МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

УРАЛЬСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

имени первого Президента России Б.Н. Ельцина ИНСТИТУТ ЕСТЕСТВЕННЫХ НАУК И МАТЕМАТИКИ

Кафедра высокопроизводительных компьютерных технологий

**ПОСТРОЕНИЕ СИСТЕМЫ МОНИТОРИНГА**

**БИЗНЕС-ПРИЛОЖЕНИЙ.**

Направление подготовки «Математика и компьютерные науки» 03.02.01

|  |  |
| --- | --- |
| Заведующий кафедрой:  д.ф.-м.н., с.н.с,  М. Ю. Филимонов | Выпускная квалификационная работа бакалавра  **Сагалова Леонида Борисовича** |
| Нормоконтролер:  к.ф.-м.н., В. С. Зверев | Научный руководитель: к.ф.-м.н., В. С. Зверев |

Екатеринбург  
2022

**РЕФЕРАТ**

Дипломная работа студента МЕН-480-207 Сагалова Леонида Борисовича по теме «Построение системы мониторинга бизнес -приложений» состоит из введения, 4-х разделов и 8 подразделов, заключения и списка литературы из 7-ми источников. Работа изложена на 36 страницах печатного текста.

Исследование в рамках дипломной работы направлено на изучение подходов к мониторингу современных технологических систем, изучение инструментов, обеспечивающих построение систем наблюдения за целевыми приложениями, реализацию собственной системы с помощью выбранного подхода к наблюдениям. Работа состоит из двух частей. В первой освещается проблема и необходимость по построению системы, происходит знакомство с основными понятиями и терминами. Вторая часть работы является практической, в ней описаны используемые инструменты и способ их применения в предоставляемом решении, а так же произведен анализ нескольких методологий, выбрана и реализована одна из них.

**СОДЕРЖАНИЕ**

1. **Введение** 4

2. **Постановка проблемы** 6

[3. **Теоретическая часть** 7](#_Toc105026214)

3.1 Понятие мониторинг 7

3.2 Инструменты мониторинга 8

3.3 SR-инженеры и контракты по доступности 11

4. **Практическая часть** 14

4.1 Реализация сборщика метрик 14

4.2 Запуск хранилища метрик VictoriaMetrics 18

4.3 Запуск сборщика vmagent 21

5. **Методологии мониторинга** 23

5.1 Описание методологий 23

5.2 Реализация целевой методологии 30

**Заключение** 35

**Список литературы** 36

**ВВЕДЕНИЕ**

Сфера информационных технологий плотно вошла в нашу жизнь. Сейчас сложно представить себе отрасль, которая могла бы обходиться без какой-то компоненты технологического прогресса. В частности, без сервисов, обеспечивающих работу как веб приложений, так и мобильных приложений. Оплата жилищно-коммунальных услуг, покупка еды, взаимодействие с банками, оплата передвижения, порталы и веб-сайты различных организаций, предоставляющих необходимую информацию и услуги — все это было бы невозможно без технологического развития специализированных приложений. В рамках работы предлагается обсудить способы достижения круглосуточной доступности сервисов подобного рода, реализацию вспомогательных инструментов и сопутствующие этому проблемы.

Второстепенной целью данной работы ставится погружение в мир информационных технологий, объяснение того, что скрывает под собой термин «мониторинг» применимо к данной сфере и конкретно к приложениям, обеспечивающим исполнение потребностей пользователя в различных сценариях. Описать компоненты целевой сущности, объяснить их отличия и пояснить необходимость каждой из компонент при эксплуатации современных технологических решений.

В заключении вступительной части поговорим про так называемых sr-engineer или SRE (от англ. Site Reliability Engineering). Так называют технических инженеров, часть деятельности которых непосредственно связана с мониторингом, его реализацией, настройкой и последующим использованием. Не обойдем стороной и тему заключения контрактов уровня надежности ИТ-систем между такого рода инженерами и заказчиками, целиком и полностью основанную на том же самом мониторинге.

Во второй части приведены детали технической реализации: разбор необходимого инструментария, посредством которого мониторинг реализуется. Так же будут описаны различные подходы к измерениям и соответствующему построению систем, реализующему эти подходы.

Основная часть работы - описание целевого метода наблюдения за прикладным приложением, реализуемого в рамках дипломной работы. Сравним плюсы и минусы выбранного решения с другими, выявим особенности. А затем перейдем к технической реализации, опишем используемые компоненты и их взаимодействие, поясним выбор при необходимости. Будут учтены сложности и особенности с оглядкой на целевой метод наблюдения.

В заключительной части будут подведены итоги. Будет описано конечное решение, рассмотрены его плюсы и минусы. Обсудим применимость реализованной системы в широком спектре потребления. Оценим трудозатраты на подобную реализацию и, конечно, прикладную пользу.

**ПОСТАНОВКА ПРОБЛЕМЫ**

Сервис подобного рода, как описанный во введении, несомненно, добавляет комфорта в жизнь простого человека. И такому человеку чаще всего необходимо, чтобы используемые им услуги были доступны в полной мере в тот момент, когда у него возникнет необходимость ими воспользоваться. И это абсолютно справедливое ожидание, так как существуют различные параметры, будь то разница в часовых поясах или образ жизни конкретного потребителя, требующие работы той или иной услуги в любой момент времени. Так что круглосуточная доступность большинства приложений является стандартом де-факто на сегодняшний день.

Рассмотрим, с какими сложностями сопряжено достижение круглосуточной доступности и с помощью каких инструментов это возможно обеспечить. Все компоненты, из которых состоит тот или иной технологический продукт, являются материальными и живут в физическом мире, пусть и по своим законам. Стойки с серверами, системные платы, электроэнергия, обновление программного обеспечения – все они подвержены различного рода изменениям, поломкам, являются очень нестабильными компонентами и в то же время основополагающими для желаемой доступности. В условиях постоянных изменений и множества факторов, на часть из которых нет возможности повлиять, появляется необходимость в инструментах наблюдения за составными частями системы, возможности анализировать текущее состояние и предсказывать возможное поведение в будущем. И это справедливо для многих сфер работы и жизни в современном мире, а в частности – для сферы информационных технологий.

**ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

**3.1 Понятие мониторинг**

Перейдем к такому понятию, как «мониторинг». Так, например, в информационном ресурсе «википедия» дается такое определение:

“Мониторинг — система постоянного наблюдения за явлениями и процессами, проходящими в окружающей среде и обществе, результаты которого служат для обоснования управленческих решений по обеспечению безопасности людей и объектов экономики. В рамках системы наблюдения происходит оценка, контроль объекта, управление состоянием объекта в зависимости от воздействия определённых факторов” [1].

Из определения становится понятно, что основная идея заключается в достаточно очевидном факте – явления и процессы подвержены наблюдению, а результаты наблюдений, в свою очередь, являются информационным оплотом для последующего принятия решений, связанных с наблюдаемыми объектами. То же самое применимо и к сфере информационных технологий.

Теперь рассмотрим данный термин применимо ИТ-системам. Справедливым будет обратиться к трактовке понятия от компании Google, чьи продукты находятся в повсеместном использовании и демонстрируют высочайший уровень надежности. Вспомните, когда в последний раз у вас не вышло воспользоваться поисковиком? А ведь этим сервисом пользуются миллионы людей по всему земному шару в каждый момент времени. Так что ссылка на данную компанию более чем уместна в контексте разговора про надежные системы и средства по поддержанию такого уровня доступности. Компания дает такое определение:

«Мониторинг — это инструмент или техническое решение, позволяющее командам отслеживать и понимать состояние своих систем. Мониторинг основан на сборе предопределенных наборов метрик или журналов. Наблюдаемость — это инструмент или техническое решение, которое позволяет командам активно отлаживать свою систему» [2].

Ключевая мысль здесь, по моему мнению, в том, что этот комплексный инструмент позволяет отслеживать и понимать состояние систем. Это своеобразный язык общения между инженерами и эксплуатируемыми системами. Чтобы последние работали стабильно и «чувствовали себя хорошо» с технической точки зрения, необходимо понимать их текущее состояние и дальнейшие поведенческие тенденции. А это становится возможным лишь опираясь на информационные показатели.

**3.1 Инструменты мониторинга**

В связи с вышесказанным возникает вопрос: каким же образом эти показатели воплощаются в жизнь, то есть что собой представляет мониторинг, из чего состоит и как трактовать добытую с помощью него информацию?

В качестве ответа на вопрос возникло два базовых инструмента:

метрики[[1]](#footnote-1) и логи[[2]](#footnote-2). Базируясь на этих столпах важное место занимает третий компонент – графическое представление, так как визуализация более доступна и воспринимаема человеком.

Лог можно представить как дневник или ежедневник, только применимо к ИТ-продукту. Это своего рода контейнер для сбора информации о том, что происходит при работе системы. В этот контейнер может быть помещена любая информация, которой оперирует рассматриваемая система и которую создатель системы посчитает нужным зафиксировать на длительный срок для каких-либо целей. На заре развития технологий такая информация писалась в текстовые файлы, но с течением времени были созданы новые подходы для работы с такими данными, а также целые системы хранения и работы с лог записями.

