sysdig

Prometheus
メトリクスを収集する

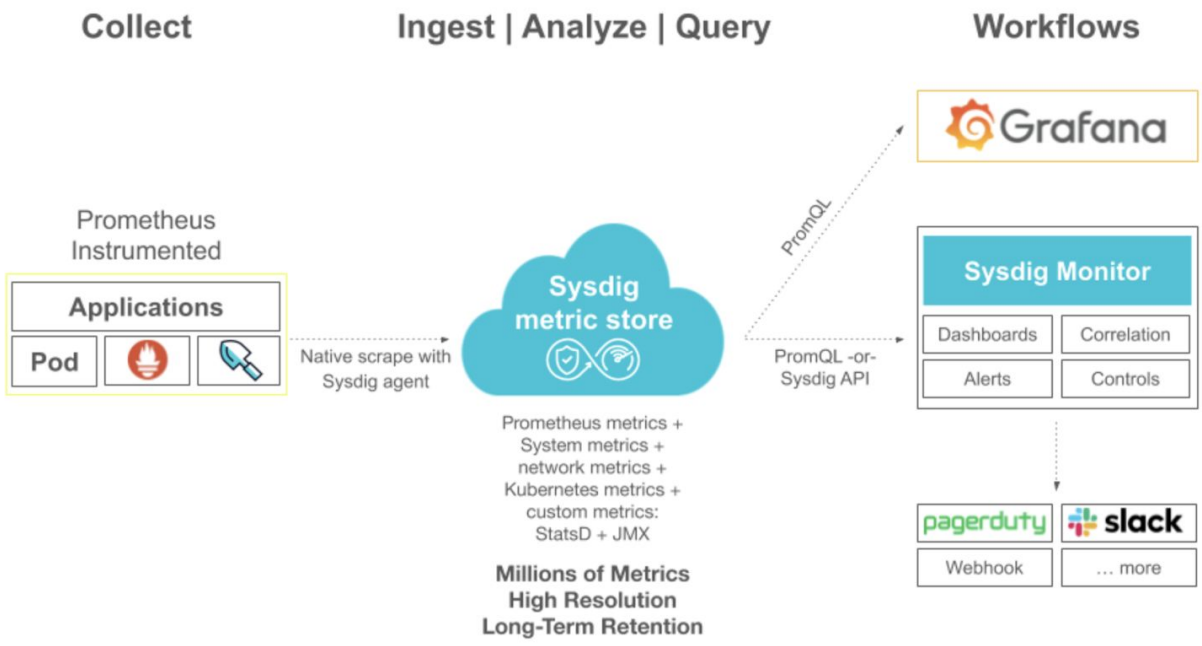
2020年12月9日現在

本文の内容は、2020年12月9日現在のCollect Prometheus Metricsのドキュメント(https://docs.sysdig.com/en/collect-prometheus-metrics.html)を元に日本語に翻訳・再構成した内容となっております。

Prometheusメトリクスを収集する

Sysdigは、Prometheusのネイティブメトリクスとラベルの収集、保存、クエリをサポートしています。 Sysdigは、Prometheusを使用するのと同じ方法で使用でき、Prometheusクエリ言語（PromQL）を利用してダッシュボードとアラートを作成できます。 SysdigはPrometheus HTTP APIと互換性があり、PromQLを使用してプログラムで監視データをクエリし、SysdigをGrafanaなどの他のプラットフォームに拡張できます。

メトリクス収集の観点から見ると、軽量のPrometheusサーバーがSysdigエージェントに直接組み込まれているため、メトリクス収集が容易になります。これは、Prometheus構文を使用したフィルタリングと再ラベル付けを行うターゲット、インスタンス、ジョブもサポートします。 独自のホストでPrometheusメトリクスエンドポイントを公開するこれらのプロセスを識別するようにエージェントを構成し、それをSysdigコレクターに送信して、保存およびその後の処理を行うことができます。



注意
このドキュメントでは、メトリクスと時系列を同じ意味で使用しています。 設定パラメーターの説明は「メトリクス」を指しますが、厳密なプロメテウスの用語では、これらは時系列を意味しま

す。 つまり、100メトリクスの制限を適用すると、すべての時系列データが同じメトリクス名を持たない可能性がある時系列に制限が適用されます。

Prometheusメトリクスコレクションでは、Prometheus製品自体をインストールする必要はありません。

注意

- Sysdigエージェントv10.5.0: Prometheusネイティブサービスディスカバリをサポートするpromscrape.v2。これをサポートするために、Prometheusネイティブサービスディスカバリを有効にしたときに使用するKubernetesポッドディスカバリのルールでデフォルトのprometheus.yamlファイルが追加されています。
- Sysdigエージェントv10.0.0以降：軽量のPrometheusサーバーであるpromscrapeは、デフォルトでPrometheusメトリクスのスクレイピングに使用されます。 このコンポーネントは、オープンソースのPrometheusに基づいています。
- Sysdigエージェントv9.8.0からv10.0：軽量のPrometheusサーバーであるpromscrapeがv9.8.0で導入され、Prometheusメトリクスをスクレイピングします。 このメソッドを使用するには、dragent.yamlファイルでpromscrapeを有効にする必要があります。
- Sysdigエージェントv0.70.0以降：Prometheusエクスポーターから自動的にメトリクスを収集するための豊富なサポートを提供します。

以下のトピックでは、Sysdigエージェントをサービスディスカバリー、メトリクス収集、およびその後の処理用に設定する方法について詳しく説明します。

- Prometheusメトリクスの操作
- 環境のセットアップ
- Sysdigエージェントの設定
- 設定例
- Prometheusメトリクスコレクションの制限の適用
- Prometheus ネイティブサービスディスカバリーを有効にする
- GrafanaのSysdigデータソースを設定する
- ロギングとトラブルシューティング
- リモートホストからのPrometheusメトリクスの収集

もっと詳しく知る

Prometheusメトリクススの詳細と、そのようなメトリクススが通常どのように使用されるかについては、次のブログ投稿を参照してください。

- Prometheusメトリクスの計測
- SysdigのPrometheusモニタリング
- Sysdigが初のクラウドスケールのPrometheusモニタリング製品を発表
- 大規模なPrometheusを使用した場合における課題

Prometheusメトリクスの操作

Sysdigエージェントは、（ホストレベルとコンテナレベルの両方で）実行中のすべてのプロセスに対する可視性を使用して、Prometheusメトリクスをスクレイピングするのに適したターゲットを見つけます。 デフォルトでは、スクレイピングは試行されません。 機能が有効になると、エージェントは対象となるターゲットのリストを作成し、フィルタリングルールを適用して、Sysdigコレクターに送り返します。

エージェントv10.0.0の新しいPrometheus機能

以下の機能を使用するためにSysdigエージェントv10.0以降が必要です。

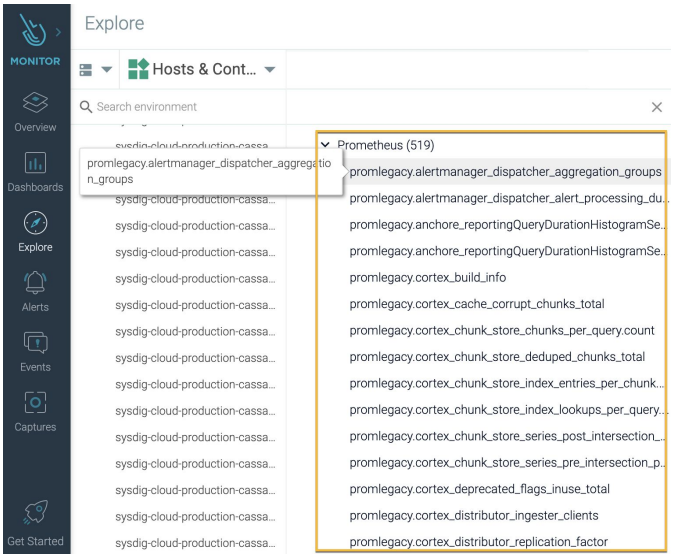
- Prometheusデータを使用する新しい機能：
    - PromQLクエリを使用してデータを視覚化する機能。 PromQLの使用を参照してください。
    - PromQLベースのダッシュボードからアラートを作成します。 パネルアラートの作成を参照してください。
    - ダッシュボードv2およびアラートの下位互換性

      注意

      ダッシュボードv2ヒストグラムを使用して新しいPromQLデータを視覚化することはできません。 ヒストグラムメトリクスには時系列ベースの視覚化を使用します。
- エージェントごとの新しいメトリクス制限：
    - カスタムメトリクス：10,000

これは、ホスト、コンテナ、Kube Stateメトリクスなど、すぐに使用できるエージェントメトリクスに追加されます。

    - Prometheusメトリクス：8000
    - StatsDメトリクス：1000
    - JMXメトリクス：500
    - AppChecks：500
- 10秒のデータ粒度
- 新しいMetric Storeの保持率が高くなりました。
- 新しいメトリクスの命名規則：
    - レガシーPrometheusメトリクスは、prefix promlegacyで利用できます。 命名規則は`promlegacy.<metrics>`です。 たとえば、`cortex_build_info`の名前は`promlegacy.cortex_build_info`に変更されます。



