AXIOM JDK

Инструкция по настройке мониторинга JVM и Libercat через Grafana

Axiom JDK I Май 2025

Программное обеспечение АКСИОМ содержит программное обеспечение с открытым исходным кодом. Дополнительная информация о коде сторонних разработчиков доступна на сайте https://axiomjdk.ru/third_party_licenses. Для дополнительной информации о том, как получить копию исходного кода, можно обратиться по адресу info@axiomjdk.ru.

ДАННАЯ ИНФОРМАЦИЯ МОЖЕТ ИЗМЕНЯТЬСЯ БЕЗ ПРЕДВАРИТЕЛЬНОГО УВЕДОМЛЕНИЯ. АКСИОМ ПРЕДОСТАВЛЯЕТ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ "КАК ЕСТЬ" БЕЗ КАКИХ-ЛИБО ГАРАНТИЙ, АКСИОМ ПРЯМО ОТКАЗЫВАЕТСЯ ОТ ВСЕХ ПОДРАЗУМЕВАЕМЫХ ГАРАНТИЙ, ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ПОДРАЗУМЕВАЕМЫМИ ГАРАНТИЯМИ ТОВАРНОЙ ПРИГОДНОСТИ И ПРИГОДНОСТИ ДЛЯ ОПРЕДЕЛЕННОЙ ЦЕЛИ.

АКСИОМ НИ ПРИ КАКИХ ОБСТОЯТЕЛЬСТВАХ НЕ НЕСЕТ ОТВЕТСТВЕННОСТИ ЗА ЛЮБЫЕ КОСВЕННЫЕ, СЛУЧАЙНЫЕ, СПЕЦИАЛЬНЫЕ, ШТРАФНЫЕ ИЛИ КОСВЕННЫЕ УБЫТКИ, ИЛИ УБЫТКИ ОТ ПОТЕРИ ПРИБЫЛИ, ДОХОДА, ДАННЫХ ИЛИ ИСПОЛЬЗОВАНИЯ ДАННЫХ, ПОНЕСЕННЫЕ ВАМИ ИЛИ ЛЮБОЙ ТРЕТЬЕЙ СТОРОНОЙ, БУДЬ ТО В РЕЗУЛЬТАТЕ ДЕЙСТВИЯ ДОГОВОРА ИЛИ ДЕЛИКТА, ДАЖЕ ЕСЛИ АКСИОМ БЫЛО ПРЕДУПРЕЖДЕНО О ВОЗМОЖНОСТИ ТАКИХ УБЫТКОВ.

Использование любого программного продукта АКСИОМ регулируется соответствующим лицензионным соглашением, которое никоим образом не изменяется условиями данного уведомления. Программные продукты и фирменные наименования: Axiom JDK, Axiom JDK Pro, Axiom Runtime Container Pro, Axiom Linux, Libercat, Libercat Certified и АКСИОМ принадлежат АКСИОМ и их использование допускается только с разрешения правообладателя.

Товарный знак Linux® используется в соответствии с сублицензией от Linux Foundation, эксклюзивного лицензиата Линуса Торвальдса, владельца знака на всемирной основе. Java и OpenJDK являются товарными знаками или зарегистрированными товарными знаками компании Oracle и/или ее аффилированных лиц. Другие торговые марки являются собственностью их соответствующих владельцев и используются только в целях идентификации.

Содержание

1. Введение

Данный документ описывает настройку и работу с дешбордом для мониторинга виртуальной машины в Axiom JDK и Сервера приложений Axiom Libercat с помощью Grafana и Prometheus. Дешборд содержит основные метрики, необходимые для мониторинга производительности и состояния JVM и сервера приложений. Метрики считываются через JMX.

2. Установка

Grafana

См. подробную инструкцию по установке на странице Install Grafana.

Windows

Выполните следующие шаги, чтобы скачать и установить Grafana.

1. Перейдите на страницу загрузки Grafana.
2. Выберите версию Grafana, которую вы хотите установить в списке **Version**. По умолчанию выбрана самая последняя версия Grafana.
3. Выберите выпуск в списке **Edition**.
    - Версия с открытым исходным кодом - Эта версия функционально идентична корпоративной версии.
    - Корпоративная версия - Функционально она идентична версии с открытым исходным кодом, но включает функции, которые вы можете разблокировать с помощью лицензии.
4. Выберите Windows. Чтобы использовать установщик Windows, выполните следующие действия:
    a. Нажмите Download the installer, чтобы загрузить установщик.
    b. Откройте и запустите установщик.
    c. Следуйте инструкциям на экране окна установщика. Нажмите **Next** для перехода на следующий шаг.

