Artículo
Bernardo Linarez · 28 oct, 2020 Lectura de 12 min

Monitorización de InterSystems IRIS y Caché con Prometheus

Prometheus es uno de los sistemas de monitorización adaptado para recoger datos de series temporales.

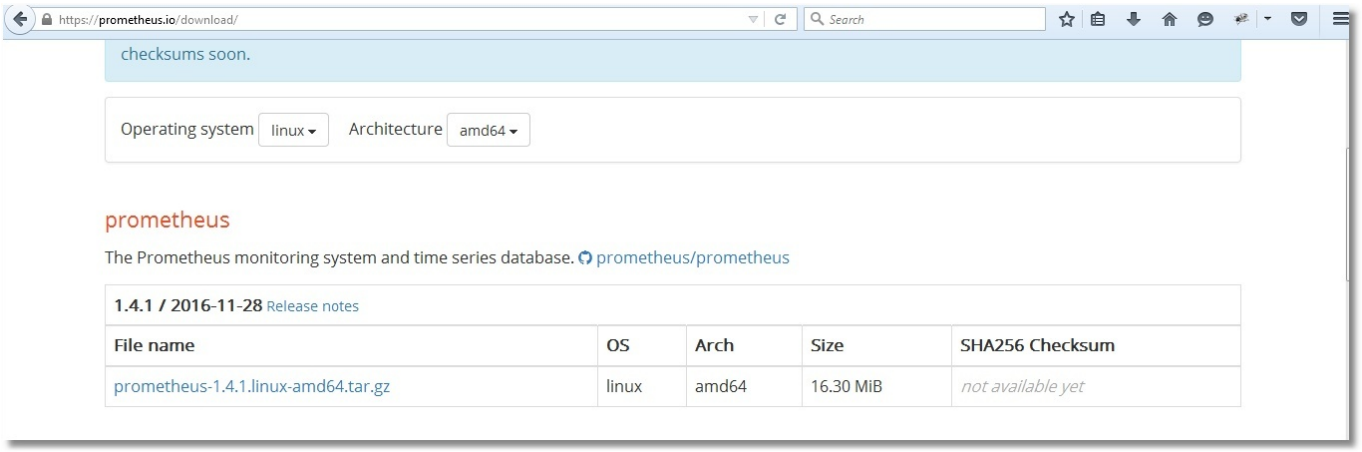
Su instalación y configuración inicial son relativamente sencillos. El sistema tiene un subsistema gráfico integrado llamado **PromDash** para la visualización de datos, pero los desarrolladores recomiendan usar un producto de otro proveedor, llamado Grafana. Prometheus puede monitorizar muchas cosas (hardware, contenedores, distintos sistemas de gestión de base de datos), pero en este artículo me gustaría analizar la monitorización de una instancia de Caché (para ser exactos, será una instancia de Ensemble, pero las métricas serán de Caché). Si te interesa, sigue leyendo.

En nuestro caso extremadamente simple, Prometheus y Caché vivirán en una única máquina (Fedora Workstation 24 x8664). Versión de Caché:

```
%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
```

Instalación y configuración

Vamos a descargar un paquete de distribución de Prometheus desde la página oficial y lo vamos a guardar en la carpeta /opt/prometheus.



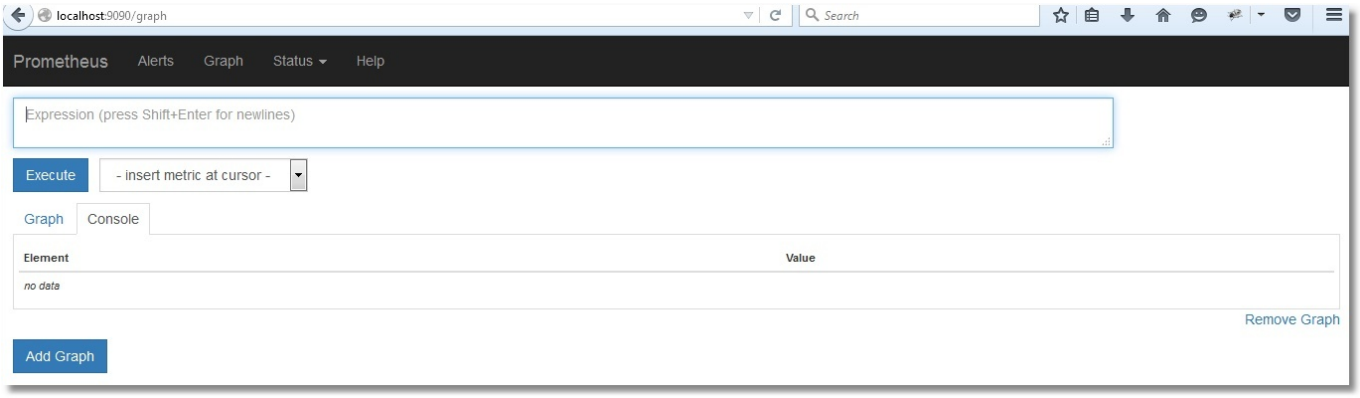
Descomprime el archivo, modifica el archivo de configuración de plantilla según tus necesidades e inicia Prometheus. De forma predeterminada, Prometheus mostrará sus registros directamente en la consola, por eso guardaremos sus registros de actividad en un archivo de log.

Iniciar Prometheus

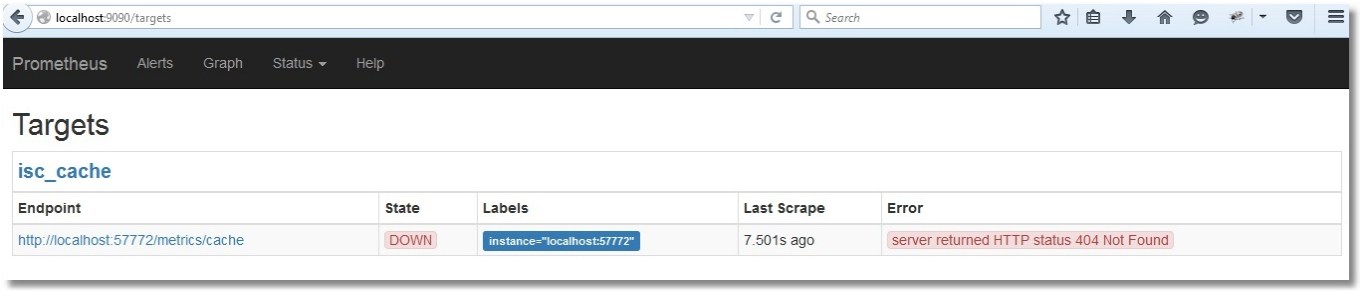
```
# pwd
/opt/prometheus
# ls
prometheus-1.4.1.linux-amd64.tar.gz
# tar -xzf prometheus-1.4.1.linux-amd64.tar.gz
# ls
prometheus-1.4.1.linux-amd64 prometheus-1.4.1.linux-amd64.tar.gz
# cd prometheus-1.4.1.linux-amd64/
# ls
consolelibraries consoles LICENSE NOTICE prometheus prometheus.yml promtool
```

```
# cat prometheus.yml
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.

scrape_configs:
  - job_name: 'isc_cache'
    metrics_path: '/metrics/cache'
    static_configs:
    - 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»
```

