【MicroProfile開発ガイド】
MicroProfile Metrics

2018/12

日本アイ・ビー・エム システムズ・エンジニアリング株式会社

Disclaimer

- この資料は日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム　システムズ・エンジニアリング株式会社の正式なレビューを受けておりません。
- 当資料は、資料内で説明されている製品の仕様を保証するものではありません。
- 資料の内容には正確を期するよう注意しておりますが、この資料の内容は2018年9月現在の情報であり、製品の新しいリリース、PTFなどによって動作、仕様が変わる可能性があるのでご注意下さい。
- 今後国内で提供されるリリース情報は、対応する発表レターなどでご確認ください。
- IBM、IBMロゴおよびibm.comは、世界の多くの国で登録されたInternational Business Machines Corporationの商標です。他の製品名およびサービス名等は、それぞれIBMまたは各社の商標である場合があります。現時点でのIBMの商標リストについては、www.ibm.com/legal/copytrade.shtmlをご覧ください。
- 当資料をコピー等で複製することは、日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム　システムズ・エンジニアリング株式会社の承諾なしではできません。
- 当資料に記載された製品名または会社名はそれぞれの各社の商標または登録商標です。
- JavaおよびすべてのJava関連の商標およびロゴはOracleやその関連会社の米国およびその他の国における商標または登録商標です。
- Microsoft, WindowsおよびWindowsロゴは、Microsoft Corporationの米国およびその他の国における商標です。
- Linuxは、Linus Torvaldsの米国およびその他の国における登録商標です。
- UNIXはThe Open Groupの米国およびその他の国における登録商標です。

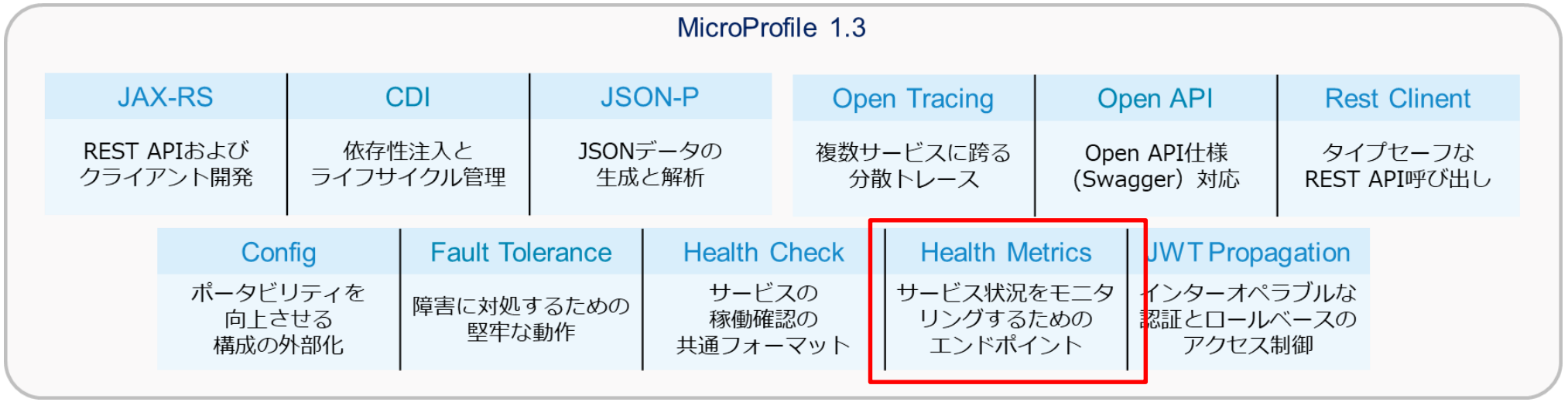
目次

MicroProfile Metrics概要

MicroProfile Metricsとは

MicroProfile Metricsはマイクロサービス環境において各サービス（アプリケーション）の実行中に計測可能な数値情報（メトリック）を取得するための機能を提供します。

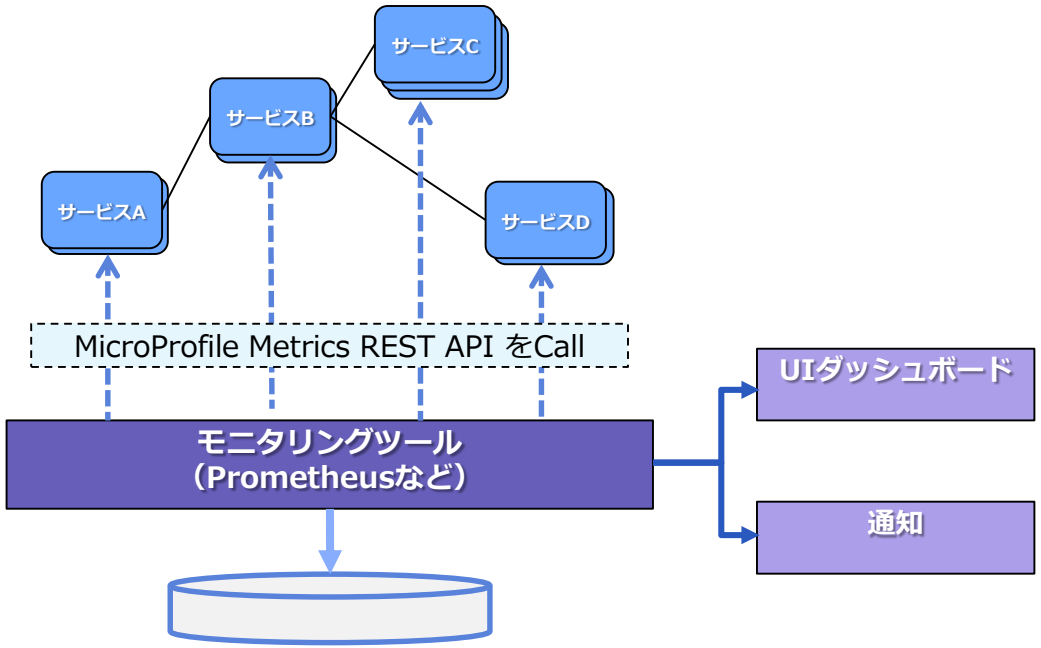
mpMetrics-1.1 フィーチャーを使用することにより、MicroProfile Metricsが提供するAPIを使用してアプリケーションをモニタリングできます。 mpMetrics-1.1フィーチャーは、MicroProfile Metrics 1.1 仕様に準拠するREST APIのインターフェースを提供します。 アプリケーション開発者は、WAS Libertyで提供されるメトリックと共に、独自のカスタム・メトリックを MicroProfile Metrics APIを使用して追加できます。



