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**

# 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 [2].

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 интерфейс для создания и экспорта метрик;
2. Быстрое развертывание облачной инфраструктуры для сбора и отображения 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 интерфейса;
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: программный интерфейс для построения метрик, анализ аномалий во временных рядах, 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;
- 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 [3].

**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 [3].

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

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

**СУБД** – это комплекс программно-языковых средств, позволяющих создать базы данных и управлять данными [4].

**API** (программный интерфейс приложения, англ. application programming interface) – описание способов, с помощью которых одна компьютерная программа может взаимодействовать с другой.

### 1.1.2 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 [3]:

- *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.

### 1.1.3 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 [5]. Ниже изображена схема мониторинга производительности приложений с описанием области применения и потенциальными преимуществами (см. рисунок 1.1.).

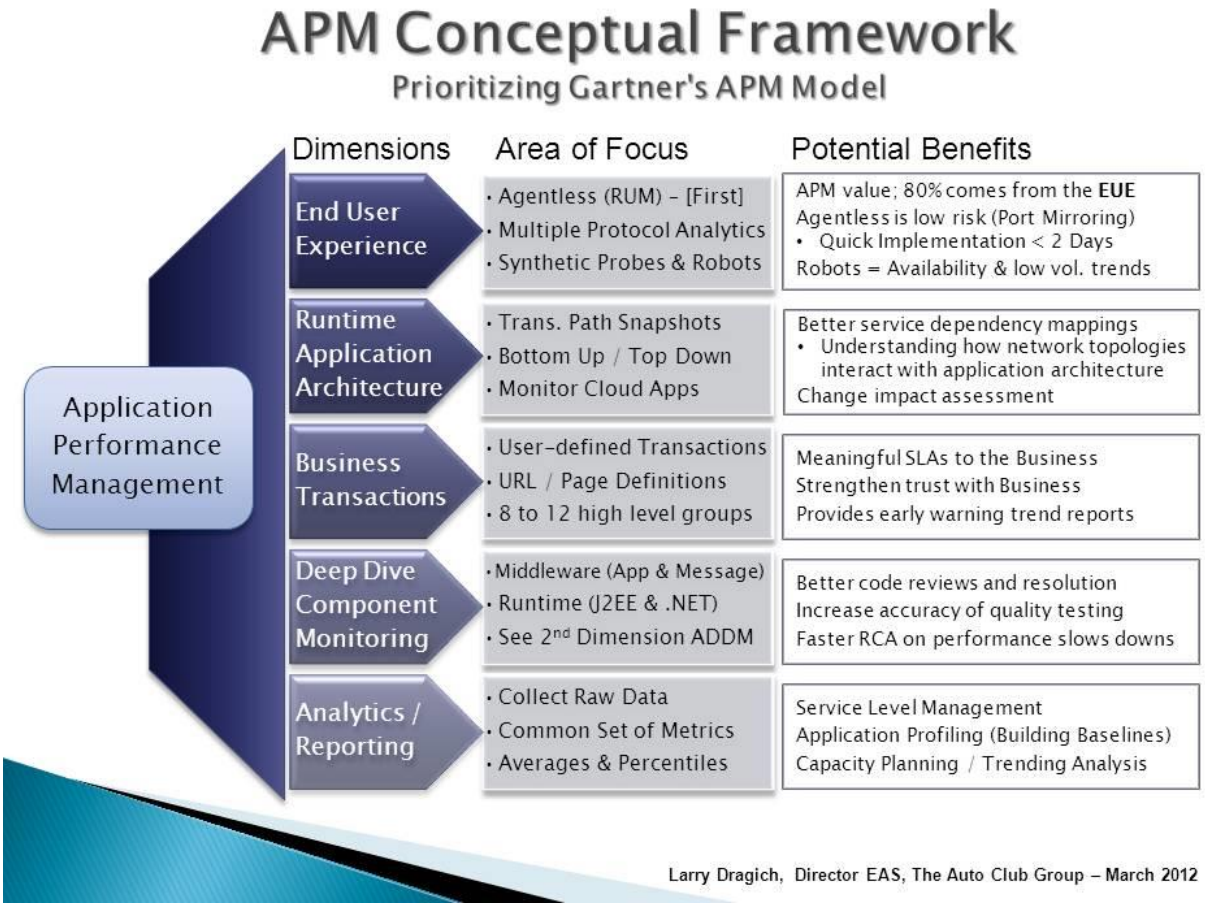


Рисунок 1.1. Мониторинг производительности приложений (Application performance monitoring (APM))

Источник: https://en.wikipedia.org/wiki/Application_performance_management

Рассмотрим каждый аспект (измерение) более подробно [6]:

- *Мониторинг взаимодействия с пользователем (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 (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 (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 (Application deep-dive analysis).* Сбор показателей производительности всех компонентов инфраструктуры приложений. Мониторинг инфраструктуры должен быть интегрирован в инструмент APM.
- *Аналитика ИТ-операций (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.

Фреймворк «Мониторинг производительности приложений» определяет 5 ключевых областей наблюдения, которыми должна обладать качественная система мониторинга.

### 1.1.4 Компоненты системы мониторинга

Любая система мониторинга должна обладать набором компонентов, которые обеспечивают функциональность данной системы. На схеме изображены компоненты современной системы мониторинга (см. рисунок 1.2.).