La configuración prometheus.yml se escribe en lenguaje YAML, que no se lleva bien con los símbolos de tabulación, por lo que solo deberías usar espacios. Ya mencionamos que las métricas se descargarán desde http://localhost:57772 y enviaremos solicitudes a /metrics/cache (el nombre de la aplicación es arbitrario), es decir, la dirección de destino para recopilar métricas será http://localhost:57772/metrics/cache. Se agregará una etiqueta "job=isc_cache" a cada métrica. Una etiqueta, de forma muy aproximada, es el equivalente a WHERE en SQL. En nuestro caso no se usará, pero funcionará bien para más de un servidor. Por ejemplo, los nombres (y/o instancias) de servidores pueden guardarse en etiquetas, y luego se pueden usar las etiquetas para parametrizar solicitudes para realizar gráficas. Vamos a asegurarnos de que Prometheus está funcionando (en la salida de arriba podemos ver el puerto que está escuchando: 9090):



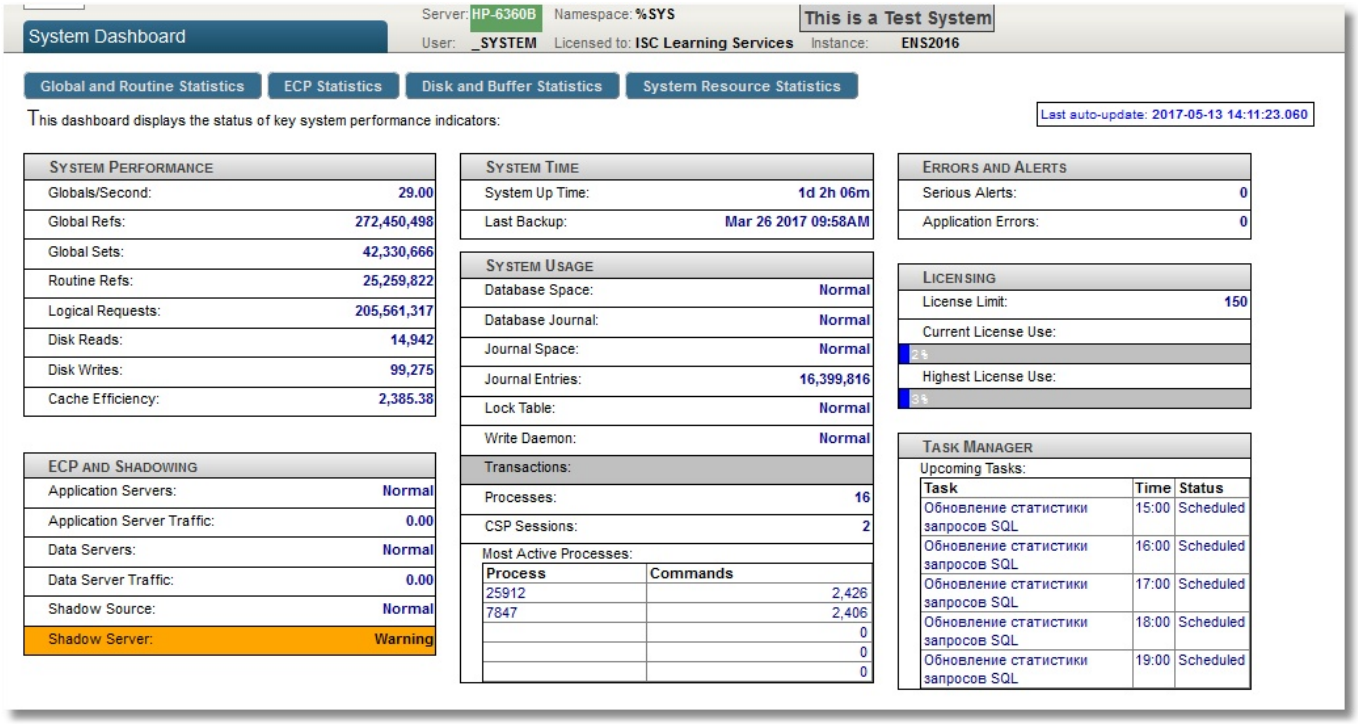
Se abre una interfaz web, lo que significa que Prometheus está funcionando. Sin embargo, aún no ve métricas de Caché (verifiquemos haciendo clic en Status → Targets):



Preparación de métricas

Nuestra tarea es dejar las métricas disponibles para Prometheus en **un formato adecuado** en http://localhost:57772/metrics/cache. Usaremos las capacidades REST de Caché debido a su simplicidad. Es necesario saber que Prometheus solo "entiende" las métricas numéricas, por lo que no exportaremos métricas de cadenas de texto. Para obtener lo último, usaremos la API de la clase SYS.Stats.Dashboard. Estas métricas las

usa el propio Caché para mostrar la barra de herramientas de Sistema:



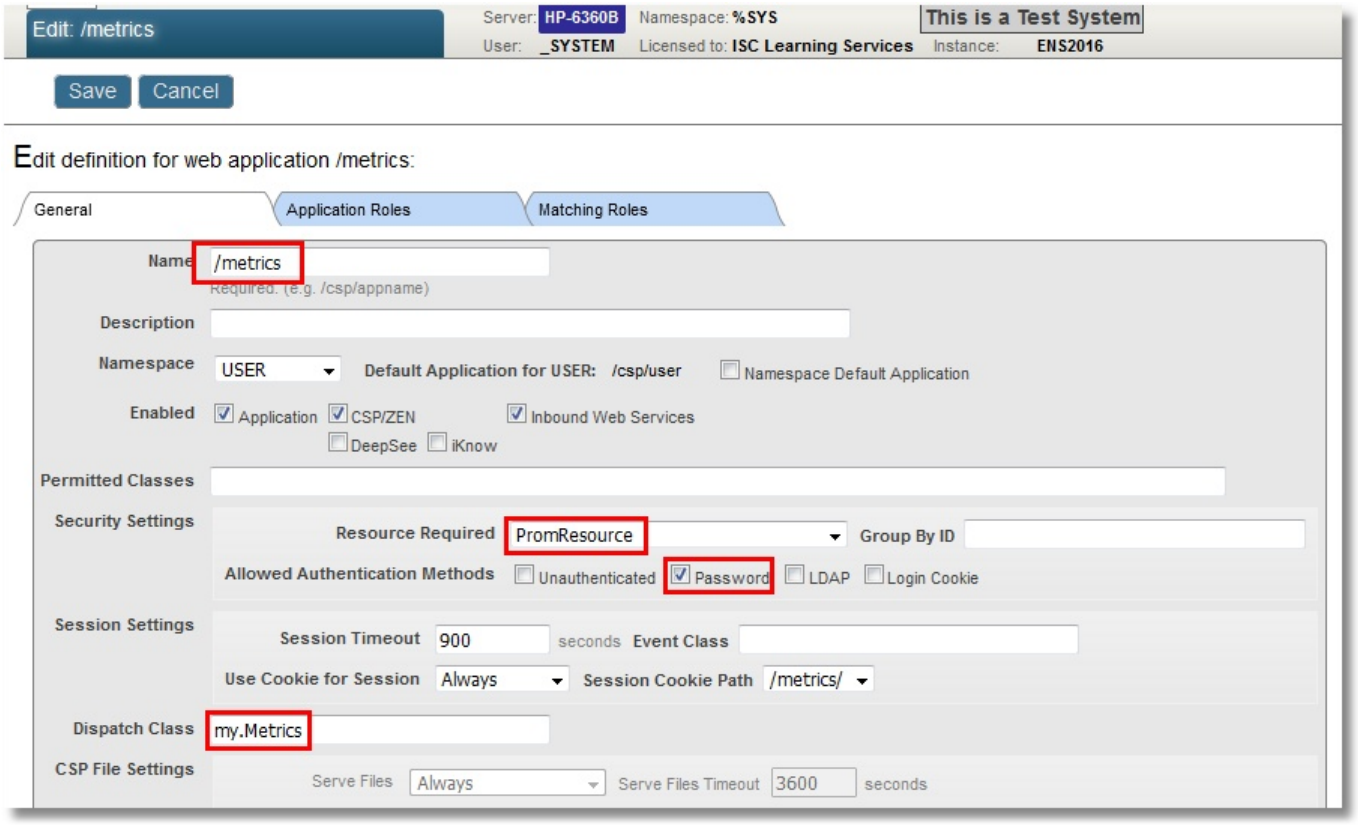
Ejemplo de lo mismo en el Terminal:

```
%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
...
```

El espacio USER será nuestro "entorno controlado" (sandbox). Para empezar, creamos una aplicación/métricas REST. Para añadir un nivel muy básico de seguridad, protegeremos nuestro inicio de sesión con una contraseña y asociaremos la aplicación web con algún recurso, que llamaremos PromResource. Necesitamos desactivar el acceso público al recurso, por lo que haremos lo siguiente:

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

La configuración de nuestra aplicación web:



También necesitaremos un usuario con acceso a este recurso. El usuario también debería poder leer desde nuestra base de datos (USER, en nuestro caso) y guardar datos en ella. Además de esto, el usuario necesitará permisos de escritura para la base de datos del sistema CACHESYS, ya que más tarde pasaremos al espacio %SYS en el código. Seguiremos el esquema estándar, es decir, crear un rol PromRole con estos permisos y luego crear un usuario PromUser asignado a este rol. Para la contraseña, usaremos "Secret":

```
%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
```

Este usuario PromUser será el que usaremos para autenticación en la configuración de Prometheus. Una vez hecho, enviaremos una señal SIGNUP al proceso del servidor, para releer la configuración.

Una configuración más segura

```
# cat /opt/prometheus/prometheus-1.4.1.linux-amd64/prometheus.yml
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.

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) # or kill -1 $(pgrep prometheus)
```

Ahora Prometheus puede pasar la autenticación con éxito, para usar la aplicación web con métricas.

Las métricas serán provistas por la clase de procesamiento de solicitudes my.Metrics. Aquí está la implementación:

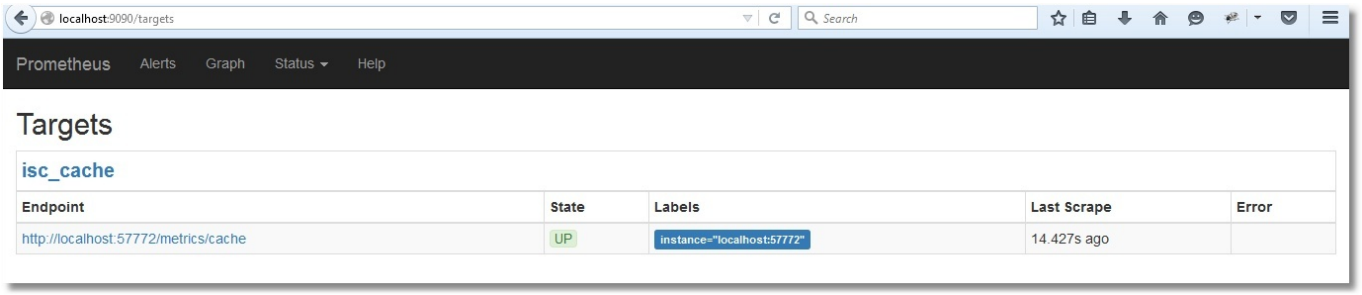
```
Class my.Metrics Extends %CSP.REST
{

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>
}

/// Output should obey the Prometheus exposition formats. Docs:
/// https://prometheus.io/docs/instrumenting/exposition_formats/
///
/// The protocol is line-oriented. A line-feed character (\n) separates lines.
/// The last line must end with a line-feed character. Empty lines are ignored.
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 supports time series database
  // so if we get empty (for example, backup metrics) or non-digital metrics
  // we just omit them.
  if ((propertyValue '= "") && ('$match(propertyValue, ".*[-A-Za-z ]+.*"))) {
   set metricsName = ..#DASHPREFIX_..camelCase2Underscore(propertyName)
   set metricsValue = propertyValue

   // Write description (help) for each metrics.
   // Format is that the Prometheus requires.
   // Multiline descriptions we have to join in one string.
   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).Properties

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

/// Converts metrics name in camel case to underscore name with lower case
/// Sample: input = WriteDaemon, output = _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")
 }

 // To lower case
 set result = $zcvt(result, "l")

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

 quit result
}
}
```