利用シナリオ

マイクロサービス環境では、分散環境で稼働する各サービス上で計測される各種メトリックをモニタリングツール（Prometheusなど）にて集約し、アプリケーション全体としての稼働状況の把握や分析、およびルールや閾値に基づく障害予兆の通知などを実現することが求められます。

MicroProfile Metricsを利用することにより、モニタリングツールが各サービスのメトリックを取得するための標準的なAPIを容易に提供することが可能になります。



収集対象メトリクス情報の例
- CPU、メモリー、ディスク仕様率
- アクセス数、エラー発生率、応答速度

ダッシュボード(Grafanaなど)

フィーチャーの有効化

WAS Liberty上でMicroProfile Metricsの機能を有効化するためには、該当サーバーのserver.xmlにmpMetrics-1.1フィーチャーを設定します。
当該フィーチャーが含まれるMicroProfile-1.3を設定することも可能です。

```
<server description="new server">

    <!-- Enable features -->
    <featureManager>
         <feature>mpMetrics-1.1</feature>
    </featureManager>
</server>
```

※mpMetrics-1.1フィーチャーが提供するMicroProfile Metrics REST APIの利用にはフィーチャーの有効化に加え、セキュリティ設定等が必要ですが、それについては後述します。

mpMetrics-1.1フィーチャーを有効化することにより、以下のフィーチャーも暗黙的ロードにより自動的に有効化されます。

- cdi-1.2
- distributedMap-1.0
- jndi-1.0
- json-1.0
- mpConfig-1.2
- servlet-3.0
- servlet-3.1
- ssl-1.0

MicroProfile Metricsの使用方法

MicroProfile Metrics REST API

MicroProfile Metrics REST APIは各サービスからメトリックを取得するためのREST APIであり、そのAPI仕様はMicroProfile Metrics仕様で標準化されています。

- Metrics REST APIのセキュリティ

  WAS Liberty上でMicroProfile Metrics REST APIを使用するためには、前述のmpMetrics-1.1フィーチャー有効化に加え、APIを保護するためのセキュリティ設定を推奨します。
  以下にserver.xmlのquickstartSecurity要素を使用してLibertyの最も単純なセキュリティ設定を行う例を示します。

```
<server description="new server">
    <!-- Enable features -->
    <featureManager>
         <feature>mpMetrics-1.1</feature>
    </featureManager>
    <!-- Enable security -->
    <quickStartSecurity userName="<user>" userPassword="<password>"/>
    <keyStore id="defaultKeyStore" location="<file>" type="<type>" password="<keystore_password>" />
    . . . .
```

  上記の設定によりMicroProfile Metrics REST APIはBasic認証で保護されるようになり、アクセス時にはquickStartSecurity要素に指定したuserNameとuserPasswordが必要となります。

MicroProfile Metrics REST API

- Metrics REST APIの認証を無効化する方法

  前述の通り、MicroProfile Metrics REST APIへのアクセスを保護するためのセキュリティ設定を推奨しますが、開発環境等で認証の必要がない場合には、認証を無効化することができます。
  以下にserver.xmlのmpMetrics要素を使用して認証を無効化する例を示します。

```
<server description="new server">
    <!-- Enable features -->
    <featureManager>
         <feature>mpMetrics-1.1</feature>
    </featureManager>
    <!-- Disable MicroProfile Metrics Authentication -->
    <mpMetrics authentication="false"/>
    . . . .
```

  上記の設定により、MicroProfile Metrics REST APIへのアクセスで、認証が不要になります。

18.0.0.3+

MicroProfile Metrics REST API

- Libertyコンポーネント・メトリック（vendorスコープ）を出力させる方法

  monitor-1.0とmpMetrics-1.1の両方のフィーチャーが使用可能になっている場合、以下のLibertyコンポーネントからのメトリックが /metrics/vendor に出力されます。

  – Webアプリケーション・メトリック
  – スレッド・プール・メトリック
  – セッション管理・メトリック
  – 接続プール・メトリック

  各メトリック出力の詳細については、後述の「vendorスコープで規定されたメトリック」で解説します。

  この場合のserver.xmlの例を示します。

```
<server description="new server">
    <!-- Enable features -->
    <featureManager>
         <feature>monitor-1.0</feature>
         <feature>mpMetrics-1.1</feature>
    </featureManager>
. . . .
```

MicroProfile Metrics REST API

- 日本語環境でMicroProfile Metric REST APIを使用する場合の設定

  WAS Libertyのデフォルト設定では日本語環境での実行時にMetrics REST APIのレスポンスが途切れて返されるという事象が発生します。それを回避するために、server.xmlのwebContainer要素のsetContentLengthOnCloseプロパティにfalseを設定します。

```
<server description="new server">
    <!-- Enable features -->
    <featureManager>
         <feature>mpMetrics-1.1</feature>
    </featureManager>
    <!-- Enable security -->
    . . . .
    <!-- Avoid metrics api response truncation -->
    <webContainer setContentLengthOnClose="false" />
    . . . .
```

18.0.0.4+

Liberty 18.0.0.4でこの問題は修正されましたので、18.0.0.4以降では、<webContainer setContentLengthOnClose="false" />の設定は不要です。この問題の修正は、Open Liberty のIssue #5164に含まれ、WebSphere Liberty 18.0.0.4にも含まれます。

GitHub OpenLiberty/open-liberty "/metrics output got truncated on Japanese locale #5164"
https://github.com/OpenLiberty/open-liberty/issues/5164

MicroProfile Metrics REST API

- MicroProfile Metrics REST APIのAPI仕様
  - –RESTエンドポイント

    MicroProfile Metrics REST APIでは以下の２つの種類のエンドポイントを提供します。

    - メトリック取得用

エンドポイント	メソッド	フォーマット	説明
/metrics	GET	JSON、Prometheus	すべてのスコープに含まれる登録済みメトリックを返します。
/metrics/<scope>	GET	JSON、Prometheus	それぞれのスコープの登録済みメトリックを返します。
/metrics/<scope>/<metric name>	GET	JSON、Prometheus	それぞれのスコープのメトリック名と一致するメトリックを返します。

    - メトリック・メタデータ取得用

エンドポイント	メソッド	フォーマット	説明
/metrics	OPTIONS	JSON	すべてのスコープに含まれる登録済みメトリック・メタデータを返します。
/metrics/<scope>	OPTIONS	JSON	それぞれのスコープの登録済みメトリック・メタデータを返します。
/metrics/<scope>/<metric name>	OPTIONS	JSON	それぞれのスコープのメトリック名と一致するメトリック・メタデータを返します。

