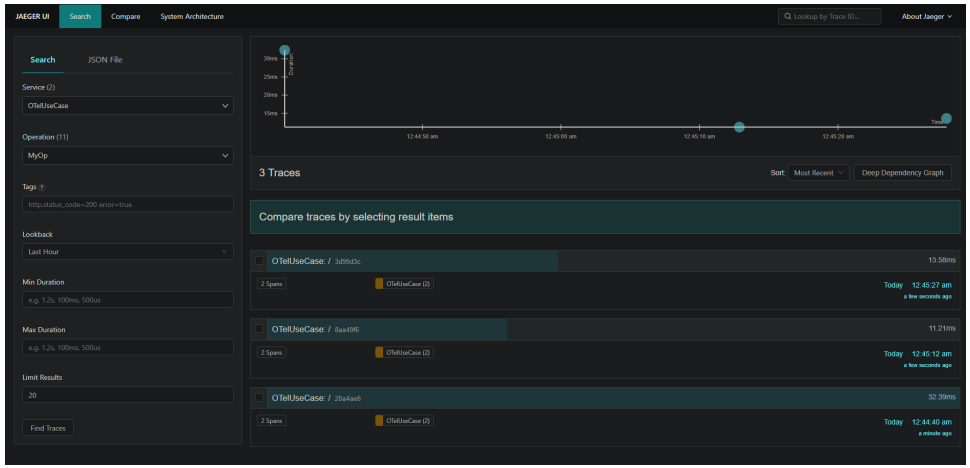
Le specifiche OTel descrivono i **requisiti** e le aspettative **cross-language** per tutte le implementazioni attraverso API, SDK e Dati (formati e standard).

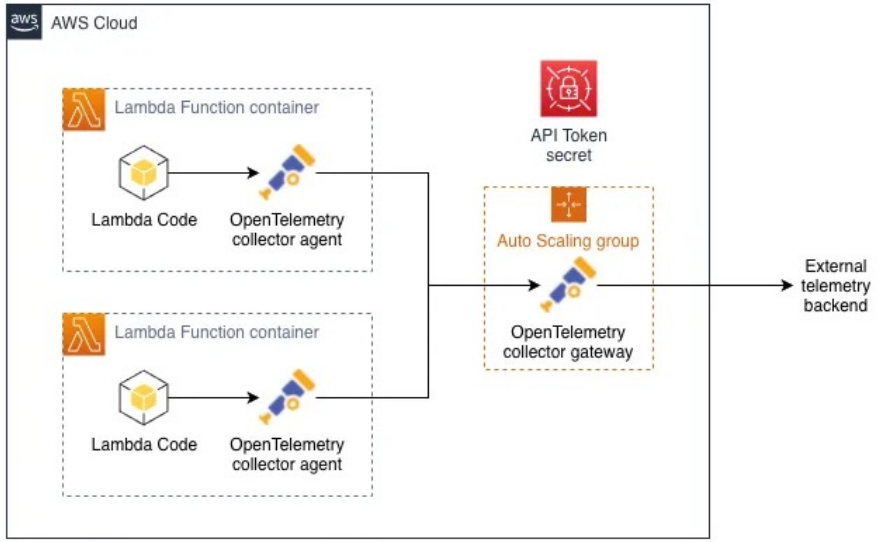
- API
  - Tracing: creare Tracer e quindi generare Span
  - Logging: spesso come integrazione agli strumenti già presenti
  - Metrics: per misurzioni sull'esecuzione dei programmi (es: numero chiamate di un metodo)
  - Baggage: per aggiungere contesto ai tre precedenti
- SDK
  - Definiscono i requisiti per le implementazioni specifiche per i vari linguaggi delle API
  - Permette di campionare, propagare il contesto ed effettuare altre varie procedure ed infine di esportare i dati, tipicamente ad un Collector OpenTelemetry (o di terze parti)
  - OTEL supporta inoltre **l'instrumentazione automatica** con integrazione a vari framework popolari e librerie così come agenti di auto instrumentazione
- Formati standard per i Dati
  - Definiscono il protocollo OpenTelemetry (OTPL) e le convenzioni semantiche indipendente dai fornitori

Raccolta, trasformazione ed esportazione dati

I dati campionati tramite SDK OpenTelemetry:

- possono essere inviati direttamente ai back-end supportati come Jaeger, Zipkin o Prometheus
- possono essere raccolti attraverso un Collector OTEL Agent che successivamente si occuperà di esportarli verso tali back-end o altri Collector di tipo Gateway.