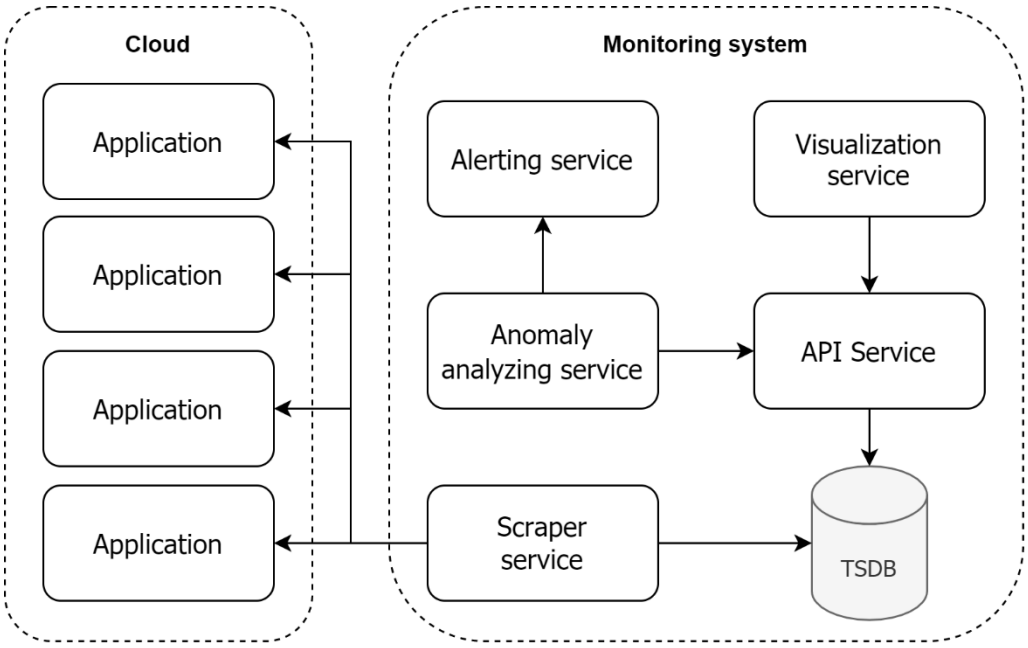

Рисунок 1.2. Компоненты системы мониторинга

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.

## 1.2 Метрики в системе мониторинга

### 1.2.2 Характеристика метрик

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

Ниже изображена метрика влажности в популярном формате построения метрик Prometheus (см. рисунок 1.3.).

```
# HELP humidity Humidity for particular city in percents
# TYPE humidity gauge
humidity{application="nile-application",city="Paris",status="SUCCESS",} 58.0
humidity{application="nile-application",city="Moscow",status="SUCCESS",} 34.0
humidity{application="nile-application",city="Rome",status="SUCCESS",} 92.0
humidity{application="nile-application",city="Ottawa",status="SUCCESS",} 76.0
humidity{application="nile-application",city="Barcelona",status="SUCCESS",} 70.0
humidity{application="nile-application",city="Canberra",status="SUCCESS",} 92.0
humidity{application="nile-application",city="Beijing",status="SUCCESS",} 41.0
humidity{application="nile-application",city="Washington",status="SUCCESS",} 73.0
humidity{application="nile-application",city="London",status="SUCCESS",} 81.0
humidity{application="nile-application",city="Berlin",status="SUCCESS",} 54.0
```

Рисунок 1.3. Метрика влажности

Данная метрика имеет название – humidity, три тэга и значение. Тэг *application* определяет название приложения, которое экспортирует данную метрику, в тэге *city* указан город, для которого отображена влажность, в тэге *status* отображается статус последнего запроса на получение влажности (если статус отличен от «SUCCESS», то это повод рассмотреть возможные проблемы с определением влажности в данном городе). Как можно понять из описание метрики, ее значением является влажность, измеряемая в процентах, 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. п.), бизнес метрики отображают бизнес состояние приложения, специфичное для конкретного бизнеса (кол-во одобренных кредитов, скорость работы скоринга и т. п.). У большинства приложений технические метрики очень

похожи. Бизнес метрики отвечают бизнес-потребностям конкретного приложения и крайне специфичны для конкретного сервиса.

### 1.2.3 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 показателя и сообщить клиентам о том, что один из них находится вне своей нормы, то качество мониторинга для сервиса можно определить как минимум как удовлетворительное.

### 1.2.4 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 количество запросов, величина задержки для которых попадает в заданные интервалы, удобные для построения гистограммы: какое количество запросов потребовали для обработки от 0 до 50 миллисекунд, от 50 до 100, от 100 до 200 и т. д. Построение подобных гистограмм – один из самых простых и эффективных способов наглядно продемонстрировать распределение характеристик различного рода временных метрик.

Ниже изображен пример метрики времени обработки запроса с разбивкой по интервалам (англ. bucket) (см. рисунок 1.4.).

```
# HELP response_time_seconds Response time for external API
# TYPE response_time_seconds histogram
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="0.001",} 0.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="0.001048576",} 0.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="0.001398101",} 0.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="0.001747626",} 0.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="0.002097151",} 0.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="0.002446676",} 0.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="0.002796201",} 0.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="0.003145726",} 0.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="0.003495251",} 0.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="0.003844776",} 0.0
                                    ▪ ▪ ▪
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="7.158278826",} 11.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="8.589934591",} 11.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="10.021590356",} 11.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="11.453246121",} 11.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="12.884901886",} 11.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="14.316557651",} 11.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="15.748213416",} 11.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="17.179869184",} 11.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="22.906492245",} 11.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="28.633115306",} 11.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="30.0",} 11.0
response_time_seconds_bucket{application="nile-application",status="SUCCESS",le="+Inf",} 11.0
response_time_seconds_count{application="nile-application",status="SUCCESS",} 11.0
response_time_seconds_sum{application="nile-application",status="SUCCESS",} 5.0072659
```

Рисунок 1.4. Метрика времени обработки запроса с разбивкой по интервалам

У данной метрики есть название – *response_time*, а также тэг *le*, определяющий интервал в секундах, в который попадает определенное количество запросов. Так, в интервал «до 0.0038 секунд» не попадает ни одного запроса, а в интервал «до 7.158 секунд» попадает 11 запросов.