    メトリック・メタデータは、メトリックについての情報を記述および要約するデータの集合です。

MicroProfile Metrics REST API

- 出力フォーマット

  メトリック取得用エンドポイントでは以下の複数の出力フォーマットをサポートします。
  出力フォーマットはリクエストのacceptヘッダーの値により選択されますが、acceptヘッダーが無い場合のデフォルトはPrometheusテキスト・フォーマットになります。
  メトリック・メタデータ取得用エンドポイントでサポートされるのはJSONフォーマットのみになります。

フォーマット	説明
Prometheus テキスト・フォーマット	Prometheusモニタリング・ツールで読み込み可能なメトリック表記フォーマット。 リクエストのaccept ヘッダーが text/plainの要求に対して返される。
JSONフォーマット	JSON形式のメトリック表記フォーマット。 リクエストのaccept ヘッダーが application/jsonの要求に対して返される。

MicroProfile Metrics REST API

- 出力フォーマット：Prometheusテキスト・フォーマット

  Prometheus 0.0.4公開フォーマットに基づいた出力フォーマットです。このフォーマットではメトリック名やタイプ等のメトリック・メタデータが各メトリックに対して記述されます。

- 出力例

```
# TYPE base:classloader_total_loaded_class_count counter
# HELP base:classloader_total_loaded_class_count Java 仮想マシンが実行を開始した以降にロードされたクラスの・・・
base:classloader_total_loaded_class_count 9413
# TYPE base:cpu_system_load_average gauge
# HELP base:cpu_system_load_average 直前の 1 分間におけるシステム負荷の平均を表示します。・・・
base:cpu_system_load_average 1.0
# TYPE base:thread_count counter
# HELP base:thread_count デーモン・スレッドと非デーモン・スレッドの両方を含めて、現在のライブ・スレッドの数・・・
base:thread_count 44
・・・
```

メトリックのフォーマット：
TYPE {スコープ}:{メトリック名} {タイプ} {タグ}
HELP {スコープ}:{メトリック名} {説明文}
{スコープ}:{メトリック名} {値}

※baseスコープの項目における#Help行の説明文言については、WAS Libertyの言語設定に応じてデフォルトで翻訳されます。

※Prometheusテキスト・フォーマットについての詳細は以下を参照ください。
https://prometheus.io/docs/instrumenting/exposition_formats/#text-format-details

MicroProfile Metrics REST API

- 出力フォーマット：JSONフォーマット
  - メトリック取得用エンドポイント（GET）
    各メトリックはスコープごとのサブツリーでまとめられ、メトリックのタイプに応じて値 (ゲージおよびカウンター) またはサブツリー(メーター、ヒストグラム、およびタイマー) のいずれかによって出力されます。
  - メトリック・メタデータ取得用（OPTIONS）
    各メトリック・メタデータはメトリックごとのサブツリーで出力されます。

※メトリックのスコープおよびタイプについては次頁を参照のこと

- 出力例

（メトリック取得用）

```
{
  "application": {
    "hitCount": 45
  },
  "base": {
     "thread.count" : 33,
     "thread.max.count" : 47,
      ・・・
  },
  "vendor": {...}
}
```

applicationスコープのメトリック

baseスコープのメトリック

vendorスコープのメトリック

（メトリック・メタデータ取得用）

```
{
  "hitCount": {
    "unit": "none",
    "type": "counter",
    "description": "The Hit Counts",
    "displayName": "Hit Counts",
     ・・・・
  },
   ・・・
}
```

hitCountメトリックのメタデータ

MicroProfile Metrics REST API

- メトリックのスコープ

スコープ	説明
base	MicroProfile Metricsで規定された標準メトリック項目。デフォルトで取得可能。
application	アプリケーションがMicroProfile MetricsのJava APIやアノテーションを使用して追加したメトリック項目。
vendor	標準化されていないベンダー固有のメトリック項目。WAS LibertyではV18.0.0.3以降で利用可能。

- メトリックのタイプ

タイプ	説明
カウンター（counter）	インクリメンタルに増減する数値によるメトリック（例：受信リクエスト数）
ゲージ（gauge）	ある対象物の値を測定して得られるメトリック (例：CPU使用率).
メーター（meter）	ある値に対する単位時間あたりの数値（スループット）および１,５,１５分間の指数加重移動平均を算出したメトリック
タイマー（timer）	ある経過時間の値とその集合から得られる各種統計値（平均、最大、最小、スループット等）を算出したメトリック
ヒストグラム（histogram）	ある値の集合に対する度数分布を算出したメトリック

メトリックのタイプごとの出力フォーマットの違いについては以下を参照ください。
https://www.ibm.com/support/knowledgecenter/ja/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/rwlp_mp_metrics_rest_api.html

MicroProfile Metrics REST API

- baseスコープで規定された標準メトリック（WAS Libertyで取得可能な項目）
  これらの標準メトリックはMicroProfile Metrics REST APIからデフォルトで取得可能です。

メトリック	説明	タイプ
memory_committed_heap_bytes	Java 仮想マシンで使用するためにコミットされているメモリー量 (バイト) を表示します。	ゲージ
memory_max_heap_bytes	Java 仮想マシンで使用可能な最大メモリー量 (バイト) を表示します。	ゲージ
memory_used_heap_bytes	Java 仮想マシンで使用中のメモリー量 (バイト) を表示します。	ゲージ
gc_copy_count	発生したガベージコレクション（gc_copy）の総数を表示します。	カウンター
gc_copy_time_seconds	概算累積のガベージコレクション（gc_copy）経過時間 (ミリ秒) を表示します。	ゲージ
gc_mark_sweep_compact_count	発生したガベージコレクション（gc_mark_sweep_compact）の総数を表示します。	カウンター
gc_mark_sweep_compact_time_seconds	発生したガベージコレクション（gc_mark_sweep_compact）経過時間 (ミリ秒) の累積を表示します。	ゲージ
jvm_uptime_seconds	Java 仮想マシンの開始時刻 (ミリ秒) を表示します。	ゲージ
thread_count	デーモン・スレッドと非デーモン・スレッドの両方を含む現在のライブ・スレッドの数を表示します。	カウンター
thread_daemon_count	現在のデーモン・スレッドの数を表示します。	カウンター
thread_max_count	デーモン・スレッドと非デーモン・スレッドの両方を含むJava 仮想マシン起動後のスレッドの最大数を表示します。	カウンター

