УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Отчет

О производственной практике

Разработка инструмента для автоматической генерации дашбордов Grafana по описанию

Студент

Митрофанов Е. P34101

Руководитель практики от профильной организации

Пономарев И.

Руководитель практики от университета

Маркина Т. А.

Санкт-Петербург, 2022 г.

Оглавление

Характеристика компании

Яндекс - крупная российская компания, которая занимается разработкой и предоставлением широкого спектра интернет-услуг. Она является лидером в области поисковых систем в России и странах СНГ, а также предоставляет услуги в области электронной коммерции, онлайн-картографии, транспортных сервисов, интернет-телевидения и многих других областях. Компания Яндекс известна своей инновационностью и высоким уровнем технологической разработки, что позволяет ей успешно конкурировать на мировом рынке.

Введение

Тема

Разработка инструмента для автоматической генерации дашбордов Grafana по описанию

Цель

Мониторинг данных необходим для того, чтобы следить за работой системы, выявлять проблемы и принимать меры по их устранению. В современных компьютерных системах сотни и тысячи компонентов, которые работают взаимосвязанно, и в случае возникновения проблемы может быть сложно понять, в какой части системы произошла ошибка. Мониторинг позволяет получать данные о состоянии системы в реальном времени, а также анализировать эти данные на предмет возможных проблем.

Ручная настройка дашбордов для каждого компонента очень большая задача, состоящая в основном из монотонных повторяемых действий, так как набор панелей мониторинга обычно совпадает и содержит набор метрик по http запросам, запросам в различные базы данных, общению с очередью сообщений и так далее. Поэтому процесс создания дашбордов можно автоматизировать. Цель работы - создание инструмента для генерации дашбордов систем мониторинга Grafana со всеми необходимыми метриками для каждого из сервисов.

Задачи и этапы выполнения

В рамках производственной практики в компании руководителем практики были выделены этапы для достижения поставленной цели и разделены на задачи в рамках этапа. Данные этапы приведены ниже. (Таблица 1)

Порядковый номер этапа	**Наименование этапа**	**Задание этапа**
1	Инструктаж обучающегося	Инструктаж обучающегося по ознакомлению с требованиями охраны труда, техники безопасности, пожарной безопасности, а также правилами внутреннего трудового распорядка
2	Ознакомление с Grafana	Ознакомление с Grafana, доступными инструментов и возможностями, изучение API для проектирования приложения, изучение доступных метрик и инструментов сбора данных
3	Проектирование приложения	Анализ необходимых для реализации метрик, описания генерации дашбордов, сбор метрик с внешних сервисов, на основании которых будет произведена визуализация
4	Разработка инструмента	Разработка инструмента, преобразующего декларативное описание дашборда в скрипт генерации Json для Grafana.
5	Тестирование разработанного приложения	Тестирование инструмента на различных сервисах, проверка и исправление ошибок
6	Анализ разработанного приложения	Анализ инструмента, выявление областей для улучшения
7	Оформление отчетной документации	Оформление отчетных документов в соответствии с требованиями

Таблица 1, этапы производственной практики

Основная часть

Ознакомление с Grafana

Визуализация данных

Grafana — это инструмент для визуализации и мониторинга данных, который позволяет создавать красивые и понятные графики, диаграммы и дашборды для анализа и отслеживания различных метрик. Он поддерживает множество источников данных, включая базы данных, системы мониторинга и API, и обладает широкими возможностями настройки и конфигурирования. Grafana может использоваться для мониторинга производительности приложений, сетей, серверов, баз данных и других систем, а также для анализа бизнес-метрик и KPI.

Рисунок 1, пример дашборда Grafana

Сбор метрик

Для сбора метрик может применяться множество систем, такие как Logstash, Kibana, а также различные базы данных. В окружении компании используется множество разных систем сбора метрик, но в данной работе будет использоваться интеграция с Prometheus.

Prometheus — это система мониторинга и алертинга, которая позволяет собирать, хранить и анализировать метрики различных систем. Она поддерживает множество протоколов для сбора метрик, включая HTTP, SNMP, JMX, StatsD и другие.

