ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛАЭКОНОМИКИ»

*Высшая школа бизнеса*

Волков Андрей Андреевич

**РАЗРАБОТКА СИСТЕМЫ ДЛЯ ПОСТРОЕНИЯ МОНИТОРИНГА,**

**АНАЛИЗА АНОМАЛИЙ И СВОЕВРЕМЕННОГО ПРЕДУПРЕЖДЕНИЯ**

**ДЛЯ СОВРЕМЕННЫХ IT КОМПАНИЙ**

Выпускная квалификационная работа

по направлению подготовки 38.03.05 Бизнес-информатика

образовательная программа «Бизнес-информатика»

|  |  |
| --- | --- |
|  | Научный руководитель |
|  | к.т.н, доцент |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | С. Г. Ефремов |

Москва 2021

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc39691355)

[ВВЕДЕНИЕ. 4](#_Toc39691356)

[Актуальность. 4](#_Toc39691357)

[Цель исследования. 5](#_Toc39691358)

[Задачи исследования. 5](#_Toc39691359)

[Методы. 6](#_Toc39691360)

[ГЛАВА I. Предметная область мониторинга бизнес-приложений. 7](#_Toc39691361)

[1 Мониторинг бизнес-приложений. 16](#_Toc39691370)

[1.1 Используемая терминология. 16](#_Toc39691371)

[1.2 Задачи систем монитоирнга. 18](#_Toc39691372)

[1.3 Компоненты системы мониторинга. 19](#_Toc39691373)

[2 Работа с метриками. 16](#_Toc39691370)

[2.1 Симптомы и причины. 16](#_Toc39691371)

[2.2 Методы черного и белого ящика. 18](#_Toc39691372)

[2.3 4 золотых сигнала. 18](#_Toc39691372)

[2.4 Перцентили и работа с выбросами в метриках. 19](#_Toc39691373)

[2.5 Уровень детализации измерений. 19](#_Toc39691373)

[2.6 Долгосрочное наблюдение. 19](#_Toc39691373)

[2.7 Раскрытие понятия SRE и соглашения: SLA / SLO / SLI. 19](#_Toc39691373)

[2.8 Инфраструктура как код (Infrastructure as a code). 19](#_Toc39691373)

[3 Архитектура мониторинговых систем. 16](#_Toc39691370)

[3.1 Инструменты для экспорта и сбора метрик приложения 16](#_Toc39691371)

[3.2 Метки в метриках 18](#_Toc39691372)

[3.3 Хранение метрик временных рядов 19](#_Toc39691373)

[3.4 Существующие системы мониторинга бизнес-приложений 19](#_Toc39691373)

[ГЛАВА 2. Архитектура сервиса 16](#_Toc39691369)

[ЗАКЛЮЧЕНИЕ 46](#_Toc39691395)

[СПИСОК ЛИТЕРАТУРЫ 47](#_Toc39691396)

# ВВЕДЕНИЕ

***Актуальность***

В современном мире все больше компаний переходят на онлайн модель ведения бизнеса, сегодня бизнес в любой сфере – начиная с новостей, кино, обучения и заканчивая банками, фондовыми биржами – более удобен для клиентов в онлайн, чем в оффлайн формате. За каждым таким IT бизнесом стоит отдельная группа приложений, которые, общаясь между собой и сторонними системами, обеспечивают работу современных IT компаний.

Однако далеко не всегда такую работу можно назвать надежной, в современном мире большое количество проблем, связанных с доступностью приложений и с корректностью выполнения их бизнес-логики. Ежедневно сотни компаний по всему миру испытывают проблемы с доступностью своих сервисов.

По данным Облачной панели состояния Google (англ. Google Cloud Status Dashboard) [1], в компании Google за последний год было выявлено более 150 инцидентов в различных сервисах компании. Наиболее критичный сбой произошел 14 декабря 2020 года, когда из-за строгого квотирования внутреннего хранилища все сервисы, требующие аутентификации Google были недоступны на протяжении 50 минут. По данным блога компании Яндекс, 5 февраля 2020 года, во время проведения регулярных работ, в компании произошел сетевой сбой, который длился 2 часа 30 минут и стал причиной отказа большинства сервисов Яндекса [3].

Каждый сбой приносит компаниям большие убытки, накладывает негативный отпечаток на репутацию бренда. Однако подобных проблем можно избежать благодаря мониторингу. Мониторинг – это отдельная система постоянного наблюдения за работой бизнес-приложений, призванная агрегировать количественные показатели сервисов и вовремя сообщать о потенциальных угрозах.

Не удивительно, что сегодня почти любая, даже небольшая ИТ компания, имеет свою систему мониторинга. Постоянное наблюдение за различными количественными показателями системы, анализ их значений и своевременное предупреждение критически важно для любого бизнеса в сфере информационных технологий.

***Цель работы***

Целью данной работы является разработка системы для построения мониторинга, анализа аномалий и своевременного предупреждения со следующими свойствами:

1. Программный API для создания и экспорта метрик
2. Развертывание облачной инфраструктуры для сбора и построения графиков с метриками
3. Реализация алгоритмов для анализа аномалий во временных рядах и экспорт метрик с аномалиями
4. Построение графиков в пользовательском интерфейсе посредством программного API
5. Пользовательский интерфейс для отображения графиков с метриками приложения
6. Система предупреждения, отправляющая уведомления о сбоях в популярные мессенджеры

***Объект и предмет исследования***

Объект исследования: экосистема мониторинга приложения.

Предмет исследования: API для построения метрик, алгоритмы для анализа аномалий, система оповещения о сбоях.

***Задачи*:**

* Изучить предметную область мониторинга бизнес-приложений;
* Изучить существующие подходы к реализации мониторинга в приложениях и существующие инфраструктурные решения;
* Определить требования к системе для построения мониторинга со стороны конфигурации метрик, анализа аномалий, пользовательского интерфейса, инфраструктуры;
* Изучить алгоритмы для анализа аномалий во временных рядах;
* Разработать API для написания и конфигурации метрик;
* Развернуть инфраструктуру для экспорта, сбора и отображения метрик в режиме реального времени;

***Методы***

В рамках работы будут проанализированы работы на тему мониторинга и анализа аномалий, а также использован опыт крупных компаний из различных IT сфер. На основе этих знаний будут составлены требования к системе и определены ограничения к ее функционалу. Описанная система будет разработана и задокументирована.