前提条件とガイドライン

最新のPrometheus機能を使用するには、Sysdigエージェントv 10.0.0以降が必要です。

`dragent.yaml`ファイルでPrometheus機能を有効にします。

```
prometheus:
   enabled: true
```

詳細については、環境の設定を参照してください。

ターゲットのエンドポイントは、エージェントへからTCP接続ができる必要があります。エージェントは、リモートまたはローカルのターゲットを、`IP: Port`またはdragent.yamlのURLで指定します。

サービスディスカバリー

Prometheusネイティブサービスディスカバリーを使用するには、Prometheusネイティブサービスディスカバリの有効化で説明したように、promscrape.v2を有効にしてください。このセクションでは、Sysdig エージェントでプロセスフィルタを設定することを含む Sysdig のサービスディスカバリの方法について説明します。

Sysdigエージェントでのサービス発見の方法は、Prometheusサーバーのそれとは異なります。Prometheusサーバーがいくつかのサービス発見メカニズムと構成設定を読み取るための prometheus.yml ファイルとの統合を内蔵しているのに対し、Sysdigエージェントは dragent.yaml, ファイルの仕様に一致するプロセス（エクスポートまたはインスツルメンテーションされたプロセス）を自動発見し、組み込みの軽量Prometheusサーバーにそこからメトリクスを取得するように指示しています。

エージェント内の軽量Prometheusサーバはpromscrapeという名前で、dragent.yamlファイル内の同名のフラグによって制御されます。詳細はSysdigエージェントの設定を参照してください。

クラスター内のすべてのマシン上で実行されているプロセスをスクレイプできるPrometheusサーバーとは異なり、エージェントはインストールされているホスト上で実行されているプロセスのみをスクレイプすることができます。

対象となるプロセス/ポート/エンドポイントのセット内で、エージェントは、Prometheusメトリクスをエクスポートしているポートのみをスクレイプし、ポートが接続してスクレイプしようとする試みにどのように反応するかに基づいて、ポート上でのスクレイプまたは再試行の試みを停止します。したがって、スクレイピングを試みるプロセスとポートを、エクスポート先の予想される最小範囲に制限する構成を作成することを強くお勧めします。これにより、エージェントとアプリケーションの両方に意図しない副作用が発生する可能性を最小限に抑えることができます。

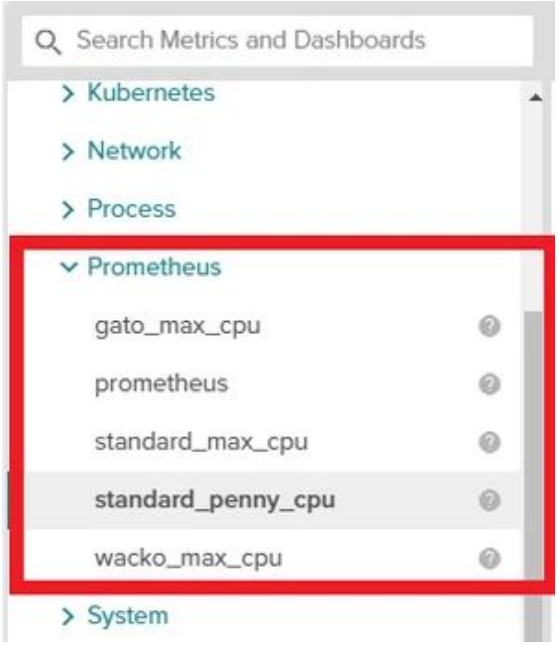
エンドツーエンドのメトリクス収集は、次のように要約できます：

1. プロセスが一連のプロセスフィルターの include/excludeルールと明確に一致する場合、プロセスはスクレイピングの対象になると判断されます。詳細については、プロセスフィルターを参照してください。
2. エージェントは、ポートのサブセットまたは別のエンドポイント名、あるいはその両方へのスクレイピングを制限する追加の設定が存在しない限り、すべてのリスニングTCPポートの/metricsエンドポイントで適格なプロセスをスクレイピングしようとします。
3. エージェントv9.8.0以降、取り込み時のメトリクスのフィルタリングを有効にすることができます。有効にすると、メトリクスを受信するときに、フィルタリングルールが取り込み時に適用されます。詳細については、Prometheusメトリクスのフィルタリングを参照してください。
4. メトリクスを受信すると、エージェントはSysdigコレクターに送信する前に以下のルールを適用します。
    - グローバル`metrics_filter`ルールを適用します。

      カスタムメトリクスを include/excludeするをご覧ください。

    - 設定された`max_metrics`を課して、メトリクスの数を制限します。
    - Prometheusメトリクスコレクションの制限の適用を参照してください。

メトリクスは、最終的にはPrometheusセクションのSysdig Monitor Exploreインターフェースに表示されます。



環境のセットアップ

Kubernetes環境のクイックスタート

すでにKubernetes Service Discovery（具体的にはこのサンプルprometheus-kubernetes.ymlのアプローチ）を利用しているPrometheusユーザーは、ポッドに、スクレイピングの対象としてマークするアノテーションがすでに添付されている可能性があります。このような環境では、いくつかの簡単な手順で、Sysdigエージェントを使用して同じメトリクスをすぐに取得できます。

1. SysdigエージェントでPrometheusメトリクス機能を有効にします。 DaemonSetsを使用してデプロイしていると想定すると、DaemonSet YAMLに以下を含めることで、必要な設定をエージェントのdragent.yamlに追加できます（sysdig-agentコンテナのenvセクションに配置します）。

```
- name: ADDITIONAL_CONF
  value: "prometheus:\n  enabled: true"
```

2. Prometheusエクスポーターを含むKubernetesポッドが次のアノテーションでデプロイされていることを確認して、スクレイピングを有効にします（リスニングエクスポーターのTCPポートを置き換えます）。

```
spec:
  template:
    metadata:
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "exporter-TCP-port"
```

   上記の設定は、エクスポーターが/metricsと呼ばれる一般的なエンドポイントを使用することを前提としています。 エクスポーターが別のエンドポイントを使用している場合は、exporter-endpoint-nameの代わりに、次のオプションのアノテーションを追加して指定することもできます。

```
prometheus.io/path: "/exporter-endpoint-name"
```

シンプルなエクスポーターのこのKubernetes Deploymentを試すと、自動検出されたPrometheusメトリクスがSysdig Monitorに表示されるのがすぐにわかります。 この実例を基礎として使用して、独自のエクスポーターに同様にアノテーションを付けることができます。

スクレイピングしたいアノテーション付きのKubernetesポッドにデプロイされていないPrometheusエクスポーターがある場合、次のセクションでは、メトリクスを検索してスクレイピングするようにエージェントを設定するためのオプションの完全なセットについて説明します。

コンテナ環境のクイックスタート

Dockerベースのコンテナ環境でPrometheusスクレイピングが機能するようにするには、アプリケーションによってメトリクスがエクスポートされる正しいポートとパスで<exporter-port>と<exporter-path>を置き換えて、次のラベルをアプリケーションコンテナに設定します。

- `io.prometheus.scrape=true`
- `io.prometheus.port=<exporter-port>`
- `io.prometheus.path=<exporter-path>`

たとえば、mysqld-exporterをスクレイピングする場合は、次のようにコンテナを起動します。

```
docker -d -l io.prometheus.scrape=true -l io.prometheus.port=9104 -l io.prometheus.path=/metrics mysqld-exporter
```

Sysdigエージェントの設定

エージェントの場合と同様に、機能のデフォルト設定はdragent.default.yamlで指定されており、dragent.yamlでパラメーターを設定することでデフォルトを上書きできます。 各パラメータについて、dragent.yamlで設定しないでください。dragent.default.yamlのデフォルトは引き続き有効です。

主な設定パラメータ

パラメータ	デフォルト	説明
`prometheus`	下記参照	Prometheusスクレイピングのオンとオフを切り替えます。
`process_filter`	下記参照	スクレイピングの対象となるプロセスを指定します。 以下の「プロセスフィルター」セクションを参照してください。
`use_promscrape`	下記参照	Prometheusメトリクスのスクレイピングにpromscrapeを使用するかどうかを決定します。 エージェントv9.8.0以降では、このパラメーターはデフォルトで無効になっています。 エージェントv10.0.0では、このパラメーターはデフォルトで有効になっています。

promscrepe

Promscrapeは、Sysdigエージェントが組み込まれた軽量のPrometheusサーバーです。 use_promscrapeパラメーターは、それを使用してPrometheusエンドポイントをスクレイピングするかどうかを制御します。

パラメータ	デフォルト	説明

`use_promscrape`	`true`	エージェントv10.0.0では、このフラグはデフォルトで有効になっています。 エージェントv9.8.0以降では、このオプションを明示的に有効にして、スクレイピングにpromscrapeを使用します。 有効にすると、ユーザーは取り込み前にソースでPrometheusメトリクスをフィルタリングできます。 詳細については、「Prometheusメトリクスのフィルタリング」を参照してください。

Prometheus

prometheusセクションでは、Prometheusメトリクスの収集と分析に関連する動作を定義します。この機能をオンにして、エージェント側からスクレイピングするメトリクスの数に制限を設定し、ヒストグラムメトリクスをレポートして、失敗したスクレイピングの試行をログに記録するかどうかを決定します。

