記事
Toshihiko Minamoto · 2020年11月12日 15m read

Prometheus で InterSystems Caché を監視する

Prometheus は時系列データの収集に適した監視システムです。

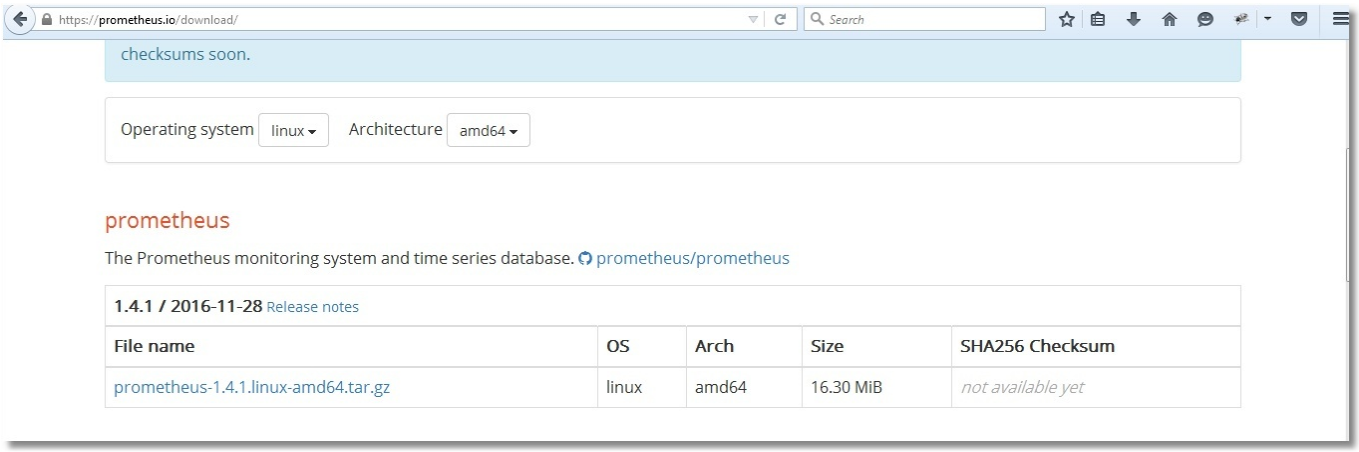
このシステムのインストールと初期構成は比較的簡単です。 このシステムにはデータ視覚化用の PromDash と呼ばれる画像サブシステムが組み込まれていますが、開発者は Grafana と呼ばれる無料のサードパーティ製品を使用することを推奨しています。 Prometheus は多くの要素（ハードウェア、コンテナ、さまざまな DBMS の構成要素）を監視できますが、この記事では Caché インスタンス（正確に言えば Ensemble インスタンスですが、メトリックは Caché からのものになります）の監視に注目したいと思います。 ご興味があれば、このまま読み進めてください。

非常に単純なケースでは、Prometheus と Caché は単一のマシン（Fedora Workstation 24 x8664）上に存在します。 Caché のバージョンは以下のとおりです。

```
 %SYS>write $zv
Cache for UNIX (Red Hat Enterprise Linux for x86-64) 2016.1 (Build 656U) Fri Mar 11 2016 17:58:47 EST
```

インストールと構成

公式サイトから適切な Prometheus の配布パッケージをダウンロードし、/opt/prometheus フォルダーに保存してください。



アーカイブを解凍し、必要に応じてテンプレート構成ファイルを変更してから Prometheus を起動します。 Prometheus はデフォルトでコンソールにログを表示するため、ここではアクティビティレコードをログファイルに保存することにします。