MicroProfile Metrics REST API

- baseスコープで規定された標準メトリック（WAS Libertyで取得可能な項目）（続き）

メトリック	説明	タイプ
classloader_total_loaded_class_count	Java 仮想マシンが実行を開始した以降にロードされたクラスの総数を表示します。	カウンター
classloader_current_loaded_class_count	Java 仮想マシンに現在ロードされているクラスの数を表示します。	カウンター
classloader_total_unloaded_class_count	Java 仮想マシンが実行を開始した以降にアンロードされたクラスの総数を表示します。	カウンター
cpu_system_load_average	直前の 1 分間におけるシステム負荷の平均を表示します。システム負荷の平均は、使用可能なプロセッサーのキューに入れられている実行可能なエンティティーの数と、使用可能なプロセッサーで実行されている実行可能なエンティティーの数の合計の一定期間の平均値です。負荷平均が使用できない一部プラットフォームでは、負の値が表示されます。	ゲージ
cpu_available_processors	Java 仮想マシンが使用可能なプロセッサーの数を表示します。	ゲージ
cpu_process_cpu_load_percent	Java 仮想マシンによる現在のCPU使用率を表示します。	ゲージ

MicroProfile Metrics REST API

- vendorスコープで規定されたメトリック（WAS Libertyで取得可能な項目）
  これらのメトリックはMicroProfile Metrics REST APIからデフォルトで取得可能です。

  – Webアプリケーション

メトリック	説明	タイプ
servlet_<アプリ名>_<サーブレット名>_request_total	サーバーの開始以降の、このサーブレットへのアクセス数を表示します。	カウンター
servlet_<アプリ名>_<サーブレット名>_response_time_total_seconds	サーバーの開始以降の、このサーブレットの応答時間の合計（※）を表示します。	ゲージ

（※）単位については、Prometheusフォーマットの場合は秒、JSONフォーマットの場合はナノ秒となり、メトリック名にも"_seconds"は付きません。

  – スレッド・プール

メトリック	説明	タイプ
threadpool_<スレッド・プール名>_active_threads	タスクを実行しているスレッドの数を表示します。	ゲージ
threadpool_<スレッド・プール名>_size	スレッド・プールのサイズを表示します。	ゲージ

MicroProfile Metrics REST API

- vendorスコープで規定されたメトリック（WAS Libertyで取得可能な項目）（続き）
  - セッション管理

メトリック	説明	タイプ
session_<仮想ホスト名>_<コンテキスト・ルート>_create_total	このメトリックが有効になって以降にログインしたセッションの数を表示します。	カウンター
session_<仮想ホスト名>_<コンテキスト・ルート>_live_sessions	現在ログインしているユーザーの数を表示します。	ゲージ
session_<仮想ホスト名>_<コンテキスト・ルート>_active_sessions	同時にアクティブなセッションの数を表示します。(注意：あるセッションがアクティブであるとは、製品がそのユーザー・セッションを使用する要求を現在処理している場合をいいます。)	ゲージ
session_<仮想ホスト名>_<コンテキスト・ルート>_invalidated_total	このメトリックが有効になって以降にログアウトしたセッションの数を表示します。	カウンター
session_<仮想ホスト名>_<コンテキスト・ルート>_invalidatedby_timeout_total	このメトリックが有効になって以降にタイムアウトによってログアウトしたセッションの数を表示します。	カウンター

MicroProfile Metrics REST API

- vendorスコープで規定されたメトリック（WAS Libertyで取得可能な項目）（続き）
  - 接続プール

メトリック	説明	タイプ
connectionpool_<接続プール名>_create_total	プールの作成以降に作成された管理接続の総数を表示します。	カウンター
connectionpool_<接続プール名>_destroy_total	プールの作成以降に破棄された管理接続の総数を表示します。	カウンター
connectionpool_<接続プール名>_managed_connections	空きプール、共有プール、および非共有プール内の管理接続の数を表示します。	ゲージ
connectionpool_<接続プール名>_connection_handles	使用中の接続の数を表示します。この数には、単一の管理接続から共有される複数の接続も含まれる場合があります。	ゲージ
connectionpool_<接続プール名>_free_connections	空きプール内の管理接続の数を表示します。	ゲージ

MicroProfile Metrics REST API

- ブラウザから/metrics/vendorへのアクセスによる出力例 (Prometheusテキスト・フォーマット)

```
# TYPE vendor:servlet_test_rest_com_nogu_db_access_servlet_response_time_total_seconds gauge
# HELP vendor:servlet_test_rest_com_nogu_db_access_servlet_response_time_total_seconds サーバーの開始以降の、このサーブレットの応答時間の合計。
vendor:servlet_test_rest_com_nogu_db_access_servlet_response_time_total_seconds 2.09212502
# TYPE vendor:session_default_host_metrics_invalidated_total counter
# HELP vendor:session_default_host_metrics_invalidated_total このメトリックが有効になって以降にログアウトしたセッションの数。
vendor:session_default_host_metrics_invalidated_total 0
# TYPE vendor:connectionpool_jdbc_db2_destroy_total counter
# HELP vendor:connectionpool_jdbc_db2_destroy_total プールの作成以降に破棄された管理接続の総数。
vendor:connectionpool_jdbc_db2_destroy_total 0
# TYPE vendor:session_default_host_metrics_live_sessions gauge
# HELP vendor:session_default_host_metrics_live_sessions 現在ログインしているユーザーの数。
vendor:session_default_host_metrics_live_sessions 1
# TYPE vendor:connectionpool_jdbc_db2_connection_handles gauge
# HELP vendor:connectionpool_jdbc_db2_connection_handles 使用中の接続の数。この数には、単一の管理接続から共有される複数の接続も含まれる場合
があります。
vendor:connectionpool_jdbc_db2_connection_handles 0
# TYPE vendor:servlet_test_rest_com_nogu_db_access_servlet_request_total counter
# HELP vendor:servlet_test_rest_com_nogu_db_access_servlet_request_total サーバーの開始以降の、このサーブレットへの訪問数。
vendor:servlet_test_rest_com_nogu_db_access_servlet_request_total 2
# TYPE vendor:session_default_host_metrics_active_sessions gauge
# HELP vendor:session_default_host_metrics_active_sessions 同時にアクティブなセッションの数。(あるセッションがアクティブであるとは、製品がその
ユーザー・セッションを使用する要求を現在処理している場合をいいます。)
vendor:session_default_host_metrics_active_sessions 1
# TYPE vendor:connectionpool_jdbc_db2_create_total counter
# HELP vendor:connectionpool_jdbc_db2_create_total プールの作成以降に作成された管理接続の総数。
vendor:connectionpool_jdbc_db2_create_total 1
・・・
```