パラメーター	デフォルト	説明
`enabled`	`false`	Prometheusスクレイピングのオンとオフを切り替えます。
`interval`	`10`	エージェントがPrometheusメトリクススのポートをスクレイピングする頻度（秒単位）
`prom_service_discovery`	`false`	値が true に設定されている場合、ネイティブの Prometheus サービスの検出を有効にします。無効にすると、promscrape がターゲットのスクレイプに使用されます。Prometheus ネイティブサービス検出を有効にする を参照してください。
`log_errors`	`true`	適格なターゲットをスクレイピングするために失敗した試行の詳細をエージェントがログに記録するかどうか

`max_metrics`	`1000`	すべてのターゲットにわたってスクレイピングされるPrometheusメトリクスの合計の最大数。この値1000は、エージェントごとの最大値であり、他のカスタムメトリクス（statsd、JMX、その他のアプリケーションチェックなど）とは別の制限です。
`max_metrics_per_process`	`1000`	エージェントが単一のスクレイピングされたターゲットから保存するPrometheusメトリクスの最大数。 （エージェントバージョン0.90.3では、デフォルトが100から1000に変更されました。） エージェントv10.0.0では非推奨
`max_tags_per_metric`	`20`	エージェントがスクレイピングされたターゲットから保存する、Prometheusメトリクスごとのタグの最大数 エージェントv10.0.0では非推奨
`timeout`	1	タイムアウトになる前に、Prometheusエンドポイントをスクレイピングするときにエージェントが待機する時間を構成するために使用されます。デフォルト値は1秒です エージェントv10.0以降、このパラメーターはpromscrapeが無効になっている場合にのみ使用されます。現在、promscrapeがデフォルトであるため、タイムアウトは非推奨と見なすことができますが、promscrapeを明示的に無効にした場合でも、タイムアウトは使用されます

`histograms`	`false`	エージェントがヒストグラムメトリクスをスクレイピングしてレポートするかどうか。詳細については、ヒストグラムメトリクスの有効化を参照してください。

Process Filter

process_filterセクションは、エージェントが認識しているプロセスのうち、スクレイピングの対象となるプロセスを指定します。

dragent.yamlでprocess_filterを指定すると、dragent.default.yamlに表示されているPrometheusのprocess_filterセクション全体（つまり、すべてのルール）が置き換えられることに注意してください。

プロセスフィルターは、エージェントが認識しているプロセスごとに上から下に評価される一連の`include`および`exclude`ルールで指定されます。プロセスがincludeルールに一致する場合、プロセスのスクレイピング方法をさらに制限するconfセクションもルール内に表示されない限り、プロセスの各リスニングTCPポートの/metricsエンドポイントを介してスクレイピングが試行されます（以下のconfを参照）。

1つのルールで複数のパターンを指定できます。その場合、ルールが一致するためには、すべてのパターンが一致する必要があります（ANDロジック）。

パターン値内では、単純な「glob」ワイルドカードを使用できます。*は任意の数の文字（なしを含む）と一致し、？任意の1文字と一致します。 YAML構文のため、ワイルドカードを使用する場合は、必ず値を引用符（"*"）で囲んでください。

以下の表は、プロセスフィルタールールでサポートされるパターンを示しています。現実的な例を示すために、以下のDockerコマンドラインを使用してコンテナとしてデプロイできる簡単なサンプルPrometheusエクスポーター（ここではソースコード）を使用します。いくつかの設定オプションを説明するために、このサンプルエクスポーターは、より一般的な/metricsエンドポイントの代わりに/promtheusにPrometheusメトリクスを提示します。これは、以下の設定例でさらに示されます。

```
# docker run -d -p 8080:8080 \
```

```
    --label class="exporter" \
    --name my-java-app \
    luca3m/prometheus-java-app

# ps auxww | grep app.jar
root    11502 95.9  9.2 3745724 753632 ?    Ssl  15:52   1:42 java -jar /app.jar
--management.security.enabled=false

# curl http://localhost:8080/prometheus
...
random_bucket{le="0.005",} 6.0
random_bucket{le="0.01",} 17.0
random_bucket{le="0.025",} 51.0
...
```

パターン名	説明	例
`container.image`	指定されたイメージを実行しているコンテナ内でプロセスが実行されているかどうかに一致します	`- include:` `container.image: luca3m/prometheus-java-app`
`container.name`	プロセスが指定された名前のコンテナ内で実行されているかどうかに一致します	`- include:` `container.name: my-java-app`
`container.label.*`	指定された値に一致するラベルを持つコンテナでプロセスが実行されている場合に一致します	`- include:` `container.label.class: exporter`

`kubernetes.<object>.annotation.*`
`kubernetes.<object>.label.*`
	指定された値と一致するアノテーション/ラベルでマークされたKubernetesオブジェクト（ポッド、ネームスペースなど）にプロセスがアタッチされている場合に一致します。
	注：このパターンは、上記のDockerのみのコマンドラインには適用されませんが、エクスポーターがこのサンプルYAMLを使用してKubernetesデプロイメントとしてインストールされた場合に適用されます。
	注：サポートされているアノテーションとラベルの完全なセットについては、下記のKubernetesオブジェクトを参照してください。
		`- include:`
		
		`kubernetes.pod.annotation.prometheus.io/scrape: true`

`process.name`	実行中のプロセスの名前と一致します	`- include:`
		
		`process.name: java`

`process.cmdline`	コマンドライン引数に一致します	`- include:`
		
		`process.cmdline: "*app.jar*"`

`port`	プロセスが1つ以上のTCPポートで待機している場合に一致します。
	単一のルールのパターンは、この例に示すように単一のポートまたは単一の範囲（例：8079-8081）を指定できますが、ポート/範囲のコンマ区切りリストはサポートしていません。
	注：このパラメーターは、プロセスがリッスンしているポートに基づいて、プロセスがスクレイピングに適格かどうかを確認するためにのみ使用されます。たとえば、プロセスが1つのポートでアプリケーショントラフィックをリッスンしており、Prometheusメトリクスをエクスポートするために2番目のポートを開いている場合、アプリケーションポート（エクスポートポートではない）を指定し、conf でエクスポートポートを指定できます。 セクション（ただし、アプリケーションポートではありません）。プロセスは適格であると一致し、エクスポートポートは廃棄されます。
		`- include:`
		
		`port: 8080`

`appcheck.match`	特定の名前またはパターンのアプリケーションチェックがプロセスに対して実行されるようにスケジュールされているかどうかに一致します。	`- exclude:`
		
		`appcheck.match: "*"`

前述の`include`の例では、それぞれがプロセスに一致しましたが、前述の単一のルールで複数のパターンを組み合わせる機能により、次の非常に厳密な設定も一致します。

```
- include:
  container.image: luca3m/prometheus-java-app
  container.name: my-java-app
  container.label.class: exporter
  process.name: java
  process.cmdline: "*app.jar*"
  port: 8080
```

Conf

port_filterの各includeルールには、適格なプロセスでスクレイピングがどのように試行されるかをさらに説明するconf部分を含めることができます。 conf部分が含まれていない場合、一致プロセスのすべてのリスニングポートの/ metricsエンドポイントでスクレイピングが試行されます。 可能な設定：

パラメータ名	説明	例

`port`	スクレイピングされる単一のTCPポートの静的番号、または中括弧で指定されたコンテナ/KubernetesラベルまたはKubernetesアノテーション。プロセスがこのラベルでマークされたコンテナで実行されているか、このアノテーション/ラベルでマークされたKubernetesオブジェクト（ポッド、ネームスペースなど）に接続されている場合、スクレイピングは、ラベル/アノテーションの値として指定されたポートでのみ試行されます。
	注：照合するラベル/注釈には、赤で表示されたテキストは含まれません。
	注：サポートされているアノテーションとラベルの完全なセットについては、Kubernetesオブジェクトを参照してください。
	注：コンテナ内でエクスポーターを実行する場合、これは、コンテナがホストに公開するポートではなく、コンテナー内のエクスポータープロセスがlistenするポート番号を指定する必要があります。
		`port: 8080`
		
		- or -
		
		`port: "{container.label.io.prometheus.port}"`
		
		- or -
		
		`port: "{kubernetes.pod.annotation.prometheus.io/port}"`

`port_filter`	スクレイピングを試行できる適格なプロセスのリスニングTCPポートの最終的なセットを定義する包含ルールと除外ルールのセット。 構文は、process_filterの上位レベルのインクルードルール内のポートパターンオプションとは異なることに注意してください。 ここで、特定のルールには、単一のポート、コンマで区切られたポートのリスト（大括弧で囲まれている）、または連続したポート範囲（大括弧なし）を含めることができます。	`port_filter:`
		
