Manuale Utente

Gruppo MILCTdev – ***Progetto OpenAPM***
milctdev.team@gmail.com

Versione	1.0.0
Redazione	Luca Dal Medico Carlo Munarini
Verifica	Leonardo Nodari
Approvazione	Isacco Maculan
Uso	Esterno
Distribuzione	Kirey Group Prof. Tullio Vardanega Prof. Riccardo Cardin Gruppo MILCTdev

Descrizione
Questo documento fornisce ad un utente una guida all'utilizzo del prodotto OpenAPM

Registro delle modifiche

Versione	Ruolo	Nominativo	Descrizione	Data
1.0.0	Responsabile	Isacco Maculan	Approvazione documento per il rilascio	2018-04-27
0.1.0	Verificatore	Leonardo Nodari	Verifica del documento	2018-04-25
0.0.4	Programmatore	Luca Dal Medico	Stesura appendice A	2018-04-23
0.0.3	Programmatore	Carlo Munarini	Stesura §3	2018-04-17
0.0.2	Programmatore	Luca Dal Medico	Stesura §2	2018-04-12
0.0.1	Programmatore	Carlo Munarini	Stesura §1	2018-04-10
0.0.0	Programmatore	Luca Dal Medico	Inserimento template del documento	2018-04-10

Indice

1 Introduzione

1.1 Scopo del documento

Questo documento è rivolto all'utente utilizzatore di OpenAPM e ha lo scopo di illustrare gli aspetti di base del prodotto e le configurazioni modificabili dall'utente.

1.2 Scopo del prodotto

Lo scopo del *prodotto*$_{\text{G}}$ è realizzare un set di funzioni basate su *Elasticsearch*$_{\text{G}}$ e *Kibana*$_{\text{G}}$ per interpretare i dati raccolti da un *Agent*$_{\text{G}}$. I dati interpretati forniranno a *DevOps*$_{\text{G}}$ statistiche e informazioni utili per comprendere il funzionamento della propria applicazione. In particolare si richiede lo sviluppo di un motore di generazione di *metriche*$_{\text{G}}$ da *trace*$_{\text{G}}$, un motore di generazione di *baseline*$_{\text{G}}$ basato sulle metriche del punto precedente, e un motore di gestione di *critical event*$_{\text{G}}$.

1.3 Informazioni utili

OpenAPM può essere visto come una pipeline di eventi che:

- date le tracce di azioni intraprese su un applicazione;
- queste vengono analizzate, creando così delle *metriche*;
- le quali servono per analizzare il comportamente dell'applicazione, tramite la creazione di *baseline*;
- queste baseline vengono poi utilizzate per controllare che il comportamento dell'applicazione di standard;
- se ciò non fosse, vengono riscontrati degli *alert*;
- successivamente vengono lanciare delle *azioni di rimedio*.

Per questo motivo è stata scelta una struttura del documento che permetta da prima di avere familiarità con le configurazioni degli elementi sopra citati, e successivamente spieghi come svilupparle al meglio.

Al fine di illustrare i concetti con maggior chiarezza, nell'appendice A è presente un glossario con i termini che MILCTdev ritiene necessitino di una definizione. L'identificazione di questi termini viene fatta marcando la prima occorrenza di questi con una $_{\text{G}}$ a pedice e il testo sarà in corsivo.

2 Istruzioni per l'utilizzo

2.1 Requisiti software

Di seguito vengono riportare le versioni minime garantite per il funzionamento del prodotto OpenAPM:

- **Sistema operativo:** Ubuntu 16.04. Il prodotto è supportato anche da altre distribuzioni Linux;
- **Java:** Versione 8 o superiore;
- **Elasticsearch:** Versione 6 o superiore.

Il prodotto mette a disposizione anche la possibilità di inviare delle email, allo scatenarsi di eventi; per tanto nel caso di utilizzo di questa funzionalità, è richiesto un server compatibile con:

- Java Mail Sender;
- Spring Mail.

2.2 Requisiti hardware

Non sono stati individuati requisiti hardware minimi per il funzionamento del prodotto OpenAPM.

2.3 Prerequisiti

Si assume che l'utilizzatore del prodotto OpenAPM possegga delle conoscenze basilari nel campo della programmazione ad oggetti e del linguaggio Java. Un altro prerequisito importante è la familiarità con Elasticsearch e sopratutto con l'inserimento e la modifica di documenti. Una guida sull'argomento può essere trovata al seguente link:
https://www.elastic.co/guide/en/elasticsearch/guide/current/data-in-data-out.html

2.4 Installazione e avvio

Per utilizzare il prodotto OpenAPM sarà sufficiente eseguire queste poche operazioni:

- Scaricare il codice;
- Configurare il file '*application.properties*';
- Eseguire il comando '*./gradlew bootJar*' per ottenere un .jar file eseguibile;
- Copiare il .jar ottenuto (*build/libs/openapm-{VERSIONE}.jar*) dove desiderato;
- Avviare il file lanciando '*java -jar openapm-{VERSIONE}.jar*.