Чтобы установить Grafana из архива, выполните следующие действия:

1. На странице загрузки Grafana выберите Windows и далее нажмите Download the zip file.
2. Извлеките ZIP-файл в любую папку.

    При необходимости сначала разблокируйте файл следующим образом: щелкните загруженный файл правой кнопкой мыши, выберите "Свойства", установите флажок "Разблокировать" и нажмите "OK".

См. страницу Установка на Windows для дополнительной информации.

Linux

Выполните следующие шаги, чтобы скачать и установить Grafana как отдельный пакет на Linux.

1. Перейдите на страницу загрузки Grafana.
2. Выберите версию Grafana, которую вы хотите установить в списке **Version**. По умолчанию выбрана самая последняя версия Grafana.
3. Выберите выпуск в списке **Edition**.
    - Версия с открытым исходным кодом - Эта версия функционально идентична корпоративной версии.
    - Корпоративная версия - Функционально она идентична версии с открытым исходным кодом, но включает функции, которые вы можете разблокировать с помощью лицензии.
4. Нажмите Linux и выберите вашу версию Linux.
5. Скопируйте и вставьте код со страницы загрузки под необходимой версией Linux в командную строку и запустите. Например, чтобы скачать и установить Grafana как отдельный пакет на Linux выполните следующие команды:

```
wget https://dl.grafana.com/oss/release/grafana-11.5.1.linux-amd64.tar.gz
tar -zxvf grafana-11.5.1.linux-amd64.tar.gz
```

См. страницу Установка на Linux для дополнительной информации.

Prometheus

Чтобы установить Prometheus, выполните следующие шаги:

1. Откройте страницу загрузки.
2. Для скачивания выберите и кликните на архив подходящий для вашей операционной системы.
3. Разархивируйте программу в каталог на диске. Например, чтобы разархивировать файл на Linux запустите следующую команду:

```
tar -zxvf prometheus-3.2.0-rc.1.linux-amd64.tar.gz
```

JMX Exporter

Скачайте JMX Exporter на ваш диск: https://github.com/prometheus/jmx_exporter/releases/download/1.1.0/jmx_prometheus_javaagent-1.1.0.jar

3. Конфигурация

JMX Exporter

Создайте файл конфигурации exporter.yaml:

```
rules:

- pattern: ".*"
```

Prometheus

Создайте файл конфигурации prometheus.yaml:

```
global:
  scrape_interval: 5s
  evaluation_interval: 5s

scrape_configs:
  - job_name: 'jmx-monitor'
    scheme: http
    static_configs:
      - targets: ['localhost:9999']
```

Обратите внимание:

- Значение параметра `job_name` "jmx-monitor" будет в дальнейшем использоваться в Grafana для считывания метрик.
- В поле targets указан endpoint JMX Exporter, который повторно будет указан при запуске экспортера (следующий шаг).

> **Примечание:**
>
> Официальная документация по конфигурационному файлу Prometheus: Configuration file.

4. Запуск

Grafana

Запустите Grafana:

```
grafana-server --homepath='/usr/share/grafana'
--config='/usr/share/grafana/conf/defaults.ini'
```

Prometheus

Запустите Prometheus, указав конфигурационный файл `prometheus.yaml` в параметре `--config.file`:

```
prometheus --config.file=prometheus.yaml --web.enable-admin-api
```

JMX Exporter

Запустите приложение вместе с агентом JMX Exporter.

В параметре `-javaagent` укажите `JMX Exporter`, как показано в примере. Также должны быть указаны `endpoint` для считывания метрик и конфигурационный файл exporter.yaml:

```
-javaagent:jmx_prometheus_javaagent
-<VERSION>.jar=[HOSTNAME:]<PORT>:<EXPORTER.YAML>
```

Пример:

```
java -javaagent:jmx_prometheus_javaagent
-1.1.0.jar=localhost:9999:exporter.yaml -XX:NativeMemoryTracking=summary -jar
test.jar
```

Примечание:

Ключ `-XX:NativeMemoryTracking=summary` нужен для работы графиков, отслеживающих native memory.