		`- include: 8080 - exclude: [9092,9200,9300] - include: 9090-9100`

`path`	スクレイピングされるエンドポイントの静的な仕様か、中括弧で指定されたコンテナ/Kubernetesラベル名またはKubernetesアノテーションのいずれか。プロセスがこのラベルでマークされたコンテナで実行されているか、このアノテーション/ラベルでマークされたKubernetesオブジェクト（ポッド、ネームスペースなど）にアタッチされている場合、ラベル/アノテーションの値として指定されたエンドポイントを介してスクレイピングが試行されます。
	パスが指定されていない場合、または指定されているがエージェントがプロセスに添付されたラベル/注釈を見つけられない場合、Prometheusエクスポーターの一般的なデフォルトである/metricsが使用されます。
	注：照合するラベル/注釈には、赤で表示されたテキストは含まれません。
	注：サポートされているアノテーションとラベルの完全なセットについては、Kubernetesオブジェクトをご覧ください。
		`path: "/prometheus"`
		
		- or -
		
		`path: "{container.label.io.prometheus.path}"`
		
		- or -
		
		`path: "{kubernetes.pod.annotation.prometheus.io/path}"`

`host`	ホスト名またはIPアドレス。 デフォルトはlocalhostです。	`host: 192.168.1.101`
		`- or -`
		`host: subdomain.example.com`
		`- or -`
		`host: localhost`

`use_https`	trueに設定すると、エクスポーターへの接続は、HTTPではなくHTTPSを介してのみ試行されます。 デフォルトではfalseです。 （エージェントバージョン0.79.0以降で使用可能）	`use_https: true`
`ssl_verify`	trueに設定すると、HTTPS接続のサーバー証明書の検証が実行されます。デフォルトではfalseです。 検証は0.79.0より前ではデフォルトで有効にされていました。 （エージェントバージョン0.79.0以降で使用可能）	`ssl_verify: true`

認証統合

エージェントバージョン0.89以降、Sysdigは認証を必要とするエンドポイントからPrometheusメトリクスを収集できます。 この機能を有効にするには、以下のパラメーターを使用します。

- ユーザー名/パスワード認証の場合：
  - `username`
  - `password`
- トークンを使用した認証の場合：
  - `auth_token_path`
- 証明書キーによる証明書認証の場合：
  - `auth_cert_path`
  - `auth_key_path`

注意

トークン置換は、すべての許可パラメーターでもサポートされています。 たとえば、次のように指定することで、Kubernetesアノテーションからユーザー名を取得できます

```
username: "{kubernetes.service.annotation.prometheus.openshift.io/username}"
```

conf Authenticationの例

以下は、OpenShift、KubernetesなどでのすべてのPrometheus認証設定オプションを示すdragent.yamlセクションの例です。

この例では：

- `username/password`は、OpenShiftによって使用されるデフォルトのアノテーションから取得されます。
- `auth token`パスは、Kubernetesデプロイメントで一般的に利用できます。
- ここでetcdに使用される`certificate`と`key`は、通常、エージェントが簡単にアクセスできない場合があります。この場合、それらはホストのネームスペースから抽出され、Kubernetesシークレットに設定されてから、エージェントコンテナにマウントします。

```
prometheus:
  enabled: true
  process_filter:
    - include:
        port: 1936
        conf:
            username: "{kubernetes.service.annotation.prometheus.openshift.io/username}"
            password: "{kubernetes.service.annotation.prometheus.openshift.io/password}"
    - include:
        process.name: kubelet
        conf:
            port: 10250
            use_https: true
            auth_token_path: "/run/secrets/kubernetes.io/serviceaccount/token"
    - include:
        process.name: etcd
        conf:
            port: 2379
            use_https: true
```

```
auth_cert_path: "/run/secrets/etcd/client-cert"
auth_key_path: "/run/secrets/etcd/client-key"
```

Kubernetesオブジェクト

上記のように、Kubernetesのラベルやアノテーションの自動検出された値に基づいて設定できる複数の設定オプションがあります。いずれの場合も、形式は`"kubernetes.OBJECT.annotation."`で始まります。`"kubernetes.OBJECT.label."`または`OBJECT`は、サポートされている次のKubernetesオブジェクトタイプのいずれかです。

- `daemonSet`
- `deployment`
- `namespace`
- `node`
- `pod`
- `replicaSet`
- `replicationController`
- `service`
- `statefulset`

最後のドットの後に追加する設定テキストは、エージェントが検索するKubernetesラベル/アノテーションの名前になります。プロセスに添付されたラベル/アノテーションが検出された場合、そのラベル/アノテーションの値が設定オプションに使用されます。
Kubernetesのラベル/アノテーションを特定のプロセスに関連付けるには複数の方法があることに注意してください。これの最も単純な例の1つは、Kubernetes環境のクイックスタートに示されているポッドベースのアプローチです。ただし、ポッドレベルでマーキングする代わりの例として、ネームスペースレベルでラベル/アノテーションを付けることができます。その場合、自動検出された設定オプションは、Deployment、DaemonSet、ReplicaSetなどにあるかどうかに関係なく、そのネームスペースで実行されているすべてのプロセスに適用されます。

Prometheus Native Service Discoveryを有効にする

Prometheus サービスディスカバリーは、メトリクスのためにスクレイピングするエンドポイントを見つけるための標準的な方法です。prometheus.yamlを設定してスクレイピングの仕組みを設定します。エージェントv10.5.0の時点で、SysdigはネイティブのPrometheusサービスディスカバリをサポートしており、ネイティブのPrometheusと同じようにprometheus.yamlで設定することができます。

dragent.yamlで有効にすると、新バージョンのpromscrapeは、エージェントがprocess_filterルールを使用して見つけたエンドポイントを使用するのではなく、設定したprometheus.yamlを使用してエンドポイントを見つけるようになります。新バージョンのpromscrapeは、promscrape.v2と名付けられています。

Promscrape V2

- promscrape.v2は、metric relabel configsに加えて、Prometheusネイティブのrelabel configをサポートしています。relabel設定では、以下のことが可能になります。
    - ラベルをスクレイピングする前にターゲットのラベルフォーマットを編集する
    - 不要なメトリクスや不要なラベルをメトリクスから削除する
- promscrape.v2は通常のサンプルフォーマット(メトリクス名、ラベル、メトリクスの読み込み)に加えて、エージェントに送られた全てのサンプルにメトリクスタイプ(カウンター、ゲージ、ヒストグラム、サマリー)を含みます。
- promscrape.v2は、フェデレーション、ブラックボックスエクスポータなど、あらゆるタイプのスクレイピング設定をサポートしています。
- メトリクスは、特定のPrometheusラベル名を既知のエージェントタグにマッピングするソースラベルを使用して、そのソース(ポッド、プロセス)にマッピングすることができます。

Promscrape V2の制限事項

- promscrape.v2 はcalculatedメトリクスをサポートしていません。
- promscrape.v2 は、レコーディングルールやアラート管理などのクラスタ全体の機能をサポートしていません。
- promscrape と promscrape.v2 のサービスディスカバリーの設定は互換性がなく、変換には使えません。
- promscrape.v2 を有効にすると、エージェントを実行しているすべてのノードで実行され、prometheus.yaml ファイルで指定されたローカルまたはリモートのターゲットからメトリクスを収集することを目的としています。

- prometheus.yaml はすべての promscrape インスタンスで共有されます。リモートターゲットをスクレイプするように promscrape を設定しても意味がありません。
- promscrape.v2はクラスタビューを持っていないため、クラスタ全体のメトリクス収集で使用されるレコーディングルールやアラートの設定を無視しています。
- そのため、以下のPrometheusの設定はサポートされていません。
    - alert_relabel_configs
    - alertmanager_config
    - Recording Rules
    - remote_write
    - remote_read
- Sysdig は __HOSTNAME__ を使用していますが、これは Prometheus の標準的なキーワードではありません。

Promscrape V2を有効にする

Prometheusネイティブサービスディスカバリーを有効にするには

1. dragent.yaml ファイルを開きます。
2. 以下の Prometheus Service Discovery パラメータを true に設定します。

    ```
    prometheus:
      prom_service_discovery: true
    ```

    true の場合、promscrape.v2 が使用されます。そうでなければ、promscrapeがターゲットをスクレイプするために使用されます。

3. エージェントを再起動します。

デフォルトのPrometheus設定ファイル

ここにデフォルトの prometheus.yaml ファイルがあります。

```
global:
  scrape_interval: 10s
scrape_configs:
- job_name: 'k8s-pods'
  tls_config:
    insecure_skip_verify: true
  kubernetes_sd_configs:
  - role: pod
  relabel_configs:
    # Trying to ensure we only scrape local targets
    # __HOSTIPS__ is replaced by promscrape with a regex list of the IP addresses
```

```
    # of all the active network interfaces on the host
  - action: keep
    source_labels: [  meta_kubernetes_pod_host_ip]
    regex:   HOSTIPS__
  - action: keep
    source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
    regex: true
  - action: replace
    source_labels: [  meta_kubernetes_pod_annotation_prometheus_io_scheme]
    target_label:  _scheme__
    regex: (https?)
  - action: replace
    source_labels: [  meta_kubernetes_pod_annotation_prometheus_io_path]
    target_label: __metrics_path__
    regex: (.+)
  - action: replace
    source_labels: [  address  , __meta_kubernetes_pod_annotation_prometheus_io_port]
    regex: ([^:]+)(?::\d+)?;(\d+)
    replacement: $1:$2
    target_label:   address
    # Holding on to pod-id and container name so we can associate the metrics
    # with the container (and cluster hierarchy)
  - action: replace
    source_labels: [  meta_kubernetes_pod_uid]
    target_label: sysdig_k8s_pod_uid
  - action: replace
    source_labels: [  meta_kubernetes_pod_container_name]
    target_label: sysdig_k8s_pod_container_name
```

