[New Guide] Download the 2017 Guide to Databases: Speed, Scale, and Security

Download Guide

Calling Stored Procedures From Spring Data JPA

by Martin Farrell MVB · Nov. 30, 15 · Java Zone

Build vs Buy a Data Quality Solution: Which is Best for You? Gain insights on a hybrid approach. Download white paper now!

Consider the following stored procedure:

```
CREATE OR REPLACE PACKAGE test_pkg AS
   PROCEDURE in_only_test (inParam1 IN VARCHAR2);
   PROCEDURE in_and_out_test (inParam1 IN VARCHAR2, outParam1 OUT VARCHAR2);
END test_pkg;
/

CREATE OR REPLACE PACKAGE BODY test_pkg AS
   PROCEDURE in_only_test(inParam1 IN VARCHAR2) AS
   BEGIN
      DBMS_OUTPUT.PUT_LINE('in_only_test');
   END in_only_test;

   PROCEDURE in_and_out_test(inParam1 IN VARCHAR2, outParam1 OUT VARCHAR2) AS
   BEGIN
      outParam1 := 'Woohoo Im an outparam, and this is my inparam ' || inParam1;
   END in_and_out_test;
END test_pkg;
```

Here we have two different procedures:

- in_only_test - takes an input parameter(inParam1), but doesn't return a value.
- in_and_out_test - takes an input parameter(inParam1), and returns a value(outParam1).

We can then call the stored procedures using the @NamedStoredProcedureQueries annotation:

```
@Entity
@Table(name = "MYTABLE")
```

```
@NamedStoredProcedureQueries({
   @NamedStoredProcedureQuery(name = "in_only_test",
                              procedureName = "test_pkg.in_only_test",
                              parameters = {
                                 @StoredProcedureParameter(mode = ParameterMode.IN, name =
                              }),
   @NamedStoredProcedureQuery(name = "in_and_out_test",
                              procedureName = "test_pkg.in_and_out_test",
                              parameters = {
                                 @StoredProcedureParameter(mode = ParameterMode.IN, name =
                                 @StoredProcedureParameter(mode = ParameterMode.OUT, name =
                              })
})
public class MyTable implements Serializable {
}
```

The key points are:

- The Stored Procedure uses the annotation @NamedStoredProcedureQuery and is bound to a JPA table.
- procedureName – This is the name of the stored procedure.
- name – This is the name of the StoredProcedure in the JPA ecosystem.
- Define the IN/OUT parameter using @StoredProcedureParameter.

We then create the Spring Data JPA repository:

```
public interface MyTableRepository extends CrudRepository<MyTable, Long> {
   @Procedure(name = "in_only_test")
   void inOnlyTest(@Param("inParam1") String inParam1);

   @Procedure(name = "in_and_out_test")
   String inAndOutTest(@Param("inParam1") String inParam1);
}
```

The key points are:

- @Procedure – the name parameter must match the name on @NamedStoredProcedureQuery
- @Param – Must match @StoredProcedureParameter name parameter
- Return types must match - so in_only_test is void, and in_and_out_test returns String

We can then call them:

```
// This version shows how a param can go in an be returned from a stored procedure
String inParam = "Hi Im an inputParam";
String outParam = myTableRepository.inAndOutTest(inParam);
Assert.assertEquals(outParam, "Woohoo Im an outparam, and this is my inparam Hi Im an inpu

// This version shows how to call a Stored Procedure which doesnt return any parameter -
myTableRepository.inOnlyTest(inParam);
```

Other Tricks

The wide range of possiblities for stored procedures has resulted in a few occasions when the above approach hasnt worked. I've solved these problems by defining a custom repository to call the stored procedures as a native query.