Ниже изображена гистограмма, построенная по данной метрике (см. рисунок 1.5.).

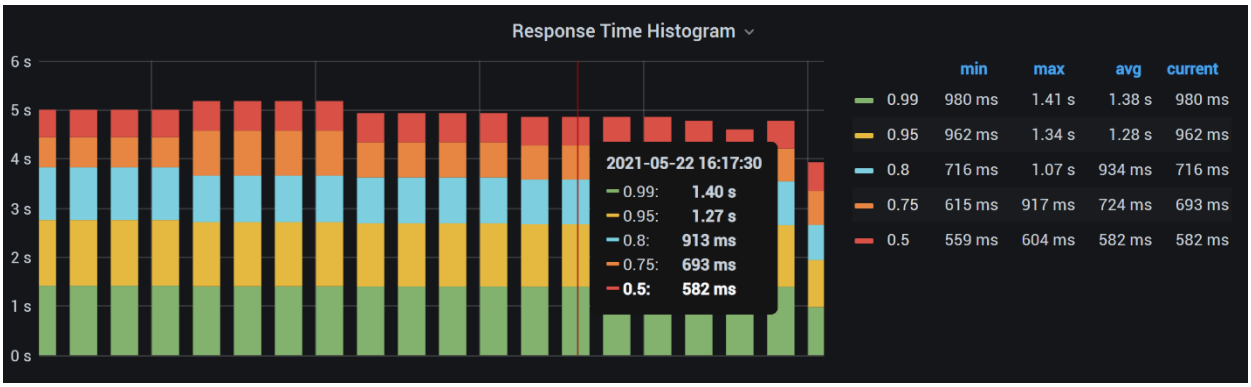


Рисунок 1.5. Гистограмма времени обработки запроса

На данной гистограмме показано время обработки запроса с разбивкой по перцентилям. Так, например, можно узнать, что половина запросов была обработана меньше, чем за 582 мс., а 4 % запросов были выполнены в интервале от 1.27 сек. до 1.4 сек.

При использовании только среднего значения было бы получено только одно значение – 582 мс., что очень поверхностно отражает действительность.

### 1.2.5 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 (см. таблица 1.1.).

| 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 1.1. 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 «а» вдвое выросло время ответа и этот факты является симптомом для сервиса «а», 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.

## 1.3 Существующие системы мониторинга бизнес-приложений

### 1.3.2 Nagios

Nagios – система мониторинга компьютерных систем и сетей [7,8,9]. 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;

### 1.3.3 Zabbix

Zabbix – система мониторинга служб и состояний компьютерной сети, предназначенное для мониторинга производительности, доступности серверов, сетевого оборудования, веб-приложений, баз данных [7,8,10]. 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.)

### 1.3.4 Anturis

Anturis – облачная (Software as a Service, англ. программное обеспечение как услуга) платформа, предназначенная для внешнего мониторинга веб-сервисов и внутреннего мониторинга ИТ-инфраструктуры [11,12,13].

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;

### 1.3.5 Instrumental

Instrumental – система мониторинга инфраструктуры и приложений, обладающая возможностью самостоятельно создавать информационные панели, графики, оповещения [11,14].

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;
4. Конфигурация панелей для визуализации обладает узкой функциональностью;

# Глава 2. Архитектура сервиса мониторинга бизнес-приложений

В данной главе рассмотрена архитектура новой системы мониторинга, представляющая из себя набор из отдельных компонентов для построения, сбора, анализа и визуализации метрик.

## 2.1 Требования к системе мониторинга

Основываясь на анализе современных подходов к организации мониторинга в IT компаниях и выводах, полученных 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.

В таблице представлены конкретные требования к системе мониторинга по каждому из этих трех пунктов (см. таблица 2.1.).

| 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. |
| **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. |
| СУБД для долгосрочного хранения метрик | Потребность в долгосрочном хранении метрик. | Возможность просматривать метрики за период до 2-х лет. |

| | | |
|---|---|---|
| Сервис для перемещения метрик в долгосрочное хранилище | Потребность в перемещении метрик из краткосрочной базы данных в долгосрочное хранилище. | Возможность доставлять метрики из одного хранилища в другое. |
| Сервис для предоставления доступа к метрикам через API | Потребность в обращении к метрикам для внешних клиентов. | Получение метрик через API за определенный промежуток времени. |

Таблица 2.1. Требования к системе мониторинга

Основные преимущества данной системы мониторинга:

- удобный и исчерпывающий пользовательский API для написания технических и бизнес-метрик приложения;
- простота развертывания инфраструктуры для мониторинга;
- удобные инструменты для визуализации мониторинга приложения;
- возможность настройки правил для системы оповещений, анализа аномалий;

## 2.2 Используемые технологии

В процессе разработки системы мониторинга были использованы технологии для написания кода, его тестирования, а также для сбора, хранения и визуализации метрик. В таблице представлен список, использованных технологий, а также обоснование выбора данной технологии (см. таблица 2.2.).