Prometheusの設定ファイルには、ローカルノード上で稼働しているポッドをスクレイピングするためのデフォルト設定が付属しています。この設定には、Kubernetes State MetricsやSysdigネイティブメトリクスとの相関性を高めるために、ポッドのUIDとコンテナ名のラベルを保存するルールも含まれています。

スクレーピング間隔

デフォルトのスクレイピング間隔は10秒です。しかし、この値はスクレイピングジョブごとに上書きすることができます。prometheus.yamlで設定されたスクレイピング間隔は、エージェントの設定に依存しません。

promscrape.v2 は prometheus.yaml を読み込み、スクレイピングジョブを開始します。

ターゲットからのメトリクスは、各ターゲットのスクレイプ間隔ごとに収集され、すぐにエージェントに転送されます。エージェントは10秒ごとにメトリクスをSysdigコレクタに送信します。最後の送信以降に受信したメトリクスのみがコレクターに送信されます。あるジョブのスクレイピングジョブ

のスクレイピング間隔が10秒よりも長い場合、エージェントの送信にはそのジョブからのすべてのメトリクスが含まれていない可能性があります。

ホスト名の選択

`__HOSTIPS__`はホストIPアドレスに置き換えられます。ホストIPアドレスによる選択が信頼性の点で好ましいです。

`__HOSTIPS__` は promscrape がターゲットのスクレイピングを開始する前に実際のホスト名に置き換えられます。これにより、promscrape は他のホスト上で実行されているターゲットを無視することができます。

リラベリング設定

デフォルトのPrometheus設定ファイルには、以下の2つのリラベル設定が含まれています。

```
- action: replace
  source_labels: [  meta_kubernetes_pod_uid]
  target_label: sysdig_k8s_pod_uid
- action: replace
  source_labels: [  meta_kubernetes_pod_container_name]
  target_label: sysdig_k8s_pod_container_name
```

これらのルールは、sysdig_k8s_pod_uid と sysdig_k8s_pod_container_name という 2 つのラベルを、それぞれポッド ID とコンテナ名を含むローカルターゲットから収集されたすべてのメトリクスに追加します。これらのラベルは、さらなる処理のためにSysdigコレクタに送信する前にメトリクスから削除されます。

Prometheusメトリクスコレクションの制限の適用

Sysdigは、処理および保存されるPrometheusメトリクスの数に制限を適用します。 したがって、すべての時系列データがSysdigモニターUIに表示されるわけではありません。 指定された制限を超えるデータは、エージェントによって破棄されます。

データ収集に制限を課すと、メトリクスの集計に必要なディスク容量や時間などのリソース使用量を削減できます。 これらの制限の実施は、2つの異なるフェーズで発生します。

- Prometheusメトリクスのフィルタリング

- Prometheusエンドポイントをスクレイピングした直後

Prometheusメトリクスのフィルタリング

Sysdigエージェント9.8.0以降、軽量のPrometheusサーバーがpromscrapeという名前のエージェントに組み込まれ、prometheus.yamlファイルが設定ファイルの一部として組み込まれています。 Sysdigは、オープンソースのPrometheus機能を使用して、Prometheusの機能を利用して、取り込み前にソースでPrometheusメトリクスをフィルタリングできます。 そのためには、次のことを行います。

- dragent.yamlファイルでPrometheusスクレイピングが有効になっていることを確認します。

  ```
  prometheus:
    enabled: true
  ```

- エージェントv9.8.0以降では、dragent.yamlでuse_promscrapeパラメータをtrueに設定して機能を有効にします。 取り込み時にフィルタリングを有効にするをご覧ください。 エージェントv10.0では、メトリクスのスクレイピングにpromscrapeがデフォルトで使用されます。
- prometheus.yamlファイルの設定を編集します。 Prometheus設定ファイルの編集を参照してください。

  Sysdig固有の設定は、prometheus.yamlファイルにあります。

取り込み時にフィルタリングを有効にする

エージェントv9.8.0では、ターゲットフィルタリングを機能させるために、dragent.yamlのuse_promscrapeパラメータをtrueに設定する必要があります。構成の詳細については、「Sysdigエージェントの設定」を参照してください。

```
use_promscrape: true
```

注意

エージェントv10.0では、`use_promscrape`がデフォルトで有効になっています。つまり、promscrapeはPrometheusメトリクスのスクレイピングに使用されます。

フィルタリング設定はオプションです。 prometheus.yamlがなくても、エージェントの既存の動作は変わりません。

Prometheus設定ファイルの編集

Prometheus設定ファイルについて

prometheus.yamlファイルには、ほとんどの場合、ターゲットプロセスの属性を表すキーと値のペアのリストにフィルタリング/再ラベル付けの構成が含まれています。

キーと値を、環境に対応する必要なタグに置き換えます。

このファイルでは、以下を設定します。

- デフォルトのスクレイプ間隔（オプション）

  例：scrape_interval：10s

- Prometheusが提供するラベル付けパラメーターのうち、Sysdigがサポートするのは metric_relabel_configsのみです。 relabel_configパラメーターはサポートされていません。
- 0個以上のプロセス固有のフィルタリング設定（オプション）

  Kubernetes環境とDocker環境をご覧ください

  フィルタリング設定には以下が含まれます。

  - フィルタリングルール

    例：`- source_labels: [container_label_io_kubernetes_pod_name]`

  - スクレイプサンプル数の制限（オプション）

    例：`sample_limit: 2000`

- デフォルトのフィルタリング設定（オプション）。デフォルト設定を参照してください。

  フィルタリング設定には以下が含まれます。

  - フィルタリングルール

例：`- source_labels: [car]`

- かき取りサンプル数の制限（オプション）

  例：`sample_limit: 2000`

prometheus.yamlファイルは、dragent.yamlと一緒にインストールされます。ほとんどの場合、prometheus.yamlの構文は標準のPrometheus設定に準拠しています。

デフォルトの設定

キーと値のペアが空の設定は、デフォルト設定と見なされます。デフォルトの設定は、一致するフィルタリング設定を持たない、スクレイピングされるすべてのプロセスに適用されます。サンプルPrometheus設定ファイルでは、`job_name: 'default'`セクションはデフォルトの設定を表しています。

Kubernetes環境

エージェントがKubernetes環境（オープンソース/ OpenShift / GKE）で実行されている場合は、次のKubernetesオブジェクトをキーと値のペアとして含めます。エージェントのインストールの詳細については、エージェントのインストール：Kubernetesをご覧ください。

例；

```
sysdig sd_configs:
- tags:
    namespace: backend
    deployment: my-api
```

前述のタグに加えて、これらのオブジェクトタイプのいずれかを照合できます：

```
daemonset: my daemon
deployment: my_deployment
hpa: my hpa
namespace: my_namespace
node: my node
pod: my pode
replicaset: my replica
replicationcontroller: my_controller
resourcequota: my quota
service: my service
stateful: my_statefulset
```

Kubernetes / OpenShift / GKEデプロイメントの場合、prometheus.yamlはdragent.yamlと同じConfigMapを共有します。

Docker環境

Docker環境では、コンテナ、ホスト、ポートなどの属性を含めます。 例えば：

```
sysdig sd_configs:
- tags:
    host: my-host
    port: 8080
```

Dockerベースのデプロイメントの場合、prometheus.yamlをホストからマウントできます。

Prometheus設定ファイルのサンプル

```
global:
  scrape interval: 20s
scrape configs:
- job name: 'default'
  sysdig sd configs: # default config
  relabel configs:
- job name: 'my-app-job'
  sample limit: 2000
  sysdig sd_configs:  # apply this filtering config only to my-app
  - tags:
      namespace: backend
      deployment: my-app
  metric relabel configs:
  # Drop all metrics starting with http_
  - source labels: [  name__]
    regex: "http_(.+)"
    action: drop
  metric relabel configs:
  # Drop all metrics for which the city label equals atlantis
  - source labels: [city]
    regex: "atlantis"
    action: drop
```

Prometheusメトリクスコレクションの制限

メトリクスの制限はSysdigバックエンドによって指示され、強制された制限はSysdigエージェントによって行われます。さらに、エージェントは、メトリクスストアに送信されたPrometheusメトリクスエンドポイントから読み込まれたメトリクスの数にも制限を課します。管理者として、値がSysdigのメ

トリクス制限を超えないことを条件に、エージェントの dragent.yaml ファイルを設定することで制限をオーバーライドすることができます。

メトリクス制限

エージェントv10.0.0以降、エージェントごとの新しいメトリクス制限は次のとおりです。

- カスタムメトリクス：10,000
- Prometheusメトリクス：8000

  他のカスタムメトリクスの制限をゼロに設定して、Prometheusメトリクスの制限を10,000に増やせます。

エージェントの設定

Sysdigが処理および保存するPrometheusメトリクスの数の制限は、dragent.yamlファイルの特定のパラメーターによって制御できます。以下に、関連する設定とデフォルトを示します。 値の変更は、エージェントを再起動した後に有効になります。

```
prometheus:
  max tags per metric: 20
  max metrics per process: 1000
  max_metrics: 1000
```