3 Manuale d'uso

3.1 Metriche

3.1.1 Configurazione di una metrica

Di seguito viene riportata un configurazione di esempio per una metrica, con la successiva spiegazione dei vari campi che l'utente può configurare:

```
{
    "name": "http-duration-by-url",
    "cron": "0 * * * * *",
    "duration": 60,
    "traces_index": "stagemonitor-spans-*",
    "metrics_index": "#'openapm-metrics-'yyyy.MM.dd",
    "filters": [
        {
            "operator": "=",
            "operands": [
                "type",
                "'http'"
            ]
        }
    ],
    "aggregation": {
        "operands": [
        "http.url"
        ]
    },
    "calculation": {
        "operator": "average",
        "operands": [
            "duration_ms"
        ]
    }
}
```

La configurazione di cui sopra va inserita nell'indice di configurazione metriche che di default è: *openapm-config-metrics*
I campi configurabili sono quindi:

- **name:** specifica il nome per la metrica generata, rappresenterà il valore del campo 'name' della metrica;
- **cron:** specifica un'espressione cron che determini quando il calcolo della metrica deve essere eseguito;

- **duration:** specifica un valore, in secondi, che determini l'intervallo in cui prelevare le trace per il calcolo la metrica (Es. 60 secondi);
- **traces_index:** specifica l'indice o la tabella da cui prelevare le trace;
- **metrics_index:** specifica l'indice o la tabella in cui salvare le metriche;
- **filters:** specifica una lista di operatori filtro per selezionare le trace valide da utilizzare per il calcolo;
- **aggregation:** specifica un operatore che, dato un unico gruppo di trace, le divida in tanti sottogruppi;
- **calculation:** specifica un operatore che, dato un gruppo di metriche, ne estragga un valore numerico rappresentante il valore finale della metrica per quel gruppo di trace.

3.2 Baselines

3.2.1 Configurazione di una baseline

Di seguito viene riportata un configurazione di esempio per una baseline, con la successiva spiegazione dei vari campi che l'utente può configurare:

```
{
    "name": "http-duration-by-url",
    "metrics_index": "#'openapm-metrics-'yyyy.MM.dd",
    "baselines_index": "openapm-baselines",
    "strategy": "daily",
    "filters": [
        {
        "operator": "=",
        "operands": [
            "name",
            "'http-duration-by-url'"
            ]
        }
    ],
    "aggregation": {
        "operands": [
        "group"
        ]
    },
    "calculation_field": "value"
}
```

La configurazione di cui sopra va inserita nell'indice di configurazione baseline che di default è: *openapm-config-baselines*
I campi configurabili sono quindi:

- **name:** specifica il nome per la baseline generata, rappresenterà il valore del campo 'baseline' della metrica;
- **metrics_index:** specifica l'indice o la tabella da cui prelevare le metriche;
- **baselines_index:** specifica l'indice o la tabella in cui salvare le baseline;
- **strategy:** specifica un identificatore per la strategia da utilzzare;
- **filters:** specifica una lista di operatori filtro per selezionare le metriche da utilizzare per il calcolo;
- **aggregation:** specifica un operatore che, dato un unico gruppo di metriche, le divida in tanti sottogruppi;
- **calculation_field:** specifica il campo con il valore numerico da utilizzare per il calcolo della baseline.

3.3 Alters

3.3.1 Configurazione di un alert

Di seguito viene riportata un configurazione di esempio per un alert, con la successiva spiegazione dei vari campi che l'utente può configurare:

```
{
    "name": "http-duration-warning",
    "evaluate_when": "all_match",
    "conditions": [ /* ... */ ],
    "verification": {
        "on": "immediately"
    },
    "actions": [
        "VK1d4GIBNvwm-_-b4nKr",
        "wa1g4GIBNvwm-_-bK34k",
        "wq1g4GIBNvwm-_-bL35Z"
    ]
}
```

La configurazione di cui sopra va inserita nell'indice di configurazione alter che di default è: *openapm-config-alerts*
I campi configurabili sono quindi:

- **name:** specifica il nome dell'alert, a puro scopo informativo;
- **evaluate_when:** specifica un identificatore che indica la strategia da utilizzare per determinare lo stato globale dell'alert a partire dallo stato delle singole condizioni;
- **conditions:** specifica una lista di singole condizioni che verificano questo alert;
- **verification:** specifica un identificatore per la strategia da utilzzare;
- **actions:** specifica un elenco di ID corrispondenti a delle azioni di rimedio.