Интеграция Prometheus с Grafana позволяет создавать графики и дашборды на основе собранных метрик. После настройки соединения можно начать создавать графики и дашборды на основе метрик, собранных Prometheus. Для этого необходимо выбрать источник данных Prometheus в Grafana и использовать функции запросов PromQL для выборки нужных метрик.

Примеры использования Prometheus и Grafana:

- **Мониторинг производительности серверов**
  Prometheus может собирать метрики производительности серверов (например, загрузку процессора или использование памяти), а Grafana может использоваться для создания графиков и дашбордов для отслеживания этих метрик.

- **Мониторинг приложений**
  Prometheus может собирать метрики производительности приложений (например, время ответа или количество запросов), а Grafana может использоваться для создания графиков и дашбордов для отслеживания этих метрик.

- **Анализ бизнес-метрик**
  Prometheus может собирать метрики, связанные с бизнес-процессами (например, количество заказов или выручка), а Grafana может использоваться для создания графиков и дашбордов для анализа этих метрик.

Prometheus является одной из лучших систем сбора метрик по нескольким причинам:

- **Масштабируемость**
  Prometheus обеспечивает масштабируемость, позволяя собирать метрики с большого количества серверов и приложений. Он может обрабатывать огромные объемы данных, не теряя при этом производительности.

- **Гибкость**
  Prometheus поддерживает различные методы сбора метрик, такие как сбор метрик через HTTP, JMX, SNMP, Exporters и другие. Он также позволяет настраивать алерты и оповещения, чтобы оперативно реагировать на проблемы.

- **Простота установки и использования**
  Prometheus представляет собой один бинарный файл, который можно быстро установить и настроить. Он имеет простой и интуитивно понятный язык запросов PromQL, который позволяет быстро и легко извлекать нужные данные.

- **Открытый и расширяемый**
  Prometheus является открытым и расширяемым инструментом. Он имеет активное сообщество разработчиков и пользователей, которые создают и поддерживают множество инструментов и библиотек для расширения его функциональности.

Реализация сбора метрик

Библиотека io.micrometer - это инструмент для сбора метрик приложений, который позволяет интегрировать различные системы мониторинга, включая Prometheus.

Использование io.micrometer с Prometheus позволяет автоматически собирать метрики приложений и отправлять их в Prometheus для дальнейшего анализа и отображения в Grafana.

Для использования io.micrometer с Prometheus необходимо добавить зависимость в проект и настроить соединение с Prometheus. Для этого можно использовать класс PrometheusMeterRegistry, который предоставляет методы для регистрации метрик и отправки их в Prometheus.

Пример реализации

Рассмотрим реализацию системы сбора метрик на практическом примере – получение информации об использовании S3 – объектной базы данных, которая является частью системы AWS.

В Prometheus есть четыре основных типа метрик:

1. **Counter (Счетчик)**
   Этот тип метрик используется для отслеживания количественных значений, которые могут только увеличиваться, например, количество запросов к серверу или количество ошибок. Каждый счетчик имеет имя и значение, которое увеличивается каждый раз при поступлении новых данных. Значение может сбрасываться или уменьшаться только в том случае, если сервер перезапущен.

2. **Gauge (Индикатор)**
   Этот тип метрик используется для отслеживания значений, которые могут как увеличиваться, так и уменьшаться во времени, например, количество пользователей онлайн или загрузка CPU. Gauge метрики могут иметь произвольное начальное значение и изменяться в любом направлении.

3. **Histogram (Гистограмма)**
   Этот тип метрик используется для измерения распределения значений, например времени ответа сервера. Гистограммы автоматически подсчитывают количество значений, которые попадают в определенные диапазоны, называемые "корзинами". Гистограммы также вычисляют сумму и квадрат суммы значений, что позволяет вычислить среднее значение и стандартное отклонение.

4. **Summary (Сводка)**
   Этот тип метрик также используется для измерения распределения значений, но предоставляет несколько более сложную информацию, такую как медиана и 90-й перцентиль. Сводки также автоматически подсчитывают количество значений и их сумму, что позволяет вычислить среднее значение.