注意

max_tags_per_metricおよびmax_metrics_per_processパラメータは、エージェントv10.0.0で廃止されました。

max_metrics

エージェントがターゲットから取得できるPrometheusメトリクスの最大数。 デフォルトは1,000です。エージェントv10.0.0以降では、上限は10,000です。 10.0.0未満のエージェントバージョンでは、上限は3,000です。

max_metrics_per_process

このパラメーターは、エージェントv10.0.0では非推奨です。

エージェントが単一のプロセスから読み取ることができるPrometheusメトリクスの最大数。 デフォルトは-1（無限大）です。 制限は、max_metricsの値によって課されます。

max_tags_per_metric

このパラメーターは、エージェントv10.0.0では非推奨です。

これは、エージェントが単一のスクレイピングターゲットから保存するPrometheusメトリクスの最大数です。

設定例

このトピックでは、デフォルトおよび特定のPrometheus設定を紹介します。

デフォルトの設定

上記の設定要素の多くを組み合わせた例として、dragent.default.yamlから継承されたデフォルトのエージェント設定を考えます。

```
prometheus:
  enabled: true
  interval: 10
  log errors: true
  max metrics: 1000
  max metrics per process: 100
  max_tags_per_metric: 20

  # Filtering processes to scan. Processes not matching a rule will not
  # be scanned
  # If an include rule doesn't contain a port or port filter in the conf
  # section, we will scan all the ports that a matching process is listening to.
  process filter:
    - exclude:
        process.name: docker-proxy
    - exclude:
        container.image: sysdig/agent
    # special rule to exclude processes matching configured prometheus appcheck
    - exclude:
        appcheck.match: prometheus
    - include:
        container.label.io.prometheus.scrape: "true"
        conf:
            # Custom path definition
            # If the Label doesn't exist we'll still use "/metrics"
            path: "{container.label.io.prometheus.path}"
```

```
        # Port definition
        # - If the Label exists, only scan the given port.
        # - If it doesn't, use port filter instead.
        # - If there is no port filter defined, skip this process
        port: "{container.label.io.prometheus.port}"
        port filter:
            - exclude: [9092,9200,9300]
            - include: 9090-9500
            - include: [9913,9984,24231,42004]
- exclude:
    container.label.io.prometheus.scrape: "false"
- include:
    kubernetes.pod.annotation.prometheus.io/scrape: true
    conf:
        path: "{kubernetes.pod.annotation.prometheus.io/path}"
        port: "{kubernetes.pod.annotation.prometheus.io/port}"
- exclude:
    kubernetes.pod.annotation.prometheus.io/scrape: false
```

このデフォルト設定については、次の点を考慮してください。

- すべてのPrometheusスクレイピングはデフォルトで無効になっています。ここに示す設定全体を有効にするには、dragent.yamlに以下を追加するだけです。

  ```
  prometheus:
    enabled: true
  ```

  このオプションを有効にすると、適切なアノテーションが設定されたポッド（Kubernetesの場合）またはラベルが設定されたコンテナ（そうでない場合）が自動的に破棄されます。

- 有効にすると、このデフォルト設定は、Kubernetes環境のクイックスタートで説明されているユースケースに最適です。
- プロセスフィルタールールは、ほとんどの環境に存在する可能性が高いが、Dockerプロキシやエージェント自体などのPrometheusメトリクスをエクスポートしないことがわかっているプロセスを除外します。
- 別のプロセスフィルタールールは、レガシーPrometheusアプリケーションチェックによってスクレイピングするように設定されたプロセスがスクレイピングされないようにします。
- 別のプロセスフィルタールールは、コンテナラベルを使用するように調整されています。コンテナラベル`io.prometheus.scrape`でマークされたプロセスはスクレイピングの対象となり、さらにコンテナラベル`io.prometheus.port`や`io.prometheus.path`でマークされた場合、このポートまたはエンドポイント、あるいはその両方でのみスクレイピングが試行されます。コンテナが指定されたパスラベルでマークされていない場合、`/metrics`エンドポイントのスクレイ

ピングが試行されます。コンテナが指定されたポートラベルでマークされていない場合、`port_filter`のリスニングポートはスクレイピングが試行されます（デフォルトのこの`port_filter`は、一般的なPrometheusエクスポーターのポートの範囲に対して設定されます。エクスポーターではない他のアプリケーションで使用されることがわかっています）。

- 最後のプロセスフィルターIncludeルールは、Kubernetes環境のクイックスタートで説明されているユースケースに合わせて調整されています。

単一のカスタムプロセスをスクレイプする

単一のカスタムプロセス、たとえば、パス/prometheusでポート9000をリッスンするJavaプロセスをスクレイピングする必要がある場合は、drangent.yamlに以下を追加します。

```
prometheus:
  enabled: true
  process filter:
    - include:
        process.name: java
        port: 9000
        conf:
          # ensure we only scrape port 9000 as opposed to all ports this process may be listening to
          port: 9000
          path: "/prometheus"
```

この設定は、デフォルト設定に示されているデフォルトのprocess_filterセクションをオーバーライドします。 関連するルールをデフォルト設定からこれに追加して、メトリクスをさらにフィルタリングできます。

留意事項

ポートは、設定のどこに置かれるかによって異なる目的を持っています。include セクションの下に置かれた場合、それは include ルールにマッチするための条件となります。

ポートをconfの下に置くことは、プロセスがリッスンしている可能性のあるすべてのポートとは対照的に、ルールがマッチしたときに、その特定のポートだけがスクレイプされることを示しています。

この例では、最初のルールは、ポート 9000 でリッスンしている Java プロセスにマッチします。9000 番ポートのみをリッスンしている Java プロセスは、スクレイプされます。

コンテナラベルに基づいて単一のカスタムプロセスをスクレイプする

それでもコンテナーラベルに基づいてスクレイピングする場合は、関連するルールをデフォルトからprocess_filterに追加します。 例えば：

```
prometheus:
  enabled: true
  process filter:
    - include:
        process.name: java
        port: 9000
        conf:
          # ensure we only scrape port 9000 as opposed to all ports this process may be listening to
          port: 9000
          path: "/prometheus"
    - exclude:
        process.name: docker-proxy
    - include:
        container.label.io.prometheus.scrape: "true"
        conf:
            path: "{container.label.io.prometheus.path}"
            port: "{container.label.io.prometheus.port}"
```

留意事項

ポートは、設定のどこに置かれるかによって異なる意味を持ちます。include セクションの下に置かれた場合は、 include ルールにマッチするための条件です。

ポートを conf の下に置くと、プロセスが listen している可能性のあるすべてのポートとは対照的に、ルールがマッチしたときにそのポートだけがスクレイプされることを示します。

この例では、ポート 9000 をリッスンしているプロセスに対して最初のルールが一致します。9000 番ポートのみをリッスンしている java プロセスは、スクラッ プされます。

コンテナ環境

このデフォルト設定を有効にすると、以下に示すエクスポーターのコンテナ化されたインストールは、エージェントを介して自動的にスクレイピングされます。

```
# docker run -d -p 8080:8080 \
    --label io.prometheus.scrape="true" \
    --label io.prometheus.port="8080" \
    --label io.prometheus.path="/prometheus" \
    luca3m/prometheus-java-app
```

Kubernetes環境

Kubernetesベースの環境では、このYAMLの例に示すように、アノテーションを使用したデプロイメントは、デフォルトの設定を有効にすることでスクレイプされます。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: prometheus-java-app
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: prometheus-java-app
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/path: "/prometheus"
        prometheus.io/port: "8080"
    spec:
      containers:
        - name: prometheus-java-app
          image: luca3m/prometheus-java-app
          imagePullPolicy: Always
```

非コンテナ環境

これは、コンテナ化されていない環境、またはラベルやアノテーションを使用しないコンテナ化された環境の例です。次のdragent.yamlはデフォルトを上書きし、サンプルエクスポーターと2番目のエクスポーターをそれぞれ500番ポートで1秒ごとにスクレイピングします。 これは控えめな「ホワイトリスト」タイプの設定と考えることができます。これは、環境内に存在することがわかっているエクスポーターと、Prometheusメトリクススをエクスポートすることがわかっているポートのみにスクレイピングを制限するためです。

```
prometheus:
  enabled: true
  interval: 1
  process filter:
    - include:
        process.cmdline: "*app.jar*"
        conf:
          port: 8080
          path: "/prometheus"
    - include:
        port: 5005
        conf:
          port: 5005
          path: "/wacko"
```

留意事項

ポートは、設定のどこに置かれるかによって異なる意味を持ちます。include セクションの下に置かれた場合は、 include ルールにマッチするための条件です。ポートを conf の下に配置すると、プロセスがリッスンしている可能性のあるすべてのポートとは対照的に、ルールがマッチしたときにそのポートだけがスクレイプされることを示します。

この例では、最初のルールはプロセス *app.jar* にマッチします。ポート 8080でのみリスニングしている java プロセスは、*app.jar* がリスニングしている可能性のあるすべてのポートとは対照的に、 スクレイプされます。2 番目のルールは 5005 番ポートにマッチし、5005 番ポートでのみリスニングしているプロセスがスクレイプされます。

ロギングとトラブルシューティング

ロギング

エージェントがPrometheusメトリクスのスクレイピングを開始してから、Sysdig Monitorでメトリクスが表示されるまでに数分程度の遅延が生じる場合があります。設定が正しいかどうかを素早く確認するために、Agentバージョン0.80.0で起動すると、起動してから初めて以下のようなログ行がAgentログに表示され、少なくとも1つのPrometheus exporterを見つけてスクレイピングに成功していることがわかります。