This is done by defining a custom repository:

```
public interface MyTableRepositoryCustom {
   void inOnlyTest(String inParam1);
}
```

We then make sure our main repository extends this interface:

```
public interface MyTableRepository extends CrudRepository<MyTable, Long>, MyTableRepositor
}
```

We then create our custom repository implementation:

```
public class MyTableRepositoryImpl implements MyTableRepositoryCustom {

   @PersistenceContext
   private EntityManager em;

   @Override
   public void inOnlyTest(String inParam1) {
      this.em.createNativeQuery("BEGIN in_only_test(:inParam1); END;")
          .setParameter("inParam1", inParam1)
          .executeUpdate();
   }

}
```

This can then be called in the normal way:

```
@Autowired
MyTableRepository myTableRepository;

// And to call the method -
myTableRepository.inOnlyTest(inParam1);
```

Build vs Buy a Data Quality Solution: Which is Best for You? Maintaining high quality data is essential for operational efficiency, meaningful analytics and good long-term customer relationships. But, when dealing with multiple sources of data, data quality becomes complex, so you need to know when you should build a custom data quality tools effort over canned solutions. Download our whitepaper for more insights into a hybrid approach.

Like This Article? Read More From DZone

My Favorite Spring Data JPA Feature

Add Custom Functionality to a Spring Data Repository

Spring Data GemFire supports Apache Geode

Free DZone Refcard
Getting Started With Play Framework

Topics: JAVA , SPRING DATA

Published at DZone with permission of Martin Farrell, DZone MVB. See the original article here.
Opinions expressed by DZone contributors are their own.

Java Partner Resources

How To Resolve Network Partitions In Seconds With Lightbend Enterprise Suite
Lightbend

Secure a Spring Microservices Architecture with Spring Security and JWTs
Okta

Migrating to Microservice Databases
Red Hat Developer Program

Advanced Linux Commands [Cheat Sheet]
Red Hat Developer Program

Book Review: Clean Architecture by Robert C. Martin

by Grzegorz Ziemoński MVB · Nov 01, 17 · Java Zone

Learn how to troubleshoot and diagnose some of the most common performance issues in Java today. Brought to you in partnership with AppDynamics.

Uncle Bob is back! His newest book, *Clean Architecture*, was released about a month ago, and it's meant to take your software engineering skills to an even higher level. Leaving all the low-level details far behind, the newest your software engineering skills to an even higher level. Leaving all the low-level details far behind, the newest book puts maximum focus on the fine art of software architecture and attempts to lay out principles for creating successful, long-lasting projects in any kind of environment, at any point in time (yep, really!).

Introduction

The book begins with a gentle introduction to the topic of architecture. Instead of the usual storm of buzzwords present in software architecture texts, Uncle Bob lays out a pragmatic, not-so-sexy goal:

“The goal of software architecture is to minimize the human resources required to build and maintain the required system.”

If you’ve read some of Martin’s previous texts, you won’t be surprised that the way to reach the goal is by keeping the codebase clean, only this time, we’re talking in terms of software architecture.

This, obviously, can set us at odds with those who want the software to earn money “right here, right now.” Here comes the reality of being a software architect: It’s a struggle!

Starting With the Bricks

The next part of the book touches on the topic of programming paradigms and their supposed evolution. After all, how can we talk about any kinds of high-level rules or principles in such a fast-evolving industry as software engineering?

The perspective that Uncle Bob presents is that the area of software engineering has not evolved that much at all over the last 70 years. Yes, we’ve got faster computers, more tools, new languages etc., but the core ideas remain the same. Any major “evolution” at the paradigm level can be presented as a constraint imposed on the preceding paradigm, so we’re actually dealing with less burden than our predecessors.

“Software — the stuff of computer programs — is composed of sequence, selection, iteration, and indirection. Nothing more. Nothing less.”

Design Principles

In the third part of the book, Martin invites us to take a deeper look into the SOLID principles. Instead of treating them as a bunch of class-level design tips, he goes on to extract the general wisdom that they carry around and explains how to apply it to software architecture.

The Single Responsibility Principle tells us to look at the different axis of change in the codebase so that a single piece of code is responsible for fulfilling the needs of a single actor.