Для запуска Libercat с JMX Exporter, параметры должны быть переданы в переменной окружения CATALINA_OPTS:

```
export CATALINA_OPTS="-javaagent:jmx_prometheus_javaagent-
1.1.0.jar=localhost:9999:exporter.yaml -XX:NativeMemoryTracking=summary" &&
./startup.sh
```

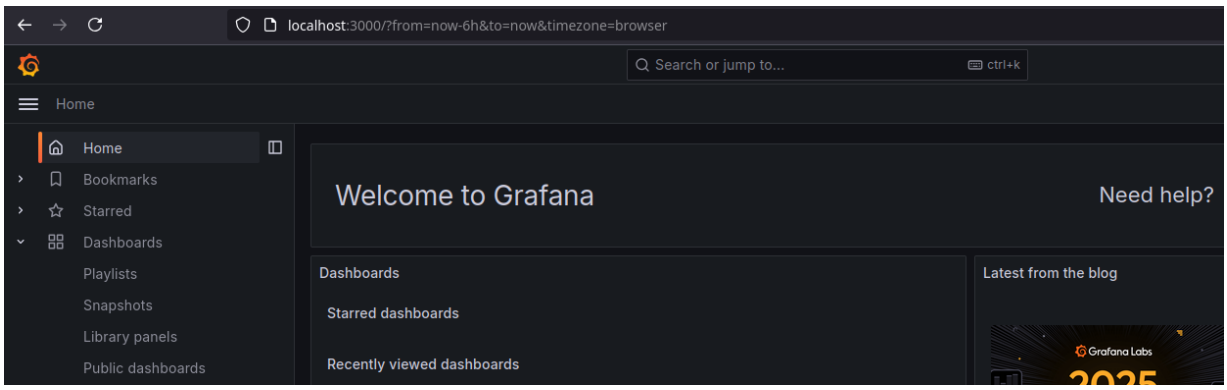
Примечание:

Официальная документация по запуску JMX Exporter: JMX Exporter.

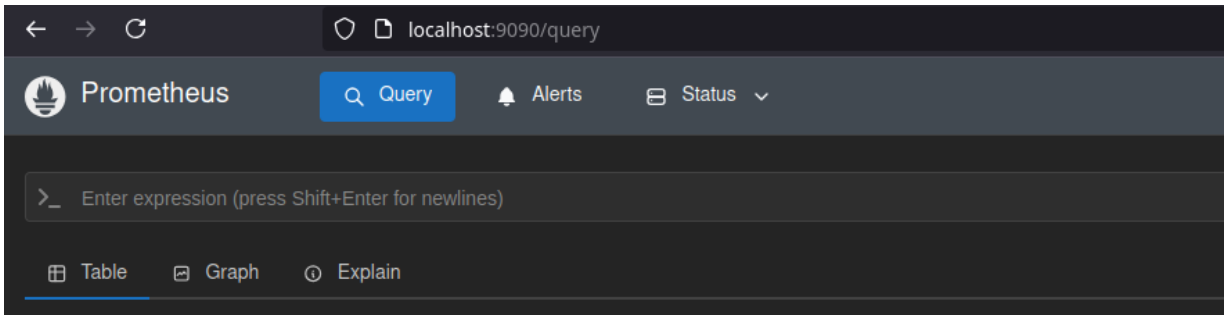
Проверка

На данном шаге должны быть доступны следующие URL:

- Grafana: http://localhost:3000



- Prometheus: http://localhost:9090



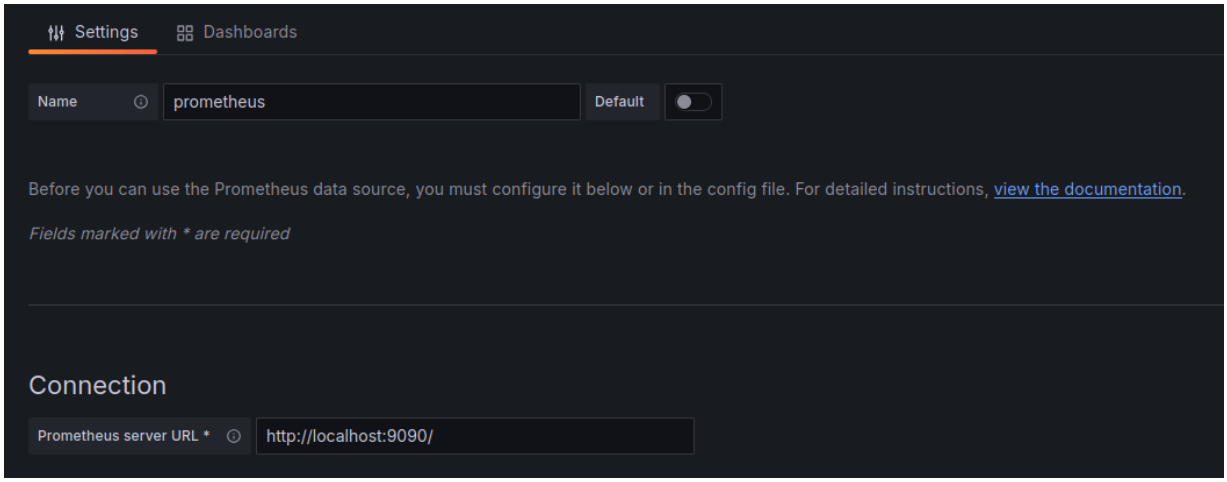
- JMX Exporter: http://localhost:9999/metrics


localhost:9999/metrics
HELP Catalina_Connector_acceptCount The accept count for this Connector Catalina:name=null,type=Connector,attribute=acceptCount
TYPE Catalina_Connector_acceptCount untyped
Catalina_Connector_acceptCount{port="8080"} 100.0
HELP Catalina_Connector_allowTrace Allow disabling TRACE method Catalina:name=null,type=Connector,attribute=allowTrace
TYPE Catalina_Connector_allowTrace untyped
Catalina_Connector_allowTrace{port="8080"} 0.0
HELP Catalina_Connector_connectionLinger Linger value on the incoming connection Catalina:name=null,type=Connector,attribute=connectionLinger
TYPE Catalina_Connector_connectionLinger untyped
Catalina_Connector_connectionLinger{port="8080"} -1.0
HELP Catalina_Connector_connectionTimeout Timeout value on the incoming connection Catalina:name=null,type=Connector,attribute=connectionTimeout
TYPE Catalina_Connector_connectionTimeout untyped
Catalina_Connector_connectionTimeout{port="8080"} 20000.0
HELP Catalina_Connector_enableLookups The 'enable DNS lookups' flag for this Connector Catalina:name=null,type=Connector,attribute=enableLookups
TYPE Catalina_Connector_enableLookups untyped
Catalina_Connector_enableLookups{port="8080"} 0.0
HELP Catalina_Connector_keepAliveTimeout The number of milliseconds Tomcat will wait for a subsequent request before closing the connection Catalina:name=null,type=Connector,attrib
TYPE Catalina_Connector_keepAliveTimeout untyped
Catalina_Connector_keepAliveTimeout{port="8080"} 20000.0
HELP Catalina_Connector_localPort The port number on which this connector is listening to requests. If the special value for port of zero is used then this method will report the a
Catalina:name=null,type=Connector,attribute=localPort
TYPE Catalina_Connector_localPort untyped
Catalina_Connector_localPort{port="8080"} 8080.0

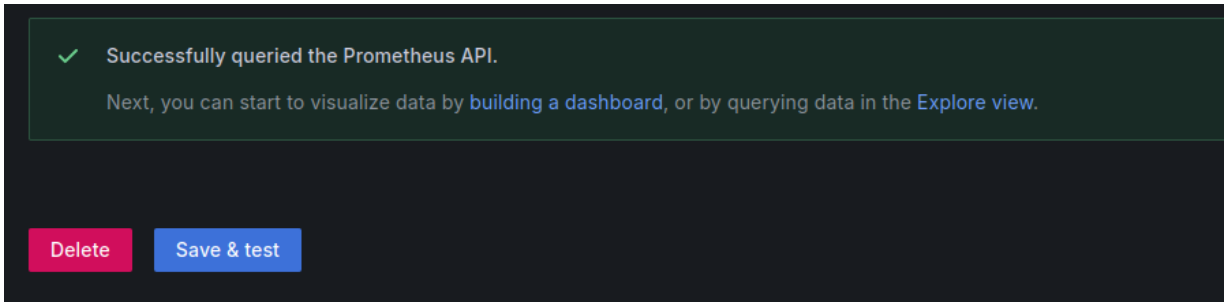
5. Настройка Grafana

Data source

1. Добавьте Prometheus data source, нажав **Add new data source** на данной странице: http://localhost:3000/connections/datasources/prometheus



2. В поле **Prometheus server URL** укажите http://localhost:9090 и нажмите **Save & test**. Должен отобразиться успешный статус "Successfully queried the Prometheus API."