| **Технология** | **Тип** | **Обоснование** |
|---|---|---|
| **Kotlin [15]** | Язык программирования | • удобный DSL для написания функционального кода;<br>• возможность использовать сопрограммы (англ. coroutines) для асинхронного сбора метрик;<br>• полная совместимость с Java (на Java написано большинство промышленных приложений) |
| **Spring Framework [16]** | Фреймворк для Java платформы | • возможности для внедрения зависимостей<br>• возможности для автоконфигурации библиотеки<br>• поддержка большинства промышленных приложений |
| **Junit [17]** | Библиотека для модульного тестирования Java | • параллельный запуск тестов<br>• подготовка тестовых данных<br>• интеграция с Spring Framework |
| **Docker [18]** | Программное обеспечение для контейнеризации приложений | • возможности для контейнеризации приложений<br>• простота конфигурации<br>• быстрое развертывание |

| | | |
|---|---|---|
| **ClickHouse [19]** | Система управления базами данных | • колоночная аналитическая СУБД<br>• движок для хранения временных рядов<br>• эффективные алгоритмы сжатия |
| **Prometheus [20]** | Система для сбора метрик | • удобство развертывания и поддержки<br>• большое сообщество по всему миру<br>• поддержка внутренней базы данных временных рядов |
| **Grafana [21]** | Система для визуализации метрик | • удобство развертывания и поддержки<br>• поддержка продвинутых инструментов для визуализации<br>• внутренняя система предупреждения |

Таблица 2.2. Используемые технологии системы мониторинга

Основной критерий при выборе данных технологий – современность и мировое признание. Система мониторинга должна использовать хорошо проверенные и надежные технологии, пользующиеся популярностью у большого количества компаний.

## 2.3 Реализация требований к системе мониторинга

Инструмент для работы с метриками, графиками и инструмент для анализа аномалий представляет из себя клиентскую библиотеку, которая состоит 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».

Сервис для перемещения метрик в долгосрочное хранилище представляет из себя отдельное приложение – NileMetrics Loader. Сервис для предоставления доступа к метрикам через API представляет из себя отдельное приложение – NileMetrics API. Оба сервиса были разработаны в рамка данной работы.

В таблице представлен набор сервисов для развертывания инфраструктуры с обоснованием выбора (см. таблица 2.3.).

| 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. |
| СУБД для долгосрочного хранения метрик | ClickHouse | Колоночная аналитическая СУБД с эффективными алгоритмами сжатия. |
| Сервис для перемещения метрик в долгосрочное хранилище | NileMetrics Loader | Разработанное приложение для эффективного перемещения метрик в долгосрочное хранилище. |
| Сервис для предоставления доступа к метрикам через API | NileMetrics API | Разработанное приложение для предоставления качественного API. |

Таблица 2.3. Используемые технологии системы мониторинга

Ниже представлена схема инфраструктуры системы мониторинга (см. рисунок 2.1.).