Raccolta, trasformazione ed esportazione dati

Questi **Collector** quindi possono fungere

- sia da agenti locali posizionati sullo stesso host dell'applicazione da monitorare
  - più dispendioso ma permette di recuperare informazioni sulla macchina Host non recuperabili altrimenti)
- sia da servizio centrale di aggregazione per vari nodi applicativi avvalendosi di Collector Gateway
  - soluzione ottima in termini di scalabilità ma che si espone ad un rischio maggiore di perdita di dati

Ogni Collector è composto da 3 parti principali:

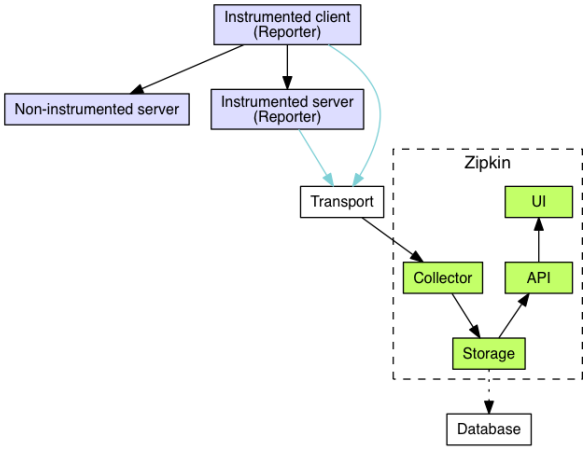
- Ricevitori: dati in ingresso in vari formati e protocolli come OTLP (OpenTelemetry Protocol), o altri formati come quello di Jaeger, Zipkin o altro
- Processatori: per effettuare aggregazioni, filtri e campionamenti
- Esportatori: per trasmettere i dati telemetrici verso uno o più back-end o altri collector

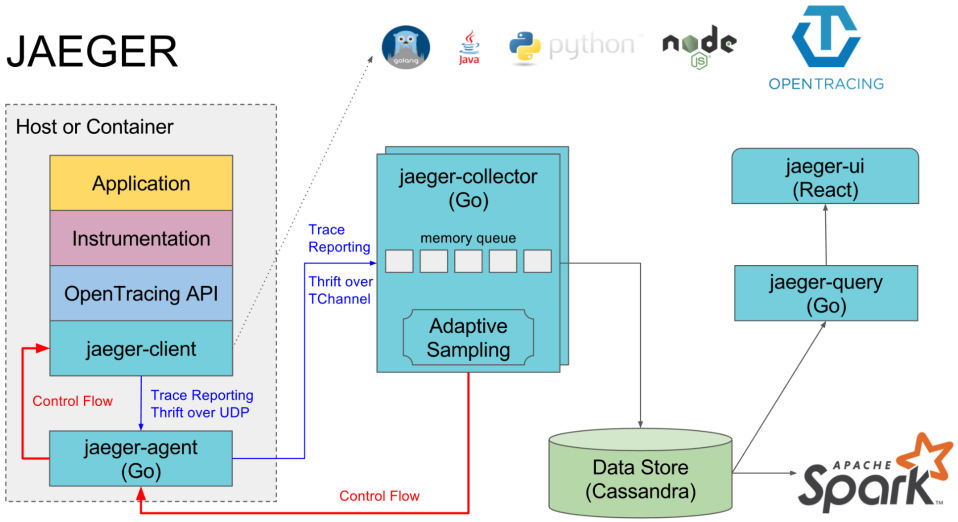
Sistemi back-end di analisi

OpenTelemetry si pone come obiettivi di standardizzare due cose:

- il modo di instrumentare il codice applicativo per generare dati telemetrici
- il formato dei dati telemetrici generati

Una volta creati i dati in formati standard essi possono essere inviati a qualsiasi back-end di analisi che supporti tali formati.





Un esempio pratico

Come esempio è stata creata una WebApp ASP.NET Core che

- mostra come inizializzare OpenTelemetry per logging e tracing
- analizzando anche i rispettivi exporter

I dati di **logging** vengono esportati direttamente sulla console

I dati di **tracing** vengono esportati utilizzando un Otel exporter per vedere un esempio di multi collector simulando un ambiente distribuito. Questi dati possono poi essere visualizzati nel back-end Jaeger.

Per avviare Jaeger e l'applicativo di test è stato creato un file di configurazione docker-compose.yml che imposta la webApp, JaegerUI e due collector.

Un esempio pratico

Per avviare l'esempio posizionarsi nella radice della soluzione OTelUseCase archiviata su GitLab al seguente indirizzo

https://gitlab.com/pika-lab/courses/ds/projects/ds-project-laddaga-1819/

ed eseguire la seguente riga di comando assicurandosi di avere Docker in esecuzione in modalità Linux:

- docker - compose up -d

Jaeger sarà cosi consultabile in localhost alla porta 16686 , l'applicativo alla porta 5000.

Nell'esempio proposto oltre all'esportazione diretta verso Jaeger sono stati impostati due collector in sequenza che ricevono i dati dall'applicativo per poi riversarli su Jaeger.

Come si può notare navigando su JaegerUI per ogni chiamata avremo quindi 3 volte gli Span di analisi.

Conclusioni

La telemetria e in particolare il mondo di OpenTelemetry sono molto più vasti di quanto qui mostrato:

- oltre al supporto e alla presenza di SDK per **molti linguaggi** (Java, Go,Ruby, Javascript, Swift, Erlang, ecc)
- e alla presenza di vari strumenti per **l'instrumentazione automatica**

c'è un'intera comunity che sta lavorando per consolidare e ampliare questo standard portandolo ad essere a tutti gli effetti un progetto **community-driven**.

Con l'incremento del numero di applicativi distribuiti che devono inter-comunicare risulta di primaria importanza tale standard per garantire uno **strumento univoco** per lo sviluppo, l'analisi a runtime, la manutenibilità e la gestione a 360 gradi dei sistemi distribuiti.

GRAZIE PER L'ATTENZIONE

Michele Laddaga
michele.laddaga@studio.unibo.it
May 2022