The Open/Closed Principle helps us in creating a hierarchy of components in which lower-level components become plugins to the high-level policies.

Liskov’s Substitution Principle once again warns us of the consequences of partial module substitutability.

The Interface Segregation Principle transforms into a rule of not depending on things that we don’t use, not just on code “interface” level.

And finally, the dependency inversion principle becomes the *dependency rule*, which is the backbone of the whole Clean Architecture concept.

Component Principles

Having moved away from the class-level design and principles, we need a more suitable building block for the needs of software architecture. Against the current trend of micro-containero-distributo-somethings, Uncle Bob calls these simply *components*, defined simply as “units of deployment”.

The important part of components being effectively units of deployment is that we need to take extra care when deciding what does and what does not go inside one. The cohesion and coupling are no longer about “doing one thing” or “mixing things together.” There’s a tension between different kinds of benefits achieved by different kinds of separation.

Granularity of releases, code reuse, code stability, and abstractness and others are the main topic of this part, alongside with the rules that help us get the most out of them: Component Cohesion and Coupling Principles.

Architecture

About halfway through the book, we finally arrive at the architecture-architecture part. From this point until the very end of the book, it’s almost all about two complementary topics: high-level policy vs. low-level detail and drawing boundaries.

If you’ve read anything about Clean Architecture before, there won’t be any surprises here. Enterprise business rules should not depend on the user’s use cases. The use cases should not depend on the UI, databases, and such. Whatever you do, follow the *dependency rule* by introducing boundaries between different kinds of building blocks.

Alongside the Clean Architecture concept itself, this part covers, in detail, a variety of topics: how to draw boundaries, the different ways of decoupling and enforcing boundaries, the relationship between layers, boundaries, tests, and so on, so forth.

This is the "meatiest" and most interesting part of the book, in my opinion, so I won’t offer more spoilers here. Go read the book, if you’re interested!

Details

The last part looks a bit like a mix of things that didn’t fit into the previous parts of the book. And so we get a few more chapters about databases, web, and frameworks being low-level details (if you weren’t already converted at that point), a very short case study, and a guest chapter from Simon Brown.

My Opinion

So far, I've done my best to give you an objective description of what’s covered in the book and which thought directions are taken here and there. This was to spark your interest and level your expectations if you haven’t read the book yet. Now I’ll give you a bunch of fully subjective thoughts, which you are fully welcome to ignore or disagree with.

I dislike the book.

I hate to say it, but I was really disappointed when I finished reading. Don’t get me wrong. I had really high hopes for the book. I pre-ordered it the moment I first saw it available on Amazon and then read it on Safari because I didn’t want to wait for the physical copy to arrive. So my opinion doesn't come from any kind of general negative bias toward Uncle Bob or the topic Clean Architecture.

My first issue with the book is that it does not contain much more than you can find in other Uncle Bob's teachings, like articles or videos. While the form a book is certainly a good way to organize information together and present it as a cohesive whole, the lack of novelty is somewhat disappointing for a long-term Uncle Bob fan.

The second issue, a bit connected with the first one, is that, in my opinion, it contains an awful lot of unnecessary noise. The whole walk through the paradigms and reiterating SOLID principles for the millionth time didn't seem like a necessity for me. If there was a limited number of pages for the book, these chapters would be the one to throw away for me.

The third, and probably biggest, issue that I have is that the book feels very disconnected from a day-to-day reality of a software developer. Yes, it tells you that you should keep databases, frameworks, and others at arm's length. Yes, it tells you what kinds of levels can you see in your code and how the dependency rule works between them. But there's close to none information about the "how" of that idea. And in my experience, the how of Clean Architecture is the hard part for aspiring clean coders. What we get here is a super-small, very basic case study that I found more confusing than useful.

