Оглавление

Введение

База данных — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. База данных обычно управляется системой управления базами данных (СУБД). Данные вместе с СУБД, а также приложения, которые с ними связаны, называются системой баз данных, или, для краткости, просто базой данных.

Скорость и бесперебойная работа систем управления базами данных (СУБД) переходит в разряд критически-важных факторов для работы многих компаний.

При увеличении количества хранимой информации растут требования к производительности: если раньше СУБД обслуживали оффлайн-процессы, то сегодня должны соответствовать требованиям онлайн-сервисов. А значит, должны быстро обрабатывать множество запросов. Например, если раньше банкам хватало пропускной способности БД в несколько тысяч запросов в секунду, то с развитием онлайн-банкинга счет идет на сотни тысяч обращений.

Непрерывность бизнеса — еще один элемент цифровой реальности: простой может обернуться прямыми или косвенными убытками. Работоспособность сервисов зависит в том числе от бесперебойной работы СУБД.

Поэтому при увеличении количества данных и росте производительности необходимо иметь систему мониторинга для отслеживания состояния работы СУБД для своевременного принятия решений по устранению появляющихся неполадок

Цель данного курсового проекта – разработать систему мониторинга СУБД RedDatabase [1] для отслеживания состояния ее работы.

Для достижения этой цели были поставлены следующие задачи:

а) провести анализ системы мониторинга Prometheus;

b) создать экспортер метрик и настроек RedDatabase для передачи их системе мониторинга;

c) реализовать графический интерфейс для представления информации в виде счетчиков и графиков;

d) произвести тестирование разработанной системы.

Таким образом, структура данного курсового проекта состоит из трех глав и выглядит следующим образом:

a) в первой главе производится анализ технического задания, который включает в себя обзор технологий Prometheus [2] и Grafana [3] для реализации графического представления мониторинга системы;

b) во второй главе описывается процесс разработки экспортера и системы;

c) в третьей главе приводится процесс тестирования системы на нагруженной базе данных.

1. Анализ технического задания

Prometheus — бесплатное программное приложение, используемое для мониторинга и оповещения о событиях. Он записывает метрики в реальном времени в базу данных временных рядов (с учетом высокой размерности), построенную с использованием модели HTTP pull, с гибкими запросами и оповещениями в реальном времени. Проект написан на Go и распространяется под лицензией Apache 2 License, исходный код доступен на GitHub.

Доступность и понятность системы стало главной причиной выбора данной системы мониторинга для выполнения поставленных перед курсовым проектом задач.

Перед проектом были поставлены задачи собирать данные об нагруженной базе данных RedDatabase, а именно:

- Размер базы данных в МБ;
- Количество текущих подключений к базе данных;
- Утилита gstat -h
- - разница между OAT и OIT
- - разница между NT и OIT;
- Firebird.log (количество данных записей за текущие сутки)
- - Firebird consistency check
- - Error while trying to read from file
- - Error while trying to write to file.

Данные Prometheus хранятся в виде метрик, причем каждая метрика имеет имя, которое используется для обращения к ней и запроса к ней. Каждую метрику можно детализировать с помощью произвольного количества пар ключ-значение (меток). Метки могут включать информацию об источнике данных (с какого сервера поступают данные) и другую информацию о разбивке для конкретного приложения, такую как код состояния HTTP (для показателей, связанных с ответами HTTP), метод запроса (GET или POST), конечная точка и т.д., Возможность указать произвольный список меток и

выполнять запросы на их основе в реальном времени. Поэтому модель данных Prometheus называются многомерной.

Prometheus хранит данные локально на диске, что способствует быстрому хранению данных и быстрым запросам.

Prometheus собирает данные в виде временных рядов. Временные ряды строятся по модели вытягивания: сервер Prometheus запрашивает список источников данных (иногда называемых экспортерами) с определенной частотой опроса. Каждый из источников данных обслуживает текущие значения метрик для этого источника данных в конечной точке, запрашиваемой Prometheus. Затем сервер Prometheus агрегирует данные по источникам данных. Prometheus имеет ряд механизмов для автоматического обнаружения ресурсов, которые следует использовать в качестве источников данных.

Для реализации поставленных задач необходимо реализовать экспортер данных, которые будут отправляться на сервер, который, в свою очередь, будет «слушать» Prometheus и агрегировать данные для дальнейшего отображения.

Prometheus сам по себе не отображает данные, а, только лишь, является системой агрегирования. В нем есть механизм отображения графиков, но он будет не очень удобным если таких метрик будет много. Поэтому для реализации визуализации нужна другая система, которая будет брать агрегированные данные из Prometheus и отображать в более удобном виде.