# Глава 1. Предметная область мониторинга бизнес-приложений

**1.1. Мониторинг бизнес-приложений**

**1.1.1 Используемая терминология**

**Мониторинг** – сбор, обработка, агрегирование и отображение в реальном времени количественных показателей системы, например общее число и тип запросов, количество ошибок и их типы, время обработки запросов и время функционирования серверов.

**Оповещение** – сообщения, на которые должен обратить внимание человек и которые направляются в конкретную систему, например в очередь запросов («тикетов»), в электронную почту или на специальное устройство — пейджер.

**SRE** (Site reliability engineering) - это дисциплина, которая включает аспекты разработки программного обеспечения и применяет их к инфраструктуре и операционным проблемам.

**Приложение** (сервис) – компьютерная программа, написанная программистом, которая подчиняется, заложенной в нее бизнес-логике и удовлетворяет потребности клиентов.

**Метрика** (показатель) – наблюдаемая количественная характеристика приложения.

* + 1. **Задачи системы мониторинга**

В современных ИТ компаниях система мониторинга критически важна для стабильной работы сервисов компании. Благодаря мониторингу владельцы сервиса могут принимать рациональные решения о влиянии изменений на приложение, грамотно и заблаговременно реагировать на критические ситуации и обосновывать необходимость самого сервиса: измерять и оценивать, насколько он соответствует бизнес-потребностям.

В книге от ведущих инженеров компании Google «Site reliability engineering» выделяются основные задачи, которые должна решать любая качественная система мониторинга:

* *Анализ долгосрочных тенденций.* Мониторинг позволяет оценивать тренды в долгосрочной перспективе, например, сравнение размера базы данных сегодня и полгода назад даст представление о скорости ее заполняемости, а сравнение кол-ва обработанных запросах может сказать росте или снижении популярности приложения.
* *Сравнение с предыдущими версиями или экспериментальными группами.* Сравнение производительности разных версий приложения позволит сопоставить значения ключевых метрик приложения и отдать предпочтение той или оной технологии / методу. Например, новая версия веб-фреймворка может работать медленнее предыдущей версии, что повлияет на решение «откатиться» на предыдущую версию, пока не будет исправлена новая.
* *Оповещение.* Наблюдение за ключевыми метриками доступности приложения может сообщить о сбое или о его предпосылках, что позволит вовремя оповестить инженеров о неисправности и спровоцировать реакцию на инцидент. Например, сервис начали больше использовать, и он стал потреблять близкое к пороговому значению кол-во ресурсов, в таком случае инженеры должны вмешаться в ситуацию и либо снизить нагрузку на сервис, либо увеличить кол-во доступных ресурсов для приложения.
* *Создание информационных панелей.* Любую метрику – будь то техническая метрика или бизнес-метрика – можно интегрировать в информационную панель. Данная панель должна содержать ответы на главные вопросы о работе сервиса. Например, на панели могут быть изображены графики скорости обработки запросов или их классификация по бизнес-признакам.
* *Ретроспективный анализ различного назначения.* Ретроспективный анализ позволяет выявлять причинно-следственные связи между различного рода событиями. Например, время ответа сервиса резко возросло вдвое, в то же время резко возросла скорость выполнения запросов в базу данных, в то же время увеличилось кол-во ошибок при попытке установить соединения с базой данных, что свидетельствует о проблемах на сети. Значит, если устранить первоначальную проблему, то можно решить ее следствие.

Исходя из поставленных задач, систему мониторинга можно рассматривать как инструмент, которым пользуются и разработчики, и аналитики, и владельцы продукта. Грамотно построенная система мониторинга должна давать исчерпывающее представление о состоянии работы сервиса, делать процессы прозрачными, доступными для анализа.

* + 1. **Компоненты системы мониторинга**

На рисунке 1 изображены компоненты современной системы мониторинга.

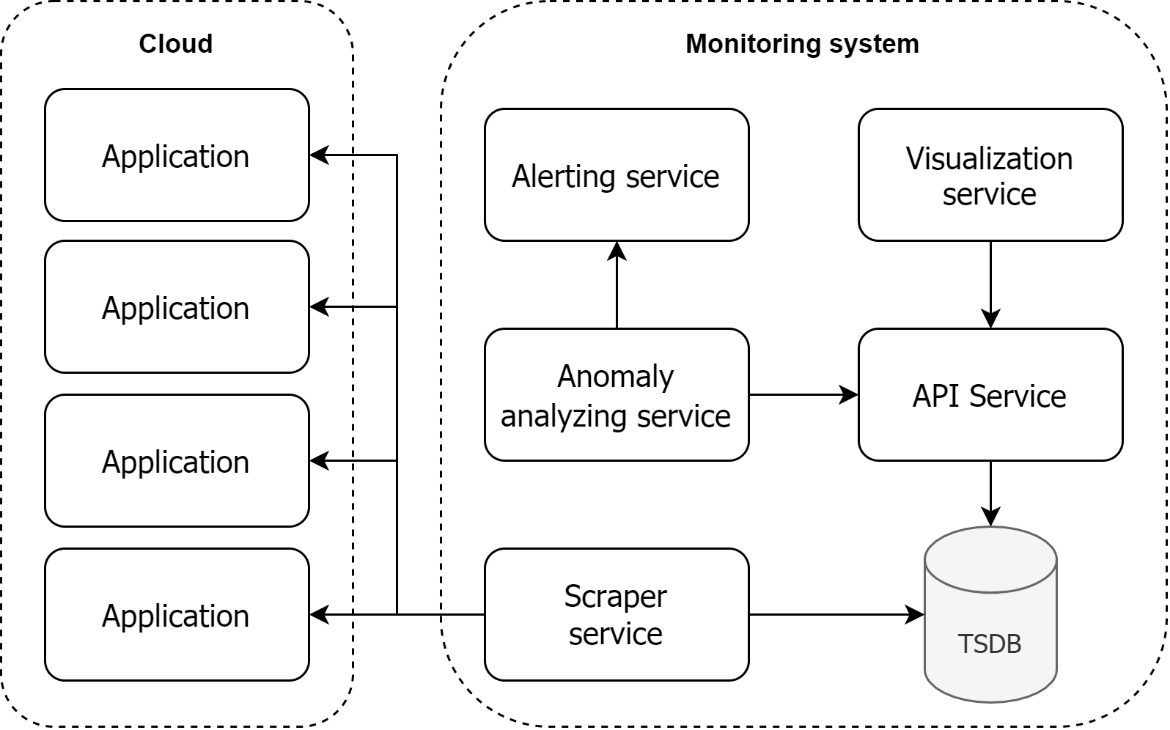


Рисунок 1. Компоненты системы мониторинга

В общем случае система мониторинга состоит из следующих компонентов:

* *TSDB (Time Series Database, англ. база данных временных рядов).* Основное хранилище метрик системы мониторинга.
* *Scraper service (англ. сервис сбора).* Сервис сбора метрик, который выполняет запросы к объектам наблюдения (приложениям) с определенным временным интервалом и сохраняет полученные метрики в базу данных временных рядов.
* *API Service (англ. сервис предоставления внешних интерфейсов).* Компонент, предоставляющий внешние интерфейсы для работы с базой данных.
* *Anomaly analyzing service (англ. сервис анализа аномалий).* Сервис анализа аномалий, который анализирует временные ряды в базе данных на наличие в них аномалий.
* *Alerting service (англ. сервис предупреждения).* Сервис предупреждения, который рассылает уведомления ответственным об аномальном поведении в метриках (сбое).
* *Visualization service (англ. сервис визуализации).* Сервис для отображения панелей с графиками различных бизнес и технических метрик.

Обратим внимание на то, что система мониторинга отделена от «облака», в котором развернуты экземпляры приложений. Отделение системы мониторинга от объектов наблюдения (приложений) на уровне инфраструктуры необходимо, поскольку позволяет системе мониторинга не зависеть от состояния кластера, в котором развернуты приложения.

* 1. **Метрики в системе мониторинга**
     1. **Характеристика метрик**

Метрика – наблюдаемая количественная характеристика приложения. У каждой метрики есть уникальное в рамках одного приложения название, метки (тэги) для большей детализации и значение.

На рисунке 2 изображена метрика температуры в формате prometheus.

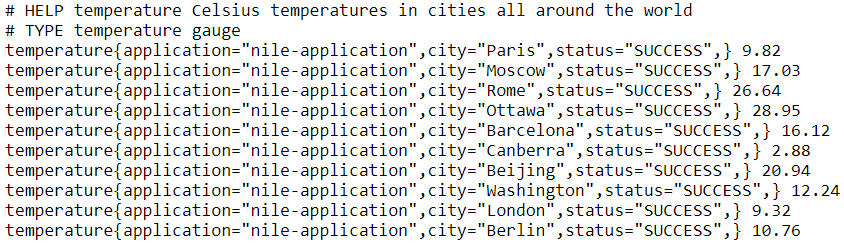


Рисунок 2. Метрика температуры

Данная метрика имеет название – *temperature*, три тэга и значение. Тэг *application* определяет название приложения, которое экспортирует данную метрику, в тэге *city* указан город, для которого отображена температура, в тэге *status* отображается статус последнего запроса на получение температуры (если статус отличен от «SUCCESS», то это повод рассмотреть возможное проблемы с получением температуры в данном городе). Как можно понять из описание метрики, ее значением является температура в градусах Цельсия специфичная для города, указанного в тэге *city*.

У каждого приложения глобально можно выделить два типа метрик: технические метрики и бизнес метрики. Технические метрики отображают техническое состояние приложения (объем использованной памяти, нагрузка на центральных процессор, кол-во входящих запросов, уровень ошибок и т. д.), у большинства приложений технические показатели очень похожи. Бизнес-метрики отвечают бизнес-потребностям конкретного приложения и крайне специфичны для конкретного сервиса.

* + 1. **4 золотых показателя**

Среди технических метрик, в книге «Site reliability engineering» выделяют «Четыре золотых показателя», которые должны присутствовать у любого приложения – время ответа, величина трафика, уровень ошибок и степень загруженности. Это основные метрики для минимальной диагностики работы любого сервиса. Рассмотрим каждый из них подробнее.

* *Время ответа.* Время, которое необходимо для выполнения запроса. В данной метрике очень важно разделять время ответа для успешных и неуспешных запросов. Например, код ошибки 401 Unauthorized возвращается очень быстро, однако, поскольку данный код ответа указывает на то, что запрос не был выполнен, то учитывать данный запрос при подсчете общей статистике некорректно.
* *Величина трафика.* Величина нагрузки, которую обрабатывает сервис. Для разных сервисов единицы измерения нагрузки будут разными, например, для веб-сервиса трафик измеряется в количестве HTTP запросов в секунду, для сервиса, который читает очередь брокера сообщений – количество прочитанных сообщений, для системы потокового аудио сигнала это скорость передачи данных по сети или количество параллельных соединений.
* *Уровень ошибок.* Количество неуспешно выполненных запросов: явно (возвращаемый код не относится к категории успешных – не 2\*\*), неявно (если код ответа является успешным, но полученные данные являются неправильными) или не соответствующих требованиям (например, если ответ должен приходить в пределах 2-ух секунд, то любой запрос, выполнившийся за большее время считается неуспешным)
* *Степень загруженности.* Показатель того, насколько сильно загружен сервис. В каждом сервисе это сразу набор метрик, которые показывают ограничения в работе сервиса, например, в приложении, ограниченных по ресурсам – это ресурсы, в приложениях имеющих ограничения на количество одновременно выполняемых запросов – это количество потоков в очереди пула потоков. Отметим, что многие сервисы начинают работать медленнее еще до того, как перейдут пороговое значение по своей степени загруженности, поэтому очень важно следить за данными метриками как за целевым показателем.

Если удается измерить все четыре сигнала и сообщить клиентам о том, что один из них находится вне своей нормы, то качество мониторинга для сервиса можно определить как минимум удовлетворительным.

* + 1. **Мониторинг производительности приложений**

Консалтинговая и исследовательская компания Gartner, специализирующаяся на рынке информационных технологий разработала концепцию «Мониторинга производительности приложений» (англ. Application performance monitoring (APM)), которая описывает 5 важных аспектов мониторинга, на которых нужно сфокусироваться в первую очередь. На рисунке 3 изображена схема мониторинга производительности приложений с описанием области применения и потенциальными преимуществами.