Заведем две новых метрики типа counter для подсчета загруженных и выгруженных байт информации в базе данных. Наследуемся от

ServiceMetricCollector, реализованного Amazon AWS, и реализовываем сбор метрик, увеличивая counter соответствующего типа на количество байт. (Рисунок 2)

Рисунок 2, реализация download и upload метрик

В наборе AWS также есть класс RequestMetricCollector, отвечающий за обработку запросов к базе. С его использованием мы можем собирать информацию о статусе ответа и о типе запроса. Для сбора метрик создадим counter и наполним его информацией с разбиением на tag. (Рисунок 3)

Позже на этом же примере отрисуем полученные метрики в Grafana.

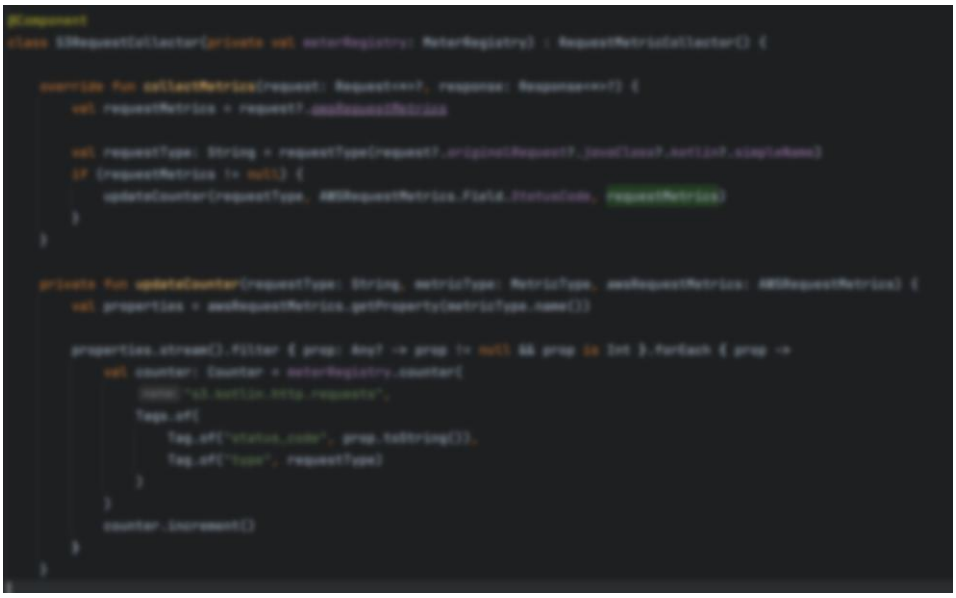

Рисунок 3, реализация метрик подсчета запросов

Описание генерируемых приложением метрик

Для корректного отображения метрик сервису необходимо сообщить приложению о том, какие метрики он умеет собирать. Это зависит от используемых баз данных, наличии обработчиков http запросов, работы с брокерами сообщений Kafka и другими данными.

Для описания данных удобно подойдет формат yaml

```yaml
s3:
  connections:
    # NDA

postgres:
  connections:
  # NDA
  databases:
  # NDA

ydb:
  databases:
  # NDA

stq-worker:
  queues:
  # NDA
```

Рисунок 4, пример service.yaml файла

Таким образом мы отметим конкретные соединения и механизмы, метрики для которых мы составляем в этом сервисе.

Язык разработки

Разрабатываемый инструмент должен запускаться мануально и не участвовать в CI/CD цикле, нет необходимости разрабатывать web-приложение, так как достаточно читать yaml описания и генерировать json для обновления дашборда с помощью API Grafana.

Для разработки используется язык Go по следующим причинам:

- **Эффективность**
  Go был разработан для быстрой разработки и эффективного выполнения кода. Он компилируется в машинный код и имеет низкий уровень абстракции, что обеспечивает высокую производительность.

- **Простота**
  Синтаксис Go очень простой и чистый. Это упрощает чтение и понимание кода и ускоряет разработку.

- **Многопоточность**
  Go имеет встроенную поддержку многопоточности, что делает его идеальным для разработки многопоточных приложений.

- **Надежность**
  Go предназначен для создания надежных и стабильных приложений. Он имеет встроенную обработку ошибок и автоматическое управление памятью, что снижает вероятность ошибок в коде.