Grafana — свободная программная система визуализации данных, ориентированная на данные систем ИТ-мониторинга. Реализована как веб-приложение в стиле «приборных панелей» с диаграммами, графиками, таблицами, предупреждениями.

Подключается к многообразным источникам данных, поддерживает расширение с помощью системы плагинов. Позволяет конечным пользователям стоить сложные панели мониторинга с помощью

интерактивных запросов. Разделена на фронтенд и бэкенд, написанные на TypeScript и Go соответственно. Популярный компонент в стеках мониторинга, часто используемый в сочетании с СУБД временных рядов, какой в свою очередь и является Prometheus.

Grafana содержит много встроенных элементов отображения данных, таких как: графики, счетчики, диаграммы, таблицы и др., что делает эту систему наиболее привлекательной для дальнейшей разработки системы мониторинга.

Таким образом, данный проект будет работать следующим образом:

1. Создается база данных и запускается нагрузка, которая будет симулировать работу с несколькими подключениями (или любой другая рабочая база, без нужды прописывать симулятор);
2. Реализуется экспортер, который в бесконечном цикле будет собирать метрики и отправлять их на сервер в виде, который будет понимать Prometheus;
3. Настраивается Prometheus на «слушание» данного сервера для агрегации данных;
4. Настраивается система визуализации Grafana для сбора и отображения данных из Prometheus.

2. Реализация экспортера и системы мониторинга

Для начала необходимо настроить систему Prometheus, чтобы сказать ей, откуда нужно брать метрики для дальнейшего агрегирования.

Это выполняется через файл prometheus.yml. Данный файл хранит все виды «работ» (job, экспортеры), которые собирает Prometheus. У каждой «работы» есть настройки временного интервала, сервера и дополнительных настроек агрегирования. Для rdb_exporter этот файл будет выглядеть так:

```
global:
  scrape_interval: 15s
  scrape_timeout: 10s
  evaluation_interval: 15s
scrape_configs:
- job_name: rdb_exporter
  honor_timestamps: true
  scrape_interval: 10s
  scrape_timeout: 10s
  metrics_path: /metrics
  scheme: http
  follow_redirects: true
  enable_http2: true
  static_configs:
  - targets:
    - localhost:9248
```

Теперь наша «работа» создана и Prometheus будет на metrics_path каждые scrape_interval отправлять данные, которые получил из targets по протоколу scheme.

Нагрузку базы данных будет симулировать tpcc.fdb база данных.

Prometheus теперь «слушает» наш порт, нужно отправить на него данные. Это будет реализовано языком Python и модулем Prometheus_client [4]. Код экспортера представлен ниже:

```
def readConfig(config_path='./rdb_exporter_config.cfg'):
    '''
    returns config file
    '''
    try:
        config = configparser.ConfigParser()
        config.read(config_path)
    except Exception:
        print(f'Check your {config_path} config file!')
        return None
    else:
        return config
```

Файл конфигурации реализован для быстрой смены окружения и отказа от концепции Hardcodding.

```
def gstat_scrape(gstat_path, database_path):
    with Popen(
        [gstat_path, '-h', database_path],
        stdout=PIPE,
        bufsize=1,
        universal_newlines=True
    ) as gstat:
        gstat_stdout = gstat.stdout.read().split('\n')
        OIT = gstat_stdout[11].split('\t')[-1]
        OAT = gstat_stdout[12].split('\t')[-1]
        NT  = gstat_stdout[14].split('\t')[-1]
        OIT, OAT, NT = list(map(int, [OIT, OAT, NT]))

        gauge_OAT_OIT_difference.set(OAT - OIT)
        gauge_NT_OIT_difference.set(NT - OIT)
```

Здесь реализована функция парсинга значений из вывода программной утилиты gstat-h. Данные собираются, преобразуются в нужный по заданию вид и устанавливаются в переменные gauge*, которые являются счетчиками в модуле Prometheus_client и отправляются на заданный порт.

```
def db_size_scrape(database_path):
    if os.path.exists(database_path):
        db_size_mb = os.path.getsize(database_path) * 1.0

        gauge_database_size.set(db_size_mb/1024/1024)

def attachments_scrape(hostname, database_path, user_name, password, fb_library_name):
    try:
        con = fdb.connect(
            host=hostname,
            database=database_path,
            user=user_name,
            password=password,
            charset='UTF8',
            fb_library_name=fb_library_name
        )
        cur = con.cursor()
        cur.execute('select count(*) from MON$ATTACHMENTS')

        gauge_database_attachments.set(cur.fetchall()[0][0])
    except:
        print('Database connection is not established')
```