※vendorスコープの項目における#HELP行の説明文言については、WAS Libertyの言語設定に応じてデフォルトで翻訳されます。

アプリケーションによるメトリックの追加

MicroProfile Metricsは開発者がアプリケーションに独自のメトリックを追加するためのAPIを提供します。開発者が追加したメトリックの値は標準メトリックと同様、MicroProfile Metrics REST APIのメトリック取得用エンドポイントで取得できます。

アプリケーションに独自メトリックを追加するには、メトリックを作成してアプリケーション用のメトリック・レジストリーに登録する必要があります。メトリック・レジストリーへの登録により、WAS Libertyは独自メトリックを認識し、MicroProfile Metrics REST APIで追加した独自メトリックが出力されるようになります。

独自メトリックの作成と登録は、以下の方法で行うことが可能です。

- MetricRegistryクラスを直接使用
  - アプリケーション・コードが明示的にメトリックを作成してメトリック・レジストリーに登録します。
- CDI を使用してメトリックを注入
  - メトリックは CDI によって暗黙的に作成され、メトリック・レジストリーに登録されます。
- メトリック・アノテーションを使用
  - メトリックは メトリック・アノテーションによって暗黙的に作成され、メトリック・レジストリーに登録されます。

MetricRegistryクラスによるメトリックの追加

メトリック・レジストリーは、全てのメトリックの値と各メトリックのメタデータを保管します。アプリケーションはMetricRegistryクラスが提供しているメソッドを使用して、独自メトリックおよびメタデータを登録および取得できます。アプリケーションが追加する独自メトリックは、applicationスコープを管理するMetricRegistryインスタンスにデフォルトで保管されます。

- メトリックの登録とメタデータ

  メトリック・メタデータは、メトリックについての情報を記述および要約するデータの集合であり、様々なタイプのメトリックの検出および処理を簡単に行うために使用されます。メトリックの登録時にはMetadataクラスに以下のメタデータ情報を設定してMetricRegistryインスタンスに登録します。

メタデータ	説明
名前	メトリックの名称。
表示名	メトリック名が人間が読めるものでない場合 (例えば、メトリック名が UUID である場合)、表示目的のための人間が読めるメトリック名。
説明	メトリックの説明文。
タイプ	メトリックのタイプ（ゲージ、カウンター、メーター、ヒストグラム、またはタイマー）。
単位	メトリックで使用する単位。
タグ	キーと値 (key=value) のペアのコンマ区切りリスト。Prometheus上でデータを抽出する際のラベルとして使用します（Prometheusについては後述します）。
再利用可能	メトリック名が再使用可能かどうかを示す真偽値。例えば複数のメソッドにアノテーションを付けるときに、同じメトリックとして再使用したい場合にTrueを設定する。

MetricRegistryクラスによるメトリックの追加

- コード例

```
@Inject
MetricRegistry registry;          // MetricRegistyのインジェクション
Counter sampleHitsCounter;

@PostConstruct
private void init() {
    // Metricの登録
    Metadata sampleHitsCounterMetadata = new Metadata(
        "sampleHits",                                  // name
        "Sample Hits",                                 // display name
        "Number of hits on the sample call",          // description
        MetricType.COUNTER,                           // type
        MetricUnits.NONE);                            // units
    sampleHitsCounter = registry.counter(sampleHitsCounterMetadata);
}

public void sampleCount() {
    sampleHitsCounter.inc();   // カウンターのインクリメント
}
```

MetricRegistryクラスやMetadataクラス、Counterクラスおよび他のメトリッククラスのAPIについては以下を参照ください。
https://openliberty.io/docs/ref/microprofile/1.3/#package=org/eclipse/microprofile/metrics/package-frame.html&class=org/eclipse/microprofile/metrics/package-summary.html

MetricRegistryクラスによるメトリックの追加

– メトリック出力例

```
・・・
# TYPE application:sample_hits counter
# HELP application:sample_hits Number of hits on the sample call
application:sample_hits 4
・・・
```

前頁のコードが実行されると、sampleCount()メソッド呼び出しにおいてsampleHitsメトリック（カウンター）がインクリメントされます。MicroProfile Metrics REST API（例：GET /metrics/application）を呼び出すことにより、 sampleHitsメトリックのその時点のカウンターの値が上記のように出力されます。

CDIを使用したメトリックの注入

CDIを使用すると、メトリックの作成とMetricRegistryへの登録を簡単に行うことができます。Metadataクラスのインスタンスを作成する代わりに@Metricアノテーションのパラメーターを使用して、メトリック・メタデータを設定することが可能です。

- アノテーション

アノテーション	クラス
@Metric	org.eclipse.microprofile.metrics.annotation.Metric

※このアノテーションは型がCounter、Meter、Timer、Histogramのメンバ変数に対して指定可能です。

- パラメータ

パラメータ	説明
name	メトリックの名称。
displayName	メトリックの表示名。表示目的のための人間が読めるメトリック名。
descrition	メトリックの説明文。
unit	メトリックで使用する単位。デフォルトはMetricUnits.NONE（単位なし）。
tag	"key=value"形式のタグ文字列の配列。
absolute	trueの場合、メトリック名を nameパラメーターに指定されたとおりの名前に設定する。 falseの場合、パッケージ・クラスおよび名前に接頭部を追加して、完全修飾名を使用。

CDIを使用したメトリックの注入

- コード例

```
@Inject
@Metric(name="sampleHits",
        displayName="Sample Hits",
        description="Number of hits on the sample call",
        absolute=true)
Counter sampleHitsCounter;      // カウンターのインジェクション

public void sampleCount() {
    sampleHitsCounter.inc();   // カウンターのインクリメント
}
```

上記は前述したsampleHitsメトリック（カウンター）のコード例と同内容の処理をCDIによるメトリックの注入を使用して実装したコード例になります。@MetricアノテーションによりMetricRegistryおよびMetadataクラスに対する操作を省略することが可能です。（出力例はp21と同内容となります）

他のメトリックのコード例については以下を参照ください。
https://www.ibm.com/support/knowledgecenter/ja/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/cwlp_mp_metrics_api.html

メトリック・アノテーションの利用

MicroProfile Metricsでは、@Counted、@Timed、@Meteredおよび@Gaugeアノテーションといったメトリック・アノテーションを提供しています。