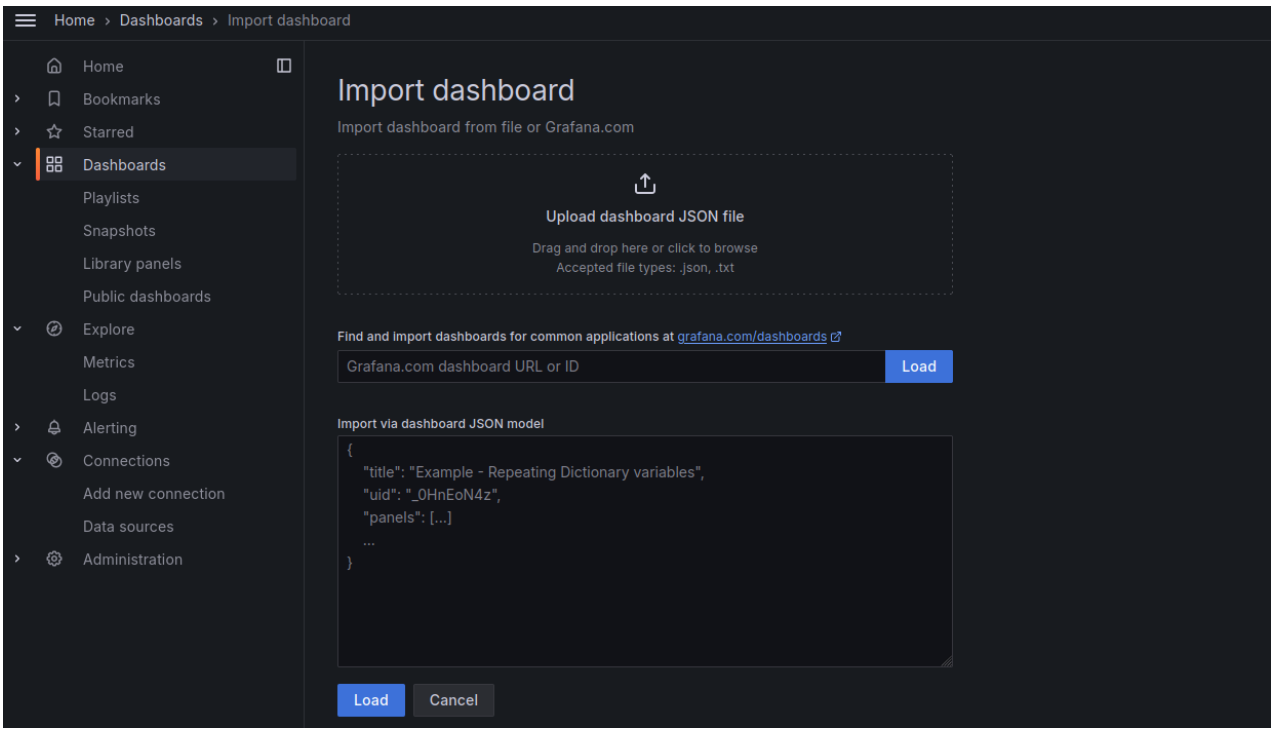


Примечание:

Официальная документация по конфигурации Prometheus data source: Configure Prometheus.

Импорт Dashboard

1. Откройте страницу импорта Dashboards: http://localhost:3000/dashboard/import
2. Выберите и загрузите JSON файл необходимого дешборда, нажав **Upload dashboard JSON file**.
    - Для JVM: https://download.axiomjdk.ru/grafana/jmx-dashboard-jvm.json
    - Для Libercat: https://download.axiomjdk.ru/grafana/jmx-dashboard-libercat.json



> **Примечание:**
>
> Официальная страница Grafana по импорту дешбордов: Import dashboards.

Использование

Откройте дешборд. В нем должны отображаться статус "up" и графики с данными.