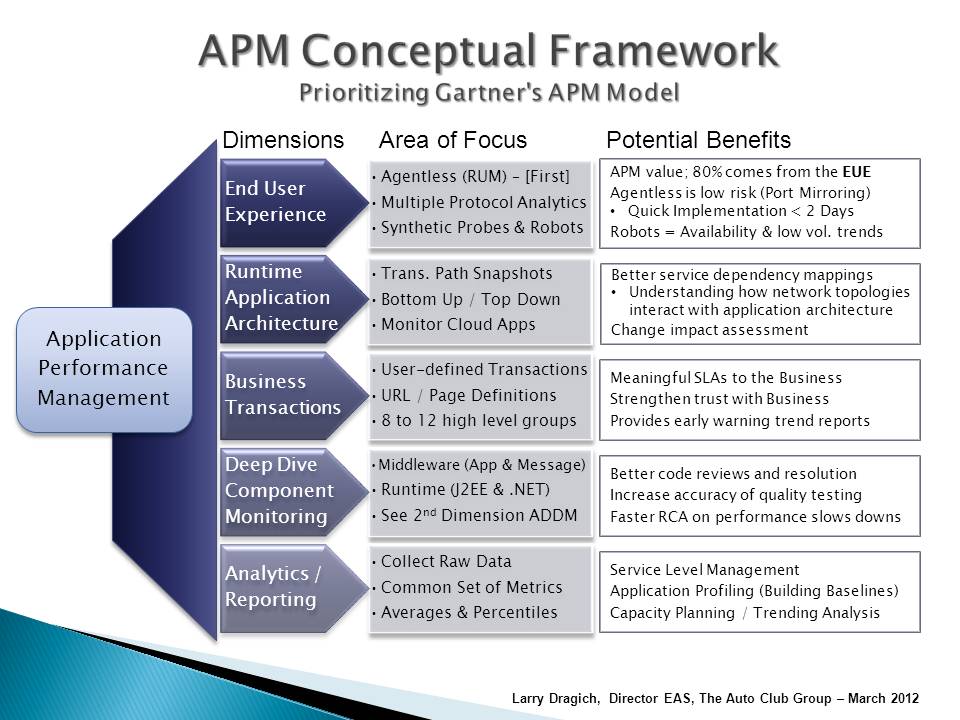


Рисунок 3. Мониторинг производительности приложений (Application performance monitoring (APM))

Рассмотрим каждый аспект (измерение) более подробно:

* *Мониторинг взаимодействия с пользователем (англ. End-user experience monitoring).* Отслеживает поведение программного приложения с точки зрения пользователя, выявляя моменты, когда они сталкиваются с медлительностью, простоями или ошибками.
* *Обнаружение, моделирование и отображение архитектуры среды выполнения приложений (англ. Application runtime architecture discovery, modeling and display).* Отображение всех компонентов вашего приложения и наблюдение за их взаимодействием друг с другом. Наличие данных в визуальной форме упрощает обнаружение проблем.
* Определяемое пользователем профилирование транзакций (англ. User-defined transaction profiling). Анализ потока каждой пользовательской транзакции и выделение конкретных взаимодействий, в которых обнаруживаются проблемы с производительностью. Трассировка позволяет проследить путь пользователя от интерфейса к серверу. Таким образом, разработчики могут найти точную строку кода, запрос к базе данных или сторонний вызов, который влияет на производительность приложения.
* *Глубокий анализ приложений (англ. Application deep-dive analysis).* Сбор показателей производительности всех компонентов инфраструктуры приложений. Мониторинг инфраструктуры должен быть интегрирован в инструмент APM.
* *Аналитика ИТ-операций (англ. IT operations analytics).* Анализ данных для выявления моделей использования, тенденций и проблем с производительностью, которые вы можете использовать для построения лучшего плана действий в нестабильных ситуациях до того, как последствия сбоя повлияют на конечных пользователей.
  + 1. **Перцентили и работа с выбросами в метриках**

При разработке системы мониторинга есть большой соблазн использовать для метрик средние значения: среднее время обработки запроса, средний процент загрузки сервиса, среднюю заполненность базы данных и т. д. Риски, связанные с такой оценкой заключаются в том, что средние значения могут скрывать проблемные места в работе сервиса. Например, среднее время обработки запроса может быть в пределах нормы – 300 миллисекунд, однако из 1000 запросов 5% отвечают за 3 секунды, а 10% за 1 секунду.

Самый простой способ различать медленное среднее время обработки и крайне медленные «хвосты» - вместо значений задержки использовать перцентили (квантили), удобные для построения гистограммы: какой процент от всех выполненных запросов обработался менее чем за 0.5 секунд, какой процент от всех запросов обработался менее чем за 0.9 секунд и т. д. Построение гистограмм с разбивкой по перцентилям – один из самых простых и эффективных способов наглядно продемонстрировать распределение характеристик различного рода временных метрик.

На рисунке 3 изображен пример метрики времени ответа с разбивкой по квантилям.

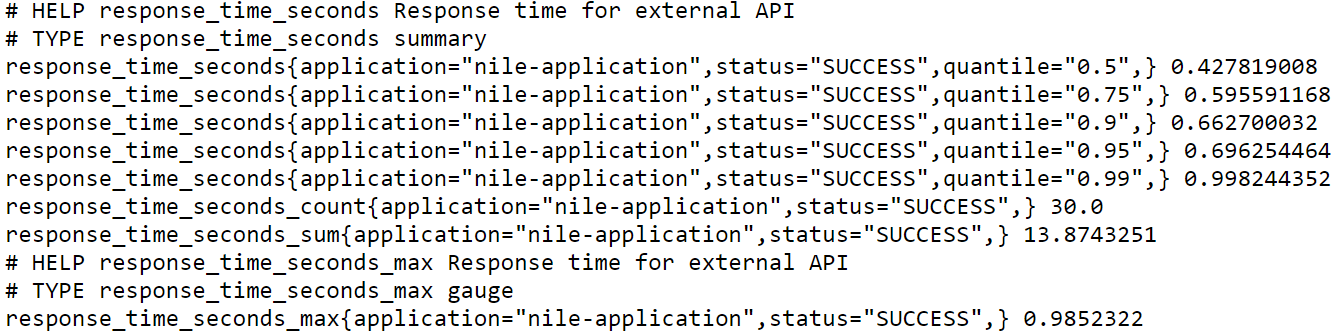


Рисунок 3. Метрика времени ответа с разбивкой по квантилям

Анализирую тэг quantile и значение данной метрики мы можем построить гистограмму распределения запросов с разбивкой по перцентилям, на которой будет видно, что 50% запросов выполнились менее, чем за 400 мс., 75% – менее, чем за 595 мс., 90% – менее, чем за 662 мс, 95% – менее, чем за 696 мс., 99% - менее, чем за 998 мс. Если бы использовали только среднее значение, то получили бы только одно значение – 427 мс, что очень поверхностно отражает действительность.