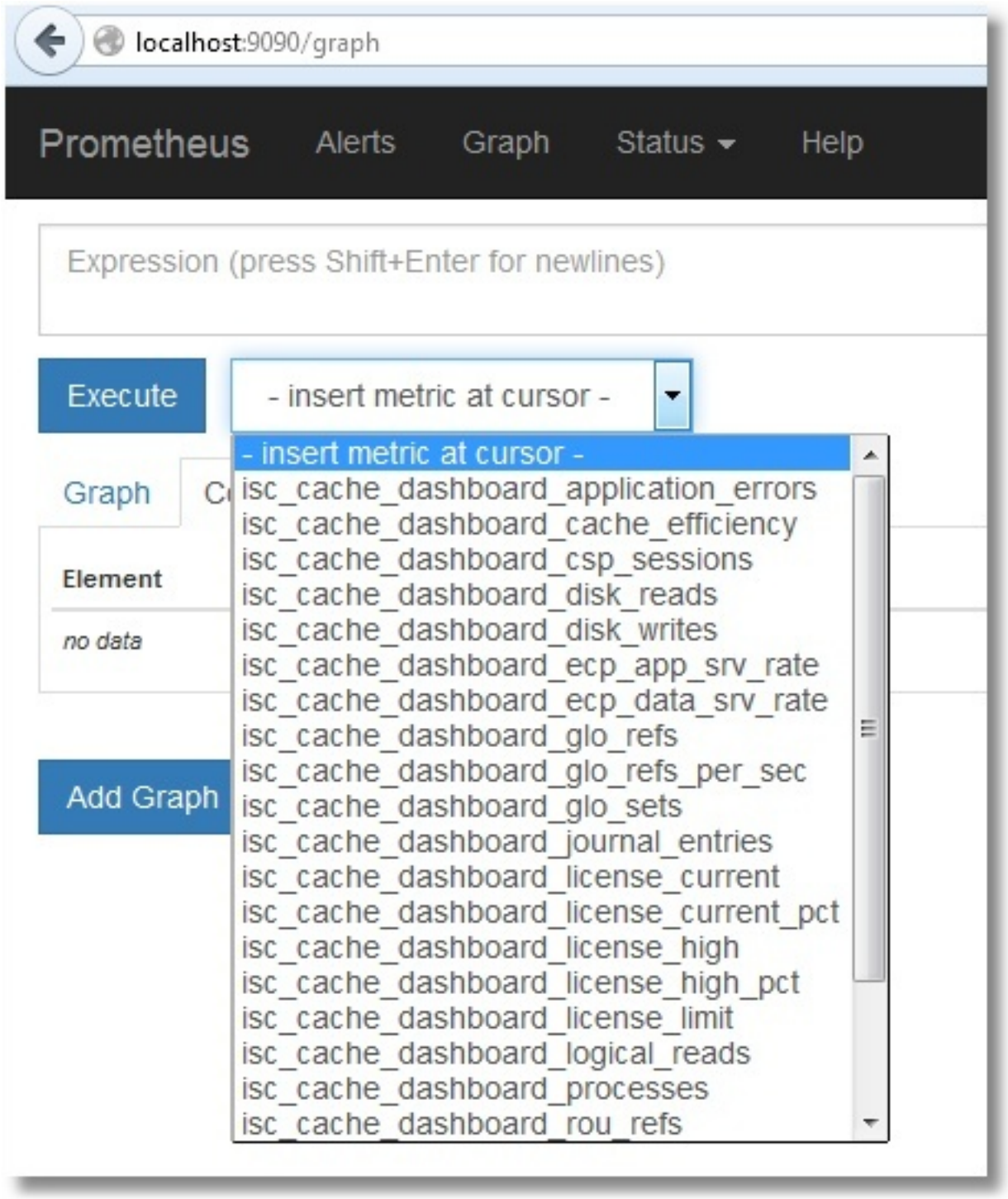
Usaremos la consola para verificar que nuestros esfuerzos no han sido en vano (agregamos la clave --silent para que curl no nos estorbe con su barra de progreso):

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

Ahora podemos verificar lo mismo en la interfaz de Prometheus:

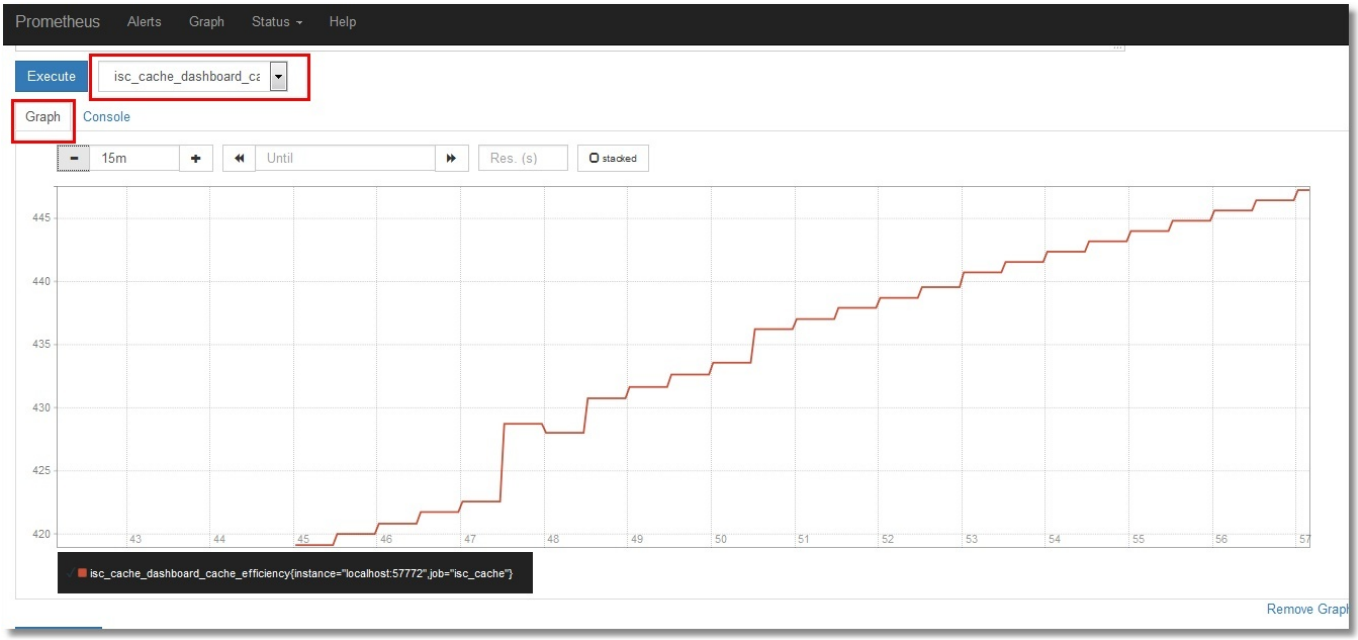


Y aquí está la lista de nuestras métricas:



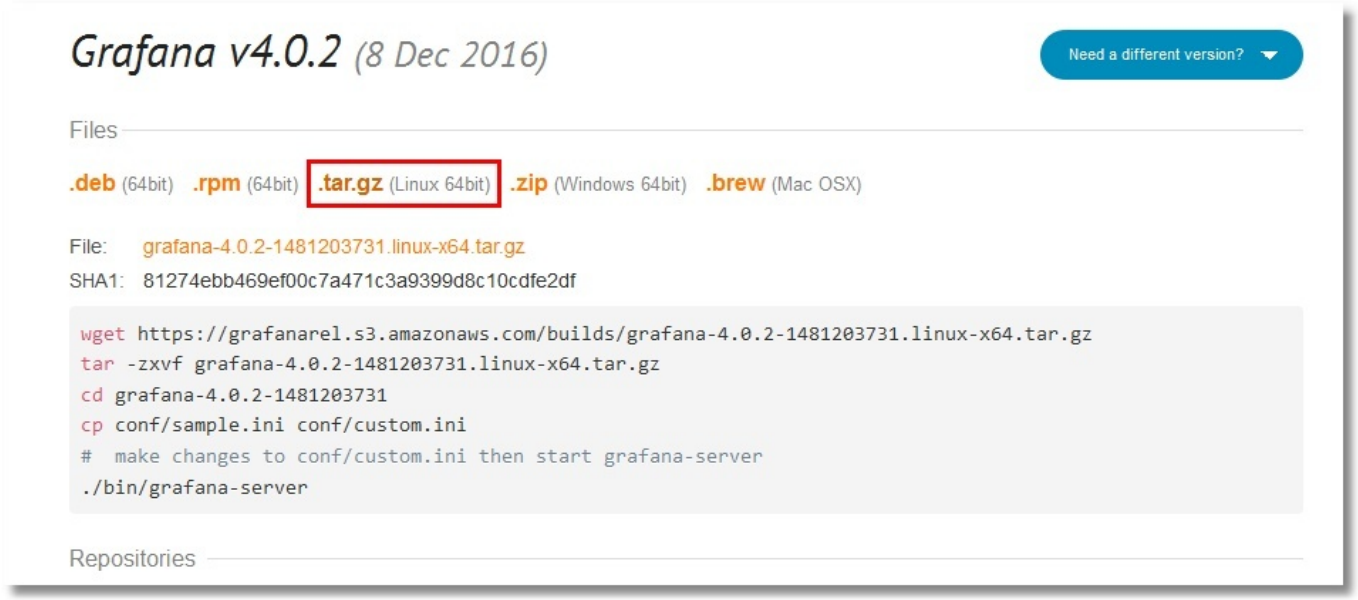
No nos enfocaremos en verlas en Prometheus. Puedes seleccionar la métrica que necesites y hacer clic en el

botón "Execute". Selecciona la pestaña "Graph" para ver la gráfica (muestra la eficiencia del caché):



Visualización de métricas

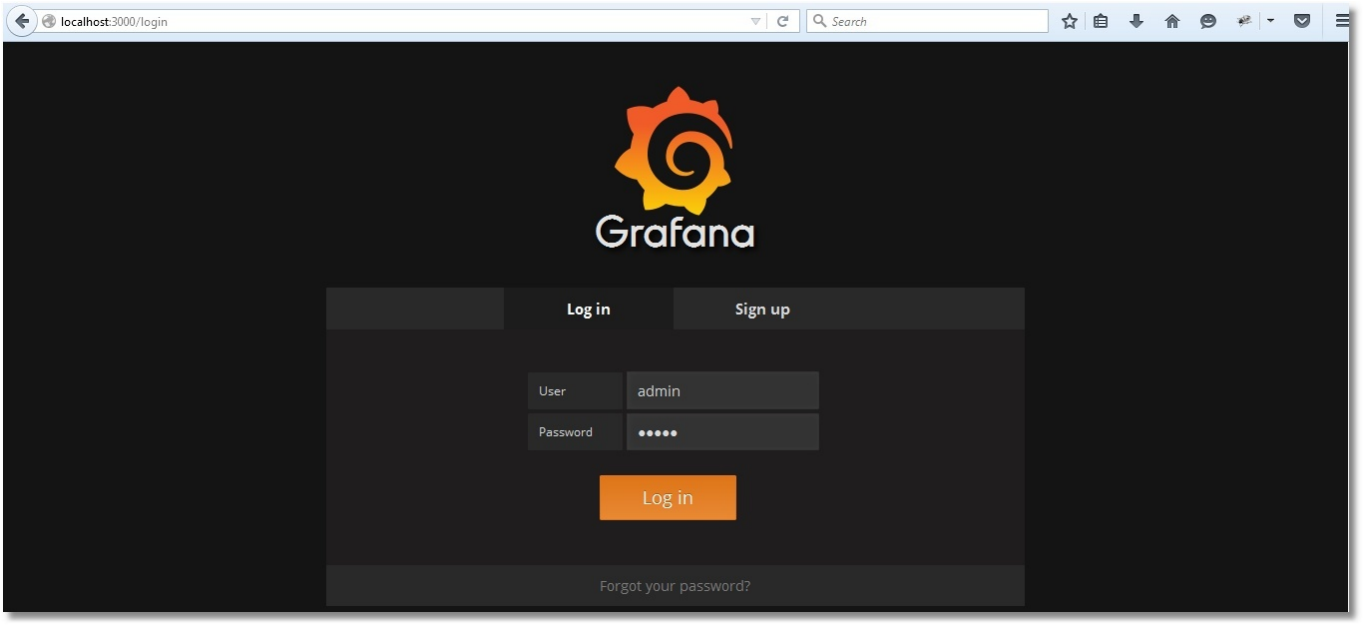
Con fines de visualización, instalemos Grafana. Para este artículo, elegí la instalación desde un tarball. Sin embargo, hay otras opciones de instalación, desde paquetes hasta un contenedor. Sigamos los siguientes pasos (después de crear la carpeta /opt/grafana y entrar a ella):