Метрики - более строгие информационные характеристики, за счет того, что основаны они на математических понятиях и оперируют цифрами. Основных выделяют четыре типа:

* Счетчик – можно инкрементировать или обнулять.
* Измеритель – можно инкрементировать и декрементировать.
* Гистограмма - выборка наблюдений, подсчет в настраиваемых временных сегментах, отображение суммы всех наблюдаемых вычислений.
* Сводка – выборка наблюдений, отражает сумму всех наблюдаемых вычислений, вычисляет квантили[[3]](#footnote-3) в скользящем временном окне [3].

Если первые два типа основаны на простейших математических операциях и представляют собой количественную индикацию события, то вторые два уже оперируют более сложными математическими понятиями из области статистики. Так, например, гистограммы являются наглядными представлениями функции плотности вероятности некоторой случайной величины. С помощью такого рода метрик становится возможным рассмотрение показателей прикладного приложения с помощью математического аппарата, который, в свою очередь, при правильном использовании предоставляет данные, достоверность которых неопровержима.

Стоит упомянуть и о графическом представлении. Чаще всего оно применимо непосредственно к метрикам, так как визуализировать численные показатели естественно для их лучшего восприятия. Суть такого представления заключается в следующем – строится координатная плоскость на осях, одной из которых, чаще всего горизонтальной, является время, а другой – количественная мера измерения (проценты, квантили, числа). Затем в этой системе отображаются данные в соответствии с форматом и значениями. Это не что иное, как оболочка над математическими данными, с ее помощью анализ и использование информации, основанной на этих данных, становятся естественнее и проще.



*Рисунок 3.1 –Визуализация метрик на примере, полученном в ходе выполнения работы*

Описанные выше типы метрик являются стандартом для клиентских библиотек, работающих с Prometheus[[4]](#footnote-4). Который, в свою очередь, представляет инструментарий для настройки и последующей работы с ключевым понятием моей работы, а именно, с мониторингом. В настоящий момент набор инструментов данной системы представляет собой стандарт в области информационных технологий, является решением с открытым исходным кодом, и активно развивается.

Разобраны основные верхнеуровневые сущности, оперируя которыми становится возможным производить наблюдение за поведением и состояние сервиса. Перейдем к разбору того, кем и как данные наблюдений будут использоваться, какой практический смысл они могут приносить.

**3.3 SR-инженеры и контракты по доступности**

В начале работы был сделан акцент на том, что конечный пользователь любого продукта потенциально желает круглосуточной доступности используемых им сервисов. Затем мы перешли к тому, что доступность невозможна без наблюдаемости и анализа текущего и возможного будущего состояния системы. Цепочка взаимосвязей постепенно приводит нас к тому, что необходим специальный человек (или группа людей), задачей которых будет трактовка полученных в ходе мониторинга данных и последующие действия, основанные на этих данных. Концепция инженеров подобного рода зародилась в 2003 году, в компании Google. Изначально это была даже не концепция инженеров, а набор принципов и практик, которые в будущем спроецировались в отдельную профессию на рынке труда. И называлась она - проектирование надежности сайта[[5]](#footnote-5) (от англ. Site Reliability Engineering[[6]](#footnote-6))[4]. Одной и основных рабочих задач такого инженера является обеспечение надежности и доступности подконтрольных ему систем, что напрямую следует из названия специальности.

Поговорим о том, в каких целях представители упомянутой профессии используют упомянутые системы мониторинга. В процессе работы команда SR-инженеров непременно столкнется с необходимостью заключения и выполнения контрактов по надежности и качеству работы вверенных им систем. Такие контракты заключаются с заказчиками и называются «соглашением об уровне предоставления услуги»[[7]](#footnote-7) (от англ. Service Level Agreement; SLA). Уровень предоставление услуги, в свою очередь, определяется с помощью различных метрик. А метрики эти становятся наблюдаемы благодаря системе мониторинга.

Контракты чаще всего придерживаются единого формата - они задают требуемый уровень обслуживания по конкретным метрикам и измеряют его в процентах за интервал времени. Весь интервал переводится в количество минут, они, в свою очередь, переводятся в проценты. Иначе – сколько минут включает в себя 1 %. И, в конечном итоге, соглашение заключается на то, что конкретная метрика будет в пределах допустимых значений 99 % времени, а 1 % времени в эквиваленте количества минут допустимо отклонение от нормы. Это допустимое время отклонения по-другому называется «бюджет ошибок».

Задача инженеров – любыми доступными им способами выполнить условия заключенного контракта. Рычагов и способы соблюдения различны, в зависимости от контекста и ситуации, но суть всегда остается неизменной.

И для того, чтобы иметь нужные данные, понимание состояния системы и статус выполнения такого договора – необходимо построить систему мониторинга, предоставляющую все необходимые опции и упрощающую работу с сущностью SLA. Пример построения такой системы - основная цель работы.

Методологий по построению таких систем множество, большинство из них описаны в книге, написанной инженерами компании Google - The Site Reliability Workbook: Practical Ways to Implement SRE. Отличие этих подходов заключается в том, каким набором параметров инженер может оперировать. В зависимости от этого набора система мониторинга предоставляет различную информацию. В дипломной работе реализован самый глубокий из подходов, оперирующий такими понятиями как скорость сгорания бюджета, бюджет ошибок, временные окна различной длины. Благодаря этому система более детально и точно указывать как на показатели сервиса, так и на текущее состояние контракта по SLA, чем значительно упрощает жизнь инженеров, генерируя точные оповещения о проблемах и предоставляя достоверную информацию. Готового продукта, реализовавшего такой подход, в открытом доступе не нашло на момент выполнения работы, в связи с этим он и был реализован собственноручно, с помощью доступных на момент выполнения инструментов.

**ПРАКТИЧЕСКАЯ ЧАСТЬ**

**4.1 Реализация сборщика метрик**

Важной частью реализации любой системы является системный дизайн. Это процесс определения всех компонент системы, продумывания архитектуры их построения, возможных состояний системы. Отдельным пунктом необходимо отметить такое свойство, как отказоустойчивость (это особенно важно в случае реализации системы мониторинга, так как ее главный функционал – предоставить наблюдение за основной системой, а при неполадках с мониторингом теряется возможность отслеживания состояния целевого продукта, что по современным меркам недопустимо). Так же в этом процессе необходимо учесть способы и технологии, посредством которых компоненты будут взаимодействовать между собой и третьими системами, при их наличии. Стоит подумать и о том, в какие моменты может понадобиться влияние человека для изменения поведения уже работающего продукта, чтобы сделать процесс внесения изменений логичным и удобным.

В процессе описания каждой из компонент и их воплощения, будут отмечены их необходимые свойства и обращено внимание на особенности, связанные с различными аспектами из области системного дизайна.

Начнем с того, каким образом связать целевое приложение с системой мониторинга. Будем считать, что оно уже запущенно в какой-либо среде и может быть использовано по назначению, те выполнять заложенный в него функционал в полной мере. Основой инструмент для наблюдения за объектом – метрики. Допустим, что последние уже реализованы на стороне приложения. Это значит, что в процессе жизненного цикла программного обеспечения будут производится некоторые подсчеты состояния его важных свойств.

Первое, что нам необходимо реализовать при имеющихся входных объектах – скрепер[[8]](#footnote-8). Это специальный утилитарный сервис, задачи которого: сбор с метрик с объекта/ов наблюдения, переработка полученных данных, возврат итогового результата следующей компоненте.

Скрепер будет представлять собой простейший веб-сервер, работающий по протоколу http[[9]](#footnote-9) в обе стороны. С помощью метода GET[[10]](#footnote-10) вышеуказанного протокола сервер обращается к целевому приложению на специальный адрес, откуда получает информацию в текстовом формате (детали реализации по предоставлению этой информации на другой стороне – остаются за рамками работы).