- Для JVM: http://localhost:3000/d/228af674/jmx-dashboard-jvm
- Для Libercat: http://localhost:3000/d/4dfa6a648/jmx-dashboard-libercat

Примечание:

Официальная документация Grafana по настройке дешбордов: Dashboards.

6. Описание графиков

Operating System

Поле **Status** отображает, запущена ли JVM. **Start time** и **Uptime** отображают, сколько времени прошло с момента запуска. В графиках используются метрики `java_lang_Runtime_Uptime` и `java_lang_Runtime_StartTime`.

System Load Average показывает среднее значение загрузки системы с помощью метрики Load Average. Метрика показывает усредненное за одну минуту число процессов, ожидающих выполнения или работающих на доступных процессорах. Интерпретация значения зависит от соседних значений (как меняется нагрузка во времени). Также высокая нагрузка характеризуется превышением Load Average числа доступных CPU cores в системе. В графике используется JVM метрика `java_lang_OperatingSystem_SystemLoadAverage`.

Open File Descriptors показывает число файлов, открытых в JVM. По колебаниям графика можно оценивать изменение нагрузки на чтение/запись с жесткого диска. В графике используется JVM метрика `java_lang_OperatingSystem_OpenFileDescriptorCount`.

Garbage Collector

График **GC Count** отображает количество процессов сборщика мусора, запущенных с момента старта приложения. Резкие колебания на графике могут свидетельствовать об "узких местах" в производительности приложения. В графике используется JVM метрика `jvm_gc_collection_seconds_count`.

GC time показывает посекундную частоту выполнения сборщика мусора за последнюю минуту (на сколько увеличивается общее время выполнения GC каждую секунду, в расчете в среднем за последнюю минуту). Колебания графика говорят об уменьшении или увеличении частоты использования сборщика мусора, и могут сигнализировать об узких местах в производительности приложения. В графике используется JVM метрика `jvm_gc_collection_seconds_sum`.

Memory Pools

График **Memory Pool Used** показывает, сколько байт задействуют пулы памяти. С его помощью можно отслеживать использование памяти и в случае утечек определять, в каком конкретно пуле памяти могла возникнуть проблема. В графике используется JVM метрика `jvm_memory_pool_used_bytes`.

График **Memory Pool Committed** показывает, сколько доступной памяти JVM гарантирует для каждого пула памяти. С его помощью можно отслеживать использование памяти и регулировать такие параметры, как размер выделенной памяти или кучи (Xmx) и параметры сборщика мусора (к примеру, `G1MaxNewSizePercent`). В графике используется JVM метрика `jvm_memory_pool_committed_bytes`.

График **Memory Heap / NonHeap Used** повторяет график Memory Pool Used с той разницей, что пулы памяти объединены в две категории - Heap и NonHeap. Это дает обобщенный взгляд на пулы памяти и позволяет регулировать параметры, связанные с ними. На графике использованы метрики `java_lang_Memory_HeapMemoryUsage_used` и `java_lang_Memory_NonHeapMemoryUsage_used`. Аналогично, график **Memory Heap / NonHeap Committed** обобщает график **Memory Pool Committed**. На графике использованы метрики `java_lang_Memory_HeapMemoryUsage_committed` и `java_lang_Memory_NonHeapMemoryUsage_committed`.

Native Memory

Для работы графиков данной секции требуется запустить приложение с параметром `-XX:NativeMemoryTracking=summary`.

График **Native Memory Committed** отображает память, гарантированную для всех пулов JVM. Можно использовать для детального анализа, когда нужно отслеживать всю память, выделенную для JVM, включая нативные пулы (code cache, metaspace, string pool и другие). В графике используется JVM метрика `jvm_native_memory_committed_bytes`.

График **Native Memory Reserved** отображает память, потенциально доступную для всех пулов JVM. График можно использовать для регулирования параметров доступной памяти JVM (Xmx, MaxDirectMemorySize, ReservedCodeCacheSize и другие). В графике используется JVM метрика **jvm_native_memory_reserved_bytes**.

Threading

График **Threads State** отображает количество потоков в каждом из состояний - NEW, RUNNABLE, BLOCKED, WAITING, TIMED_WAITING и TERMINATED. График может помочь в выявлении проблем, связанных с блокировкой потоков, и "узких мест" в производительности. В графике используется JVM метрика `jvm_threads_state`.