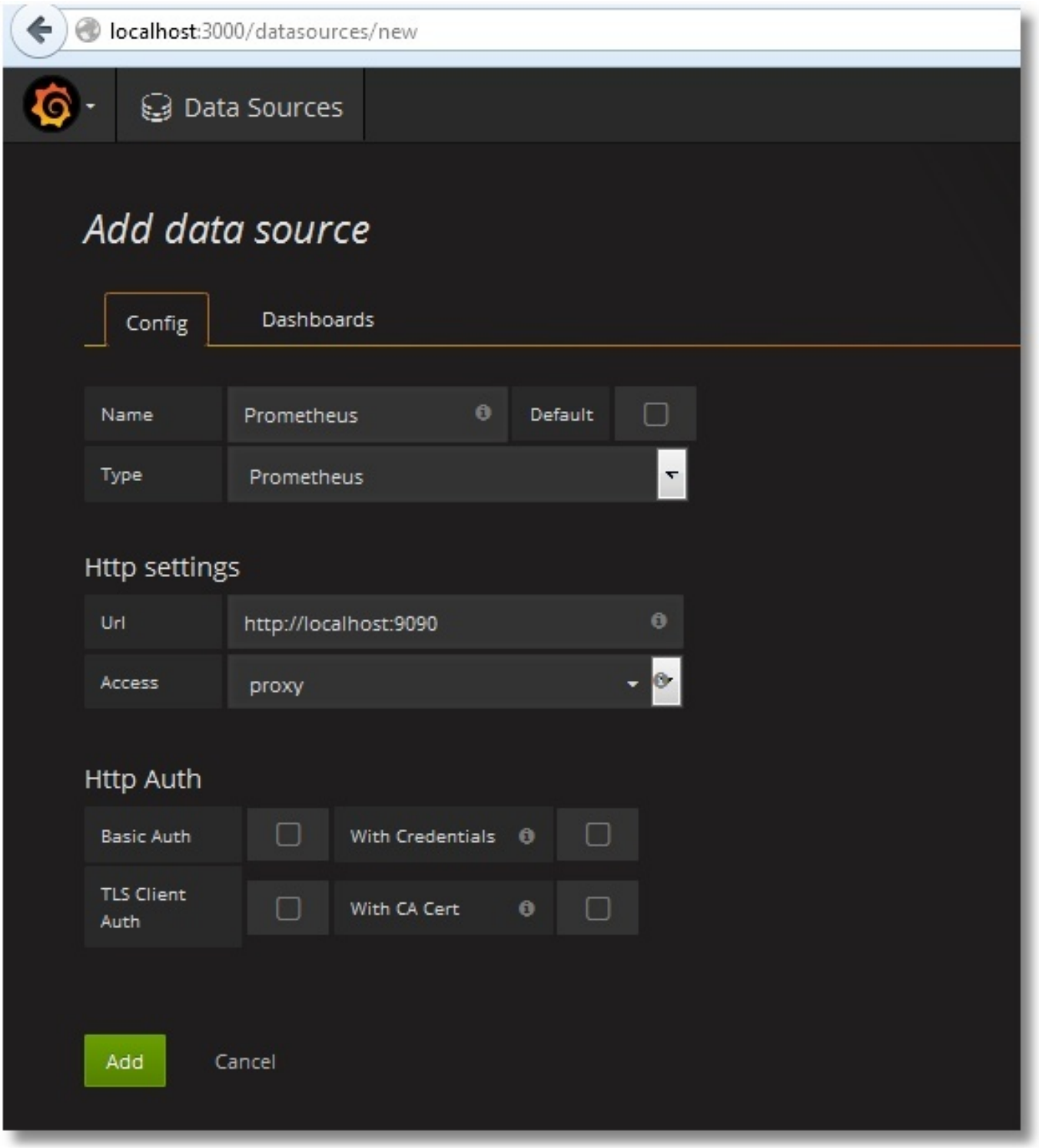
Dejemos la configuración sin cambios por ahora. En nuestro último paso, iniciamos Grafana en segundo plano (background mode). Guardaremos el log de Grafana a un archivo, igual que lo hicimos con Prometheus:

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

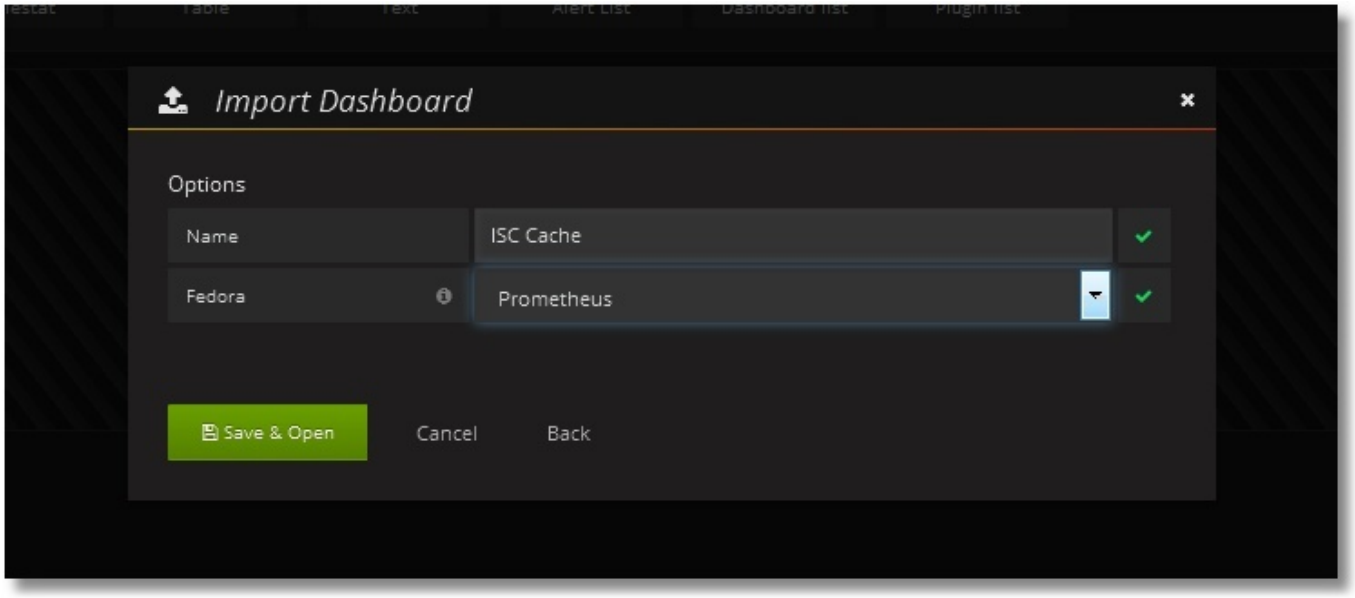
Por defecto, la interfaz web de Grafana está accesible a través el puerto 3000. Usuario/contraseña: admin/admin.



