Article <u>Mikhail Khomenko</u> · May 15, 2017 12m read

# Making Prometheus Monitoring for InterSystems IRIS and Caché

<u>Prometheus</u> is one of the monitoring systems adapted for collecting <u>time series data</u>.

Its installation and initial configuration are relatively easy. The system has a built-in graphic subsystem called <u>PromDash</u> for visualizing data, but developers recommend using a free third-party product called <u>Grafana</u>. Prometheus can monitor a lot of things (hardware, containers, various DBMS's), but in this article, I would like to take a look at the monitoring of a <u>Caché</u> instance (to be exact, it will be an Ensemble instance, but the metrics will be from Caché). If you are interested – read along.

In our extremely simple case, Prometheus and Caché will live on a single machine (Fedora Workstation 24 x86<u>6</u>4). Caché version:

%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	
	_

Installation and configuration

Let's download a suitable Prometheus distribution package from the official site and save it to the /opt/prometheus folder.

checksums soon.						
Operating system linux  Architecture amd64	4 -					
<b>prometheus</b> The Prometheus monitoring system and time series	database. 🔿 prometh	eus/promethe	us			
prometheus The Prometheus monitoring system and time series 1.4.1 / 2016-11-28 Release notes	database. <b>O</b> prometh	eus/promethe	us			
prometheus The Prometheus monitoring system and time series 1.4.1 / 2016-11-28 Release notes File name	database. O prometh	eus/promether	us Size	SHA256 Ch	ecksum	

Unpack the archive, modify the template config file according to our needs and launch Prometheus. By default, Prometheus will be displaying its logs right in the console, which is why we will be saving its activity records to a log file.

Launching Prometheus



scrapeinterval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute. 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»

The prometheus.yml configuration is written in the YAML language, which doesn't like tabulation symbols, which is why you should use spaces only. We have already mentioned that metrics will be downloaded from <a href="http://localhost:57772">http://localhost:57772</a> and we'll be sending requests to /metrics/cache (the name of the application is arbitrary), i.e. the destination address for collecting metrics will be <a href="http://localhost:57772/metrics/cache">http://localhost:57772</a> and we'll be sending requests to /metrics/cache (the name of the application is arbitrary), i.e. the destination address for collecting metrics will be <a href="http://localhost:57772/metrics/cache">http://localhost:57772/metrics/cache</a>. A "job=isccache" tag will be added to each metric. A tag, very roughly, is the equivalent of WHERE in SQL. In our case, it won't be used, but will do just fine for more than one server. For example, names of servers (and/or instances) can be saved to tags and you can then use tags to parameterize requests for drawing graphs. Let's make sure that Prometheus is working (we can see the port it's listening to in the output above – 9090):

Cecelhost:9090/graph		☆ 自 、	<b>ŀ</b> ∧	❷ ≉ - ◙ ≣	=
Prometheus Alerts Graph Status - Help					
Expression (press Shift+Enter for newlines)		h.			
Execute - insert metric at cursor -					
Graph Console					_
Element	Value				
no data					
Add Graph				Remove Grap	h

A web interface opens, which means that Prometheus is working. However, it doesn't see Caché metrics yet (let's check it by clicking Status Targets):

localhost:9090/targets			∀ C Q Searc	h	☆自♣		9	* -	Ξ
Prometheus Alerts Graph Status -	Help								
Targets									
isc_cache									
Endpoint	State	Labels	Last Scrape	Error					
http://localhost:57772/metrics/cache	p://localhost:57772/metrics/cache DOWN instance="localhost:57772"				status 404 Not F	ound			

### Preparing metrics

Our task is to make metrics available to Prometheus in <u>a suitable format at http://localhost:57772/metrics/cache</u>. We'll be using <u>the REST capabilities of Caché</u> because of their simplicity. It should be noted that Prometheus only " understands " numeric metrics, so we will not export string metrics. To get the latter, we will use the API of the <u>SYS.Stats.Dashboard</u> class. These metrics are used by Caché itself for displaying the System toolbar:

## Making Prometheus Monitoring for InterSystems IRIS and Caché Published on InterSystems Developer Community (https://community.intersystems.com)

stem Dashboard	User:	_SYSTEM Licensed	d to: ISC Learning Services	Instance:	EN S2016		
lobal and Routine Statistics ECI s dashboard displays the status of key s	P Statistics Disk	and Buffer Statistics	System Resource Sta	tistics	Last auto-up	date: 201	17-05-13 14:
System Performance		System Time			ERRORS AND ALERTS		
Globals/Second:	29.00	System Up Time:		1d 2h 06m	Serious Alerts:		0
Global Refs:	272,450,498	Last Backup:	Mar 26 201	7 09:58AM	Application Errors:		0
Global Sets:	42,330,666	Cuerra lles en			L		
Routine Refs:	25,259,822	SYSTEM USAGE		Normal	LICENSING		
Logical Requests:	205,561,317	Database Journal:		Normal	License Limit:		150
Disk Reads:	14,942	Lournal Space:		Normal	Current License Use:		
Disk Writes:	99,275	Journal Space.		10 200 040	23 Highest License Lise:		
Cache Efficiency:	2,385.38	Journal Entries:		16,399,816	3%		
		Lock lable:		Normai			
		Write Daemon:		Normal	TASK MANAGER		
ECP AND SHADOWING		Transactions:			Upcoming Tasks:		
Application Servers:	Normal	Processes:		16	Task	Time	Status
Application Server Traffic:	0.00	CSP Sessions:		2	запросов SQL	15:00	Scheduled
Data Servers:	Normal	Most Active Proces	ses:		Обновление статистики	16:00	Scheduled
Data Server Traffic:	0.00	Process	Commands	0.400	запросов SQL Обновление статистики	17:00	Scheduled
Shadow Source:	Normal	25912		2,426	sanpocoB SQL		
Shadow Server:	Warning			0	Обновление статистики запросов SQL	18:00	Scheduled
				0	Обновление статистики	19:00	Scheduled

Example of the same in the Terminal:

%SYS>set dashboard = ##class(SYS.Stats.Dashboard).Sample()
SYS>zwrite dashboard
tasinbualu=<0bject Reference>[2@313.3tals.basinbualu]
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
GloRetsPerSec = "70.00"
GloSets = 42330792
JournalEntries = 16399816
JournalSpace = "Normal"
JournalStatus = "Normal   LestBackup = "Mer 26 2017 00 59 MM"
Lasidackup =   Mai   20   20   7   09.36 AM
$\  \text{LicenseQuiterit} = 3$

The USER space will be our sandbox. To begin with, let's create a REST application /metrics. To add come very basic security, let's protect our log-in with a password and associate the web application with some resource – let's call it PromResource. We need to disable public access to the resource, so let's do the following:

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

Our web app settings:

### Making Prometheus Monitoring for InterSystems IRIS and Caché Published on InterSystems Developer Community (https://community.intersystems.com)

Edit: /metrics	Server: HP-6360B Namespace: %SYS This is a Test System
Save Cano	User: _SYSTEM Licensed to: ISC Learning Services Instance: ENS2016
Save	
dit definition for we	eb application /metrics:
General	Application Roles Matching Roles
Name	/metrics Required. (e.g. /csp/appname)
Description	
Namespace	USER    Default Application for USER: /csp/user  Namespace Default Application
Enabled	Application CSP/ZEN Inbound Web Services
Permitted Classes	
Security Settings	Resource Required PromResource Group By ID
	Allowed Authentication Methods Unauthenticated Password LDAP Login Cookie
Session Settings	Session Timeout 900 seconds Event Class
	Use Cookie for Session Always   Session Cookie Path /metrics/
Dispatch Class	Use Cookie for Session Always   Session Cookie Path /metrics/  my.Metrics

We will also need a user with access to this resource. The user should also be able to read from our database (USER in our case) and save data to it. Apart from this, this user will need read rights for the CACHESYS system database, since we will be switching to the %SYS space later in the code. We'll follow the standard scheme, i.e. create a PromRole role with these rights and then create a PromUser user assigned to this role. For password, let 's use "Secret":

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

%SYS>write ##class(Security.Users).Create("PromUser","PromRole","Secret")

It is this user PromUser that we will used for authentication in the Prometheus config. Once done, we'll re-read the config by sending a SIGNUP signal to the server process.

A safer config



Prometneus can now successfully pass authentication for using the web application with metrics.