Prometheus の起動

```
 # pwd
/opt/prometheus
# ls
prometheus-1.4.1.linux-amd64.tar.gz
# tar -xzf prometheus-1.4.1.linux-amd64.tar.gz
# ls
prometheus-1.4.1.linux-amd64 prometheus-1.4.1.linux-amd64.tar.gz
# cd prometheus-1.4.1.linux-amd64/
# ls
```

```
consolelibraries consoles LICENSE NOTICE prometheus prometheus.yml promtool
# cat prometheus.yml
global:
  scrapeinterval: 15s # スクレイプ間隔を 15 秒に設定します。 デフォルトは 1 分ごとです。
scrapeconfigs:
  - jobname: 'isccache'
  metricspath: '/metrics/cache'
  staticconfigs:
  - targets: ['localhost:57772']
# ./prometheus > /var/log/prometheus.log 2>&1 &
[1] 7117
# head /var/log/prometheus.log
time=«2017-01-01T09:01:11+02:00» level=info msg=«Starting prometheus (version=1.4.1, branch=master,
revision=2a89e8733f240d3cd57a6520b52c36ac4744ce12)» source=«main.go:77»
time=«2017-01-01T09:01:11+02:00» level=info msg=«Build context (go=go1.7.3, user=root@e685d23d8809,
date=20161128-09:59:22)» source=«main.go:78»
time=«2017-01-01T09:01:11+02:00» level=info msg=«Loading configuration file prometheus.yml»
source=«main.go:250»
time=«2017-01-01T09:01:11+02:00» level=info msg=«Loading series map and head chunks...»
source=«storage.go:354»
time=«2017-01-01T09:01:11+02:00» level=info msg=«23 series loaded.» source=«storage.go:359»
time=«2017-01-01T09:01:11+02:00» level=info msg="Listening on :9090" source=«web.go:248»
```

prometheus.yml 構成ファイルは YAML 言語で記述されているため、タブ文字の使用は好ましくありません。したがって、スペースのみを使用する必要があります。 また、すでにお伝えしたとおり、メトリックは http://localhost:57772 からダウンロードされ、リクエストは /metrics/cache に送信されます（アプリケーション名は任意）。したがって、メトリック収集用の宛先アドレスは http://localhost:57772/metrics/cache になります。 「job = isccache」タグが各メトリックに追加されます。大まかに言えば、タグは SQL の WHERE に相当するものです。
ここではタグを使用しませんが、複数のサーバーを使用する場合は役に立つでしょう。 例えばサーバー（またはインスタンス）の名前をタグに保存し、タグを使用してグラフ描画用のリクエストをパラメーター化することができます。 では、Prometheus が動作していることを確認しましょう（上記の出力では、9090 番ポートでリッスンしていることが分かります）。

Web インターフェイスが開きます。これは、Prometheus が機能していることを意味します。 ただし、Caché のメトリックはまだ表示されていません（[Status] → [Targets] をクリックして確認しましょう）。

メトリックの準備

目標は、Prometheus が http://localhost:57772/metrics/cache で適切なフォーマットでメトリックにアクセスできるようにすることです。
ここではその単純さを考慮し、Caché の REST 機能を使用します。 Prometheus は数値メトリックのみを「理解」するため、ここでは文字列メトリックをエクスポートしません。
後者を取得するには、SYS.Stats.Dashboard クラスの API を使用します。 このようなメトリックは、Caché 自体がシステムツールバーを表示する目的で使用されています。

同じ内容をターミナルで表示した例:

```
 %SYS>set dashboard = ##class(SYS.Stats.Dashboard).Sample()

%SYS>zwrite dashboard
dashboard=
<OBJECT REFERENCE>
 [2@SYS.Stats.Dashboard]
+---------------- general information ---------------
| oref value: 2
| class name: SYS.Stats.Dashboard
| reference count: 2
+---------------- attribute values ------------------
| ApplicationErrors = 0
```

```
 CSPSessions = 2
 CacheEfficiency = 2385.33
 DatabaseSpace = "Normal"
 DiskReads = 14942
 DiskWrites = 99278
 ECPAppServer = "OK"
 ECPAppSrvRate = 0
 ECPDataServer = "OK"
 ECPDataSrvRate = 0
 GloRefs = 272452605
 GloRefsPerSec = "70.00"
 GloSets = 42330792
 JournalEntries = 16399816
 JournalSpace = "Normal"
 JournalStatus = "Normal"
 LastBackup = "Mar 26 2017 09:58AM"
 LicenseCurrent = 3
 LicenseCurrentPct = 2
...
```