Somehow continuing the topic of disconnection, the book also makes close to no reference to other trending ideas in software development like DDD, event sourcing, microservices, etc. Yes, the latter two are mentioned in a few sentences, but that's far from helping people to map ideas to each other and help them get most of everything together. It feels like everything that "counts" is that the book's contents make sense together. The rest of the world is not important.

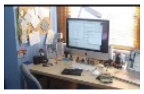
Conclusion

Now, that I've laid out both the objective and subjective things about the book, it's your turn to decide whether to invest your time and read it. My suggestion is that people who haven't read or watched a lot of Uncle Bob should definitely give it a read. You will find a good, cohesive way to think about software development. On the other hand, if you're a long-time Uncle Bob fan (and somehow haven't read it by now?!), you can put it further down your reading list. There's nothing there that you must read right here, right now.

Understand the needs and benefits around implementing the right monitoring solution for a growing containerized market. Brought to you in partnership with AppDynamics.

Like This Article? Read More From DZone

Reactive Clean Architecture With Vert.x


The Single Responsibility Principle Explained

Finding Inner Peace With the Liskov Substitution Principle

Free DZone Refcard
Getting Started With Play Framework

Topics: CLEAN ARCHITECTURE, UNCLE BOB, SOLID PRINCIPLES, JAVA

Opinions expressed by DZone contributors are their own.

Monitoring an Eclipse MicroProfile 1.2 Server With Prometheus

by Heiko Rupp MVB · Nov 01, 17 · Java Zone

Download Microservices for Java Developers: A hands-on introduction to frameworks and containers. Brought to you in partnership with Red Hat.

Eclipse MicroProfile has added a Monitoring specification in its 1.2 release. This allows for a common way of monitoring servers that implement the specification. In this article, you will learn how to monitor MicroProfile 1.2 servers with the popular Prometheus monitoring system.

Overview

I have described the concepts of Eclipse MircoProfile (MP) Monitoring in a previous article—servers expose a basic set of system metrics that are common for each implementation of the MP-Metrics specification. Applications can in addition also make specific metrics available.

The server then exposes the gathered metric data over HTTP(s) endpoints. Monitoring agents can then connect to the server's `/metrics` endpoint and poll the data.

Setting Up the (Server) Runtimes to be Monitored

I'll now show how to set up the runtimes to use MicroProfile metrics. For this, I am showcasing WildFly Swarm and OpenLiberty as two of the early adopters of the Metrics specification. I will start with WildFly Swarm.

WildFly Swarm

WildFly Swarm, or Swarm for short, uses a so-called fat-jar approach: You build your application and then get a jar file that contains the application and all of the server logic, which you then just start via `java -jar application-swarm.jar`. To enable the MicroProfile Metrics to support, you need to pull in the MicroProfile *fraction* in your build.

For Apache Maven users it looks like this:

```
    <dependency>
```

```
<dependency>
<groupId>org.wildfly.swarm</groupId>
    <artifactId>microprofile-metrics</artifactId>
</dependency>
```

After this is done, you just build your project as usual with `mvn install`, which will create the usual uber-jar in a target.

When you run it, you will see a line like the following in the server log:

```
WFSWARM0013: Installed fraction:     Microprofile-Metrics - EXPERIMENTAL    org.wildfly.swa
```

Once the server is ready, it is exposing metrics under `http://localhost:8080/metrics` by default.

OpenLiberty

You can download OpenLiberty from its homepage. After downloading and unpacking it, you need to create a server configuration. Pass the respective template in to obtain a MicroProfile configuration.

```
$ bin/server create  mp --template=microProfile1
```

The `/metrics` endpoint is secured by default on OpenLiberty.

You need to add a small addition to the config file under `usr/servers/mp1/server.xml` inside the `<server>` element.

```
1  quickStartSecurity userName="theUser" userPassword="thePassword"/>
2  <keyStore id="defaultKeyStore" password="Liberty"/>
```

When this is done, you can start the server via `bin/server run mp`. With the standard settings, the metrics can then be found under `https://localhost:9443/metrics`. Credentials are *theUser/thePassword* as seen in line 2 of the above snippet.