* + 1. **Анализ метрик**

При анализе метрик любая система мониторинга отвечает на два вопроса: что сломалось и почему это сломалось. Ответ на вопрос «Что сломалось?» подразумевает под собой определение симптома проблемы, а ответ на вопрос «Почему это сломалось?» определяет причину этой проблемы. В таблице приведены примеры симптомов и соответствующих им причин.

|  |  |
| --- | --- |
| **Симптом** | **Причина** |
| Сервис возвращает ответы HTTP 408 | Сторонний сервис начал долго обрабатывать запросы |
| Сервис возвращает ответы HTTP 401 | Сервис авторизации изменил алгоритм шифрования токенов |
| Время ответа сервиса увеличилось вдвое | База данных начала испытывать аномальное кол-во запросов от других сервисов |
| Сервис использует в 1.5 раза больше ресурсов по памяти | Выросло кол-во одновременно обрабатываемых запросов |

Таблица 1. Симптомы и причины

Хорошо спроектированная система мониторинга должна выявлять причинно-следственные связи «что» и «почему» и демонстрировать их инженерам.

В любой системе мониторинга широко используются методы черного и белого ящика. Метод черного ящика заключается в наблюдении за симптомами, которые выявляют реально возникшие проблемы. Метод белого ящика – это возможности наблюдать за внутренним устройством системы. Метод белого ящика позволяет обнаруживать потенциальные проблемы, а метод черного ящика работает с существующими проблемами.

Обратим внимание, что в многоуровневой системе то, что можно классифицировать как симптом для одной системы, для другой системы может являться причиной. Например, если у сервиса «a» вдвое выросло время ответа и этот факты является симптомом для сервиса «a», то для сервиса «b», который вызывает сервис «a» данный факт будет причиной. Поэтому очень важно проводить причинно-следственные связи, чтобы искать проблему не в соединении между двумя сервисами, а сразу исследовать симптом сервиса «a».

Метод черного ящика хорошо применять для экстренных оповещений, поскольку такое оповещение произойдет только в том случае, если проблема уже точно существует и дает реальные симптомы. Однако для потенциальных проблем более уместен метод белого ящика с правильно подобранными пороговыми значениями для реагирования.

Для разных компонентов системы измерения должны проводиться с разным уровнем детализации. Например, контроль загрузки центрального процессора с периодичностью в минуту будет нерепрезентативной метрикой, а, например, проверка доступности сервиса с той же периодичностью будет качественной метрикой, поскольку результат наблюдения меняется не так часто. Важно подходить к детализации каждой метрики отдельно и экспериментировать с настройками скрейпинга (англ. scrape – царапать) – временной интервал между каждый сбором.

Мониторинг обеспечивает слежение за постоянно меняющимися системами, у которых меняется степень нагрузки и целевой уровень производительности. Важно, чтобы при построении мониторинга для приложений инженеры руководствовались долгосрочными перспективами. При определении целевых показателей часто придется прибегать к компромиссным решениям, отказываться от высоких показателей доступности, чтобы иметь возможность и время улучшить данные показатели в будущем.

* 1. **Существующие системы мониторинга бизнес-приложений**
     1. **Nagios**

Nagios – система мониторинга компьютерных систем и сетей. Проект с открытым исходным кодом, который был создан в 1999 году, с тех пор постоянно обновлялся и улучшался. На данный момент система является универсальным решением как для малых сетей, так и для корпоративных.

Nagios осуществляет мониторинг операционных систем, сетевых протоколов, приложений, веб-сайтов, веб-серверов и т. д. Система обладает возможностью добавления собственного плагина для интеграции с любым типов стороннего программного обеспечения.

Основные возможности Nagios:

* Централизованный сбор и визуализация информации о состоянии инфраструктуры;
* Система обработки сбоев в работе приложений;
* Многопользовательский доступ;
* Возможность настройки прав доступа к компонентам внутри системы;
  + 1. **Zabbix**

Zabbix – система мониторинга служб и состояний компьютерной сети, предназначенное для мониторинга производительности, доступности серверов, сетевого оборудования, веб-приложений, баз данных. Zabbix является очень распространенной системой в промышленной разработке и используются такими компаниями, как DELL, ICANN, Salesforce, Orange и т. д.

Системная архитектура данной системы опирается на использование центрального сервера и агентов. Zabbix-сервер и Zabbix-агент могут быть установлены на такие платформы, как Linux, MacOS X, FreeBSD, OpenBSD.

Основные возможности Zabbix:

* Мониторинг Java-серверов приложений через технологию JMX (Java Management Extensions);
* Защищенность от атак пользовательского интерфейса Zabbix на стороне клиента;
* Расширение функциональности за счет поддержки сторонних скриптов, написанных на Ruby, Python, PHP, Java;
* Возможности для интеграции со сторонними инструментами системного менеджмента (Puppet, cfengine, Chef и т. д.)
  + 1. **Anturis**

Anturis – облачная (Software as a Service, англ. программное обеспечение как услуга) платформа, предназначенная для внешнего мониторинга веб-сервисов и внутреннего мониторинга ИТ-инфраструктуры.

Основные возможности Anturis:

* Мониторинг аппаратного и программного обеспечения серверов;
* Мониторинг веб-серверов, загрузки веб-страниц, сетей;
* Оповещение о проблемах через email, SMS, телефонный звонок;
* Создание отчетов, визуализация данных, анализ инцидентов;
  + 1. **Instrumental**

Instrumental – система мониторинга инфраструктуры и приложений, обладающая возможностью самостоятельно создавать информационные панели, графики, оповещения.

Основные преимущества системы:

* Минимальные требования к настройке системы;
* Большой выбор инструментов для мониторинга (Docker, MySQL, Memcached, MongoDB, PostgreSQL, Redis, PHP, Ruby, Java, Python);
* Быстрый поиск по всем сущностям системы мониторинга;
* Система оповещения о проблемах через SMS, email, slack;

**Выводы**

Существующие комплексные системы мониторинга предоставляют широкие возможности для мониторинга любого вида аппаратного и программного обеспечения. Однако у подобных систем мониторинга есть ряд общих недостатков:

1. Системы имеют строгие требования к развертыванию и поддержке;
2. Настройка метрик и правил их сбора сосредоточена в самой системе мониторинга;
3. Добавление бизнес-метрик либо невозможно, либо требует больших усилий со стороны разработчика;

# Глава 2. Архитектура сервиса мониторинга бизнес-приложений

* 1. **Требования к системе мониторинга**

Основываясь на анализе современных подходов к организации мониторинга в IT компаниях и полученных выводах из прошлой главы, можно выделить три направления в требованиях к архитектуре системы мониторинга: требования по работе с метриками, требования для инструмента отображения метрик и требования к анализу аномалий.