ここでは USER スペースがサンドボックスになります。 まず、REST アプリケーションの /metrics を作成しましょう。 非常に基本的な安全対策を行うため、ログインをパスワードで保護し、Web アプリケーションをリソースに関連付けます。このようなリソースを PromResource と呼びましょう。 リソースへの公開アクセスを無効にするため、次の内容を実行してください。

```
 %SYS>write ##class(Security.Resources).Create("PromResource", "Resource for Metrics web page", "")
1
```

Web アプリの設定:

このリソースにアクセスできるユーザーも必要です。 このユーザーもデータベース（この場合は USER）から読み取り、そこへデータを保存できる必要があります。 また、別件ですがコードの後半部では %SYS スペースに切り替えるため、このユーザーには CACHESYS システムデータベースの読み取り権限が必要になります。
ここでは標準のスキームに従います。すなわち、これらの権限を持つ PromRole ロールを作成した後にこのロールに割り当てられた PromUser ユーザーを作成します。
パスワードには「Secret」を使いましょう。

```
 %SYS>write ##class(Security.Roles).Create("PromRole","Role for PromResource","PromResource:U,%DBUSER:RW,%DBCACHESYS:R"
1
%SYS>write ##class(Security.Users).Create("PromUser","PromRole","Secret")
1
```

Prometheus の構成では、この PromUser ユーザーを認証に使用します。 完了後はサーバープロセスに SIGNUP シグナルを送信し、構成を再読み込みします。

より安全な構成

```
 # cat /opt/prometheus/prometheus-1.4.1.linux-amd64/prometheus.yml
global:
 scrape_interval: 15s # スクレイプ間隔を 15 秒に設定します。 デフォルトは 1 分ごとです。

scrape_configs:
 - job_name: 'isc_cache'
 metrics_path: '/metrics/cache'
 static_configs:
 - targets: ['localhost:57772']
 basic_auth:
 username: 'PromUser'
 password: 'Secret'
#
# kill -SIGHUP $(pgrep prometheus) # または kill -1 $(pgrep prometheus)
```

以上で Prometheus がメトリックを含む Web
アプリケーションを使用するための認証をパスできるようになりました。

メトリックは、my.Metrics リクエスト処理クラスによって提供されます。以下にその実装を示します。

```
Class my.Metrics Extends %CSP.REST
{

Parameter ISCPREFIX = "isc_cache";

Parameter DASHPREFIX = {..#ISCPREFIX_"_dashboard"};

XData UrlMap [ XMLNamespace = "http://www.intersystems.com/urlmap" ]
{
<Routes>
<Route Url="/cache" Method="GET" Call="getMetrics"/>
</Routes>
}

/// ??? Prometheus ????????????????? ???????????????
/// https://prometheus.io/docs/instrumenting/exposition_formats/
///
/// ????????????? ?????\n??????????
/// ?????????????????????? ??????????
ClassMethod getMetrics() As %Status
{
    set nl = $c(10)
    do ..getDashboardSample(.dashboard)
    do ..getClassProperties(dashboard.%ClassName(1), .propList, .descrList)

    for i=1:1:$ll(propList) {
        set descr = $lg(descrList,i)
        set propertyName = $lg(propList,i)
        set propertyValue = $property(dashboard, propertyName)

        // Prometheus?????????????????
        // ????????????????????????????????
        // ?????????????????
        if ((propertyValue '= "") && ('$match(propertyValue, ".*[-A-Za-z ]+.*"))) {
            set metricsName = ..#DASHPREFIX_..camelCase2Underscore(propertyName)
            set metricsValue = propertyValue

            // ??????????????????
            // ???????Prometheus??????????
            // ???????1???????????????
            write "# HELP "_metricsName_" "_$replace(descr,nl," ")_nl
            write metricsName_" "_metricsValue_nl
        }
    }

    write nl
    quit $$$OK
}

ClassMethod getDashboardSample(Output dashboard)
{
    new $namespace
    set $namespace = "%SYS"
```

```
    set dashboard = ##class(SYS.Stats.Dashboard).Sample()
}

ClassMethod getClassProperties(className As %String, Output propList As %List, Output
 descrList As %List)
{
    new $namespace
    set $namespace = "%SYS"

    set propList = "", descrList = ""
    set properties = ##class(%Dictionary.ClassDefinition).%OpenId(className).Properti
es

    for i=1:1:properties.Count() {
        set property = properties.GetAt(i)
        set propList = propList_$lb(property.Name)
        set descrList = descrList_$lb(property.Description)
    }
}

/// ??????????????????????????????????
/// ?: ?? = WriteDaemon??? = _write_daemon
ClassMethod camelCase2Underscore(metrics As %String) As %String
{
    set result = metrics
    set regexp = "([A-Z])"
    set matcher = ##class(%Regex.Matcher).%New(regexp, metrics)
    while (matcher.Locate()) {
        set result = matcher.ReplaceAll("_"_"$1")
    }

    // ???????
    set result = $zcvt(result, "l")

    // _e_c_p (_c_s_p) ? _ecp (_csp) ????
    set result = $replace(result, "_e_c_p", "_ecp")
    set result = $replace(result, "_c_s_p", "_csp")

    quit result
}
}
```