Setting Up Prometheus

Now that we have our targets set up to be monitored, we can install Prometheus to monitor them. Prometheus is relatively easy to get going. Just download, unpack and start it. Before you can start it, you need to provide a configuration file. Create a file `prom.yml` with the following content:

```
scrape_configs:
  # Configuration to poll from WildFly Swarm
  - job_name: 'swarm'
    scrape_interval: 15s

    # translates to http://localhost:8080/metrics
    static_configs:
      - targets: ['localhost:8080']

  # Configuration to poll from OpenLiberty
  - job_name: 'liberty'
    scrape_interval: 15s
    scheme: https
    basic_auth:
      username: 'theUser'
      password: 'thePassword'

    tls_config:
      insecure_skip_verify: true

    # translates to https://localhost:9443/metrics
    static_configs:
      - targets: ['localhost:9443']
```

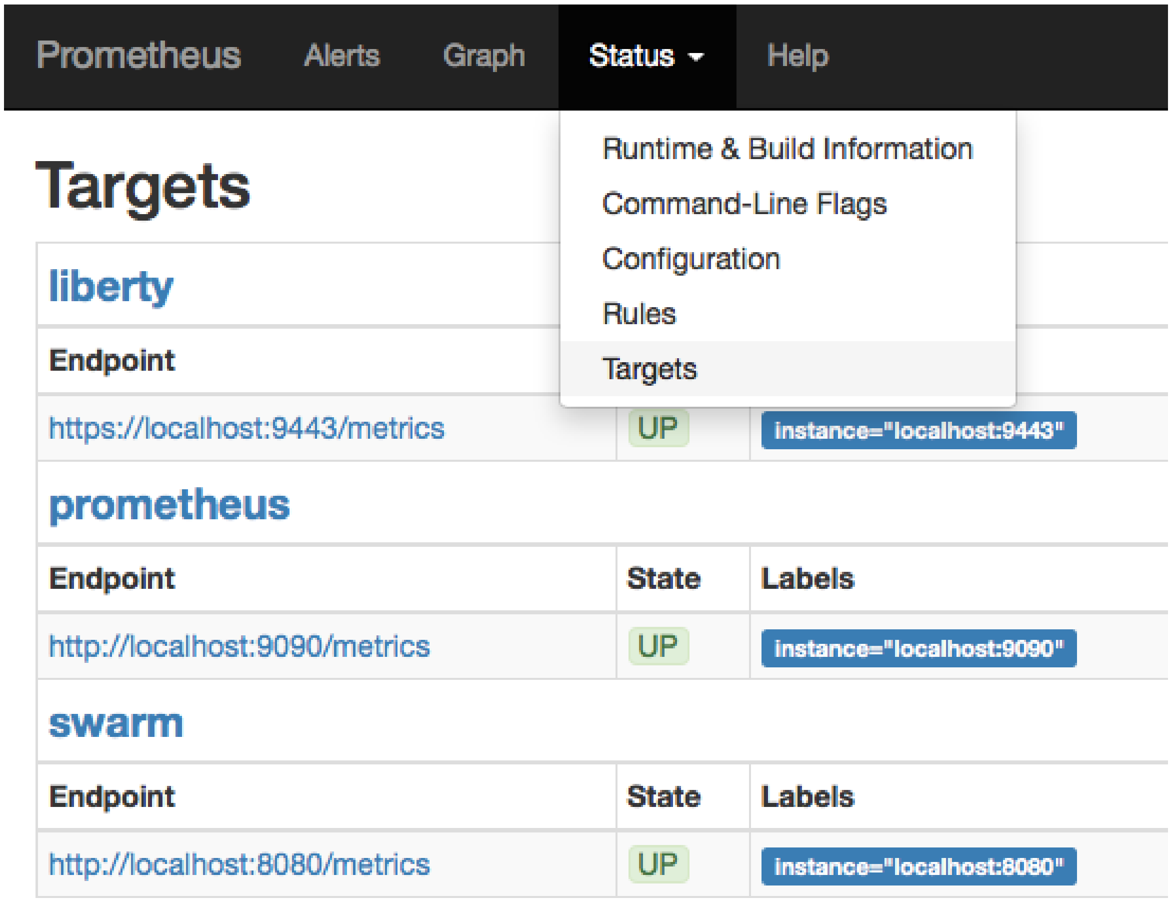
After editing the file, you can start Prometheus via;

```
$prometheus -config.file=prom.yml
```