@Metricアノテーションと同様、これらのアノテーションを直接使用することによってメタデータ・パラメーターを設定して新たなメトリックを作成できます。パラメーターで名前が指定されていない場合、メトリックの名前はアノテーションが指定されたメソッドの名前から決定されます。

各メトリック・アノテーションは、クラスおよびメソッドに指定することが可能です。
但し、これらのアノテーションは各メトリック・アノテーション用のJavaインターセプターによって処理され、対象のメトリックの値が更新されるため、Javaインターセプターが介在できないクラス内部のprivateメソッドの呼出や、Javaインターセプターを使用できないfinalクラス等ではメトリック・アノテーションが使用できません。

メトリック・アノテーションの利用

- アノテーション

アノテーション	クラス
@Counted	org.eclipse.microprofile.metrics.annotation.Counted
@Timed	org.eclipse.microprofile.metrics.annotation.Timed
@Metered	org.eclipse.microprofile.metrics.annotation.Metered
@Gauge	org.eclipse.microprofile.metrics.annotation.Gauge

（アノテーションのパラメータについては次頁に記載）

メトリック・アノテーションの利用

- パラメータ
  アノテーションにより各パラメータの使用可否が異なります。

パラメータ	説明	使用可能
name	メトリックの名称。	全て
displayName	メトリックの表示名。表示目的のための人間が読めるメトリック名。	全て
description	メトリックの説明文。	全て
unit	メトリックで使用する単位。デフォルトはMetricUnits.NONE（単位なし）。	@Gauge以外
tag	"key=value"形式のタグ文字列の配列。	全て
absolute	trueの場合、メトリック名を nameパラメーターに指定されたとおりの名前に設定します。 falseの場合、パッケージ・クラスおよび名前に接頭部を追加して、完全修飾名を使用します。	全て
reusable	メトリック名が再使用可能かどうかを示す真偽値。例えば複数のメソッドにアノテーションを付けるときに、同じメトリックとして再使用したい場合にTrueを設定します。	@Gauge以外
monotonic	trueに設定されている場合、カウンターは単調に増分されて、アノテーションを付けられたメソッドの呼び出し総数がカウントされます。 falseに設定されている場合、アノテーションを付けられたメソッドが呼び出されるとカウンターは増分、リターンすると減分され、アノテーションを付けられたメソッドの現在処理中の数がカウントされます。 デフォルトはfalse。	@Countedのみ

メトリック・アノテーションの利用

- コード例

```
@Counted(name="sampleHits",
        displayName="Sample Hits",
        description="Number of hits on the sample call",
        absolute=true
        monotonic=true)
public void sampleCount() {

    // カウント対象メソッドの処理
}
```

上記は前述したsampleHitsメトリック（カウンター）のコード例と同内容の処理を@Countedアノテーションを使用して実装したコード例になります。@CountedアノテーションによりMetricRegistryやMetadataクラス、Counterクラスに対する操作を省略することが可能です。（出力例はp21と同内容となります）

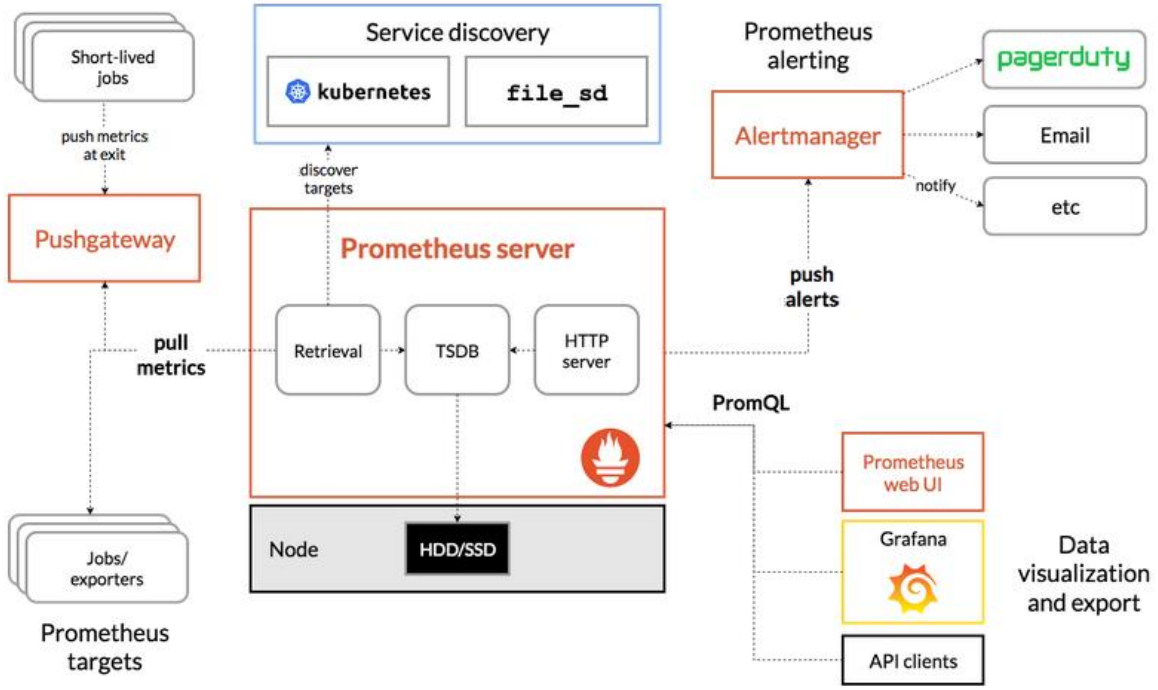
他のメトリック・アノテーションのコード例については以下を参照ください。
https://www.ibm.com/support/knowledgecenter/ja/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/cwlp_mp_metrics_api.html

Prometheusとの連携

Prometheusによるメトリックの収集

- Prometheus(https://prometheus.io/)とは
  - 時系列メトリックデータの監視に特化したオープンソースのサービスモニタリングツール
  - メトリックデータの収集方式としてPull型を採用しており、「クラウドへの適合性」「単一バイナリで動作するシンプル性」「実装レベルの高さ」などの特徴を持つ
  - Kubernetes環境でよく利用されることもあり、現在最も注目を集めるモニタリングツール