График **Threads Used** отображает количество фоновых потоков в сравнении с основными. Высокое или низкое количество фоновых потоков может свидетельствовать о возможностях оптимизации, чрезмерном или, наоборот, недостаточном использовании ресурсов. В графике используются JVM метрики `java_lang_Threading_ThreadCount` и

`java_lang_Threading_DaemonThreadCount`.

ClassLoading

График **Class Loaded / Unloaded** отображает количество загруженных и выгруженных классов в JVM. Позволяет выявлять потенциальные проблемы с загрузкой классов или использованием памяти, оптимизировать загрузку классов, уменьшать количество загружаемых классов. В графике используются JVM метрики `java_lang_ClassLoading_LoadedClassCount` и `java_lang_ClassLoading_UnloadedClassCount`.

Catalina Requests

График **Request Processor Bytes** отображает общее количество полученных и отправленных байт. График отслеживает только байты с данными, исключая заголовки и другие non-payload данные. График поможет в мониторинге нагрузки Axiom Libercat. В графике используются Catalina метрики `Catalina_GlobalRequestProcessor_bytesReceived` и `Catalina_GlobalRequestProcessor_bytesSent`.

График **Requests Count** отображает число обработанных запросов, а также число ошибок при обработке запросов. График поможет проанализировать нагрузку на сервер. В графике используются Catalina метрики `Catalina_GlobalRequestProcessor_requestCount` и `Catalina_GlobalRequestProcessor_errorCount`.

График **Request Processing Time, avg** отображает среднее время, понадобившееся Axiom Libercat, чтобы обработать все поступающие запросы. График поможет обнаружить узкие места в производительности сервера. В графике используется отношение метрик `Catalina_GlobalRequestProcessor_processingTime` и `Catalina_GlobalRequestProcessor_requestCount`.

Catalina Threads

График **Catalina ThreadPool** отображает число свободных и занятых потоков в пуле потоков, используемом для обработки поступающих запросов. Позволяет отслеживать загруженность сервера. В графике используются следующие Catalina метрики `Catalina_ThreadPool_currentThreadCount`, `Catalina_ThreadPool_currentThreadsBusy`, `Catalina_ThreadPool_connectionCount` и `Catalina_ThreadPool_keepAliveCount`.

График **MaxThreads** отображает максимальное количество потоков, которое может быть создано (параметры `current` и `current busy` из графика **Catalina ThreadPool** не могут превышать значения Max Threads). В графике используется Catalina метрика

`Catalina_ThreadPool_maxThreads`.

График **Min Spare Threads** отображает минимальное число потоков, которые постоянно поддерживаются активными (параметр `current` из графика **Catalina ThreadPool** по умолчанию равен значению `Min Spare Threads`). В графике используется Catalina метрика `Catalina_ThreadPool_minSpareThreads`.

График **Accept Count** указывает максимальную длину очереди входящих запросов, когда все потоки, обрабатывающие запросы, заняты. В графике используется Catalina метрика `Catalina_ThreadPool_acceptCount`.

График **Poller Thread Count** отображает количество потоков, занимающихся распределением входящих соединений среди рабочих потоков, которые их обрабатывают (задается атрибутом acceptorThreads в конфигурации Connector). В графике используется Catalina метрика `Catalina_ThreadPool_pollerThreadCount`.

Catalina Servlets

На графике **Top 10 servlets average processing time per request** указаны 10 сервлетов с наибольшим средним временем обработки запросов. График полезен для понимания нагрузки на приложение и определения узких мест в производительности. В графике используются Catalina метрики `Catalina_Servlet_processingTime` и `Catalina_Servlet_requestCount`.

На графике **Top 10 servlet error count** указаны 10 сервлетов с наибольшим количеством ошибок. График полезен для отслеживания ошибок в работе приложения. В графике используется Catalina метрика `Catalina_Servlet_errorCount`.

На графике **Top 10 servlet request count** указаны 10 сервлетов с наибольшим количеством запросов к ним. График полезен для отслеживания нагрузки на приложение и определения узких мест в производительности. В графике используется Catalina метрика `Catalina_Servlet_requestCount`.

Catalina Sessions

График **Top 10 Session Count** отображает 10 страниц с наибольшим количеством пользовательских сессий. График полезен для понимания, как пользователи используют приложение, а также для отслеживания нагрузки на сервер. В графике используется Catalina метрика `Catalina_Manager_sessionCounter`.