Prometheus will show a few lines about starting and a few seconds later it is ready.

Head over to your browser and go to `http://localhost:9090/` .

As a first step select Status -> Targets from the menu and make check that both servers are marked as UP.



Targets

liberty

Endpoint	State	Labels
https://localhost:9443/metrics	UP	instance="localhost:9443"

prometheus

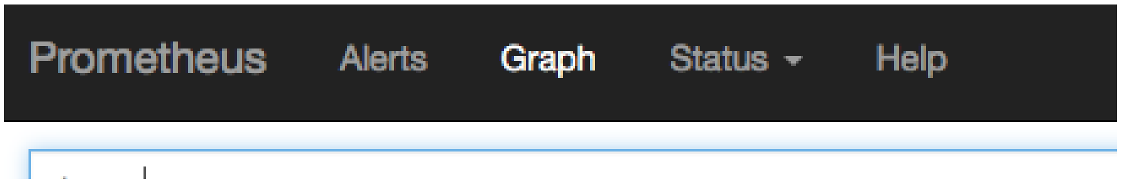
Endpoint	State	Labels
http://localhost:9090/metrics	UP	instance="localhost:9090"

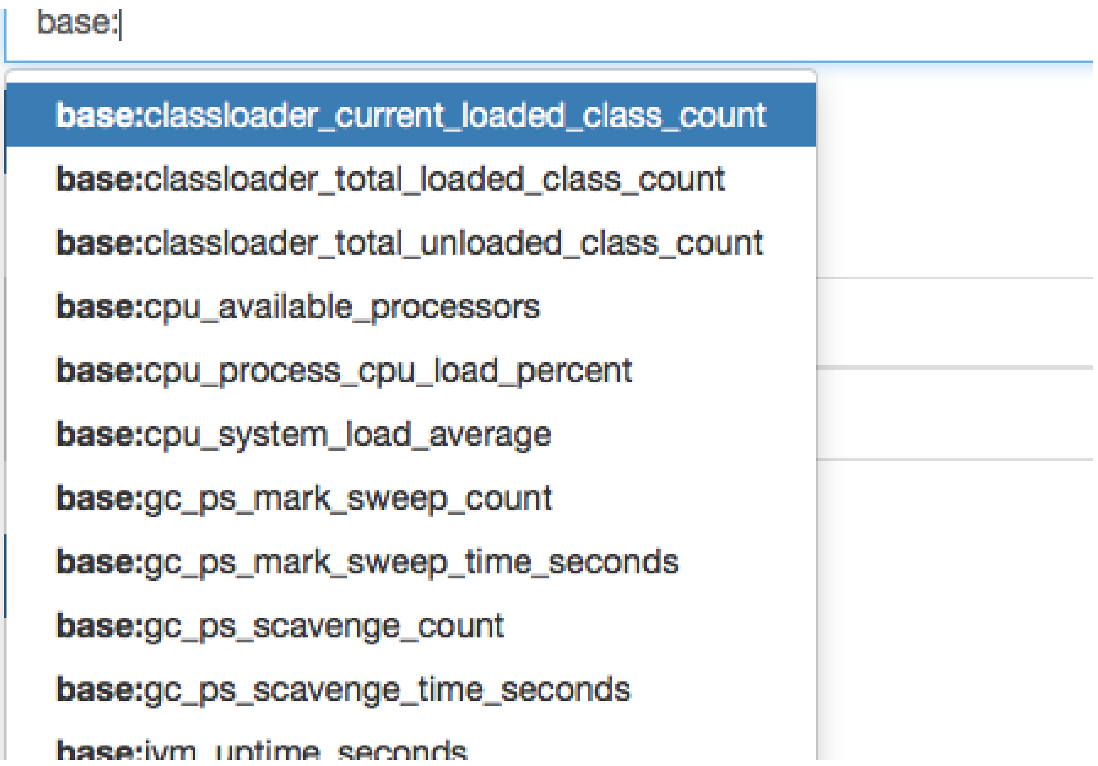
swarm

Endpoint	State	Labels
http://localhost:8080/metrics	UP	instance="localhost:8080"

List of endpoints to be scraped from along with their corresponding status

Chose Metrics for display.