コンソールを使用して、私たちの作業が無駄ではなかったことを確認しましょう（curl がプログレスバーの表示を邪魔しないように --silent キーを追加しました）。

```
 # curl --user PromUser:Secret --silent -XGET 'http://localhost:57772/metrics/cache' | head -20
# HELP iscachedashboardapplicationerrors Number of application errors that have been logged.
iscachedashboardapplicationerrors 0
# HELP iscachedashboardcspsessions Most recent number of CSP sessions.
iscachedashboardcspsessions 2
# HELP iscachedashboardcacheefficiency Most recently measured cache efficiency (Global references / (physical reads + writes))
iscachedashboardcacheefficiency 2378.11
# HELP iscachedashboarddiskreads Number of physical block read operations since system startup.
iscachedashboarddiskreads 15101
# HELP iscachedashboarddiskwrites Number of physical block write operations since system startup
iscachedashboarddiskwrites 106233
# HELP iscachedashboardecpappsrvrate Most recently measured ECP application server traffic in bytes/second.
iscachedashboardecpappsrvrate 0
# HELP iscachedashboardecpdatasrvrate Most recently measured ECP data server traffic in bytes/second.
iscachedashboardecpdatasrvrate 0
# HELP iscachedashboardglorefs Number of Global references since system startup.
iscachedashboardglorefs 288545263
```

```
# HELP isccachedashboardglorefspersec Most recently measured number of Global references per second.
isccachedashboardglorefspersec 273.00
# HELP isccachedashboardglosets Number of Global Sets and Kills since system startup.
isccachedashboardglosets 44584646
```

これで、Prometheus のインターフェースで同じ内容を確認できるようになりました。

以下は上記メトリックのリストです。

これらのメトリックの Prometheus での表示内容については詳述しません。
必要なメトリックを選択して「Execute」ボタンをクリックしてください。
「Graph」タブを選択すると、グラフが表示されます（キャッシュの効率が表示されます）。

メトリックの視覚化

メトリックを視覚化するため、Grafana をインストールしましょう。 この記事では、tarball からインストールすることにしました。
ただし、パッケージからコンテナまで、他のインストール方法もあります。
次の手順を実行してみましょう（/opt/grafana フォルダーを作成し、そこに切り替えた後）。

とりあえず設定は変更せずにそのままにしておきましょう。 最後のステップでは、Grafana をバックグラウンドモードで起動します。 Prometheus の場合と同じように、Grafana のログをファイルに保存します。

```
# ./bin/grafana-server > /var/log/grafana.log 2>&1 &
```