3.3.1.1 Il campo ‘conditions’

Espandiamo ora la clausola ‘conditions’ per illustrarne composizione e campi configurabili:

```
{
    "metrics": [
        [
            {
            "operator": "=",
            "operands": [
                "name",
                "'http-duration-by-instance'"
                ]
            },
            {
            "operator": "=",
            "operands": [
                "group",
                "'/'"
                ]
            }
        ],
        [
            {
                "operator": "=",
                "operands": [
                    "name",
                    "'http-duration-by-instance'"
                    ]
            },
            {
                "operator": "=",
                "operands": [
                    "group",
                    "'/vets.html'"
                    ]
            }
        ]
    ],
    "baseline": { /* ... */ },
    "calculation": {
        "operator": "+",
        "operands": [
            "value"
            ]
    },
    "evaluation": {
        "operator": ">",
        "operands": [
```

```
                "value",
                "baseline_value"
                ]
        }
    }
```

I campi configurabili sono quindi:

- **metrics:** specifica un elenco di metriche da considerare per calcolare lo stato della condizione. Qui ogni metrica è rappresentata da un elenco di filtri, in AND logico tra loro, per determinare quali sono valide.
  (Es. nel caso di cui sopra si considerano due metriche, entrambe con nome 'http-duration-by-instance', appartenenti però a due gruppu diversi, rispettivamente '/' e '/vets.html');
- **baseline (opzionale):** specifica una baseline da utilizzare per il calcolo della condizione;
- **calculation:** specifica un operatore che, dato delle metriche, ricavi un valore numerico da utilizzare per il confronto;
- **evaluation:** specifica un operatore che, preso il valore numerico calcolato in precedenza ('value') e opzionalmente un valore della baseline ('baseline_value'), determini se la condizione è vera o falsa.

3.3.1.1.1 Il campo 'baselines'

Espandiamo ora la clausola opzionale 'baselines' per illustrarne composizione e campi configurabili:

```
{
    "index": "openapm-baselines",
    "filters": [
        {
            "operator": "=",
            "operands": [
                "name",
                "'http-duration-by-url'"
                ]
        },
        {
            "operator": "=",
            "operands": [
                "group",
                "'/'"
                ]
        }
    ],
    "strategy": "daily",
    "calculation": {
        "operator": "+",
        "operands": [
            "mean",
            "deviation"
            ]
    }
}
```

I campi configurabili sono quindi:

- **index:** specifica un indice da cui prelevare le baseline;
- **filters:** specifica un elenco di filtri per determinare quali baseline ottenere;
- **strategy:** specifica una strategia per indicare il periodo di interesse delle baseline;
- **calculation:** specifica un operatore per determinare un valore numerico da una baseline precedentemente ottenuta.

3.4 Azioni di rimedio

3.4.1 Configurazione

Di seguito viene riportata un configurazione di esempio per un'azione di rimedio, con la successiva spiegazione dei vari campi che l'utente può configurare:

```
{
    "name": "save-alert",
    "type": "save",
    "data": { /* ... */ }
}
```

La configurazione di cui sopra va inserita nell'indice di configurazione baseline che di default è: *openapm-config-actions*
I campi configurabili sono quindi:

- **name:** specifica un nome per l'azione di rimedio a solo scopo informativo;
- **type:** sperifica la tipologia di azione da intraprendere e può assumere valore:
  - save;
  - email;
  - exec.
- **data:** specifica la struttura di ogni azione.

3.4.1.1 Salvataggio alert

Espandiamo la clausola 'data' per illustrarne composizione e campi configurabili nel caso di azione di tipo 'save':

```
{
    "name": "save-alert",
    "type": "save",
    "data": {
        "index": "#'openapm-alerts-'yyyy.MM.dd",
        "type": "alerts"
    }
}
```

- **type:** specifica il tipo di azione, in questo caso 'save';
- **data.index:** specifica l'indice o la tabella in cui salvare l'allarme;
- **data.type:** specifica la tipologia di documento per l'allarme; alcuni DBMS richiedono che venga dichiarato un tipo per il documento, come nel caso di Elasticsearch.