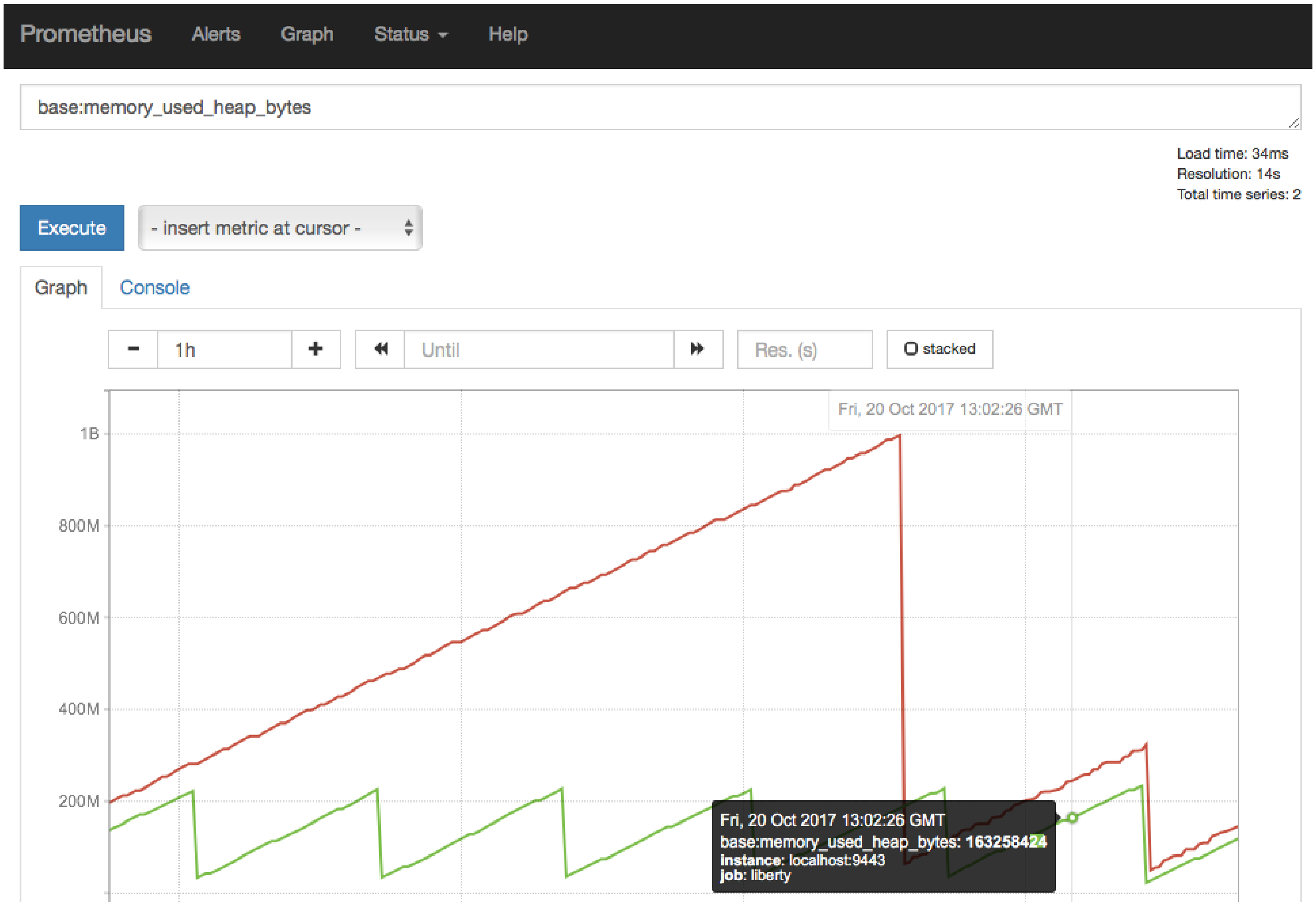


base:

base:classloader_current_loaded_class_count
base:classloader_total_loaded_class_count
base:classloader_total_unloaded_class_count
base:cpu_available_processors
base:cpu_process_cpu_load_percent
base:cpu_system_load_average
base:gc_ps_mark_sweep_count
base:gc_ps_mark_sweep_time_seconds
base:gc_ps_scavenge_count
base:gc_ps_scavenge_time_seconds
base:jvm_uptime_seconds

Prometheus metric selector with a list of base: metrics

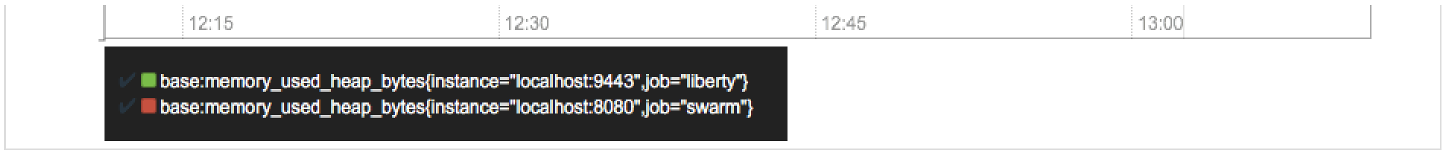
When the servers are running, we are ready to display some metrics (that gets more interesting when you wait a while so that Prometheus has polled more data).



base:memory_used_heap_bytes

Load time: 34ms
Resolution: 14s
Total time series: 2

Fri, 20 Oct 2017 13:02:26 GMT
base:memory_used_heap_bytes: 163258424
instance: localhost:9443
job: liberty



Metrics as shown in Prometheus UI

In the last chart, you can see that we only asked for the metric, `base:memory_used_heap_bytes` but got two graphs, one per MP-server, as both expose the same metric under the same name. Prometheus is adding labels on the fly, which can then be used to distinguish the servers (green is OpenLiberty, brown is Swarm).

Conclusion

MicroProfile Metrics defines a common way to expose metrics from systems. Monitoring tools like Prometheus are thus easily able to monitor those servers in a vendor-independent way.

At the time of this article, the MP-Metrics code may not yet be in a Swarm release, but it will be soon.

Download Building Reactive Microservices in Java: Asynchronous and Event-Based Application Design. Brought to you in partnership with Red Hat.

Like This Article? Read More From DZone

The Monitoring Aspects of Eclipse MicroProfile 1.2

Service Discovery With Wildfly Swarm

Creating Documented REST APIs With Wildfly Swarm

Free DZone Refcard
Getting Started With Play Framework

Topics: JAVA, MICROPROFILE, ECLIPSE, WILDFLY SWARM, APACHE MAVEN, OPENLIBERTY, PROMETHEUS, MONITORING

Published at DZone with permission of Heiko Rupp, DZone MVB. See the original article here.
Opinions expressed by DZone contributors are their own.