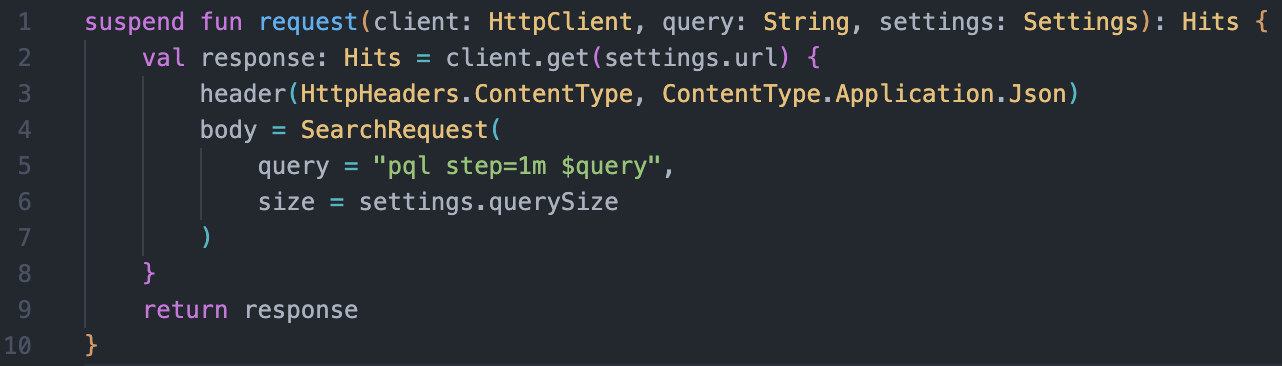


Рисунок 4.1.1 - Листинг фрагмента кода с http get-запросом в источник метрик

Что необходимо учесть при создании этого компонента:

* возможность извне регулировать набор запрашиваемых объектов по именам (названиям метрик)
* наличие нескольких одновременно работающих экземпляров, заменяющих друг друга в случае отказа
* работа в режиме многопоточности[[11]](#footnote-11), для более эффективного выполнения поставленных перед компонентом задач

Функционал по корректировке набора запрашиваемых объектов реализован посредством переменных окружения[[12]](#footnote-12). Такой подход является простым для реализации и соответствует принятому паттерну в мире разработки ПО, подразумевающему под собой, что изменение поведения сервиса должно происходить снаружи, то есть не с помощью изменения программного кода, а с помощью передаваемых настроек, которые обрабатываются исполняемым кодом.

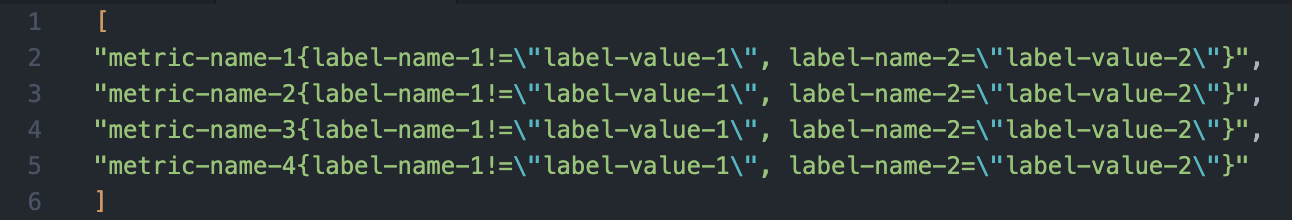


Рисунок 4.1.2 - Листинг с набором метрик в переменной окружения в формате JSON

Инфраструктура, в которой располагается скрепер, подразумевает два физически независимых дата-центра. Это значит, что в схеме работы будут присутствовать как минимум два одновременно функционирующих инстанса. Такая архитектура уже подразумевает под собой очень неплохую отказоустойчивость, но мы можем увеличить количество экземпляров до 2 в каждом из дата-центров. В таком случае мы имеем уже 4 инстанса, при необходимости лишь одного в конкретный момент времени. Этого вполне достаточно, для гарантии постоянной доступности описываемого сервиса.

Многопоточность реализована в контексте http запросов с помощью корутин[[13]](#footnote-13) – средства выбранного языка программирования. Используя данные синтаксические конструкции, мы закладываем в функционал возможность параллельного выполнения нескольких запросов за интересующими нас данными. Что в свою очередь означает более эффективное получение необходимой информации за один цикл работы.

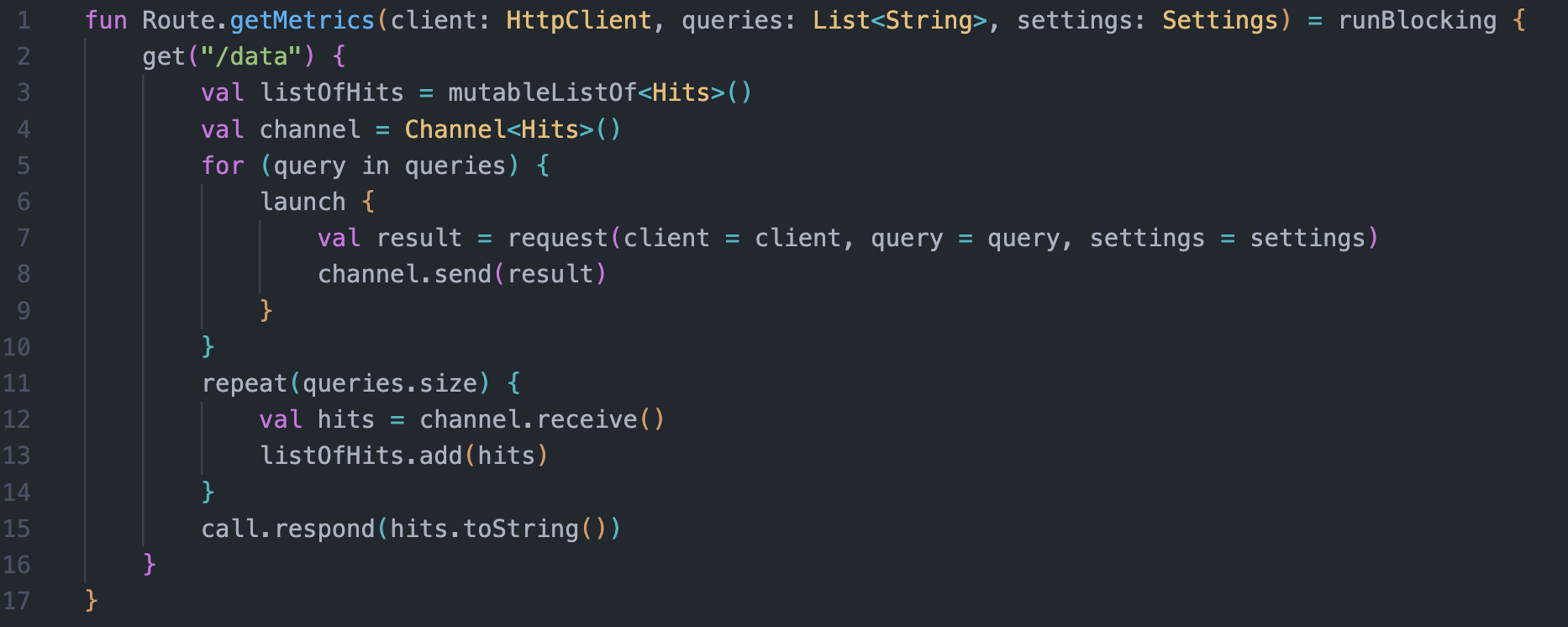


Рисунок 4.1.3 - Листинг фрагмент кода с функционалом корутин при http запросе

Описанная утилитарная компонента по итогам реализации представляет собой достаточно отказоустойчивый за счет инфраструктуры и надежный за счет минимального функционала сервис. Вмешательство инженера в его работу подразумевается только в том случае, когда появляется необходимость изменения списка запрашиваемых данных. Для того случая продуман удобный интерфейс взаимодействия – через переменные окружения. Потенциал для расширения – горизонтальный, то есть увеличение пропускной способности будет достигаться путем увеличения количества экземпляров.

Теперь, когда мы смогли агрегировать все интересующие показатели наблюдаемого приложения в одном месте (в веб-сервере, описанном выше), встает вопрос о том, каким образом хранить эту информацию. Как отмечалось ранее, мониторинг не только показывает текущее состояние объекта, но также предоставляет возможности анализа показателей на длительном промежутке времени, что подразумевает под собой хранение анализируемых данных на этом промежутке.

**4.2 Запуск хранилища метрик VictoriaMetrics**

Первое, что приходит в голову – хранение данных в оперативной памяти реализованного веб сервера. Но это будет не оптимально применительно к нашей реализации, так как такого рода память предназначена для быстрого доступа к часто запрашиваемым данным, а также она более дорогостоящая. Нас вполне устраивает хранение информации на так называемых накопителях, такого рода память предназначена для длительного хранения больших данных с достаточной скоростью их чтения.

В первой части работы мы уже упоминали о формате метрик Prometheus. Это общепринятый формат, на базе которого основано несколько крупных технических решений, предоставляющих инструменты для настройки и работы с мониторингом. Мы обратимся к решению под названием VictoriaMetrics и некоторым ее дополнительным компонентам.

Так как мы перешли к проблеме хранения, то первым рассмотрим непосредственно саму базу данных временных рядов (так называют метрики в контексте их хранения). Это специализированное решение, предназначенное для хранения метрик в том числе в формате Prometheus, распространяющееся по лицензии. Опять же нас не интересуют вопросы ее внутренней реализации, так как мы пользователи готового и зарекомендовавшего себя решения. Выбранный продукт распространяется в виде исполняемого файла[[14]](#footnote-14).