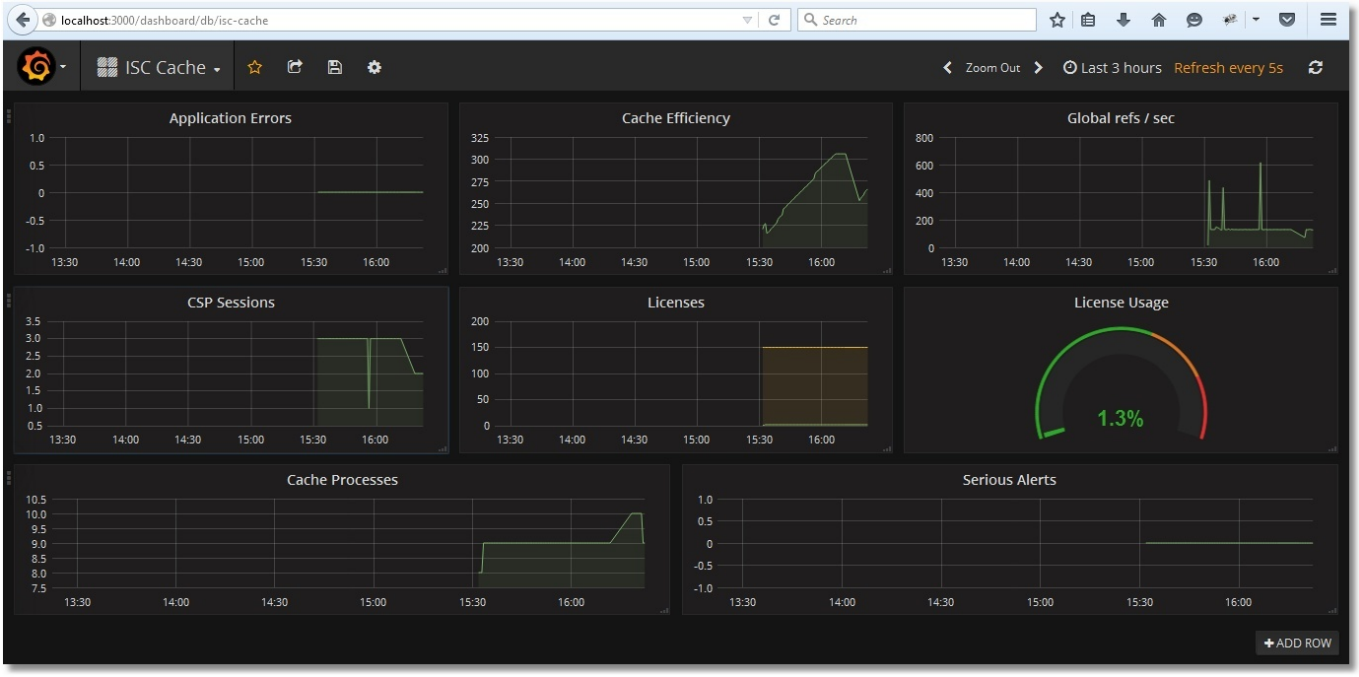
Para tener instrucciones detalladas sobre cómo hacer que Prometheus funcione con Grafana, hac clic aquí**.** En resumen, necesitamos añadir una nueva Fuente de Datos del tipo Prometheus. Selecciona tu opción para acceso directo/proxy:



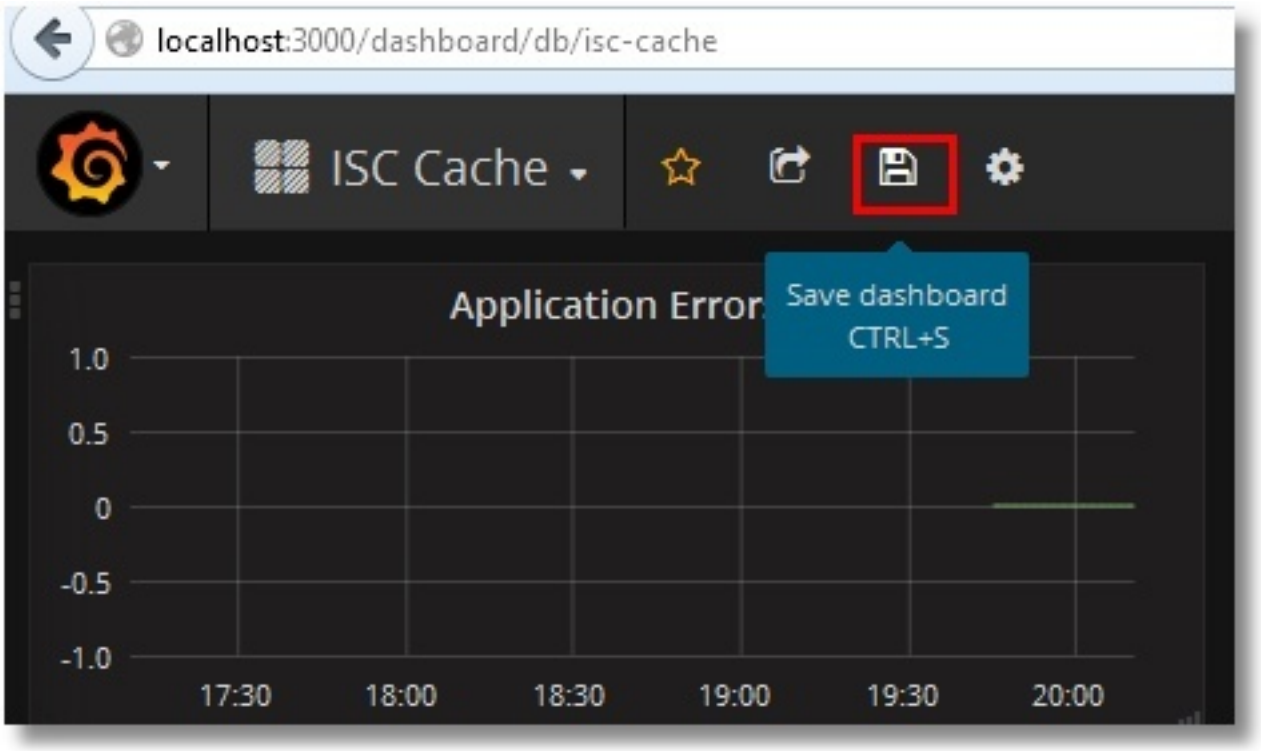
Una vez hecho esto, necesitamos añadir un tablero con los paneles necesarios. El ejemplo de prueba de un tablero está disponible públicamente, junto con el código de la clase de recolección de métricas. Es posible importar fácilmente un tablero a Grafana (Dashboards → Import):



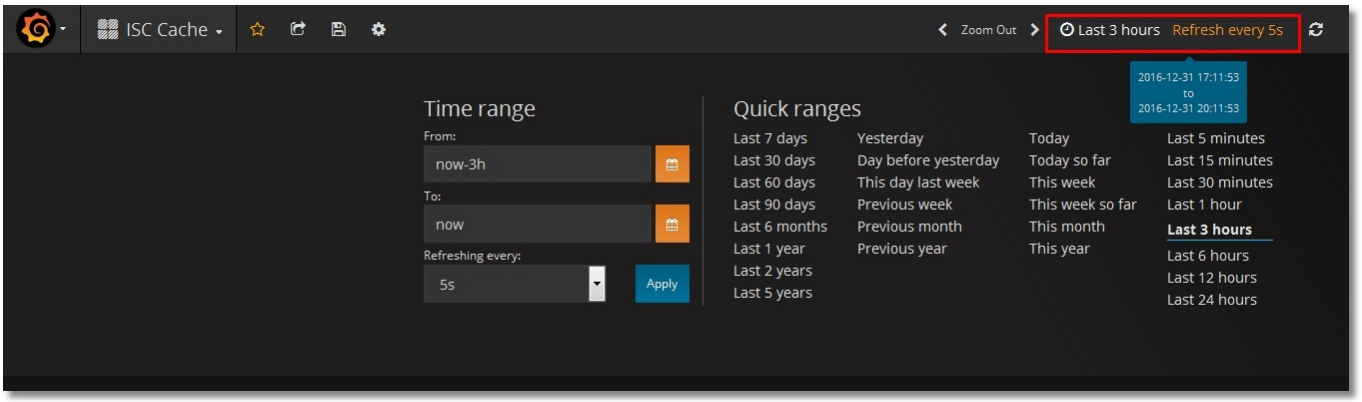
Tras la importación, obtendremos lo siguiente:



Guarda el tablero:



Puedes seleccionar el intervalo de tiempo y el plazo de actualización en la esquina superior derecha:

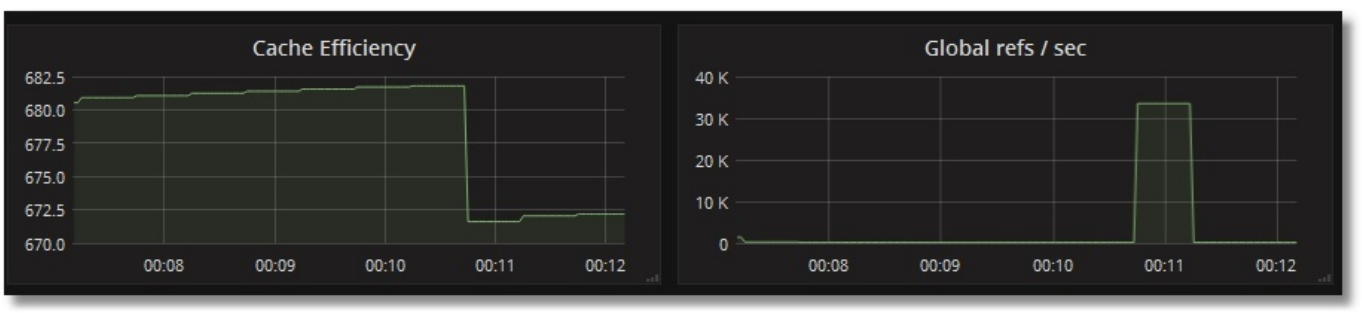


Ejemplos de tipos de monitorización

Probemos la monitorización de llamadas a globales:

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

Podemos ver que el número de referencias a globales por segundo ha aumentado, mientras que la eficiencia del caché cayó (la global ^Prometheus aún no se ha guardado en caché):



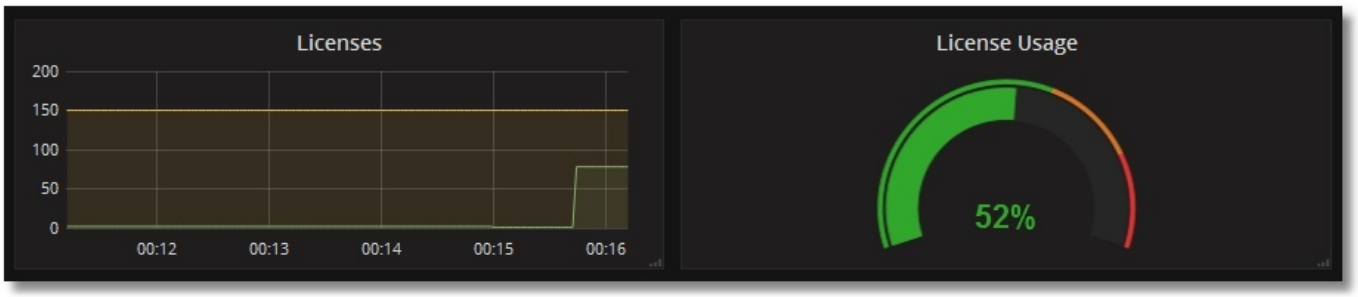
Veamos nuestro uso de licencias. Para esto, vamos a crear una página CSP primitiva, llamada PromTest.csp, en el namespace USER:

```
<html>
<head><title>Prometheus Test Page</title></head>
<body>Monitoring works fine!</body>
</html>
```

Y la visitaremos tantas veces (asumimos que la aplicación /csp/user no está protegida por contraseña):

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

Veremos la siguiente imagen de uso de licencias:



Conclusiones

Como podemos ver, implementar la funcionalidad de monitorización no es nada difícil. Incluso después de unos pocos pasos iniciales, podemos obtener información importante sobre el trabajo del sistema, como: uso de licencias, eficiencia del caché de globales (globals caching) y errores de aplicación. Para este tutorial usamos el tablero SYS.Stats.Dashboard, pero otras clases de paquetes SYS, %SYSTEM, %SYS también merecen atención. También puedes escribir tu propia clase que genere métricas personalizadas para tu propia aplicación. Por ejemplo: el número de documentos de un tipo en particular. Algunas métricas útiles, con el tiempo, se compilarán en una plantilla separada para Grafana.

Continuará

Si le interesa aprender más sobre este tema, escribiré más sobre el mismo. Estos son mis planes:

1. Preparar una plantilla para Grafana con métricas para el daemon de registro. Sería interesante hacer algún tipo de gráfico equivalente de la herramienta ^mgstat – al menos para alguna de sus métricas.
2. La protección con contraseña para las aplicaciones web es buena, pero sería bueno verificar la posibilidad de usar certificados.
3. Usar Prometheus, Grafana y algunos exportadores para Prometheus como contenedores Docker.
4. Usar servicios de descubrimiento (discovery services) para añadir automáticamente nuevas instancias de Caché a la lista de monitorización de Prometheus. Aquí es también donde querría demostrar (en la práctica) lo conveniente que son Grafana y sus plantillas. Esto es parecido a los paneles dinámicos, donde se muestran las métricas de un servidor seleccionado en particular, todo en el mismo tablero.
5. Administrador de Alertas de Prometheus (Prometheus Alert Manager).
6. Ajustes de configuración de Prometheus relacionados con la duración de los datos almacenados, así como posibles optimizaciones para sistemas con una gran cantidad de métricas y un intervalo corto de recopilación de estadísticas.
7. Otras ideas útiles y detalles que surgirán en el camino.

Enlaces útiles

Al preparar este artículo, visité varios sitios web útiles y vi una gran cantidad de videos:

- Prometheus project website

- Grafana project website
- Blog of one of Prometheus developers called Brian Brazil
- Tutorial on DigitalOcean
- Some videos from Robust Perception
- Many videos from a conference devoted to Prometheus

¡Gracias por leer hasta aquí!

#Administración del sistema #Mejores prácticas #Monitorización #Visualización #Caché #Ensemble

---

URL de fuente:https://es.community.intersystems.com/post/monitorizaci%C3%B3n-de-intersystems-iris-y-cach%C3%A9-con-prometheus