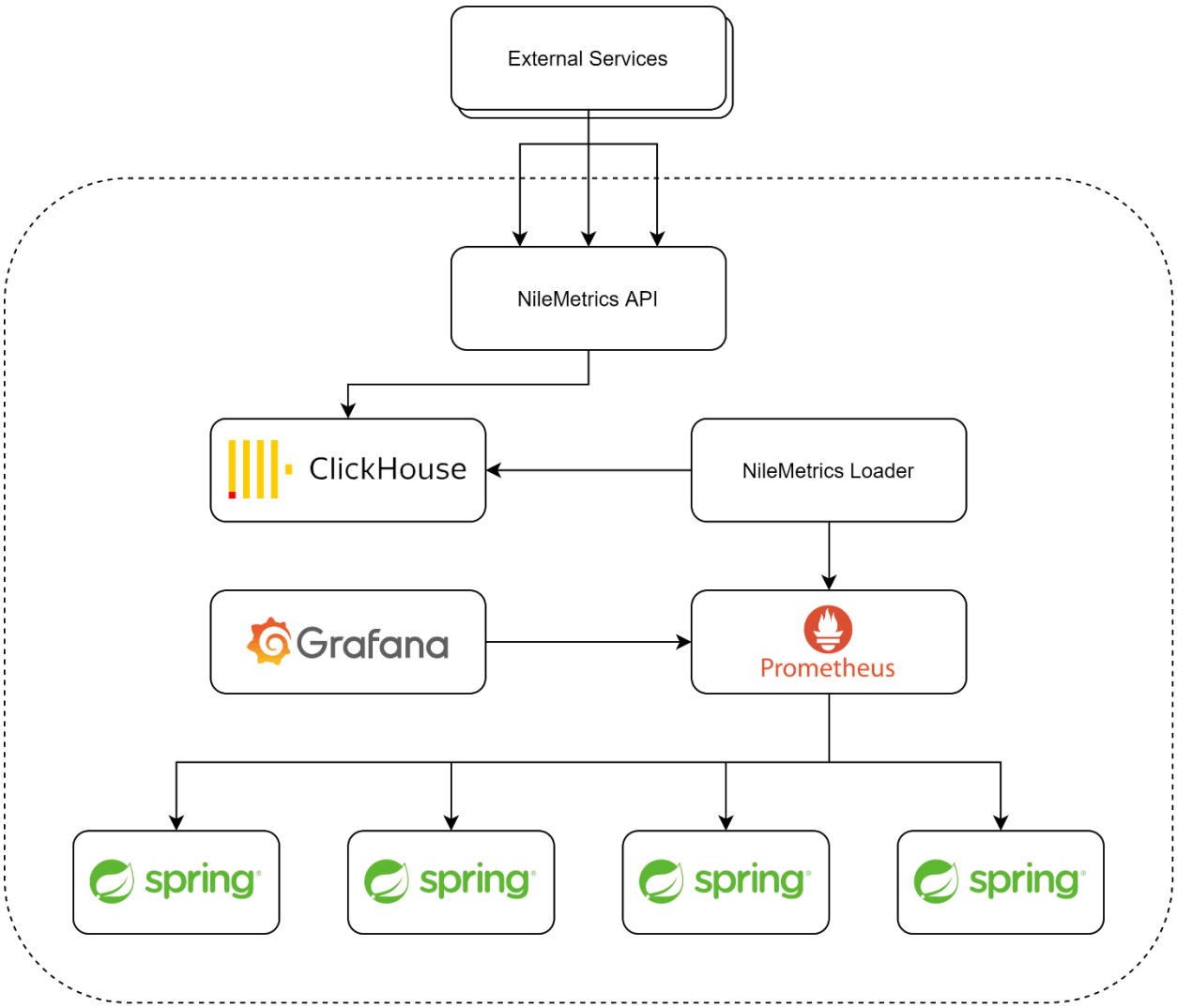


Рисунок 2.1. Схема инфраструктуры системы мониторинга

На данной схеме представлены следующие компоненты:

- *Spring приложения.* Набор приложений, которые выполняют бизнес-логику и используют библиотеки *micrometer, anomaly, grafana*.
- *Prometheus.* Система для сбора и краткосрочного хранения метрик приложений. Данная система обращается к наблюдаемым приложениям, собирает с них метрики и сохраняет в базу данных временных рядов.
- *Grafana.* Сервис для построения графиков с метриками. Данный сервис выполняет запросы к Prometheus с целью отображения графиков в реальном времени.
- *ClickHouse.* Колоночная аналитическая СУБД для долгосрочного хранения метрик.
- *NileMetrics Loader.* Разработанное приложение для перемещения метрик из Prometheus в ClickHouse. Данное приложение выполняет запросы в Prometheus и сохраняет данные с метриками в ClickHouse.
- *NileMetrics API.* Разработанное приложение для предоставления API внешним системам для проведения дополнительной аналитики за долгосрочный период.
- *External Services.* Внешние системы, выполняющие запросы к API для получения сведений о метриках за долгосрочный период.

## 2.4 Ограничения системы мониторинга

### Ограничения на уровне наблюдаемых приложений

Приложения должны удовлетворять следующим критериям:

1. Приложения должны быть написаны на языке Java или Kotlin.
2. Приложения должны использовать Spring Framework.

### Ограничения на уровне инфраструктуры

Инфраструктура для развертывания должна удовлетворять следующим критериям:

1. Инфраструктура должна поддерживать контейнеризацию Docker
2. Инфраструктура должна выделить дисковое пространство для СУБД ClickHouse, Prometheus и Grafana.
3. Инфраструктура должна гарантировать сетевую доступность между всеми компонентами системы мониторинга.

# Глава 3. Разработка сервиса мониторинга бизнес-приложений

## 3.1 Модуль для построения метрик

### 3.1.1 Метрики типа Counter

Метрики типа Counter – метрики-счетчики, позволяющие отслеживать кол-во выполненных запросов, сохраненных записей в базу данных и т. д. Ниже представлен пример метрики типа Counter (см. рисунок 3.1.).

```
# HELP weather_requests_counter_total How many requests were sent to weather API
# TYPE weather_requests_counter_total counter
weather_requests_counter_total{application="nile-application",status="SUCCESS",} 120.0
weather_requests_counter_total{application="nile-application",status="FAILURE",} 1.0
```

Рисунок 3.1. Метрика типа Counter

Данная метрика показывает кол-во выполненных запросов к стороннему API погодных условий. У данной метрики есть два тэга – *application* (название приложения, которое экспортирует данную метрику) и *status* (статус, с которым выполнился запрос).

На графике представлена визуализация метрики количества запросов к API погодных условий (см. рисунок 3.2).