Реализованы функции расчета размера файла базы данных и количестве подключений к базе с помощью модуля fdb [5] языка Python. Данные также присваиваются к gauge метрикам для отправки на сервер.

```
def find_date(line):
    if (m := re.search(

pattern=r'(Jan)*(Feb)*(Mar)*(Apr)*(May)*(Jun)*(Jul)*(Aug)*(Sep)*(Oct)*(Nov)*(Dec)*\s[0-
9]{1,2}\s[0-9]{2}:[0-9]{2}:[0-9]{2}\s[0-9]{4}',
```

```
            string=line
        )) is not None:
            return m[0]
        else:
            return None

    def log_errors_scrape(firebird_log_path, index):
        today = date.today()
        day = today.strftime('%d')
        mon = today.strftime('%B')[:3]
        year =today.strftime('%Y')
        file = open(firebird_log_path, 'r')
        lines = islice(file, index, None)
        check = False
        text = ''
        for line in lines:
            if check:
                text += line
            elif (s := find_date(line)) is not None:
                spl = s.split(' ')
                if day == spl[1] and mon == spl[0] and year == spl[-1]:
                    check = True
            index += 1
        Fcc = text.count('Firebird consistency check')
        Err_read = text.count('Error while trying to read from file')
        Err_write = text.count('Error while trying to write to file')
        gauge_Err_consistency.set(Fcc)
        gauge_Err_read.set(Err_read)
        gauge_Err_write.set(Err_write)
        return index
```

Данные две функции считывают из файла firebird.log данные об определенных ошибках за текущие сутки. Они также отправляются на порт.

```
def process_request(config) -> None:
    '''
    1.Run gstat in a loop and update stats per line
    2.Check size of database file
    3.Get amount of attachments to database
    4.Firebird log error scraping (0 in 0:00:00 every day)
    '''
    index = 0
    up.set(1)
    gstat_scrape(config['PATHS']['gstat'], config['PATHS']['database'])
    db_size_scrape(config['PATHS']['database'])
    attachments_scrape(
        config['DATABASE']['host'],
        config['PATHS']['database'],
        config['DATABASE']['username'],
        config['DATABASE']['password'],
        config['PATHS']['fb_library_name']
    )
    index = log_errors_scrape(config['PATHS']['firebird_log'], index)
```

Это и есть функция, которая бесконечно вызывается, собирая метрики по всем нужным параметрам системы.

```
up = Gauge(
    'up',
    'The value of this Gauge is always 1 when the exporter is up'
)

gauge_OAT_OIT_difference = Gauge(
    'gstat_OAT_OIT_difference',
    'The difference between oldest active transaction and oldest transaction',
)
```

```
gauge_NT_OIT_difference = Gauge(
    'gstat_NT_OIT_difference',
    'The difference between next transaction and oldest transaction',
)
gauge_database_size = Gauge(
    'database_size',
    'The database size',
)
gauge_database_attachments = Gauge(
    'database_attachments',
    'The count of attachments to the database',
)
gauge_Err_consistency = Gauge(
    'Firebird_Err_consistency',
    'The amount of consistency error in firebird log file today',
)
gauge_Err_read = Gauge(
    'Firebird_Err_read',
    'The amount of reading error in firebird log file today',
)
gauge_Err_write = Gauge(
    'Firebird_Err_write',
    'The amount of writing error in firebird log file today',
)
```

Здесь приведено описание всех счетчиков, которые будут передаваться на порт, который будет «слушать» Prometheus.

Данные отправлены на порт 9248 и выглядят следующим образом (рис.1).

```
up 1.0
# HELP gstat_OAT_OIT_difference The difference between oldest active transaction and oldest transaction
# TYPE gstat_OAT_OIT_difference gauge
gstat_OAT_OIT_difference 10438.0
# HELP gstat_NT_OIT_difference The difference between next transaction and oldest transaction
# TYPE gstat_NT_OIT_difference gauge
gstat_NT_OIT_difference 10526.0
# HELP database_size The database size
# TYPE database_size gauge
database_size 686.109375
# HELP database_attachments The count of attachments to the database
# TYPE database_attachments gauge
database_attachments 32.0
# HELP Firebird_Err_consistency The amount of consistency error in firebird log file today
# TYPE Firebird_Err_consistency gauge
Firebird_Err_consistency 0.0
# HELP Firebird_Err_read The amount of reading error in firebird log file today
# TYPE Firebird_Err_read gauge
Firebird_Err_read 0.0
# HELP Firebird_Err_write The amount of writing error in firebird log file today
# TYPE Firebird_Err_write gauge
Firebird_Err_write 0.0
```

Рис. 1 – Собранные метрики

Теперь необходимо настроить систему Grafana для отображения этих метрик в виде графиков и счетчиков.