График **Top 10 Expired / Rejected Sessions** отображает 10 узлов с наибольшим количеством истекших и отклоненных сессий - по 5 на каждую метрику. График полезен для отслеживания аномалий в использовании приложения. В графике используются Catalina метрики

`Catalina_Manager_expiredSessions` и `Catalina_Manager_rejectedSessions`.

7. Добавление метрик

Полный список метрик, доступных для использования в Grafana, представлен на странице JMX Exporter, указанной в настройках javaagent. Например, при данной конфигурации:

```
-javaagent:jmx_prometheus_javaagent-1.1.0.jar=localhost:9999:exporter.yaml
```

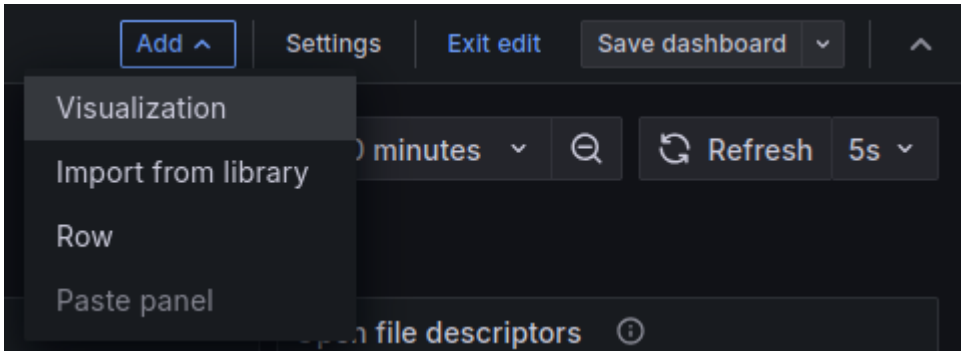
Метрики будут отображаться на странице http://localhost:9999/metrics.

Метрики представлены в следующем формате:

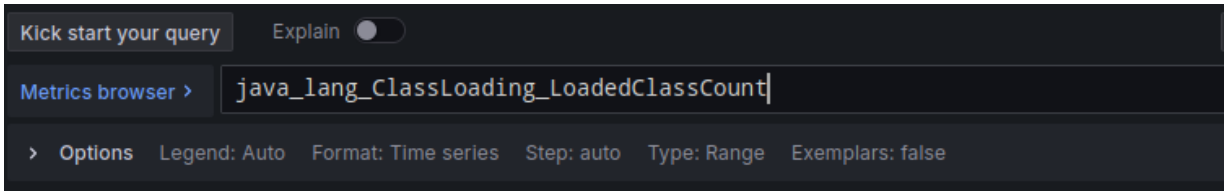
```
# HELP Users_UserDatabase_readonly No persistent save of the user database
Users:name=null,type=UserDatabase,attribute=readonly
# TYPE Users_UserDatabase_readonly untyped
Users_UserDatabase_readonly{database="UserDatabase"} 1.0
# HELP Users_UserDatabase_writable Check if user database is writable
Users:name=null,type=UserDatabase,attribute=writable
# TYPE Users_UserDatabase_writable untyped
Users_UserDatabase_writable{database="UserDatabase"} 1.0
# HELP java_lang_ClassLoading_LoadedClassCount
java.lang:name=null,type=ClassLoading,attribute=LoadedClassCount
# TYPE java_lang_ClassLoading_LoadedClassCount untyped
java_lang_ClassLoading_LoadedClassCount 5283.0
```

Для добавления графика:

1. Откройте дешборд и нажмите в правом верхнем углу **Add** > **Visualisation**:



2. Добавьте метрику в поле **Enter a PromQL query**. Например, для добавления метрики `java_lang_ClassLoading_LoadedClassCount` укажите:



Совет:

Дополнительную информацию о метриках можно найти на страницах документации Java Beans, из которых та или иная метрика экспортирована. К примеру, информацию о метрике `java_lang_OperatingSystem_SystemLoadAverage` можно найти поиском по словосочетанию "SystemLoadAverage" на странице OperatingSystemMXBean.

Для полной информации о панелях и визуализациях Grafana, см. официальную документацию.

Инструкция по настройке мониторинга JVM и Libercat через Grafana