Основные преимущества данной системы мониторинга:

* Удобный и исчерпывающий пользовательский API для написания технических и бизнес-метрик приложения;
* Удобство развертывания инфраструктуры для мониторинга;
* Удобные инструменты для визуализации мониторинга приложения;
* Возможность настройки правил для системы оповещений, анализа аномалий;

В таблице 2 представлены конкретные требования к системе мониторинга по каждому из этих трех пунктов.

|  |  |  |
| --- | --- | --- |
| **Требование** | **Обоснование** | **Примеры** |
| **Инструмент для работы с метриками** | | |
| Измерение времени обработки входящих и исходящих запросов | Возможность контролировать время выполнения запросов к сервису. | За какое кол-во времени выполняется get запрос, обрабатывается вызов стороннего сервиса. |
| Построение метрик времени выполнения для отдельных участков кода | Возможность отслеживать время выполнения отдельных участков кода. | За какое кол-во времени выполняется вызов отдельных методов, походов в базу данных и т. д. |
| Построение метрик для экспорта конкретных значений, времени выполнения (включая перцентили), счетчика | Возможность отслеживать текущий бизнес-показатели, временные тенденции, анализировать утилизацию ресурсов. | Кол-во заявок с распределением по статусам, время выполнения запросов к базе данных с разбивкой по перцентилям, кол-во прочитанных сообщений из очереди событий. |
| Запуск сбора метрик по заданному расписанию | Необходимость постоянно обновлять метрику раз в определенный интервал времени. | Обновление раз в 30 минут кол-ва записей в таблице базы данных. |
| Построение метрик на основе результатов запросов к базе данных | Потребность в работе с метриками, основанными на данных из базы данных. | Кол-во записей с разбивкой по конкретным колонкам в базе данных. |
| Удобный API для построения и настройки всех типов метрик | Возможность быстро добавлять, изменять и удалять метрики там, где это необходимо. | API для построения метрики времени выполнения в несколько строчек кода. |
| **Инструмент для отображения метрик** | | |
| Удобный пользовательский интерфейс для построения и отображения панелей с графиками | Необходимость в работе с метриками, их визуализации. | Графики скорости обработки запросов с разбивкой по перцентилям в секундах, графики размера базы данных в гигабайтах. |
| Построение разных типов графиков (графы, таблицы, счетчики) с поддержкой разных единиц измерения (проценты, мегабайты, секунды) | Потребность в том, чтобы анализировать разные типы метрик в удобном интерфейсе. | Построение графика обработки запросов, построение таблицы с метриками, построение счетчиков кол-ва запросов к базе данных. |
| Система оповещения при сбоях в популярные мессенджеры | Потребность в своевременном предупреждении о неполадках и некорректной работе. | Оповещение в telegram при обнаружении сбоя в работе приложения. |
| Построение графиков с панелями из кода приложения через DSL | Необходимость в работе с панелями Grafana через код. | Построение панели с графиками, описанными с помощью предметно-ориентированного языка. |
| Настройка правил для оповещений и реакций на аномалии из кода приложения через DSL | Необходимость в работе с настройками панелей Grafana через код. | Конфигурация пороговых значений, по достижению которых значение метрики считается аномальным. |
| **Инструмент для анализа аномалий** | | |
| Анализ временных рядов на наличие аномалий с использованием разных методов | Необходимость анализировать временные ряды разными способами. | Аномалии во времени ответа от сторонних сервисов. |
| Возможность анализировать временной ряд для любых типов метрик (конкретные значения, время выполнения, счетчики) | Необходимость в поиске отклонений от нормального поведения в разного типа метриках. | Аномалии в метриках базы данных, счетчиках кол-ва прочитанных сообщений из очереди. |
| Экспорт метрик с характеристиками аномалий | Потребность в получении более детальной информации для конкретных аномалий. | Отдельные метрики для обозначения верхней и нижней границы доверительного интервала. |
| **Инфраструктура** | | |
| Timeseries база данных | Необходимость в хранении большого кол-ва данных временных рядов. | Хранение временных рядов за последние полгода. |
| Сервис для сбора метрик | Необходимость в сборе метрик с приложения. | Сбор метрик приложений с целью их последующего анализа. |
| Сервис для отображения метрик | Потребность в отображении графиков с метриками. | Возможность просматривать метрики и графики в удобном интерфейсе. |

Таблица 2. Требования к системе мониторинга

* 1. **Используемые технологии**
     1. **Язык программирования Kotlin**

В качестве языка программирования был выбран Kotlin. Kotlin – это статически типизированный, объектно-ориентированный язык программирования, выполняющийся на виртуальной машине Java (JVM) и разрабатывается российской компанией JetBrains.

Данный язык разрабатывается с 2010 года в качестве альтернативы Java. По структурам данных язык очень похож на Java, по синтаксису ближе всего к C#.

Основными отличиями от java являются:

* краткость – значительно меньше шаблонного кода (boilerplate code);
* безопасность – специальные механизмы для предотвращения выбрасывания целых классов ошибок, таких как Null Pointer Exception (NPE);
* совместимость – возможность использовать один и тот же язык для JVM, Android, браузера;
* поддержка сред разработки – можно использовать любую среду разработки для Java, есть специальная поддержка языка в Intellij Idea;

Основным преимуществом использования Kotlin является то, что он полностью совместим с Java, а это значит, что разработанную систему мониторинга можно будет использовать в большинстве промышленных приложений, написанных на Java.

* + 1. **Spring Framework**

В качестве основного фреймворка для разработки системы мониторинга был использован Spring framework. Spring framework – универсальный фреймворк для разработки приложений с открытым исходным кодов для языков на Java платформе.

Spring был написан, как универсальное средство для решения множества задач, с которыми сталкиваются разработчики, которые пишут приложения на Java.