Metrics will be provided by the my.Metrics request processing class. Here is the implementation:

{

```
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/
111
/// 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)
ł
 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)
ł
 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, "1")
 // _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
ł
}
```

Let's use the console to check that our efforts have not been vain (added the --silent key, so that curl doesn't impede us with its progress bar):



We can now check the same in the Prometheus interface:

## Making Prometheus Monitoring for InterSystems IRIS and Caché

Published on InterSystems Developer Community (https://community.intersystems.com)

🗲 闭 localhost:909	0/targets							. △	Q Search		☆ [	€ 4	Â	ø	* -		≡
Prometheus	Alerts	Graph	Status 🗸	Help													
Targets																	
isc_cache																	
Endpoint						State	La	bels		L	ast Scr	ape			Erro	r	
http://localhost:57	7772/metri	s/cache				UP	in	stance="localhost:577	72"	1	4.427s	ago					

## And here is the list of our metrics:

🗲 🛞 localhos	<b>t</b> :9090/graph	
Prometheu	<b>s</b> Alerts Graph Status <del>-</del> Help	
Expression	(press Shift+Enter for newlines)	
Execute	- insert metric at cursor -	
Graph C	- insert metric at cursor - isc_cache_dashboard_application_errors isc_cache_dashboard_cache_efficiency	
Element	isc_cache_dashboard_csp_sessions isc_cache_dashboard_disk_reads	L
no data	isc_cache_dashboard_disk_writes isc_cache_dashboard_ecp_app_srv_rate isc_cache_dashboard_ecp_data_srv_rate	
	isc_cache_dashboard_glo_refs	=
Add Graph	isc_cache_dashboard_glo_rets_per_sec isc_cache_dashboard_glo_sets	
	isc cache dashboard license current	
	isc_cache_dashboard_license_current_pct	
	isc_cache_dashboard_license_high	
	isc cache dashboard license limit	
	isc_cache_dashboard_logical_reads	
	isc_cache_dashboard_processes	
	isc_cache_dashboard_rou_refs	*

We won 't focus on viewing them in Prometheus. You can select the necessary metric and click the "Execute" button. Select the "Graph" tab to see the graph (shows the cache efficiency):

Promethe	eus Alerts	Graph Status <del>-</del>	Help										
Execute	isc_cache_c	lashboard_c: 💌											
Graph	Console												
-	15m	+ Until		▶ Res. (s)	O stacked								
445													
440													
435													
430													
425													
420 -	43	44 45	46	47	48 49	50	51	52	53	54	55	56	57
×.	isc_cache_dashboa	ard_cache_efficiency{ins	tance="localhost:57772",	job="isc_cache"}									
												F	Remove Grap

#### Visualization of metrics

For visualization purposes, let 's instation from a tarball. However, there are other installation options, from packages to a container. Let's perform the following steps (after creating the /opt/grafana folder and switching to it):

Grafana v4.0.2 (8 Dec 2016)	Need a different version? 👻
Files         .deb (64bit)       .rpm (64bit)         .tar.gz (Linux 64bit)         .zip (Windows 64bit)         .brew (Mac OSX)	
File:         grafana-4.0.2-1481203731.linux-x64.tar.gz           SHA1:         81274ebb469ef00c7a471c3a9399d8c10cdfe2df	
<pre>wget https://grafanarel.s3.amazonaws.com/builds/grafana-4.0.2-1481203731 tar -zxvf grafana-4.0.2-1481203731.linux-x64.tar.gz cd grafana-4.0.2-1481203731 cp conf/sample.ini conf/custom.ini # make changes to conf/custom.ini then start grafana-server ./bin/grafana-server</pre>	.linux-x64.tar.gz
Repositories	

Let's leave the config unchanged for now. On our last step, we launch Grafana in the background mode. We'll be saving Grafan's log to a file, just like we did with Prometheus:

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

By default, Grafana 's web interface is accessible via port 3000. Login/password: admin/admin.

Making Prometheus Monitoring for InterSystems IRIS and Caché Published on InterSystems Developer Community (https://community.intersystems.com)

localhost:3000/login			☆ 自 →	ŀ ♠	9	* -	
	Grafa	na					
	Log in	Sign up					
	User admin Password •••••						
	Forgot your pass	word?					