Хранилища различного рода в большей степени требуют архитектуры типа statefull, те сохраняющей состояние. Подразумевается тот факт, что при восстановлении сервиса после прекращения работы по любым причинам, необходимо восстановить его состояние на последний момент времени. Иными словами – потеря данных, полученных в процессе работы до ее прерывания недопустима. Это допущение в полной мере применимо к такой компоненте как хранилище. В связи с этим необходимо запускать базу данных временных рядов в инфраструктуре, отвечающей таким параметрам. Из доступных в момент реализации вариантов лучше всего подходит такая технологическая платформа, как виртуальная машина.

Необходимо заранее просчитать, какой примерный объем жесткого диска необходим для хранения данных, желательно с запасом. Приблизительными подсчетами было выяснено, что на хранение одного объекта временного ряда (иными словами – данных об одной метрике в один момент времени) требуется около пяти тысяч байт. Затем нужно умножить этот размер на количество единиц данных, записываемых в хранилище за минуту. Будем считать, что таковых будет двадцать. Итого имеем общий размер записываемых данных в минуту – сто тысяч байт. И, наконец, переведя интервал хранения информации в минуты, а затем умножив объем данных, поступающий за этот отрезок, становится возможным оценить объемы памяти, которые надо заложить под хранение. При первом приближении и использовании системы для мониторинга за одним крупным приложением затраты жесткого диска – более тринадцати гигабайт. С учетом относительной дешевизны данного типа памяти это вполне приемлемо. Таким образом, мы смогли оценить размер хранилища и понимаем тенденции для его расширения с увеличением количества наблюдаемых приложений или метрик.

Единственное, о чем еще осталось подумать, с точки зрения запуска в работу описываемого хранилища – отказоустойчивость. С развитием технологических решений это стал важный аспект и практически каждая команда инженеров реализуя свой продукт, задумывается об этом. Разработчики базы данных временных рядов VictoriaMetrics не исключение. Они предлагают два режима работы своего продукта – в единичном экземпляре и в режиме кластера. Если с первым все понятно, то что же предлагает инсталляция второго типа? А представляет она собой классическую схему горизонтального размещения нескольких одинаковых хранилищ и настраиваемой между ними репликацией[[15]](#footnote-15). Естественно, это подразумевает в том числе и использование дополнительных мощностей (в нашем случае виртуальных машин). Используя на официальную документацию, которая рекомендует использовать единичную инсталляцию в том случае, если количество записываемых данных в секунду менее миллиона (и режим кластера в ином случае), а так же опираясь на предполагаемые объемы данных при эксплуатации, на первом этапе было принято решение использовать хранилище в одном экземпляре.

В остальном этот компонент системы мониторинга достаточно независимый после запуска и не требует стороннего вмешательства во время его работы. Ему необходимо гарантирующая сохранение состояния платформа и достаточный объем ее физических ресурсов для стабильного функционирования. Все это было учтено на достаточном уровне.

Итак, что мы имеем на данный момент. Продумана и реализована первая компонента системы, которая отвечает непосредственно за сбор целевой информации. Затем независимо запущено хранилище на основе зарекомендовавшего себя открытого решения. Появляется логичный вопрос – как доставить данные из точки их агрегации в точку хранения.

**4.3 Запуск сборщика vmagent**

Отвечая на вопрос доставки данных из точки сбора в точку хранения, мы опишем следующую часть системы – еще один скрепер, но уже на основе аналогично стандартизированного решения и имеющего необходимые интерфейсы для работы с хранилищем. Называется он vmagent, и идеологически он мало отличается от реализованного до этого сборщика. Основная причина его использования в том, что он умеет напрямую взаимодействовать с базой данных по протоколу http и записывать туда наборы данных в нужном формате. И реализуется этот функционал с минимальными затратами, что делает использование vmagent-a в качестве некого дубляжа оправданным.

Работает он схожим образом с веб-сервером, по pull[[16]](#footnote-16) модели. Роль сервера в данной модели выполняет первый скрепер, именно от «порождает ответ». А роль клиента остается за описываемым, так как его задача запросить данные.

Касаемо устойчивости к отказам – здесь все ровным счетом так же, как с первым компонентом. Два дата-центра, от двух и более одновременно работающих копий. Такая схема позволяет нам реализовать «систему сборщиков» в абсолютно идентичной архитектуре, что снимает лишнюю нагрузку по их обслуживанию и поддержке в случае необходимости.

Все, что остается сделать – корректно настроить необходимые параметры работы для vmagent-а. Тот, на который непременно стоит обратить внимание, называется remoteWrite\_url и задается через переменную окружения. Подразумевает под собой адрес целевого хранилища, куда полученная информация будет записываться. Этот адрес и будет адресом виртуальной машины, на которой работает VictoriaMetrics.

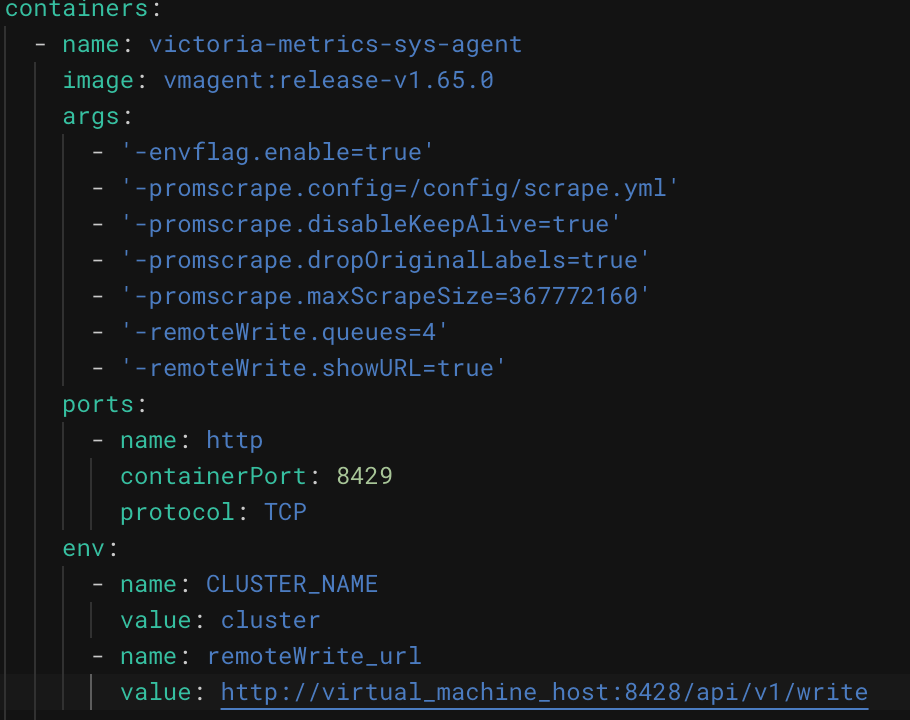


Рисунок 4.3.1- Листинг фрагмента описания манифеста для запуска vmagent в среде Kubernetes

Таким способом мы доставили данные до места, где они складируются в ожидании использования.

**МЕТОДОЛОГИИ МОНИТОРИНГА**

Всю первую часть работы описывалось, что из себя представляют интересующие данные, как и откуда их собирать, где хранить и каким образом в это место доставить. Перейдем ко второй части, в которой обсудим сам мониторинг, подходы по измерению и анализу, связанные с этим детали.

Вспомним про контракты с заказчиками – SLA. Как уже упоминалось, они задают требуемый уровень обслуживания и отражают его состояние в процентном соотношении за выбранный период. И, конечно, важной частью системы мониторинга являются оповещения для инженеров о угрожающих невыполнению контракта инцидентах.

Подходов по настройке такого рода оповещений – несколько, есть как самые тривиальные, так и требующие концентрации для понимания того, как результаты данного подхода коррелируют с реальной ситуацией и какой смысл в себе несут. На простом примере мы разберем существующие методологии, их плюсы и минусы, отталкивать от того, что имеется одно наблюдаемое значение – количество обработанных с ошибкой события в системе.

**5.1 Описание методологий**

Первый способ измерений основан на простом факте – сервис начинает не соответствовать заявленному контракту в том случае, когда количество ошибок начнет превышать заявленную границу за небольшое временное окно, установленную в рамках соглашения. В таком случае система мониторинга сразу же оповещает об этом факте. Ниже показан график взаимосвязи между временем обнаружения и уровнем ошибок, с учетом окна оповещения в 10 минут.