<Prometheusのアーキテクチャー>



https://prometheus.io/docs/introduction/overview/より

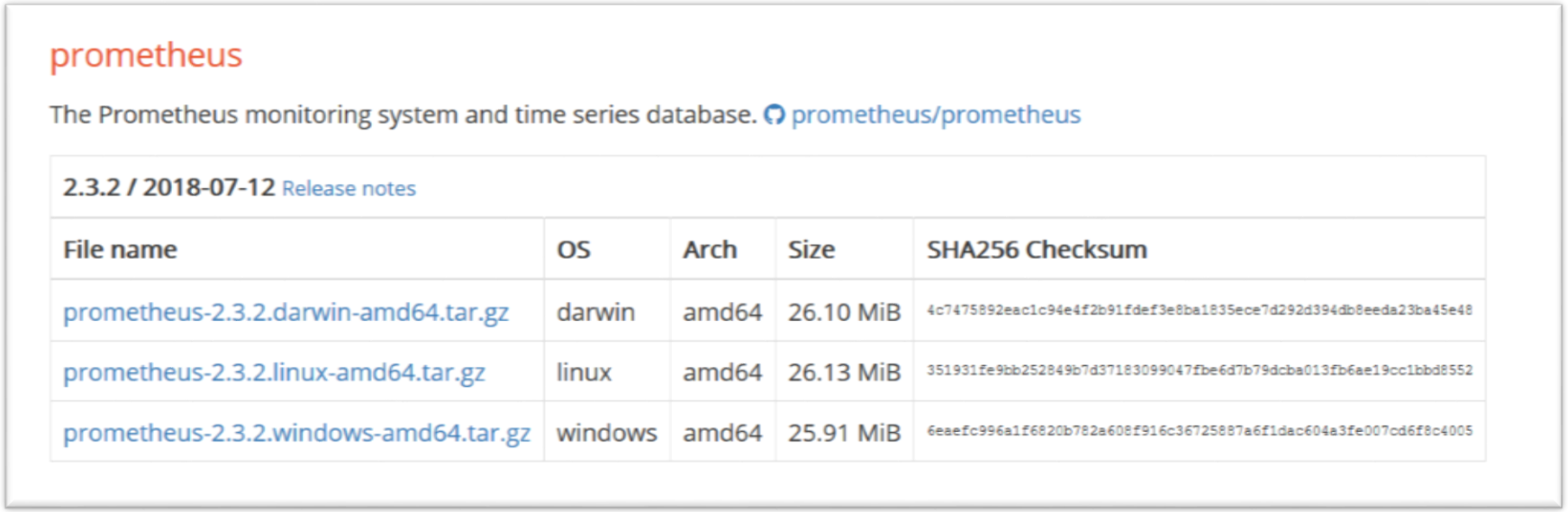
Prometheusによるメトリックの収集

- Prometheusの導入と設定

  Prometheusを導入し、MicroProfile Metrics REST APIを監視対象に設定することにより、WAS Liberty上で稼働するマイクロサービスのメトリックデータをPrometheusで収集することが可能です。

  – Prometheusの導入

    Prometheusの公式サイトから実行可能モジュールをダウンロードすることが可能です。(https://prometheus.io/download/)
    圧縮ファイルをダウンロードして解凍すればすぐに使用することができます。
    また、Docker環境向けのコンテナイメージも提供されています。



prometheus

The Prometheus monitoring system and time series database. prometheus/prometheus

2.3.2 / 2018-07-12 Release notes

File name	OS	Arch	Size	SHA256 Checksum
prometheus-2.3.2.darwin-amd64.tar.gz	darwin	amd64	26.10 MiB	4c7475892eac1c94e4f2b91fdef3e8ba1835ece7d292d394db8eeda23ba45e48
prometheus-2.3.2.linux-amd64.tar.gz	linux	amd64	26.13 MiB	351931fe9bb252849b7d37183099047fbe6d7b79dcba013fb6ae19cc1bbd8552
prometheus-2.3.2.windows-amd64.tar.gz	windows	amd64	25.91 MiB	6eaefc996a1f6820b782a608f916c36725887a6f1dac604a3fe007cd6f8c4005

Prometheusによるメトリックの収集

– Prometheusの設定

Prometheusではyaml形式の設定ファイルに監視対象を設定します。このファイルにMicroProfile Metrics REST APIを監視対象にするための設定を記述します。

- 設定例

```
scrape_configs:

  - job_name: 'mpMetrics'
    scrape_interval: 5s
    scheme: https

    basic_auth:
      username: 'user'
      password: 'password'

    tls_config:
      insecure_skip_verify: true

    static_configs:
      - targets: ['localhost:9443']
```

メトリック取得のインターバルやプロトコルを指定

Liberty側で設定したBasic認証のuserName/userPasswordを指定

サーバー側で自己証明書を使用する場合に指定（検証を省略）

左記の指定により監視対象のURLはhttps://localhost:9443/metricsとなる

Prometheusの詳細な設定方法については以下を参照ください。
https://prometheus.io/docs/prometheus/latest/configuration/configuration/

Prometheusによるメトリックの収集

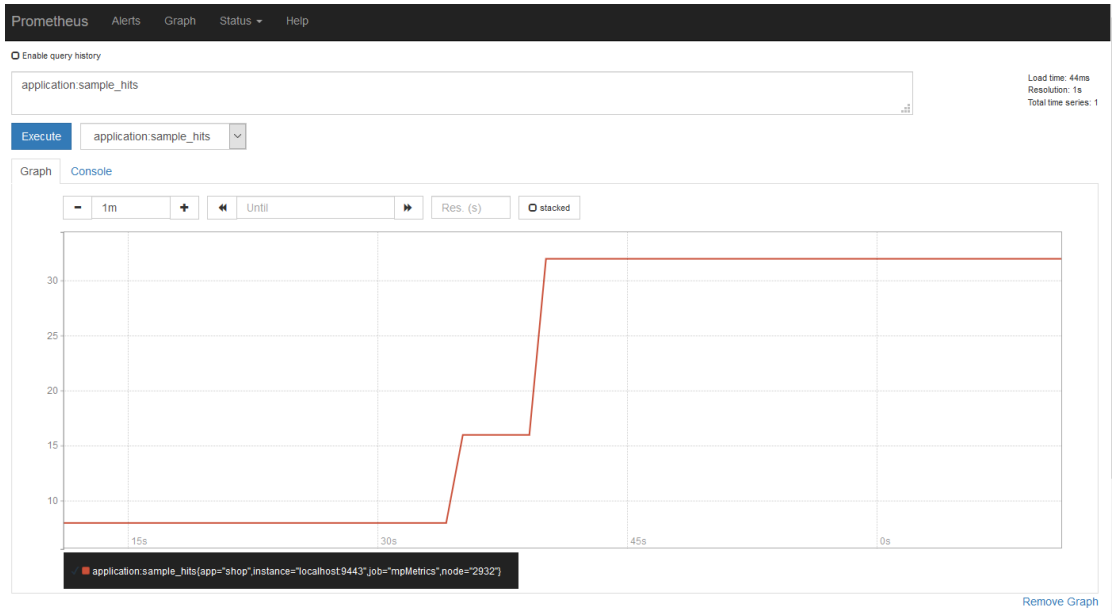
– メトリックの確認

設定ファイルを指定してPrometheusを起動すると、設定に従い監視対象にアクセスしてメトリックを取得します。Prometheusが提供するWeb UI（Expression Browser）を使用して、収集されたメトリックの値を表やグラフ形式で簡易に確認することが可能です。