Spring можно представить, как набор небольших фреймворков:

* IOC (Inversion of control) контейнер – конфигурация компонентов приложения (beans) и управление их жизненным циклом
* аспектно-ориентированное программирование – функциональность, которая не может быть реализована возможностями ООП
* доступ к данным (spring data) – работа с реляционными базами данных на Java с использованием JDBC (java database connectivity)
* управление транзакциями – API для управление транзакциями
* MVC (Model View Controller) – поддержка HTTP и servlet, предоставляющий возможности для кастомизации
* аутентификация и авторизация (spring security) – конфигурируемый инструмент для процессов аутентификации и авторизации
* тестирование – фреймворк для поддержки unit тестов

Благодаря тому, что большинство промышленных приложений написано с использованием Spring Framework, разработанная система мониторинга может быть доступна как можно большему числу клиентов.

* + 1. **Средство для сбора и хранения метрик Prometheus**

Prometheus — система мониторинга серверов и программ с открытым исходным кодом. Она была выпущена в 2012 году для мониторинга музыкальной социальной сети SoundCloud. Из-за специфики архитектуры SoundCloud традиционные системы мониторинга не подходили, поэтому по новой технологии был создан Prometheus-мониторинг. Позже новый мониторинг сервисов стал интересен за пределами музыкальной сети. Создатели доработали систему и предложили её более широкому рынку. Релиз обновлённой программы прошёл в 2015 году.

Prometheus функционирует благодаря таким компонентам, как:

* Prometheus-server — главное звено всей системы. Он отвечает за сбор и хранение данных. Есть простой веб-интерфейс, но для полноценной работы с системой лучше устанавливать дополнительный.
* Exporters — это часть ПО, которая собирает и передаёт Prometheus-метрики серверу. Существуют разные экспортёры, например HAProxy, StatsD, Graphite. Они устанавливаются на целевые объекты и собирают определённые метрики. Если ни один экспортёр вам не подходит, то можно написать свой.

Prometheus является одним из главных элементов разрабатываемой инфраструктуры мониторинга, данный сервис удобен для развертывания и поддержки.

* + 1. **Средство визуализации метрик Grafana**

Grafana — это платформа с открытым исходным кодом для визуализации, мониторинга и анализа данных. Этот инструмент часто работает в связке с Prometheus: Prometheus собирает и хранит метрики, а Grafana делает запросы в Prometheus и на их основе строит графики, дашборды.

Grafana позволяет пользователям создавать дашборды с панелями, каждая из которых отображает определенные показатели в течение установленного периода времени. Каждый дашборд универсален, поэтому его можно настроить для конкретного проекта или с учетом любых потребностей разработки и/или бизнеса.

К основным компонентам Grafana можно отнести следующие:

* Панель – базовый элемент визуализации выбранных показателей. Grafana поддерживает панели с графиками, единичными статусами, таблицами, тепловыми картами кликов и произвольным текстом, а также интеграцию с официальными и созданными сообществом плагинами (например, карта мира или часы) и приложениями, которые также можно визуализировать. Можно настроить стиль и формат каждой панели; все панели можно перетаскивать на новое место, перестраивать и изменять их размер.
* Дашборд – набор отдельных панелей, размещенных в сетке с набором переменных (например, имя сервера, приложения и датчика). Изменяя переменные, можно переключать данные, отображаемые на дашборде (например, данные с двух отдельных серверов). Все дашборды можно настраивать, а также секционировать и фрагментировать представленные в них данные в соответствии с потребностями пользователя. В проекте Grafana участвует большое сообщество разработчиков кода и пользователей, поэтому существует большой выбор готовых дашбордов для разных типов данных и источников.

Grafana выбрана как основное средство для визуализации метрик на дашбордах. Grafana так же, как и Prometheus удобна для развертывания и поддержки.

* 1. **Реализация требований к системе мониторинга**

Инструмент для работы с метриками, графиками и инструмент для анализа аномалий будет представлять из себя клиентскую библиотеку, которая будет состоять из следующих компонентов:

* *Micrometer.* Модуль, который предоставляет удобный и полезный API для построения и запуска сбора метрик по расписанию.
* *Anomaly.* Модуль для анализа аномалий временных рядов.
* *Grafana.* Модуль для построения графиков и построения дашбордов в Grafana по принципу «Grafana как код».

В таблице представлен набор сервисов для развертывания инфраструктуры с обоснованием выбора.

|  |  |  |
| --- | --- | --- |
| **Требование** | **Сервис** | **Обоснование** |
| Timeseries база данных | Prometheus | Сервис поддерживает timeseries базу данных, а также выполнение запросов на поиск временных рядов. |
| Сервис для сбора метрик | Prometheus | Open source решение, позволяющее настраивать сбор метрик с приложений по общепризнанным стандартам. |
| Сервис для отображения метрик | Grafana | Сервис обладает качественной визуализацией, поддержкой множества типов источников данных. |

На Рисунке 4 изображена схема инфраструктуры системы мониторинга.