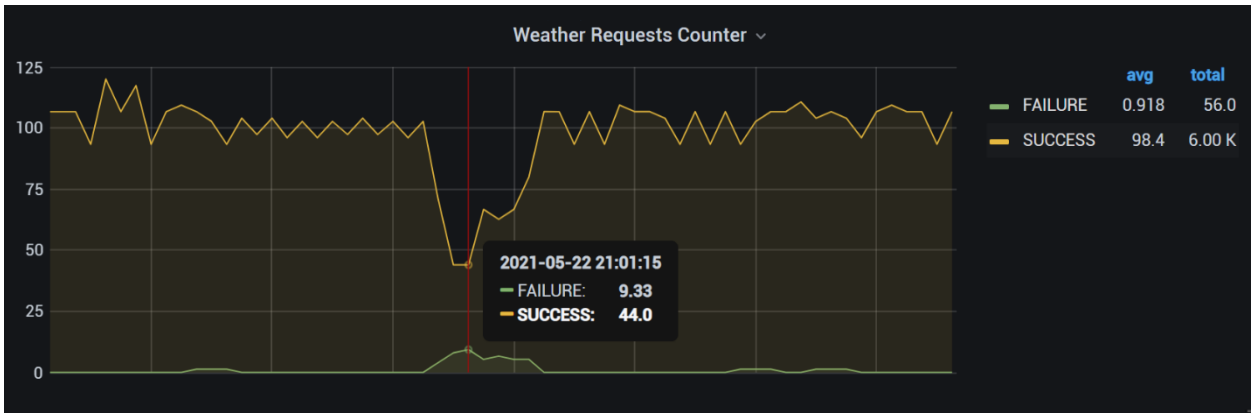


Рисунок 3.2. Визуализация метрики количества запросов с API погодных условий

### 3.1.2 Метрики типа 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.). Ниже показан пример метрики типа Timer (см. рисунок 3.3.).

```
# HELP weather_requests_timer_seconds How long does it take to process single request
# TYPE weather_requests_timer_seconds summary
weather_requests_timer_seconds_count{application="nile-application",status="SUCCESS",} 2480.0
weather_requests_timer_seconds_sum{application="nile-application",status="SUCCESS",} 210.6348671
weather_requests_timer_seconds_count{application="nile-application",status="FAILURE",} 16.0
weather_requests_timer_seconds_sum{application="nile-application",status="FAILURE",} 19.0365139
# HELP weather_requests_timer_seconds_max How long does it take to process single request
# TYPE weather_requests_timer_seconds_max gauge
weather_requests_timer_seconds_max{application="nile-application",status="SUCCESS",} 0.3091498
weather_requests_timer_seconds_max{application="nile-application",status="FAILURE",} 0.0
```

Рисунок 3.3. Метрика типа Timer

Данная метрика показывает какое кол-во запросов и за какое время было выполнено к API погодных условий. Если разделить время, затраченное на выполнение запросов на кол-во запросов, то можно получить среднее время выполнения одного запроса.

На графике представлена визуализация метрики-таймера запросов к API погодных условий (см. рисунок 3.4.).

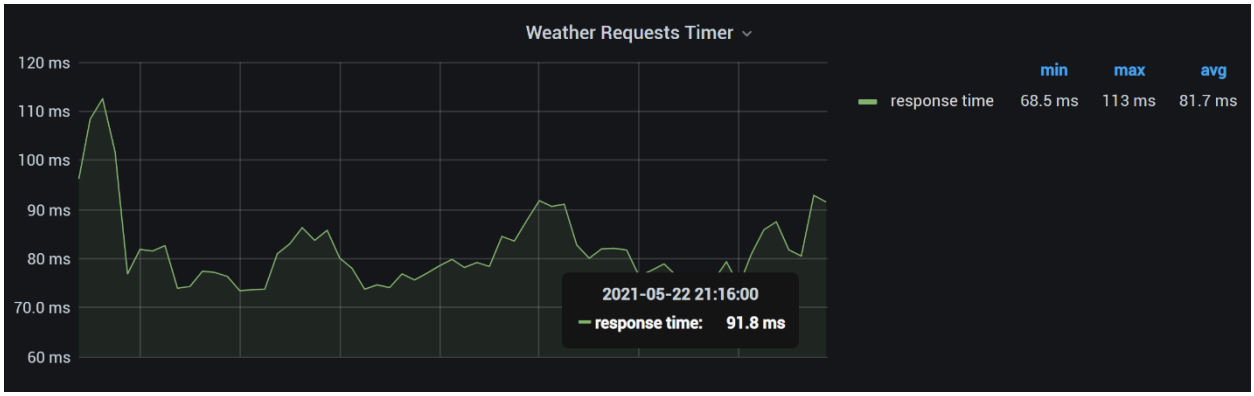


Рисунок 3.4. Визуализация метрики количества запросов с API погодных условий

### 3.1.3 Метрики типа Gauge

Метрики типа Gauge – метрики, которые отображают конкретные вещественные значения. Например, текущий объем очереди сообщений, текущее количество активных потоков приложения. Ниже представлен пример метрики типа Gauge (см. рисунок 3.5.).

```
# HELP temperature Celsius temperatures in cities all around the world
# TYPE temperature gauge
temperature{application="nile-application",city="Paris",status="SUCCESS",} 12.84
temperature{application="nile-application",city="Moscow",status="SUCCESS",} 17.26
temperature{application="nile-application",city="Rome",status="SUCCESS",} 28.02
temperature{application="nile-application",city="Ottawa",status="SUCCESS",} 27.11
temperature{application="nile-application",city="Barcelona",status="SUCCESS",} 16.99
temperature{application="nile-application",city="Canberra",status="SUCCESS",} 0.66
temperature{application="nile-application",city="Beijing",status="SUCCESS",} 18.67
temperature{application="nile-application",city="Washington",status="SUCCESS",} 16.61
temperature{application="nile-application",city="London",status="SUCCESS",} 12.77
temperature{application="nile-application",city="Berlin",status="SUCCESS",} 13.12
```

Рисунок 3.5. Метрика типа Gauge

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*.

На графике представлена визуализация метрики температуры в разных городах (см. рисунок 3.6.).