- **Быстрая разработка**
  Go имеет большую стандартную библиотеку и множество сторонних библиотек, что делает его идеальным для быстрой разработки.

- **Кроссплатформенность**
  Go может быть скомпилирован на различные платформы, такие как Windows, Linux, Mac OS и т. д.

Это идеальный выбор при работе с большим количеством данных.

Библиотечные панели

Для реализации поставленной задачи прекрасно подойдет механизм библиотечных панелей Grafana – шаблонов, которые достаточно составить один раз, и используя различные переменные окружения, менять вызываемую метрику. Для вышеописанных метрик S3 мы можем реализовать подобную панель. (Рисунок 5)

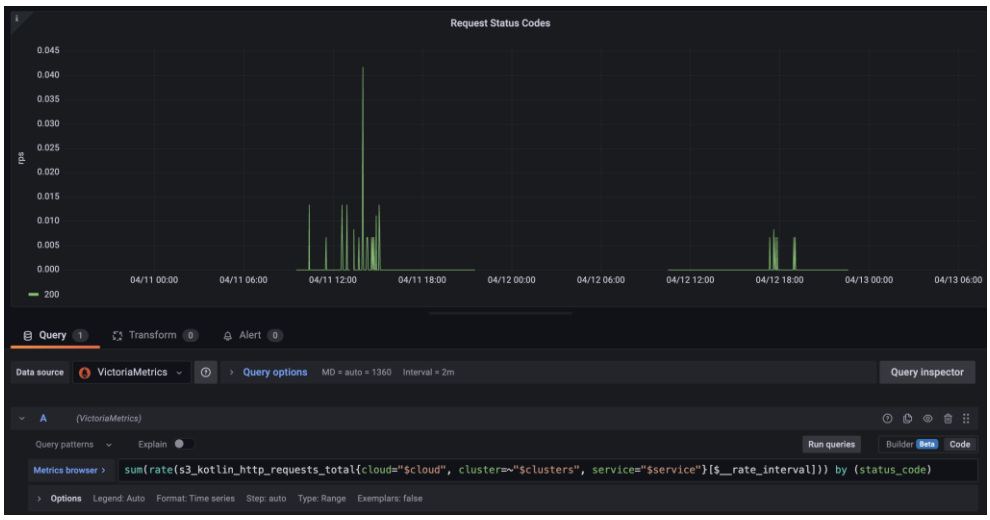

Рисунок 5, библиотечная панель метрики S3

Как видно в вызове метрики, мы подставляем переменные окружения, такие как название сервиса или тип окружения (тестовый или продуктовый).

Кроме того, мы может агрегировать все метрики по статусу ответа (200 в данном случае) и суммировать их по интервалам.

Подобные шаблоны мы реализуем для всех метрик. Ниже приведена небольшая часть списка библиотечных панелей.