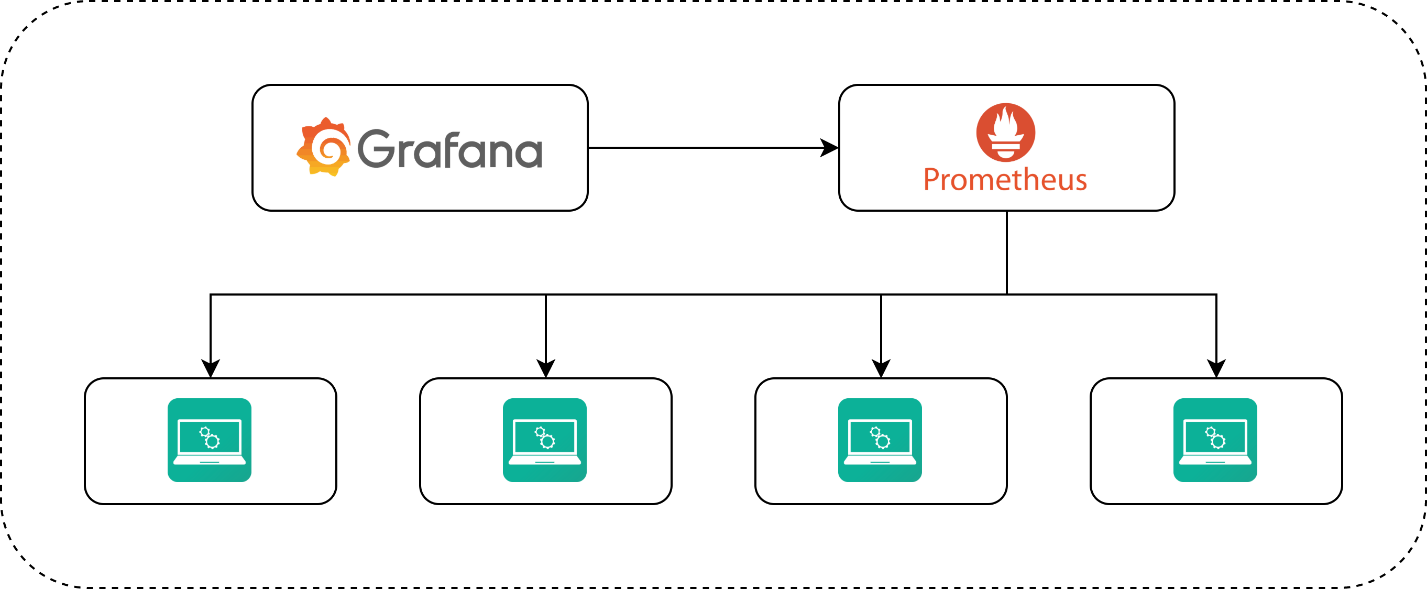


Рисунок 4. Схема инфраструктуры системы мониторинга

* 1. **Ограничения системы мониторинга**
     1. **Ограничения на уровне приложений**

Приложения должны удовлетворять следующим критериям:

1. Приложения должны быть написаны на языке Java или Kotlin.
2. Приложения должны использовать Spring Framework
   * 1. **Ограничения на уровне инфраструктуры**

Инфраструктура для развертывания должна удовлетворять следующим критериям:

1. Инфраструктура должна поддерживать контейнеризацию Docker
2. Инфраструктура должна предоставлять сетевую доступность развернутых в ней приложений и других инфраструктурных сервисов (Prometheus, Grafana)

# Глава 3. Разработка сервиса мониторинга бизнес-приложений

* 1. **Модуль построения метрик**
     1. **Метрики Counter**

Метрики типа Counter – метрики-счетчики, позволяющие отслеживать кол-во выполненных запросов, сохраненных записей в базу данных и т. д. На рисунке 4 представлен пример метрики типа Counter.

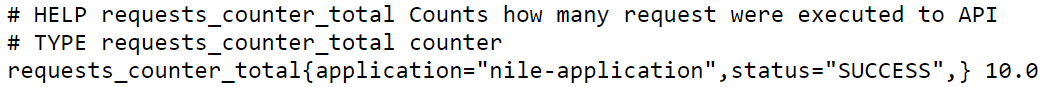


Рисунок 5. Метрика типа Counter

Данная метрика показывает кол-во запросов выполненных к стороннему API.

* + 1. **Метрики Timer**

Метрики типа Timer – метрики-таймеры, которые показывают, за какое кол-во времени выполнилось определенное действие (поход в базу данных, вызов определенного метода и т. д.). На рисунке 6 показан пример метрики типа Timer.

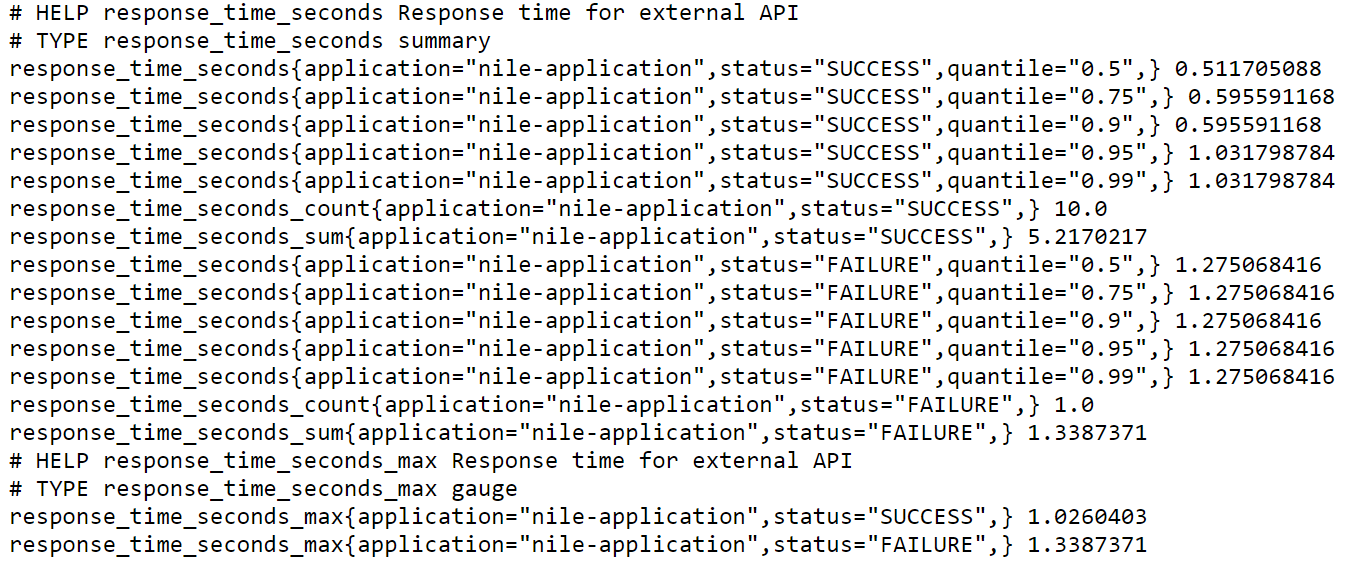


Рисунок 6. Метрика типа Timer

Данная метрика показывает какое кол-во запросов было выполнено и за какое время они были выполнены. Если разделить время, затраченное на выполнение запросов на кол-во запросов, то можно получить время выполнения одного запроса.

* + 1. Метрики Gauge
    2. Метрики Distribution Summary
    3. Запуск метрик по расписанию
  1. **Модуль для построения графиков отображения метрик**
     1. Построение панелей с графиками в Grafana
     2. Система предупреждения
  2. **Модуль для анализа аномалий**
     1. Обнаружение аномалий во временном ряде
     2. Добавление обработчика обнаружения аномалий
  3. **Конфигурация архитектуры для развертывания**
     1. Конфигурация для развертывания Prometheus
     2. Конфигурация для развертывания Grafana
     3. Конфигурация для развертывания приложения

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Formalizing and Securing Relationships on Public Networks [Электронный ресурс] – URL: <https://status.cloud.google.com/summary> (Дата обращения: 01.05.2021)
2. Formalizing and Securing Relationships on Public Networks [Электронный ресурс] – URL: <https://firstmonday.org/ojs/index.php/fm/article/view/548> (Дата обращения: 01.05.2021)