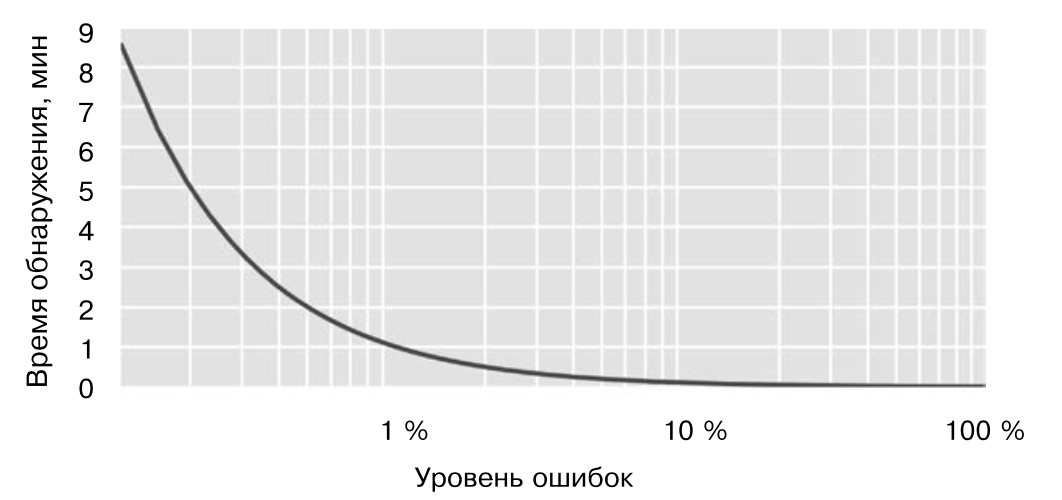


Рисунок 5.1.1 - Время обнаружения для примера сервиса с окном оповещения 10 мин и SLO 99,9 %

Рассмотрим преимущества и недостатки такого подхода. В качестве преимуществ можно выделить практически мгновенное время обнаружения проблемы – меньше 1 секунды в случае, если ошибки будут составлять 100% от общего числа событий. И отличный уровень отзывчивости, так как информирование произойдет при любом количестве ошибок, угрожающем невыполнению SLO. Что касается недостатков, то они таковы: если учитывать, что оповещение функционирует в разрезе контракта, то такой подход чреват его срабатыванием при событиях, которые соблюдению контракта могут не угрожать. Например, частота ошибок 0,1 % в течение 10 мин может привести

к оповещению, при этом расходуется всего 0,02 % ежемесячного бюджета ошибок. К такому информированию можно утратить доверие и однажды пропустить действительно критичный порог ошибок.

Описанный метод подлежит некоторому улучшению без значимых трудовых затрат. Достаточно увеличить размер окна, на котором производится наблюдение. Это приведет к тому, что время обнаружение незначительно увеличится, оставаясь еще на хорошем уровне. Но в то же время точность будет выше, чем в предыдущем примере, за счет гарантии того, что уровень ошибок сохраняется дольше и оповещение, скорее всего, будут говорить о серьезной угрозе для бюджета ошибок. Но тут есть другая проблема, заключающаяся в том, что в случае, когда в начале окна количество ошибок достигло 100%, а затем система восстановилась, велика вероятность продолжать получать оповещение на оставшемся интервале, хотя в действительности проблемы уже не существует.

Следующий подход основан на регулировании задержки для отправки сообщения о неполадке. Если в течение определенного времени контролируемое значение не будет непрерывно превышать пороговое, то не сгенерируется оповещение. Плюс этого подхода очевиден – требование устойчивого уровня ошибок на протяжении какого-то интервала до отправки информационного сообщения гарантирует с большой вероятностью, что оно соответствует значимому проблемному событию. Но очевиден существенный недостаток: низкая отзывчивость и длительное время обнаружения, поскольку продолжительность не зависит от серьезности инцидента, оповещение о 100%-ных сбоях в работе приходит через час — то же время обнаружения, что и для 0,2%-ного сбоя. 100 % сбоев за час потребляют 140% 30-дневного бюджета. Если показатель хотя бы на мгновение возвращается к уровню SLO, таймер продолжительности сбрасывается. SLI, скачущий между попаданием и непопаданием в рамки, оповещения не выдаст.

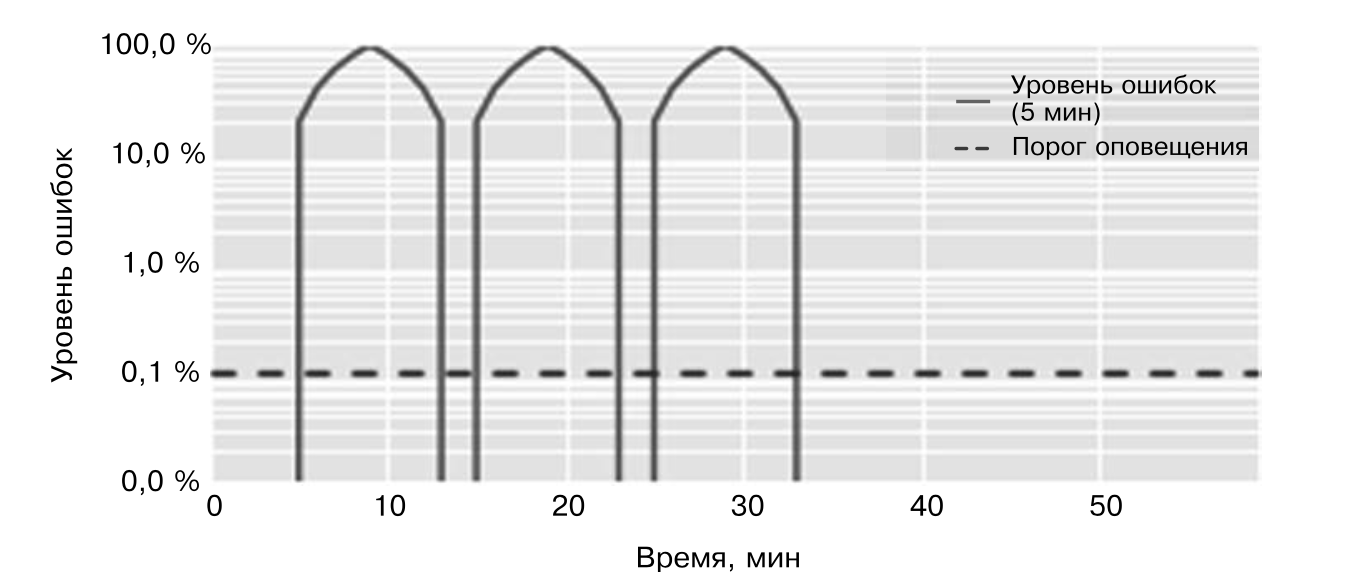
Данный факт наглядно отображается на следующем графике: 

Рисунок 5.1.2 - Сервис со 100% пиковыми уровнями ошибок каждые 10 минут

Каждый всплеск потреблял почти 12 % 30-дневного бюджета, но оповещение никогда не выдавалось. По такой причине подход не рекомендуется к использованию.

Перейдем к стратегиям, более усложненным, но вместе с тем и более точным, за счет того, что они оперируют таким понятием как скорость сгорания (от англ. burn rate) бюджета ошибок. Скорость сгорания — показатель того, как быстро сервис расходует бюджет ошибок в пересчете на период, предусмотренный SLO. Ниже приведена взаимосвязь между скоростью сгорания и соответствующими бюджетами.

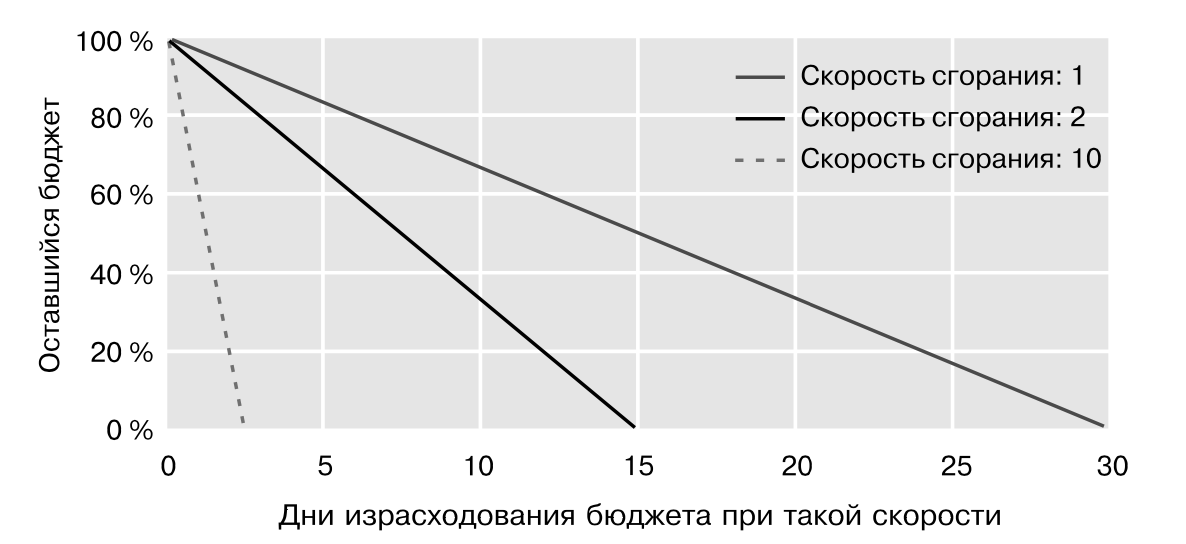


Рисунок 5.1.3 - Бюджеты ошибок относительно скоростей сгорания

Скорость сгорания равная единице подразумевает под собой трату бюджета ошибок с так называемой единичной скоростью относительно периода, на который заключен контракт. Иначе это значит, что израсходуются все квоты на возможные ошибки за период, но договор при этом не будет нарушен. Наглядно можно увидеть зависимость между различными скоростями и временем исчерпания недоступности во времени.