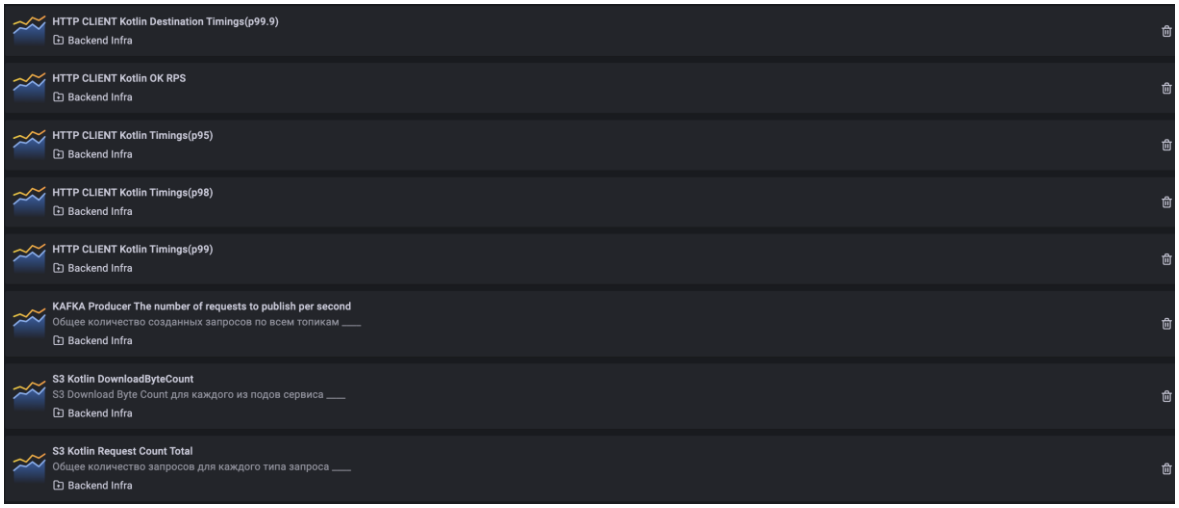

Рисунок 6, список библиотечных панелей

Архитектура приложения

Ниже представлена архитектура приложения. (Рисунок 7)

Система собирает информацию о собираемых метриках из *service.yaml* и использует ее для заполнения переменных, которые будут использованы в библиотечных панелях.

Вся полученная информация, в том числе и сами панели, полученные по API в виде json – а используются для формирования общего дашборда сервиса со всеми панелями.

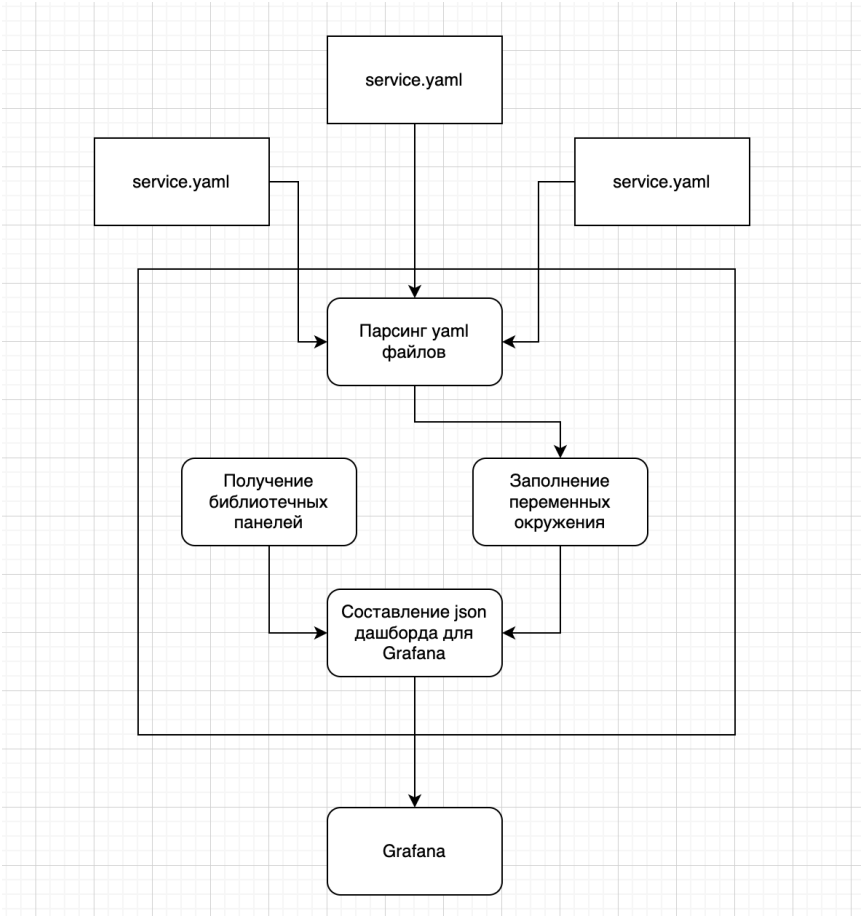

Рисунок 7, диаграмма архитектуры приложения

Разработка инструмента

Процесс разработки можно разбить на несколько составляющих, в соответствии с архитектурой. Для иллюстрации взяты лишь небольшие фрагменты кода.