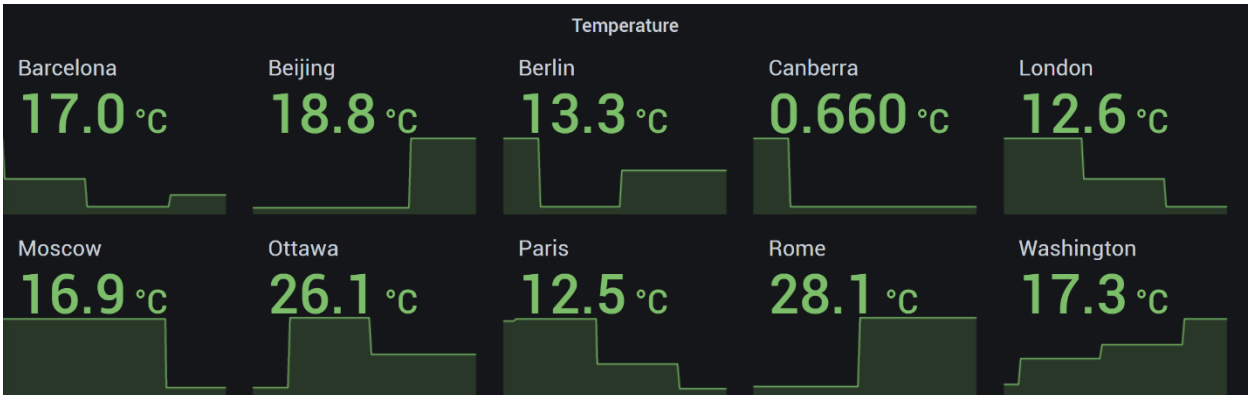


Рисунок 3.6. Визуализация метрики температуры в разных городах

### 3.1.4 Метрики типа Distribution Summary

Метрика типа Distribution Summary – метрика, которая отображает перцентили по заданным значениям.

```
# HELP temperature_distribution_summary_ Distribution summary of temperature in different cities
# TYPE temperature_distribution_summary_ summary
temperature_distribution_summary_{application="nile-application",status="SUCCESS",quantile="0.5",} 16.984375
temperature_distribution_summary_{application="nile-application",status="SUCCESS",quantile="0.75",} 18.984375
temperature_distribution_summary_{application="nile-application",status="SUCCESS",quantile="0.9",} 26.984375
temperature_distribution_summary_{application="nile-application",status="SUCCESS",quantile="0.95",} 28.984375
temperature_distribution_summary_{application="nile-application",status="SUCCESS",quantile="0.99",} 28.984375
temperature_distribution_summary_count{application="nile-application",status="SUCCESS",} 4913.0
temperature_distribution_summary_sum{application="nile-application",status="SUCCESS",} 80495.94000000114
```

Рисунок 3.7. Метрика типа Distribution Summary

Данная метрика показывает распределение температуры в градусах Цельсия по перцентилям.

На графике представлена визуализация метрики распределения температуры в разных городах с разбивкой по перцентилям.

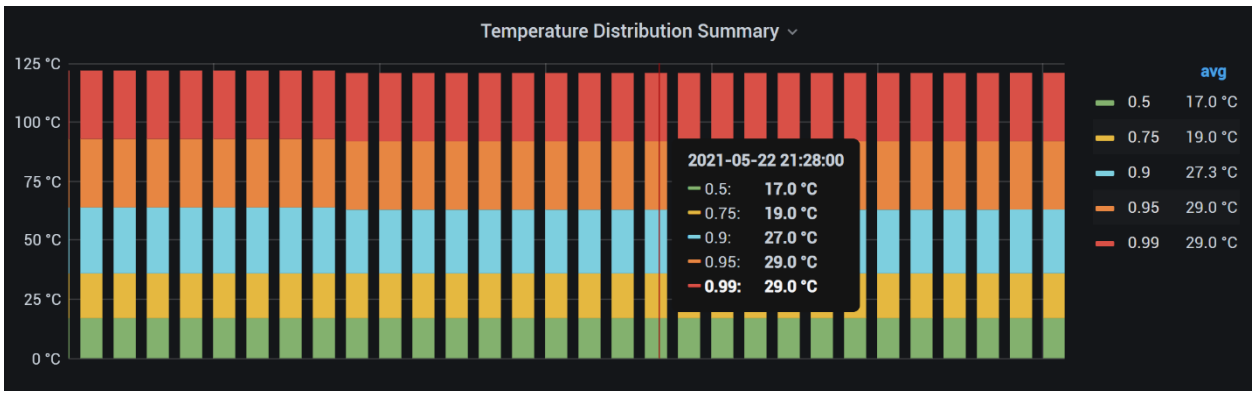


Рисунок 3.8. Визуализация метрики температуры в разных городах с разбивкой по перцентилям

## 3.2 Модуль для анализа аномалий

Модуль анализа аномалий – отдельный модуль, который предоставляет API для конфигурации обработчика метрик, который определяет наличие аномалии во временном ряде.

На стороне клиента (приложения) реализуется класс-обработчик, в который каждый раз при обновлении метрики любого типа поступает объект метрики и ее обновленное значение. Реализация в данном классе принимает решение, считать данное значение за аномалию или нет. В случае принятия положительного решения, доступен метод для срабатывания системы предупреждения.

## 3.3 Модуль для построения графиков

Модуль для построения графиков – отдельный модуль, предназначенный для конфигурации графиков в Grafana с помощью предметно ориентированного языка Kotlin DSL. Всякий раз при запуске приложения описанная конфигурация обновляет графики в Grafana. Таким образом, реализуется подход к управлению конфигурацией «Инфраструктура как код» (англ. Infrastructure as a code).

## 3.4 Приложение для перемещения метрик в долгосрочное хранилище

NileMetrics Loader – отдельное приложение, которые перемещает метрики в долгосрочное хранилище ClickHouse. Приложение с определенной периодичностью делает запросы в Prometheus для выгрузки последних обновлений метрик и пачками по несколько тысяч записей сохраняет обновленные метрики в ClickHouse.

## 3.5 Приложение для предоставления доступа к метрикам через API

NileMetrics API – отдельное приложение, которое по требованию выполняет запросы в ClickHouse и предоставляет API для работы с внешними запросами к хранилищу.

# ЗАКЛЮЧЕНИЕ

В данной работе была достигнута поставленная цель: была разработана система для построения мониторинга, анализа аномалий и своевременного предупреждения.