3.4.1.2 Esecuzione script

Espandiamo la clausola 'data' per illustrarne composizione e campi di configurabili nel caso di azione di tipo 'exec':

```
{
    "name": "exec-hello-world",
    "type": "exec",
    "data": {
        "script": "#!/usr/bin/env bash\n\ncurl \"http://
127.0.0.1:8081/?name=$ALERT_NAME&when=$ALERT_WHEN\";"
    }
}
```

- **type:** specifica il tipo di azione, in questo caso 'exec';
- **data.sript:** specifica il contenuto dello script da eseguire, contenente obbligatoriamente lo shebang. Sono disponibili due variabili d'ambiente:
  - **ALERT_NAME:** con il nome dell'alert scatenato;
  - **ALERT_WHEN:** con l'unix timestamp dell'ora in cui è stato scatenato.

3.4.1.3 Invio di un email

Espandiamo la clausola 'data' per illustrarne composizione e campi di configurabili nel caso di azione di tipo 'email':

```
{
    "name": "send-email",
    "type": "email",
    "data": {
        "to": "alert@example.com",
        "subject": "Alert {{ name }} has been triggered",
        "body": "Alert <b>{{ name }}</b> has been triggered at
<i>{{ when | date('yyyy.MM.dd HH:mm:ss') }}</i>"
    }
}
```

- **type:** specifica il tipo di azione, in questo caso 'email';
- **data.to:** specifica il destinatario dell'email;
- **data.subject:** specifica l'oggetto dell'email, elemento processato da $jTwig_{G}$;
- **data.body:** specifica il corpo dell'email, elemento processato da jTwig.

Il template mette a disposizione anche alcune variabili come:

- **name:** Nome dell'alert scatenato;

- **when:** Orario dell'allarme (java.lang.Date).

3.5 Operatori

Il prodotto OpenAPM utilizza un interfaccia *Operator* implementata da una classe astratta *AbstractOperator*. Gli operatori estendono quindi la classe astratta di cui sopra e lavorano su oggetti, così da definire ad esempio, se all'interno di una metrica è presente il campo 'value'. Gli operatori implementati sono:

- **AttributeOperator:** recupera un attributo di un elemento;
- **AverageOperator:** calcola la media di gruppi di elementi;
- **EqualsOperator:** controlla se i valori di due elementi sono uguali;
- **GreaterOperator:** controlla se i valori di un primo elemento sono maggiori di quelli del secondo;
- **LessOperator:** controlla se i valori di un secondo elemento sono maggiori di quelli del primo;
- **NullOperator:** non raggruppa, ne calcola;
- **SumOperator:** calcola la somma di gruppi di elementi;

3.5.1 Templator di operandi

Gli operatori eseguono controlli e operazioni su elementi e per agevolarne il compito, è stato introdotta la classe *OperandTemplator* che:

- Dato in input un oggetto ed un identificatore;
- Controlla che il secondo sia effettivamente un identificatore, quindi se e finisce con un apostrofo (Es. 'valore');
- Elimina gli apostrofi;
- Ritorna una stringa contenente il valore richiesto, se presente nell'oggetto;

Prendendo quindi la configurazione di una metrica:

```
{
    "name": "http-duration-by-url",
    /* ... */
    "filters": [
        {
            "operator": "=",
            "operands": [
                "type",
                "'http'"
            ]
        }
    ],
    "aggregation": { /* ... */ },
    "calculation": { /* ... */ }
}
```

Possiamo notare come *"type"* sia statico a differanza di *"'http'"*, che invece è dinamico.

3.6 Strategie

Nel processare delle baseline, vengono messe a disposizione tre diverse strategie di calcolo:

- **Daily:** la quale, data un ora di interesse, recupera elementi di n giorni prima, con n configurabile;
- **Weekly:** la quale, data un ora ed un giorno di interesse, recupera elementi di n settimane prima (Es. dato Lunedì alle 15, recupererà n baseline dei Lunedì delle settimane precedenti);
- **Monthly:** la quale, data un ora ed un giorno di interesse, recupera elementi di n mesi prima (Es. dato il 12 Aprile alle 15, recupererà n baseline di ogni giorno 12 dei mesi precedenti).

Il numero di elementi da recuperare durante il calcolo è configurabile dall'utente, tramite la modifica di *application.properties*. Di default il valore è settato a 4.

3.7 Valutatori

Per permettere di controllare un alert, si è deciso di implementare valutatori che, controllando un set di condizioni, determinano lo stato dell'alert stesso. I valutatori esistenti mettono a disposizione il metodo *boolean evaluate(List<Condition>)* e sono:

- **AllMatchEvaluator:** dato un set di istruzioni non nullo, ritorna vero solo se tutte le condizioni sono soddisfatte;
- **AnyMatchEvaluator:** dato un set di istruzioni non nullo, ritorna vero solo se almeno una condizioni è soddisfatta.