Парсинг yaml файла

Генерация панелей всех типов

Пример плагина для S3

```
func (plugin s3Plugin) AddPanels(d *grafana.Dashboard, context *plugins.Context) error {
    row := grafana.NewRow( title: "S3",  collapsed: true)

    row.AddPanel(grafana.NewLibraryPanel( name: "S3 Kotlin DownloadByteCount"))
    row.AddPanel(grafana.NewLibraryPanel( name: "S3 Kotlin UploadByteCount"))
    row.AddPanel(grafana.NewLibraryPanel( name: "S3 Kotlin Request Count Total"))
    row.AddPanel(grafana.NewLibraryPanel( name: "S3 Kotlin Status Code OK"))

    d.AddRow(row)
    return nil
}
```

Добавление переменных окружения

Выгрузка дашборда в Grafana

В процессе разработки появилась необходимость хранения библиотечных панелей в json формате внутри приложения. Так как пока что

данные хранятся только внутри Grafana, их повреждение может иметь серьезные последствия на всех дашбордах сразу. Поэтому созданные ранее библиотечные панели на этапе генерации также будут сохраняться в файлы в качестве бэкапа. (Рисунок 8)

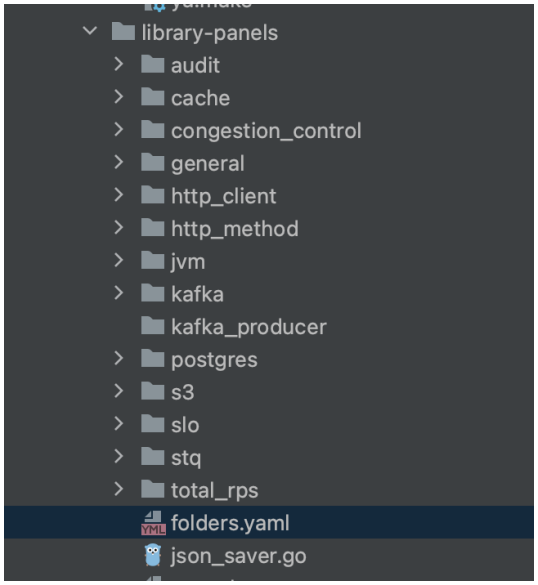

Рисунок 8, разделы для всех типов панелей

Тестирование разработанного приложения

При запуске приложения происходит парсинг аргументов командной строки, в которых содержится перечисление сервисов для генерации дашборда, данные авторизации в Grafana и другие флаги режимов.

После чего происходит процесс получения всех библиотечных панелей для необходимых метрик, наполнение панелей переменными окружения и формирование общего большого json-а, описывающего весь дашборд. Процесс генерации занимает около минуты, из-за того, что присутствует множество панелей для разных целей (более 80), некоторые панели

дублируются (http запросы для каждого из url), некоторые панели агрегируются для составления общих показателей.

В результате тестирования, были обнаружены некоторые ошибки и неточности в документации к API Grafana, в результате чего данные иногда приходили неполными. Для исправления таких ошибок потребовалось написать несколько собственных модулей, дополняющих данные, получаемые из API.

Несколько примеров метрик на дашборде

Рисунок 9, информация о доступности сервиса

Рисунок 10, информация о HTTP запросах

Рисунок 11, информация о взаимодействии с Postgres

Анализ разработанного приложения

В результате тестирования и опыта использования инструмента генерации были выделены несколько параметров для дальнейшего улучшения

- **Уменьшение времени генерации дашборда**
  Независимо от того, что регенерация происходит обычно достаточно редко, процесс ожидания при обновлении всех дашбордов сразу может занимать до получаса. Сократить врем работы можно путем хранения json данных о панелях в собственной базе данных, чтобы не отправлять запросы на их получение в Grafana

- **Доработка описания сервиса в yaml формате**
  Так как для генерации всех дашбордов используются общие шаблоны, их изменение возможно только в ручном режиме после генерации. Поэтому учесть указания стилей, типа визуализации и подобных параметров на этапе генерации невозможно. Для этого необходимо доработать формат описания генерации и расширить пул используемых шаблонов.

Заключение

В результате прохождения практики я получил новые знания касательно мониторинга данных, систем сбора и генерации метрик, разработки на языках Go и Kotlin, и использовал их для создания инструмента, решающего задачу серьезной инфраструктурной оптимизации в большой компании. Я перенял опыт коллег и использую его вместе с полученными знаниями в дальнейшей работе и профессиональном развитии.