Были решены все поставленные задачи. В первой главе была подробно изучена предметная область мониторинга бизнес-приложений, были изучены существующие подходы к реализации мониторинга. Во второй главе, основываясь на анализе современных подходов к организации мониторинга в IT компаниях и выводах, полученных в ходе исследования, были сформулированы требования к разрабатываемой системе мониторинга со стороны конфигурации метрик, анализа аномалий, пользовательского интерфейса и инфраструктуры. В третьей главе была затронута разработка системы мониторинга с описанием конкретных разработанных модулей.

В ходе исследования было выявлено, что в существующих системах мониторинга достаточно строгие требования к развертыванию и поддержке, конфигурация бизнес-метрик является крайне затруднительной, а панели для визуализации обладают очень узкой функциональностью. В данной работе была предложна альтернативная архитектура системы мониторинга, состоящая из нескольких компонентов, удобных для развертывания и поддержки.

Подробная документация и код разработанной системы мониторинга доступны в открытом доступе в репозитории на Github – VolkovTech/nile [22]. Результатами данной работы могут пользоваться разработчики и компании по всему миру.

В дальнейшем данную систему мониторинга планируется поддерживать с помощью сообщества разработчиков, добавляя новую функциональность и исправляя возможные ошибки.

# СПИСОК ЛИТЕРАТУРЫ


1. Google Cloud Infrastructure Components Incident #20013 [Электронный ресурс] / Google Cloud Status Dashboard – URL: https://status.cloud.google.com/incident/zall/20013. (Дата обращения: 01.05.2021).
2. Проблемы с сетевой доступностью 5 февраля 2020 года [Электронный ресурс] / Блог компании «Яндекс» – URL: https://cloud.yandex.ru/blog/posts/2020/02/incident-report-february-5. (Дата обращения: 01.05.2021).
3. *Бетси Бейер, Крис Джоунс, Дженнифер Петофф, Нейл Ричард Мёрфи*. Site Reliability Engineering. Надежность и безотказность как в Google. – Издательство: Питер, 2019 г.
4. Что такое СУБД [Электронный ресурс] / Регистратор доменных имен Ru-Center – URL: https://www.nic.ru/help/chto-takoe-subd_8580.html. (Дата обращения: 01.05.2021).
5. Application Performance Management: What it is & How it works [Электронный ресурс] / Компания «Aimultiple» – URL: https://research.aimultiple.com/apm/. (Дата обращения: 01.05.2021).
6. Application Performance Monitoring 101: A Guide on How APM Works, Use Cases & Best Practices [Электронный ресурс] / Компания «Sematext» – URL: https://sematext.com/guides/application-performance-monitoring. (Дата обращения: 01.05.2021).
7. 5 лучших бесплатных систем мониторинга ИТ-инфраструктуры [Электронный ресурс] / Веб-сайт networkguru.ru – URL: https://networkguru.ru/5-besplatnykh-sistem-monitoringa-it-infrastruktury. (Дата обращения: 01.05.2021).
8. Федорова Людмила Михайловна. Системы мониторинга. Обзор и сравнение. // Московский технический университет связи и информатики, г. Москва – Вестник науки и образования № 10(88). Часть 4. 2020– С. 16–18.
9. Nagios – Official website [Электронный ресурс] / Компания «Nagios» – URL: https://www.nagios.org. (Дата обращения: 01.05.2021).
10. Zabbix – Official website [Электронный ресурс] / Компания «Zabbix» – URL: https://www.zabbix.com. (Дата обращения: 01.05.2021).
11. 51 инструмент для APM и мониторинга серверов [Электронный ресурс] / Блог компании «Сервер Молл» на Хабр – URL:

https://habr.com/ru/company/pc-administrator/blog/304356. (Дата обращения: 01.05.2021).
12. Anturis – Official website [Электронный ресурс] / Компания «Anturis» – URL: https://anturis.com. (Дата обращения: 01.05.2021).
13. Monitoring-as-a-service для инфраструктур на примере сервиса Anturis [Электронный ресурс] / Блог компании «Infobox» на Хабр – URL: https://habr.com/ru/company/infobox/blog/256497. (Дата обращения: 01.05.2021).
14. Easy AWS Infrastructure Monitoring [Электронный ресурс] / Компания «Instrumental» – URL: https://instrumentalapp.com. (Дата обращения: 01.05.2021).
15. Документация языка программирования Kotlin [Электронный ресурс] / Компания «JetBrains» – URL: https://kotlinlang.org. (Дата обращения: 01.05.2021).
16. Документация Spring Framework [Электронный ресурс] / Сообщество «Spring» – URL: http://spring.io. (Дата обращения: 01.05.2021).
17. Документация Junit 5 [Электронный ресурс] / Сообщество «Junit» – URL: https://junit.org/junit5. (Дата обращения: 01.05.2021).
18. Документация Docker [Электронный ресурс] / Компания «Docker» – URL: https://www.docker.com. (Дата обращения: 01.05.2021).
19. Документация ClickHouse [Электронный ресурс] / Компания «Яндекс» – URL: https://clickhouse.tech/docs/ru. (Дата обращения: 01.05.2021).
20. Документация Prometheus [Электронный ресурс] / Компания «Prometheus» – URL: https://prometheus.io. (Дата обращения: 01.05.2021).
21. Документация Grafana [Электронный ресурс] / Сообщество «Grafana» – URL: https://grafana.com. (Дата обращения: 01.05.2021).
22. Исходный код системы мониторинга Nile [Электронный ресурс] / Github - VolkovTech/nile – URL: https://github.com/VolkovTech/nile. (Дата обращения: 01.05.2021).