3.8 Verificatori

Una volta determinato lo stato di un alert (Attivo o Inattivo), le decisioni sulle tempistiche di lancio delle azioni di rimedio spettano ai verificatori i quali, in base alla strategia, decideranno se e quando lanciare le azioni di rimedio. I verificatori implementati sono:

- **AfterVerifier:** strategia per la quale, il lancio di azioni, avviene all'ennesimo cambio di stato in Attivo di un alert;
- **ImmediateVerifier:** strategia per la quale, il lancio di azioni, avviene non appena un alert cambia stato in Attivo;
- **TerminationVerifier:** strategia per la quale, il lancio di azioni, avviene non appena un alert cambia stato in Inattivo.

3.9 Templator di indice

Per facilitare la navigazione tra gli indici, è stato creata la classe *IndexTemplator*. Esso il suo funzionamento:

- Accetta in input una stringa oppure una stringa ed una data;

- Controlla se tale stringa è un template, quindi se inizia con '#';
- Elimina il '#' dalla stringa;
- Ritorna un oggetto *SimpleDateFormat* dato il pattern nella stringa e l'orario corrente;

Vediamo un esempio prendendo la configurazione di una metrica:

```
{
    "name": "http-duration-by-url",
    "cron": "0 * * * * *",
    "duration": 60,
    "traces_index": "stagemonitor-spans-*",
    "metrics_index": "#'openapm-metrics-'yyyy.MM.dd",
    "filters": [ /* ... */ ],
    "aggregation": { /* ... */ },
    "calculation": { /* ... */ }
}
```

In questo caso *metrics_index* è un template valido e quindi:

- Ponendo che sia il 7 Maggio 2018 alle ore 17:00;
- IndexTemplator controlla che metrics_index sia template;
- Elimina il '#' dalla stringa;
- Ritorna un oggetto *SimpleDateFormat* con pattern *yyyy.MM.dd*;
- La metrica viene salvata nell'indice *openapm-metrics-2018.05.07*.

4 Risoluzione dei problemi

4.1 Segnalazione di bug

OpenAPM potrebbe contenere dei piccoli bug, oppure potrebbe essere desiderabile apportare qualche modifica al prodotto. MILCTdev si rende quindi disponibile alla ricezione di email contenenti segnalazioni o richieste, all'inidirizzo: milctdev.team@gmail.com
Vengono quindi fornite delle istruzioni per la creazione della mail:

- **Oggetto:** L'oggetto della comunicazione deve necessariamente avere come prefisso "*[OpenAPM]*" seguito dalla dichiarazione del tipo di problema o modifica (Es. "*[OpenAPM] Segnalazione errore*");
- **Corpo:** Nel corpo del messaggio chiediamo vengano inseriti:
  - Sistema operativo utilizzato, completo di versione;
  - Configurazione utilizzata;
  - Descrizione del problema ed eventuale messaggio di errore;
  - Eventuali screenshot.

A Glossario

A

Agent

In informatica un Agent è un programma che agisce per un utente esterno o per un altro programma con una relazione di "agency". In parole povere, un Agent si può chiamare anche Bot. Uno degli esempi più semplici di Agent è l'assistente personale di Apple, Siri.

D

DevOps

In informatica, DevOps rappresenta una metodologia di sviluppo software. DevOps prevede che il team di sviluppo software lavori anche alle altre operazioni non inerenti al solo sviluppo, come ad esempio logistica e ricerca.

E

ElasticSearch

ElasticSearch è uno dei server di ricerca più utilizzati nel mondo. Si basa su Lucene, supporta ricerca Full Text, supporta architetture distribuite e le sue funzionalità sono esposte tramite interfaccia RESTful. Le informazioni sono gestite come documenti JSON.

J

jTwig

jTwig è un motore di creazione di template per Java. Questo framework è configurabile, estensibile e potente, grazie all'utilizzo di un ottima sintassi e la configurabilità in moltissimi suoi aspetti.

K

Kibana

Kibana è un plugin open source di ElasticSearch per la visualizzazione di dati mediante grafici.

M

Metrica

Una metrica è un indicatore utile a misurare andamenti di interesse. Viene utilizzata per controllare, ad esempio, costi di produzione, qualità del prodotto. Esistono metriche standard ma possono anche essere create ad-hoc in base ai parametri che si vogliono misurare.

P

Prodotto

Per prodotto si intende un bene materiale o immateriale, risultato di un'attività di progetto.

T

Trace

Una trace, in ambito di monitoring, è un insieme di informazioni relative all'esecuzione di una singola operazione o richiesta da parte di un'applicazione. Possiamo ritrovare lo stesso concetto, anche se leggermente ridotto di significato, nel termine log.