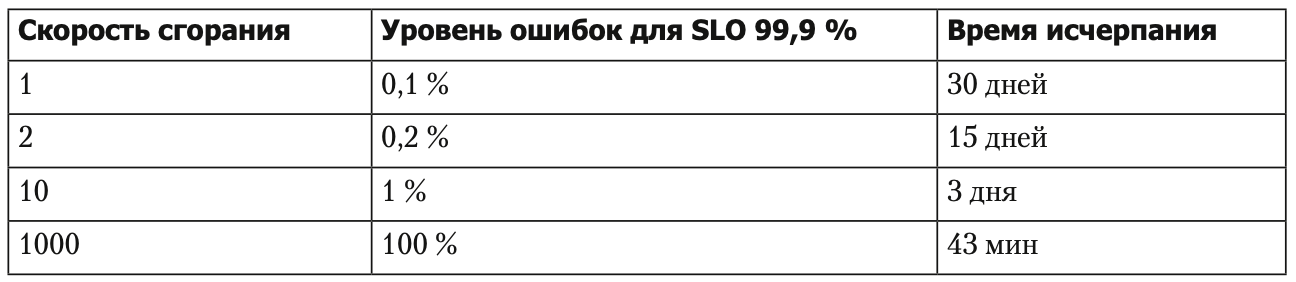


Рисунок 5.1.4 - Скорости сгорания и время, требуемое на исчерпание бюджета.

Очевидны основные плюсы такого решения: оповещение будет прислано при расходе значительной части бюджета ошибок за выбранное временное окно, так же инцидент будет достаточно быстро обнаружен. Но могут возникнуть трудности с выбором размера интервала, что может привести к таким ситуациям, когда оповещение не будет доставлено в срок, но при это скорость сгорания будет постоянно больше единичной, что по итогу приведет к несоблюдению договора.

Далее, возможно улучшить мониторинг на основе скорости сгорания за счет оперирования сразу несколькими скоростями. Эта настройка позволяет сохранить преимущества оповещений на основе скорости сгорания и гарантировать, что вы не пропустите более низкий (но все же значительный) уровень ошибок. Еще в данном подходе впервые оповещения делятся на два типа: критичные и информационные (в терминологии page и ticket). В то время как первые требуют безотлагательного вмешательства в разбор инцидента, вторые предупреждают о наличии крупной проблемы, но все еще находящейся в рамках разумного и не требующей мгновенной реакции. Работа со скоростями сгорания позволяет настроить оповещения, устанавливая соответствующий приоритет в зависимости от необходимой скорости реакции. Если бюджет ошибок из-за нерешенной проблемы исчерпывается в течение нескольких часов или нескольких дней, отправка активного (page) уведомления целесообразна. В противном случае более подходящим является уведомление через ticket, при котором оповещение будет обработано в следующий рабочий день.

Такое решение покрывает практически все требования к хорошей системе мониторинга с информированием. Теперь есть инструменты для адаптации конфигурации под различные ситуации в зависимости от их критичности. Точность информирования высокая, как и отзывчивость все системы в общем. Из недостатков стоит отметить сложность управления такой системой: большое количество параметров всегда приводит к повышению риска ошибиться. А еще при примитивности системы, отвечающей за отправку оповещения, может получится так, что произойдет несколько информирований по одной и той же проблеме, что является бессмысленным действием.

Остался последний подход, усложняющий предыдущий за счет введения дополнительного параметра – короткого временного окна, чтобы проверить, расходуется ли бюджет по-прежнему на момент генерации оповещения или нет.

Хорошая отправная точка — сделать так, как показано на рисунке ниже, чтобы короткое окно составляло 1/12 длинного окна. На графике показаны оба порога оповещения. После появления 15% ошибок на протяжении 10 минут, среднее значение для короткого окна сразу же превышает пороговое значение для оповещения, а среднее значение длинного окна превышает пороговое значение через 5 минут, что приводит к выдаче оповещения. Среднее значение для короткого окна становится ниже порогового значения через 5 минут после окончания ошибок, после чего выдача оповещений прекращается. Среднее значение для длинного окна станет ниже порогового значения через 60 мин после прекращения ошибок.

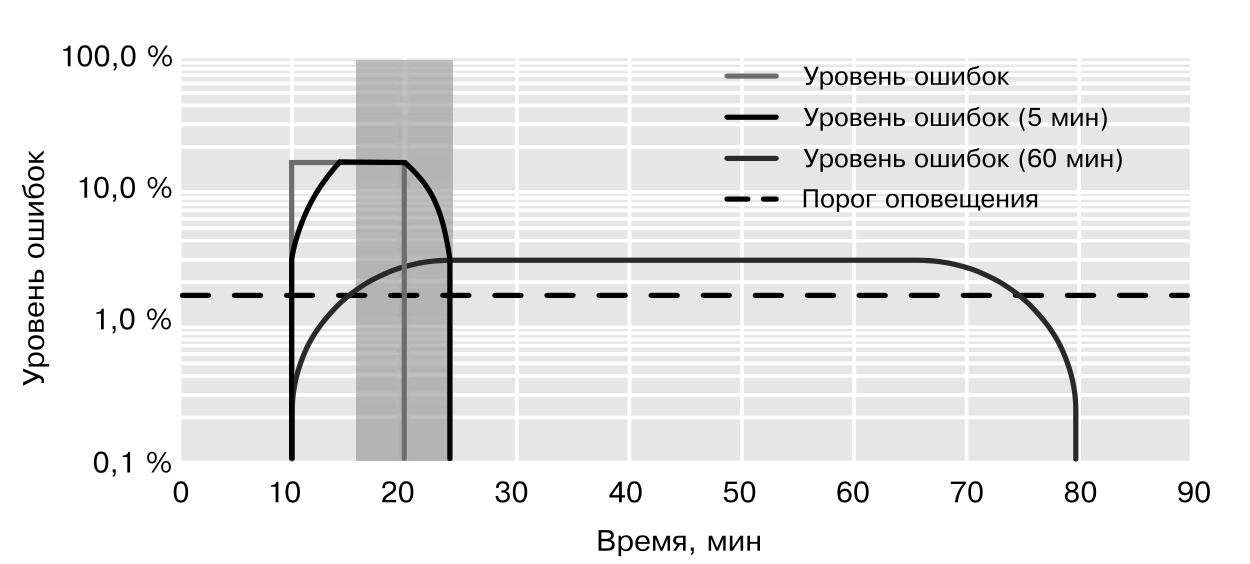


Рисунок 5.1.5 - Короткие и длинные окна оповещений

Инженеры компании Google рекомендуют в качестве отправной точки для конфигурации системы использовать следующие параметры:

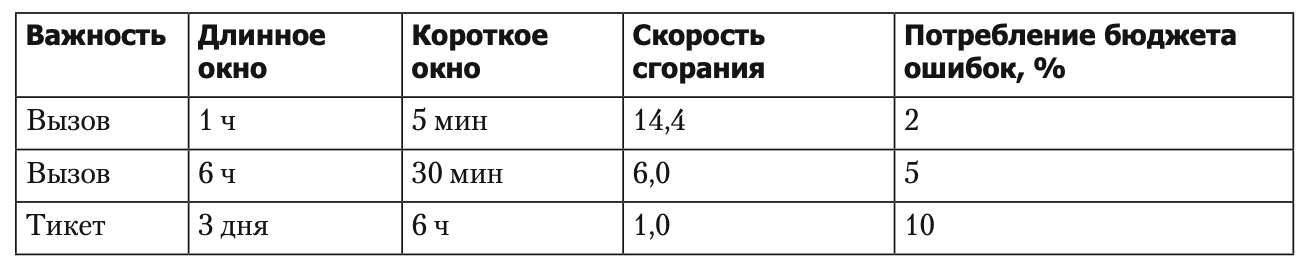


Рисунок 5.1.6 - Рекомендуемые параметры для конфигурации системы оповещения при SLO 99,9 %

Это самый сложный из всех перечисленный методов мониторинга целевой системы. Но в то же время, самый эффективный, учитывающий одновременно множество факторов.

К сожалению, на практике мало кто настолько углубляется в проработку своей системы оповещений, чаще всего используются первые 3 подхода, за счет простоты реализации и настройки. В следствие чего на рынке нет открытых готовых решений, предлагающий систему мониторинга, основанную на последнем методе. В связи с этим он и был реализован в процессе воплощения системы мониторинга.

**5.2 Реализация целевой методологии**

Вернемся к нашим компонентам системы и вспомним, какие из них у нас уже реализованы. Скреперы, задача которых в сборе метрик в одном месте и отправка их в хранилище. И, непосредственно, само хранилище.