```
2018-05-04 21:42:10.048, 8820, Information, 05-04 21:42:10.048324 Starting export of Prometheus metrics
```

これはINFOレベルのログメッセージなので、デフォルトのロギング設定を使用してAgentに表示されます。さらに詳細を明らかにするには、Agentのログレベルを DEBUGに上げると、最初に検出された特定のメトリクスの名前を明らかにする以下のようなメッセージが表示されます。その後、まもなくSysdig Monitorでこのメトリクスが表示されることを探すことができます。

```
2018-05-04 21:50:46.068, 11212, Debug, 05-04 21:50:46.068141 First prometheus metrics since agent start: pid 9583: 5 metrics including: randomSummary.95percentile
```

トラブルシューティング

スクレイピング成功時に予想されるログメッセージについては、前のセクションを参照してください。Prometheusを有効にしていて、Starting exportメッセージが表示されない場合は、設定を再検討してください。

また、設定オプションのデフォルト設定である log_errors: true のままにしておくことをお勧めします。

例えば、以下はTCPポートのスクレイプに失敗した場合のエラーメッセージです。

```
2017-10-13 22:00:12.076, 4984, Error, sdchecks[4987] Exception on running check prometheus.5000:
Exception('Timeout when hitting http://localhost:5000/metrics',)
2017-10-13 22:00:12.076, 4984, Error, sdchecks, Traceback (most recent call last):
2017-10-13 22:00:12.076, 4984, Error, sdchecks, File "/opt/draios/lib/python/sdchecks.py", line 246,
in run
2017-10-13 22:00:12.076, 4984, Error, sdchecks, self.check instance.check(self.instance conf)
2017-10-13 22:00:12.076, 4984, Error, sdchecks, File "/opt/draios/lib/python/checks.d/prometheus.py",
line 44, in check
2017-10-13 22:00:12.076, 4984, Error, sdchecks, metrics = self.get_prometheus_metrics(query_url,
timeout, "prometheus")
2017-10-13 22:00:12.076, 4984, Error, sdchecks, File "/opt/draios/lib/python/checks.d/prometheus.py",
line 105, in get prometheus metrics
2017-10-13 22:00:12.077, 4984, Error, sdchecks, raise Exception("Timeout when hitting %s" % url)
2017-10-13 22:00:12.077, 4984, Error, sdchecks, Exception: Timeout when hitting
http://localhost:5000/metrics
```

以下は、/metrics エンドポイントの HTTP 要求には応答していたが、有効な Prometheus 形式のデータでは応答していなかったポートのスクレイプに失敗した場合のエラーメッセージの例です。無効なエンドポイントは以下のように応答しています。

```
# curl http://localhost:5002/metrics
This ain't no Prometheus metrics!
```

エージェントログには、最初の失敗の後、それ以上のスクレイピングが試みられないことを示す、対応するエラーメッセージが表示されます。

```
2017-10-13 22:03:05.081, 5216, Information, sdchecks[5219] Skip retries for Prometheus error: could
not convert string to float: ain't
2017-10-13 22:03:05.082, 5216, Error, sdchecks[5219] Exception on running check prometheus.5002:
could not convert string to float: ain't
```

リモートホストからのPrometheusメトリクスの収集

Sysdig Monitorは、最小限の設定でリモートエンドポイントからPrometheusメトリクスを収集できます。リモートエンドポイント（リモートホスト）は、Sysdigエージェントをデプロイできないホストを指します。たとえば、ユーザーワークロードをデプロイできないGKEやEKSなどのマネージドKubernetesサービス上のKubernetesマスターノードは、エージェントが関与していないことを意味します。このようなホストでリモートスクレイピングを有効にするのは、エージェントを識別してスクレイピングを実行し、エージェント設定ファイルのリモートサービスセクションでエンドポイント設定を宣言するだけです。

収集されたPrometheusメトリクスは、それらをプロセスに関連付けるのではなく、スクレイピングを実行したエージェントの下に報告され、関連付けられます。

設定ファイルの準備

複数のエージェントが同じ設定を共有できます。したがって、dragent.yamlファイルを使用して、これらのエージェントのどれがリモートエンドポイントをスクレイピングするかを決定します。これは両方に適用されます

- prometheus設定の下のエージェント設定ファイルに、リモートサービス用の個別の設定セクションを作成します。
- 各リモートエンドポイントの設定セクションを含め、URLまたはホスト/ポート（およびオプションのパス）パラメーターを各セクションに追加して、スクレイピングするエンドポイントを識別します。オプションのパスは、エンドポイントでリソースを識別します。空のパスパラメータは、スクレイピング用の`"/metrics"`エンドポイントにデフォルト設定されます。
- 必要に応じて、リモートサービスのエンドポイント設定ごとにカスタムタグを追加します。タグがない場合、複数のエンドポイントが関係していると、メトリクスレポートが期待どおりに機能しない可能性があります。エージェントは、タグによって一意に識別されない限り、複数のエンドポイントからスクレイピングされた類似のメトリクスを区別できません。

参考となる、Kubernetesの設定例を以下に示します。

```
prometheus:
  remote_services:
        - prom_1:
            kubernetes.node.annotation.sysdig.com/region: europe
            kubernetes.node.annotation.sysdig.com/scraper: true
            conf:
                url: "https://xx.xxx.xxx.xy:5005/metrics"
                tags:
```

```
            host: xx.xxx.xxx.xy
            service: prom 1
            scraping_node: "{kubernetes.node.name}"
    - prom 2:
        kubernetes.node.annotation.sysdig.com/region: india
        kubernetes.node.annotation.sysdig.com/scraper: true
        conf:
            host: xx.xxx.xxx.yx
            port: 5005
            use https: true
            tags:
                host: xx.xxx.xxx.yx
                service: prom 2
                scraping_node: "{kubernetes.node.name}"
    - prom 3:
        kubernetes.pod.annotation.sysdig.com/prom_3_scraper: true
        conf:
            url: "{kubernetes.pod.annotation.sysdig.com/prom_3_url}"
            tags:
                service: prom 3
                scraping_node: "{kubernetes.node.name}"
    - haproxy:
        kubernetes.node.annotation.yourhost.com/haproxy_scraper: true
        conf:
            host: "mymasternode"
            port: 1936
            path: "/metrics"
            username: "{kubernetes.node.annotation.yourhost.com/haproxy username}"
            password: "{kubernetes.node.annotation.yourhost.com/haproxy_password}"
            tags:
                service: router
```

上記の例では、ノードとポッドアノテーションによってスクレイピングがトリガーされます。次のようにkubectl annotateコマンドを使用して、ノードとポッドにアノテーションを追加できます。

```
kubectl annotate node mynode --overwrite sysdig.com/region=india sysdig.com/scraper=true
haproxy_scraper=true yourhost.com/haproxy_username=admin yourhost.com/haproxy_password=admin
```

この例では、ノードにアノテーションを設定して、上記の設定で定義されているように、prom2およびhaproxyサービスのスクレイピングをトリガーします。

コンテナ環境の準備

Docker環境の設定例を以下に示します。

```
prometheus:
  remote services:
        - prom container:
            container.label.com.sysdig.scrape_xyz: true
            conf:
```

```
        url: "https://xyz:5005/metrics"
        tags:
            host: xyz
            service: xyz
```

Dockerベースのコンテナ環境でリモートスクレイピングを機能させるには、`com.sysdig.scrape_xyz=true`ラベルをエージェントコンテナに設定します。 例えば：

```
docker run -d --name sysdig-agent --restart always --privileged --net host --pid host -e
ACCESS KEY=<KEY> -e COLLECTOR=<COLLECTOR> -e SECURE=true -e TAGS=example tag:example value -v
/var/run/docker.sock:/host/var/run/docker.sock -v /dev:/host/dev -v /proc:/host/proc:ro -v
/boot:/host/boot:ro -v /lib/modules:/host/lib/modules:ro -v /usr:/host/usr:ro --shm-size=512m
sysdig/agent
```

<`KEY`>, <`COLLECTOR`>, `TAGS`をそれぞれアカウントキー、コレクター、タグに置き換えます。

ルールの構文

`remote_services`のルールの構文は、include/excludeルールを除いて、process_filterのルールとほぼ同じです。 `remote_services`セクションは、include/excludeルールを使用しません。 `process_filter`は、プロセスに対する最初の一致のみが適用されるルールを含めたり除外したりしますが、`remote_services`セクションでは、各ルールに対応するサービス名があり、一致するすべてのルールが適用されます。

ルール条件

ルール条件は、`process_filter`の条件と同じように機能します。唯一の注意点は、リモートプロセス/コンテキストが不明であるため、ルールがエージェントプロセスおよびコンテナに対して照合されることです。したがって、コンテナのラベルとアノテーションの一致は以前と同じように機能しますが、エージェントコンテナにも適用できる必要があります。たとえば、エージェントコンテナはノードで実行されるため、ノードアノテーションが適用されます。

アノテーションの場合、単一のルールで複数のパターンを指定できます。その場合、ルールが一致するためには、すべてのパターンが一致する必要があります（AND演算子）。次の例では、両方の注釈が一致しない限り、エンドポイントは考慮されません。