- Prometheusの起動

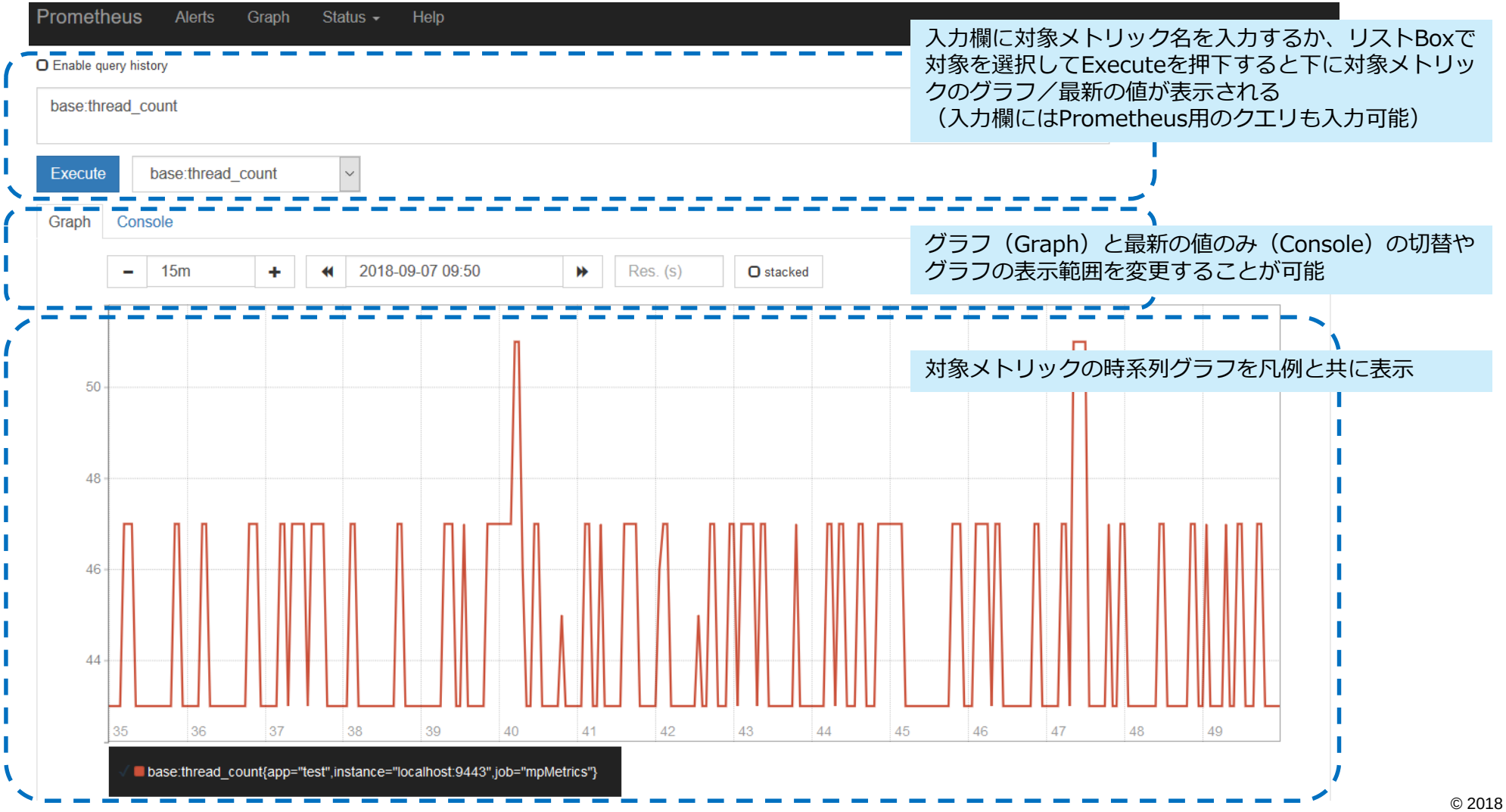
```
$./prometheus --config.file={設定ファイル}
```

- Prometheus Web UI
  (http://localhost:9090/graph)



Prometheusによるメトリックの収集

- Web UIによるメトリックの確認方法



Prometheusによるメトリックの収集

- Prometheusのラベルとクエリ機能

  Prometheusでは蓄積されたデータに対し、その値をそのまま単純にグラフ化するだけでなく、さまざまな条件を指定してクエリ処理を行って集計し、その結果をグラフ化するという機能も備えています。

  以下の形式でメトリック・データにラベルを付与しておくことにより、このクエリ機能で指定したラベルを含むデータやより複雑な条件に該当するデータのみを抽出して表示させることが可能です。

```
<メトリック名>{<ラベル名1>=<値>, <ラベル名2>=<値>, ...}
```

（例）

```
node_cpu {cpu="cpu0", mode="system"}
```

MicroProfile Metricではメトリック・メタデータのタグを指定すると、出力されるメトリック・データにタグの情報がPrometheusのラベルとして付与され、クエリ機能による抽出に使用することができます。

Prometheusのクエリ機能の詳細については以下を参照ください。
https://prometheus.io/docs/prometheus/latest/querying/basics/

参考リンク

- MicroProfile Metrics 1.1
  https://github.com/eclipse/microprofile-metrics/releases/download/1.1/metrics_spec-1-1.pdf

- IBM Knowledge Center - MicroProfile メトリックを使用したモニタリング
  https://www.ibm.com/support/knowledgecenter/ja/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/twlp_mp_metrics_monitor.html

- Providing metrics from a microservice
  https://openliberty.io/guides/microprofile-metrics.html

- MicroProfile Metrics 1.1 ベンダー・メトリック **18.0.0.3+**

https://www.ibm.com/support/knowledgecenter/ja/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/rwlp_monitor_metrics_rest_api.html

End of File