Следующий инструмент, который нам необходим и с помощью которого и будет реализован последний из описанных методов мониторинга – vmalert. Это еще один из продуктов с открытой лицензией и умеющий напрямую работать с базой VitoriaMetrics. Он оперирует напрямую метриками с помощью декларативного языка YAML[[17]](#footnote-17). Используя этот язык, мы можем задать вполне конкретные правила манипуляции с метриками для vmalert-а. И правила эти будут основаны на методологии мониторинга с оповещениями с использованием понятий бюджета ошибок, скорости сгорания и временных окон. Рассмотрим на конкретном примере формат описания правил и отметим места с отличительными особенностями, присущими подходу, взятому за основу.

Рассмотрим пример расчёта значения целевого показателя на интервале в 5 минут:

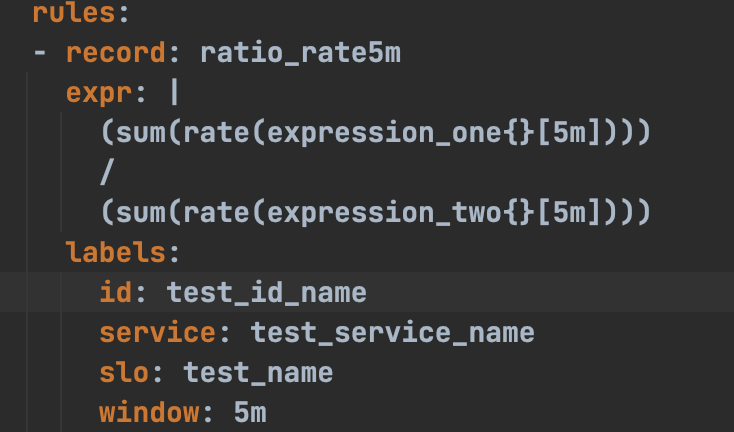


Рисунок 5.2.1 - Генерация целевого показателя на 5 минутном отрезке

Так в наше хранилище попадает новая метрика, с именем ratio\_rate5m. В поле expr находится целевое выражение, оперирующее метрикой и с помощью различных математических преобразований генерирующее текущее числовое состояние на отрезке в 5 минут (указан в квадратных скобках). Содержимое списка labels содержит различные обогащающие информативные поля, с помощью которых метрику можно получить из хранилища. Создав нужное исходя из методики количество правил с различными временными окнами, мы тем самым обеспечили себе одну из составляющих для ее реализации.

Второй шаг – создание правил оповещения:



Рисунок 5.2.2 - Правило генерации оповещения

Самое важное содержимое находится в поле expr. В нем содержится большое логически-математическое выражение, суть которого заключается в следующем: с помощью логического оператора and происходит сравнение показателей метрики на временных интервалах 5 минут и 1 час, аналогично 30 минут и 1 день, затем с помощью логического оператора or вычисляется результат логического выражения. В случае, если результат равен 1 (это значит, что показатель вышел за пределы как минимум одной из двух комбинаций временных окон) то оповещение будет отправлено. В ином случае – не будет. Если мы обратимся к Рисунку 8, представленному в работе ранее, то сможем заметить, что все эти параметры появляются прямиком из описанного подхода и являются рекомендуемыми ее создателями. Таким образом реализована основная его суть.

Реализуем дополнительные метрики, а с их помощью сможем визуализировать интересные параметры.

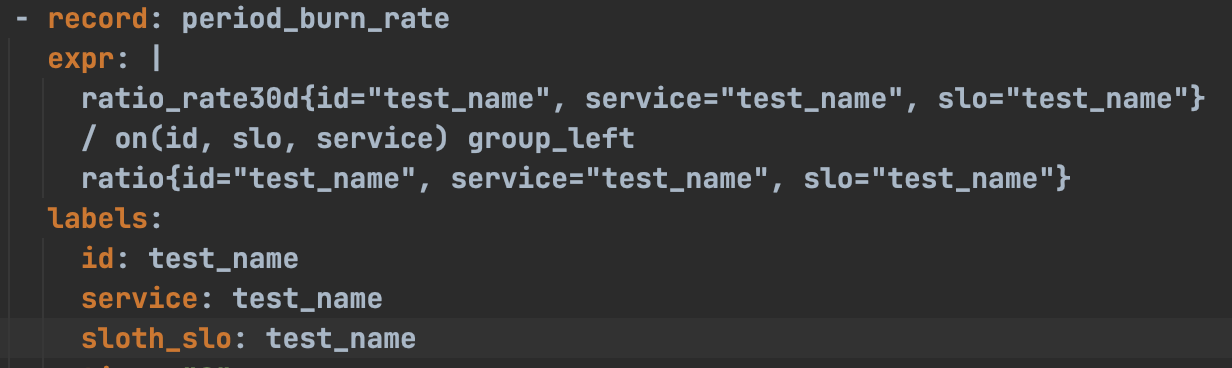


Рисунок 5.2.3 - Правило генерации метрики, отображающей скорость траты бюджета ошибок

На основе подобных правил в системе появляются метрики, использовав которые, можно построить наглядные визуализации. В результате мы можем видеть графики скорости сгорания бюджета относительно единичной, в абсолютных цифрах и в процентном соотношении периода, на котором производится подсчет.

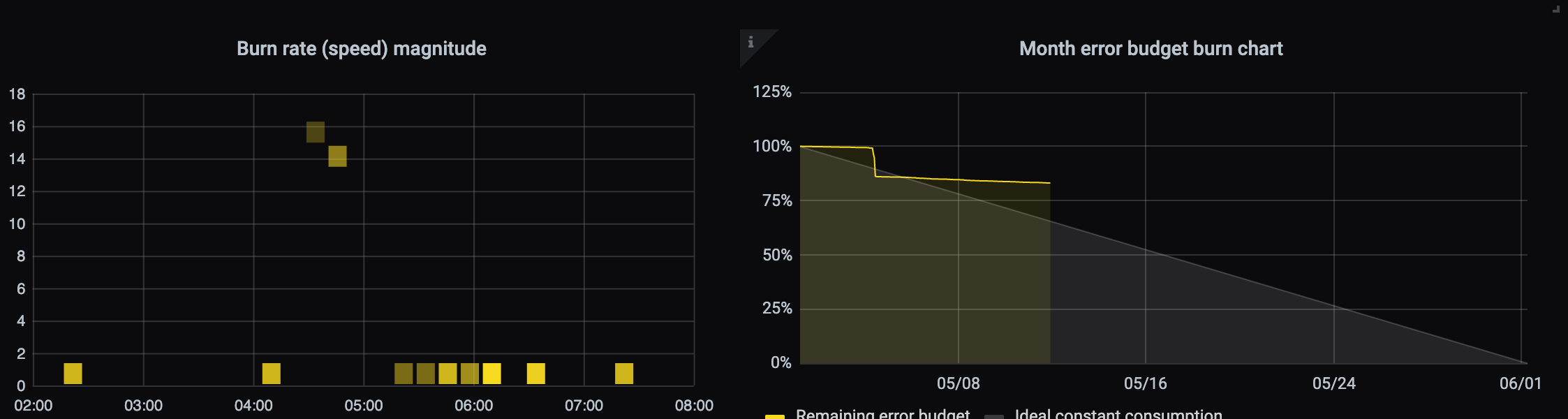


Рисунок 5.2.4 - Скорость сгорания бюджета на графиках

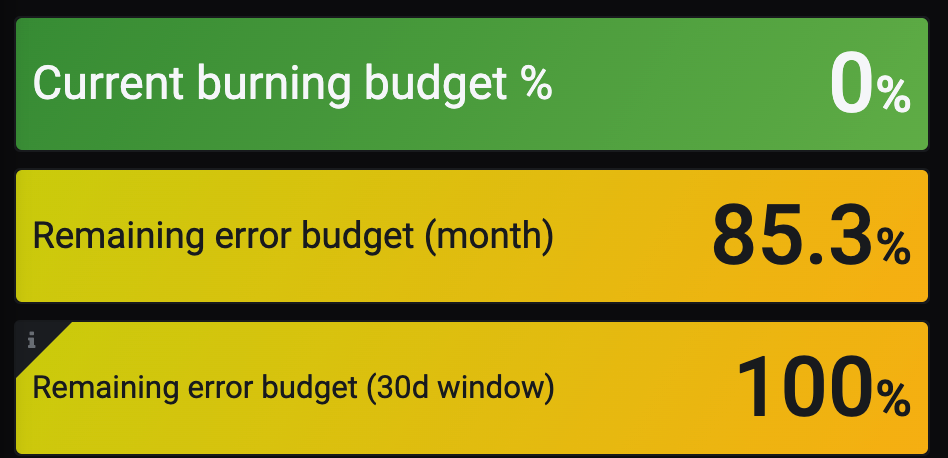


Рисунок 5.2.5 - Состояние бюджета ошибок на временных интервалах

Становится доступной в режиме реального времени сама метрика в сравнении с целевым показателем в процентах, который ей установлен.