```
kubernetes.node.annotation.sysdig.com/region scraper: europe
kubernetes.node.annotation.sysdig.com/scraper: true
```

つまり、ヨーロッパ地域のみに属するKubernetesノードはスクレイピングの対象と見なされます。

Sysdigエージェントの認証

Sysdigエージェントは、メトリクスを収集するためにリモートホストで必要な権限を必要とします。ローカルスクレイピングの認証方法は、リモートホスト上のエージェントの認証にも機能しますが、許可パラメータはエージェントコンテキストでのみ機能します。

- 証明書とキーのペアに基づく認証では、Kubernetesシークレットに構築してエージェントにマウントする必要があります。
- トークンベースの認証では、エージェントトークンがリモートエンドポイントでスクレイピングを行うためのアクセス権を持っていることを確認してください。
- プレーンテキストで渡すのではなく、アノテーションを使用してユーザー名/パスワードを取得します。中括弧で囲まれたアノテーションは、アノテーションの値に置き換えられます。アノテーションが存在しない場合、値は空の文字列になります。トークン置換は、すべての許可パラメーターでサポートされています。認証はエージェントコンテキストでのみ機能するため、認証情報をターゲットポッドから自動的に取得することはできません。したがって、エージェントポッドのアノテーションを使用してそれらを渡します。これを行うには、選択したKubernetesオブジェクトのアノテーションにパスワードを設定します。

次の例では、HAProxyアカウントは、エージェントノードのyourhost.com/haproxy_passwordアノテーションで指定されたパスワードで認証されます。

```
- haproxy:
            kubernetes.node.annotation.yourhost.com/haproxy_scraper: true
            conf:
                host: "mymasternode"
                port: 1936
                path: "/metrics"
                username: "{kubernetes.node.annotation.yourhost.com/haproxy username}"
                password: "{kubernetes.node.annotation.yourhost.com/haproxy_password}"
                tags:
                    service: router
```

GrafanaのSysdigデータソースを設定する

Sysdigを使用すると、GrafanaユーザーはSysdigからメトリクスを照会し、Grafanaダッシュボードでそれらを視覚化できます。 SysdigをGrafanaと統合するには、データソースを構成します。サポートされるデータソースには2つのタイプがあります。

- Prometheus

  PrometheusデータソースはGrafanaに付属しており、PromQLとネイティブ互換です。 Sysdigは、Proftheus互換のAPIを提供して、GrafanaとのAPIのみの統合を実現します。

  - Grafanaバージョン6.7.0以降の場合、Grafana v6.7以降でのPrometheus APIの使用に記載されているようにデータソースを設定します。
  - 6.7.0より前のバージョンのGrafanaの場合、Grafana v6.6以下でのGrafana APIの使用に記載されているようにデータソースを設定します。
- Sysdig

  Sysdigデータソースには追加の設定が必要であり、単純な“form-based”のデータ設定との互換性が高くなります。Prometheus APIの代わりにSysdigネイティブAPIを使用します。詳細については、Sysdig Grafanaデータソースを参照してください。

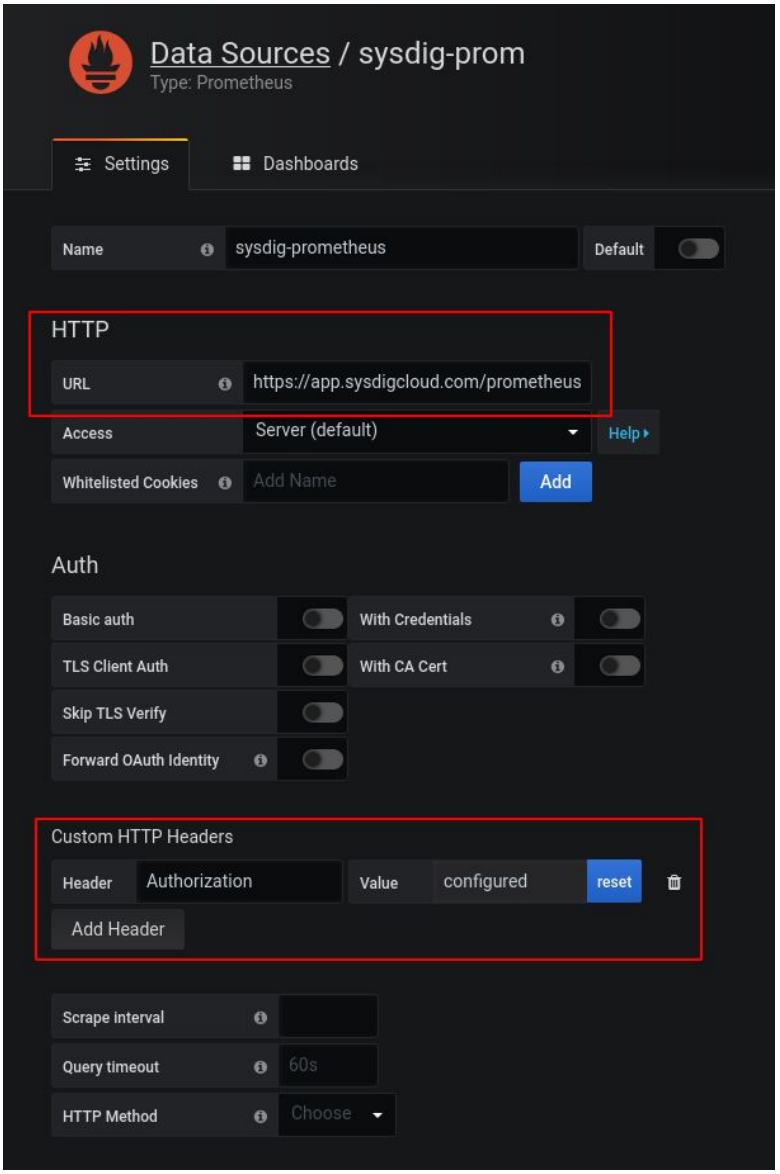
Grafana v6.7以降でのPrometheus APIの使用

Sysdig Prometheus API を使用して、Grafana で使用するデータソースを設定します。GrafanaがSysdigのメトリクスを消費する前に、Grafanaは自身をSysdigに認証する必要があります。そのためには、現在GrafanaではUIのサポートがないため、Sysdig API Tokenを使用してHTTP認証を設定する必要があります。

1. Grafanaを使用していない場合は、次のようにGrafanaコンテナを起動します。

   ```
   $ docker run --rm -p 3000:3000 --name grafana grafana/grafana
   ```

2. Grafanaに管理者としてログインし、次の情報を使用して新しいデータソースを作成します。



- **URL**: `https://<Monitor URL for Your Region>/prometheus`
  SaaS リージョンと IP レンジを参照して、Sysdig アプリケーションとリージョンに関連付けられた正しい URL を特定してください。
- **Authentication**: 認証メカニズムを選択しないでください。
- **Access**: Server (default)
- **Custom HTTP Headers**:
  - **Header**: Authorization
  - **Value**: Bearer *<Your Sysdig API Token>*
    API トークンは **Settings > User Profile > Sysdig Monitor API**.

Grafana v6.6以下でのGrafana APIの使用

Grafana APIを使用して、Sysdigデータソースを設定します。

1. Grafanaをダウンロードしてコンテナで実行します。

```
docker run --rm -p 3000:3000 --name grafana grafana/grafana
```

2. JSONファイルを作成します。

```
cat grafana-stg-ds.json
{
    "name": "Sysdig staging PromQL",
    "orgId": 1,
    "type": "prometheus",
    "access": "proxy",
    "url": "https://app-staging.sysdigcloud.com/prometheus",
    "basicAuth": false,
    "withCredentials": false,
    "isDefault": false,
    "editable": true,
    "jsonData": {
        "httpHeaderName1": "Authorization",
        "tlsSkipVerify": true
    },
    "secureJsonData": {
        "httpHeaderValue1": "Bearer your-Sysdig-API-token"
    }
}
```

3. Sysdig APIトークンを取得して、上記のJSONファイルにプラグインします。

```
“httpHeaderValue1”: “Bearer your_Sysdig_API_Token”
```

4. データソースをGrafanaに追加します。

```
curl -u admin:admin -H "Content-Type: application/json" http://localhost:3000/api/datasources
-XPOST -d @grafana-stg-ds.json
```

5. Grafanaを実行します。

```
http://localhost:3000
```

6. デフォルトの資格情報admin：adminを使用して、Grafanaにサインインします。

7. Grafanaの[Configuration]の下にある[Data Source]タブを開き、追加したものがページに表示されていることを確認します。