デフォルトでは、3000 番ポートで Grafana の Web インターフェースにアクセスできます。
ログイン/パスワードは、admin/admin です。

詳細な Prometheus と Grafana の連携手順については、こちらを参照してください。 簡単に言えば、Prometheus タイプの新しいデータソースを追加する必要があります。 また、次のように direct/proxy アクセスのオプションを選択してください。

完了後、必要なパネルを含むダッシュボードを追加する必要があります。
ダッシュボードのテストサンプルは、メトリック収集クラスのコードと共に公開されています。
ダッシュボードは Grafana に簡単にインポートできます（[Dashboards] → [Import]）。

インポート後、次のようになります。

ダッシュボードを保存します。

時間範囲と更新間隔は右上で選択できます。

監視種類の例

グローバルへの呼び出しの監視をテストしてみましょう。

```
 USER>for i=1:1:1000000 {set ^prometheus(i) = i}
USER>kill ^prometheus
```

以下のグラフでは、1秒あたりのグローバルへの参照数が増加してキャッシュ効率が低下していることが分かります（^Prometheus グローバルがまだキャッシュされていない）。

ライセンスの使用状況を確認しましょう。 そのためには、次のように PromTest.csp というプリミティブな CSP ページを USER ネームスペースに作成しましょう。

```

```

監視は正常に機能しています！

そして、何度もアクセスしてください（/csp/user
アプリケーションがパスワード保護されていないことを想定しています）。

```
# ab -n77 http://localhost:57772/csp/user/PromTest.csp
```

ライセンスの使用状況について、次の図が表示されます。

まとめ

ご覧のとおり、監視機能の実装はまったく難しくありません。 いくつかの初期手順を実行しただけでも、ライセンスの使用状況、グローバルキャッシュの効率、アプリケーションエラーなど、システムの動作に関する重要な情報を取得できます。 このチュートリアルでは SYS.Stats.Dashboard を使用しましたが、SYS / %SYSTEM / %SYS パッケージの他のクラスも注目に値します。 また、特定タイプのドキュメントの数など、独自アプリケーションのカスタムメトリックを提供する独自のクラスを作成することもできます。
いくつかの有用なメトリックは、最終的に Grafana 用の個別テンプレートにコンパイルされます。

今後の予定

本件についてより詳細な情報が必要な場合は、このテーマを詳しく説明するつもりです。
以下に私の予定を記しておきます。

1. ログデーモンのメトリックを含む Grafana テンプレートの準備について。 ^mgstat と同等の、少なくともそのメトリックに対応した何らかのグラフィカルツールを作成するのが望ましいと考えています。
2. Web アプリケーションのパスワード保護は優れていますが、証明書を使用できる可能性を確認するのが望ましいと考えています。
3. Prometheus、Grafana、および Prometheus を Docker コンテナとしてエクスポートするツールの使用について。
4. 新しい Caché インスタンスを Prometheus の監視リストに自動追加するための検出サービスの使用について。 また、Grafana とそのテンプレートの利便性を（実際に）説明したいと考えています。 これは、選択した特定のサーバーのメトリックがすべて同じダッシュボードに表示される動的なパネルのようなものです。
5. Prometheus Alertmanager について。
6. データの保存期間に関連する Prometheus の構成設定、および多数のメトリックと短い統計収集間隔を持つシステムに考えられる最適化について。
7. 途中で発生するさまざまで微妙な差異について。

リンク

この記事を準備中にいくつかの有益なサイトにアクセスし、次のようなたくさんの動画を視聴しました。

- Prometheus プロジェクトの Web サイト
- Grafana プロジェクトの Web サイト
- Brian Brazil という Prometheus 開発者のブログ
- DigitalOcean のチュートリアル
- Robust Perception の動画数点
- Prometheus を対象とする多数のカンファレンス動画

最後までお読みいただき、ありがとうございました！

#システム管理 #監視 #視覚化 #Caché

ソースURL:https://jp.community.intersystems.com/post/prometheus-%E3%81%A7-intersystems-cach%C3%A9-%E3%82%92%E7%9B%A3%E8%A6%96%E3%81%99%E3%82%8B