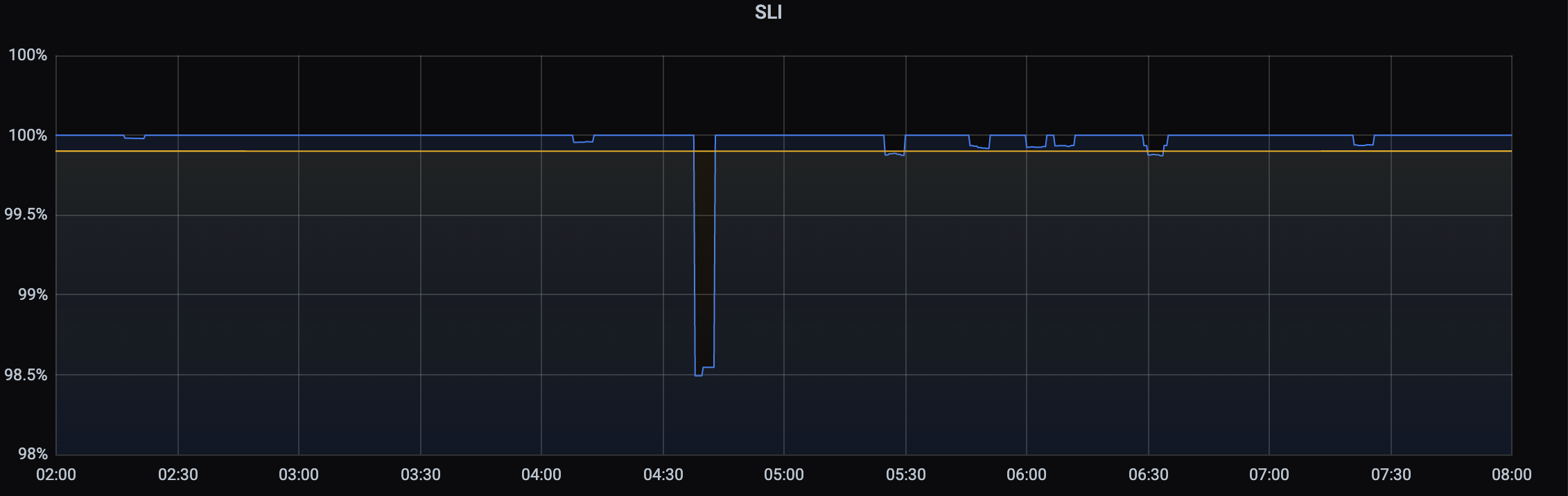


Рисунок 5.2.6 - График состояния значения метрики относительно целевого показателя в 99.9%

Появляется дополнительная информация о статусе оповещений обоих типов.



Рисунок 5.2.7 - Индикация статусов нотификаций типа page и ticket

**ЗАКЛЮЧЕНИЕ**

В процессе выполнения работы были рассмотрены основные понятия из области мониторинга технических систем и поставлена проблема необходимости контроля доступности таких систем в круглосуточном режиме.

Для решения этой проблемы было предложено реализовать один из подходов контроля за показателями, предложенный инженерами компании «Google». Попутно была описана и реализована вся необходимая инфраструктура, с аспектами из области системного дизайна, чтобы система мониторинга корректно функционировала.

В результате мы получили готовую систему, способную отражать в режиме реального времени состояние любого приложения по конкретным индикаторам состояния, зависящим от специфики последнего. А использование выбранной методологии упрощает контроль за выполнением контрактов доступности и корректной работы в целом.

**СПИСОК ИСТОЧНИКОВ И ЛИТЕРАТУРЫ**

**Ссылки на источники**

1. Общедоступная многоязычная универсальная интернет-энциклопедия со свободным контентом. URL: <https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%BD%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3> (дата обращения: 02.06.2022).
2. Google cloud documentation. Cloud Architecture Center. URL: <https://cloud.google.com/architecture/devops/devops-measurement-monitoring-and-observability> (дата обращения: 02.06.2022)
3. Prometheus documentation. URL: <https://prometheus.io/docs/concepts/metric_types/> (дата обращения: 02.06.2022)
4. Общедоступная многоязычная универсальная интернет-энциклопедия со свободным контентом. URL: <https://en.wikipedia.org/wiki/Site_reliability_engineering> (дата обращения: 05.06.2022)

**Список литературы**

1. Betsy Beyer, Chris Jones, Jennifer Petoff and Niall Richard Murphy Site Reliability Engineering: How Google Runs Production Systems
2. Betsy Beyer, Niall Richard Murphy, David K. Rensin, Kent Kawahara and Stephen Thorne The Site Reliability Workbook: Practical Ways to Implement SRE
3. Википедия - общедоступная многоязычная универсальная интернет-энциклопедия со свободным контентом, реализованная на принципах вики.
4. Документация Prometheus - an open-source systems monitoring and alerting toolkit.
5. Документация VictoriaMetrics - is a fast, cost-effective and scalable monitoring solution and time series database.
6. Medium - an open platform where readers find dynamic thinking, and where expert and undiscovered voices can share their writing on any topic
7. Хабр - русскоязычный веб-сайт в формате системы тематических коллективных блогов с элементами новостного сайта, созданный для публикации новостей, аналитических статей, мыслей, связанных с информационными технологиями, бизнесом и интернетом.

1. 1 Мера, позволяющая получить численное значение некоторого свойства программного обеспечения или его спецификаций. [↑](#footnote-ref-1)
2. Файл журнала (протокол, журнал; англ. log) — файл с записями о событиях в хронологическом порядке, простейшее средство обеспечения журналирования. Различают регистрацию внешних событий и протоколирование работы самой программы — источника записей (хотя часто всё записывается в единый файл). [↑](#footnote-ref-2)
3. Значение, которое заданная случайная величина не превышает с фиксированной вероятностью. Если вероятность задана в процентах, то квантиль называется процентилем или перцентилем. [↑](#footnote-ref-3)
4. Это набор инструментов для мониторинга и оповещения систем с открытым исходным кодом, изначально созданный в SoundCloud. С момента его создания в 2012 году многие компании и организации приняли Prometheus, и проект имеет очень активное сообщество разработчиков и пользователей. Теперь это отдельный проект с открытым исходным кодом, который поддерживается независимо от какой-либо компании. [↑](#footnote-ref-4)
5. Под сайтом в данном контексте подразумевается платформа, как предоставляемый сервис. [↑](#footnote-ref-5)
6. Проектирование надежности сайта — это набор принципов и практик, которые включают аспекты разработки программного обеспечения и применяют их к инфраструктуре и операционным проблемам. Основные цели - создание масштабируемых и высоконадежных программных систем. [↑](#footnote-ref-6)
7. Термин методологии ITIL, обозначающий формальный договор между заказчиком (в рекомендациях ITIL заказчик и потребитель — разные понятия) услуги и её поставщиком, содержащий описание услуги, права и обязанности сторон и, самое главное, согласованный уровень качества предоставления данной услуги. [↑](#footnote-ref-7)
8. Скрепер (англ. scraper) – сборщик, в данном контексте подразумевается сборка метрик. [↑](#footnote-ref-8)
9. HTTP (англ. HyperText Transfer Protocol — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных, изначально — в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных. [↑](#footnote-ref-9)
10. Используется для запроса содержимого указанного ресурса. [↑](#footnote-ref-10)
11. Многопото́чность (англ. Multithreading) — свойство платформы (например, операционной системы, виртуальной машины и т. д.) или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть без предписанного порядка во времени. При выполнении некоторых задач такое разделение может достичь более эффективного использования ресурсов вычислительной машины. [↑](#footnote-ref-11)
12. Переменные окружения — именованные переменные, содержащие текстовую информацию, которую могут использовать запускаемые программы. [↑](#footnote-ref-12)
13. Корутины — это облегчённые потоки. Облегчённый поток означает, что он не привязан к нативному потоку, поэтому он не требует переключения контекста на процессор, поэтому он быстрее. [↑](#footnote-ref-13)
14. Набор инструкций, который заставляет компьютер выполнить определённую задачу. [↑](#footnote-ref-14)
15. Репликация (англ. replication) — механизм синхронизации содержимого нескольких копий объекта (например, содержимого базы данных). Репликация — это процесс, под которым понимается копирование данных из одного источника на другой (или на множество других) и наоборот. [↑](#footnote-ref-15)
16. Технология pull (англ. pull technology, pull coding или client pull - сбор чего-либо ) — технология сетевой коммуникации, при которой первоначальный запрос данных производится клиентом, а ответ порождается сервером. [↑](#footnote-ref-16)
17. YAML (акроним англ. «Yet Another Markup Language» — «Ещё один язык разметки», позже — рекурсивный акроним англ. «YAML Ain't Markup Language» — «YAML — не язык разметки») — «дружественный» формат сериализации данных, концептуально близкий к языкам разметки, но ориентированный на удобство ввода-вывода типичных структур данных многих языков программирования. [↑](#footnote-ref-17)