A detailed instruction on making Prometheus work with Grafana is available <u>here</u>. In short, we need to add a new Data Source of the Prometheus type. Select your option for direct/proxy access:

Contemporaries and the second									
🧔 • 😡 Dat	ta Sources								
Add dat									
Config	Config Dashboards								
Name	Prometheus 🕕 Default								
Туре	Prometheus								
Http setting	zs								
Url	http://localhost:9090								
Access	proxy 👻 🎯								
Http Auth									
Basic Auth	With Credentials								
TLS Client Auth	With CA Cert 📵								
Add	Cancel								

Once done, we need to add a dashboard with the necessary panels. The test sample of a dashboard is <u>publicly</u> <u>available</u>, along with the code of the metrics collection class. A dashboard can be simply imported to Grafana (Dashboards Import):

🔹 Import Da	shboard		×
Options			
Name	ISC Cache		*
Fedora	Prometheus	-	*
🖺 Save & Open	Cancel Back		

## We'll get the following after import:



Save the dashboard:



Time range and update period can be selected in the top right corner:

<b>@</b> -	🗱 ISC Cache 🗸	☆	C	B	٥			Zoom Out	: 👌 🕑 Last 3 hou	<b>ırs</b> Refresh every 5s	ខ
						Time range From: now-3h To: now Refreshing every: 5s	Quick rang Last 7 days Last 30 days Last 60 days Last 90 days Last 6 months Last 1 year Last 2 years Last 5 years	ES Yesterday Day before yesterday This day last week Previous week Previous month Previous year	2 Today Today so far This week This week so far This month This year	016-12-31 17:11:53 to 016-12-31 20:11:53 Last 5 minutes Last 15 minutes Last 30 minutes Last 1 hour Last 6 hours Last 12 hours Last 12 hours Last 24 hours	

Examples of monitoring types

Let's test the monitoring of calls to globals:



We can see that the number of references to globals per second has increased, while cache efficiency dropped (the ^Prometheus global hasn ' t been cached yet):



Let's check our license usage. To do this, let's create a primitive CSP page called PromTest.csp in the USER namespace:

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

And visit it so many times (we assume that the /csp/user application is not password-protected):

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

We'll see the following picture for license usage:



#### Conclusions

As we can see, implementing the monitoring functionality is not hard at all. Even after a few initial steps, we can get important information about the work of the system, such as: license usage, efficiency of globals caching, application errors. We used the <u>SYS.Stats.Dashboard</u> for this tutorial, but other classes of SYS, %SYSTEM, %SYS packages also deserve attention. You can also write your own class that will supply custom metrics for your own application – for instance, the number of documents of a particular type. Some useful metrics will eventually be compiled into a separate template for Grafana.

#### To be continued

If you are interested in learning more about this, I will write more on the subject. Here are my plans:

- 1. Preparing a Grafana template with metrics for the logging daemon. It would be nice to make some sort of a graphical equivalent of the <u>Amgstat</u> tool at least for some of its metrics.
- 2. Password protection for web applications is good, but it would be nice to verify the possibility of using certificates.
- 3. Use of Prometheus, Grafana and some exporters for Prometheus as Docker containers.
- 4. Use of discovery services for automatically adding new Caché instances to the Prometheus monitoring list. It's also where I'd like to demonstrate (in practice) how convenient Grafana and its templates are. This is something like dynamic panels, where metrics for a particular selected server are shown, all on the same Dashboard.
- 5. Prometheus Alert Manager.
- 6. Prometheus configuration settings related to the duration of storing data, as well possible optimizations for systems with a large number of metrics and a short statistics collection interval.
- 7. Various subtleties and nuances that will transpire along the way.

#### Links

During the preparation of this article, I visited a number of useful sites and watched a great deal of videos:

- Prometheus project website
- Grafana project website
- Blog of one of Prometheus developers called Brian Brazil

- <u>Tutorial on DigitalOcean</u>
- Some videos from Robust Perception
- Many videos from a conference devoted to Prometheus

Thank you for reading all the way down to this line!

<u>#Best Practices</u> <u>#Monitoring</u> <u>#System Administration</u> <u>#Visualization</u> <u>#Caché</u>

Source

URL: https://community.intersystems.com/post/making-prometheus-monitoring-intersystems-iris-and-cach%C3%A9