В системе Grafana есть удобный интерфейс для реализации всевозможных элементов отображения (вплоть до пользовательских).

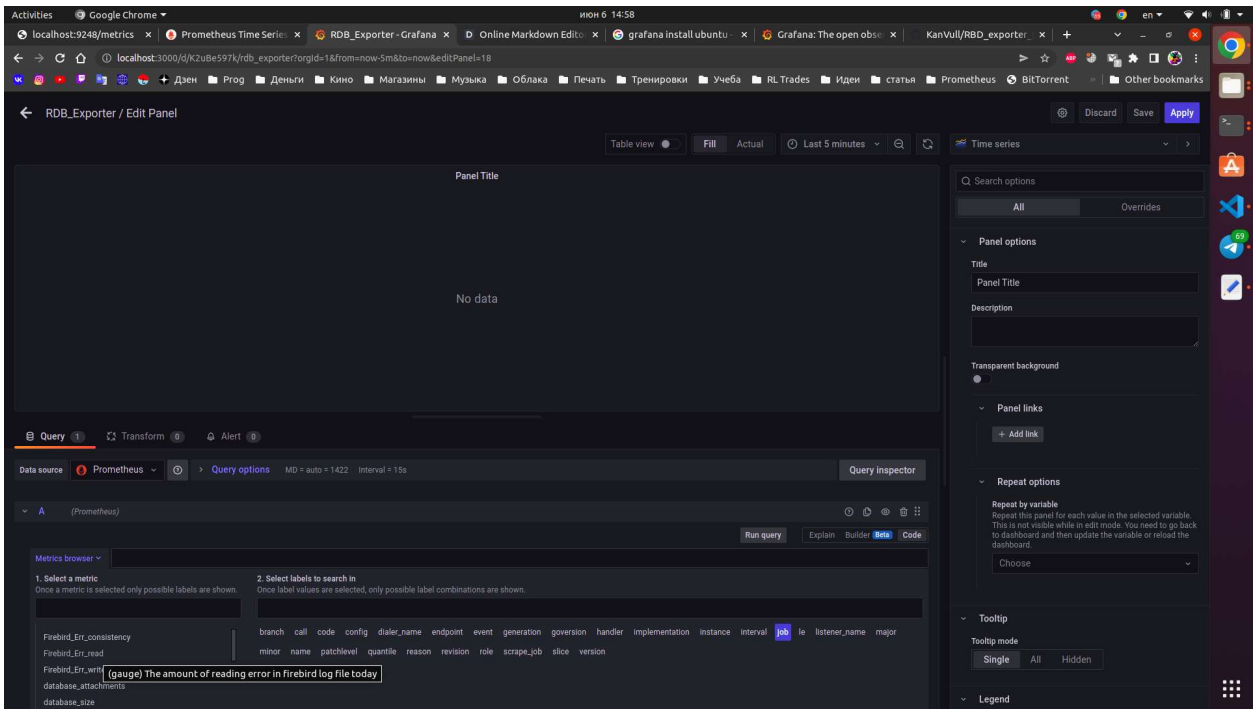

Рис. 2 – Добавление элементов в Grafana

Справа сверху можно выбрать тип элемента отображения. Справа будут его свойства (такие как текст, цвет текста, шрифт, свойства графика, параметры отображения значений, которые превышают некоторые значения).

Снизу можно выбрать источник данных. Выбор представлен в виде списка переменных, которые слушает Prometheus, которые мы в свою очередь отправляем экспортером в бесконечном цикле. Если на данном порту не будет данных, то Grafana продолжит работу со статусом NO DATA.

3. Тестирование системы мониторинга

Тестирование будет происходить на базе данных tpcc.fdb. Было запущено нагрузочной тестирование, которое будет работать некоторое время. Этого времени хватит, чтобы собрать и отобразить все необходимые метрики.

Конфигурационный файл, который был создан для экспортера выглядит следующим образом:

```
[EXPORTER]
port = 9248

[PATHS]
gstat = /opt/RedDatabase/bin/gstat
database = /var/rdb/tpcc.fdb
fb_library_name = /opt/RedDatabase/lib/libfbclient.so
firebird_log = /opt/RedDatabase/firebird.log

[DATABASE]
host = localhost
username = SYSDBA
password = masterkey
```

Здесь указана вся необходимая информация для экспортера, а именно: порт, на который должна быть реализована отправка метрик, пути до базы и программных утилит и файлов, а также информация о доступе к базе данных.

Окно системы Grafana было отрисовано для каждой метрики и отображает графики и счетчики для данных (рис.3).

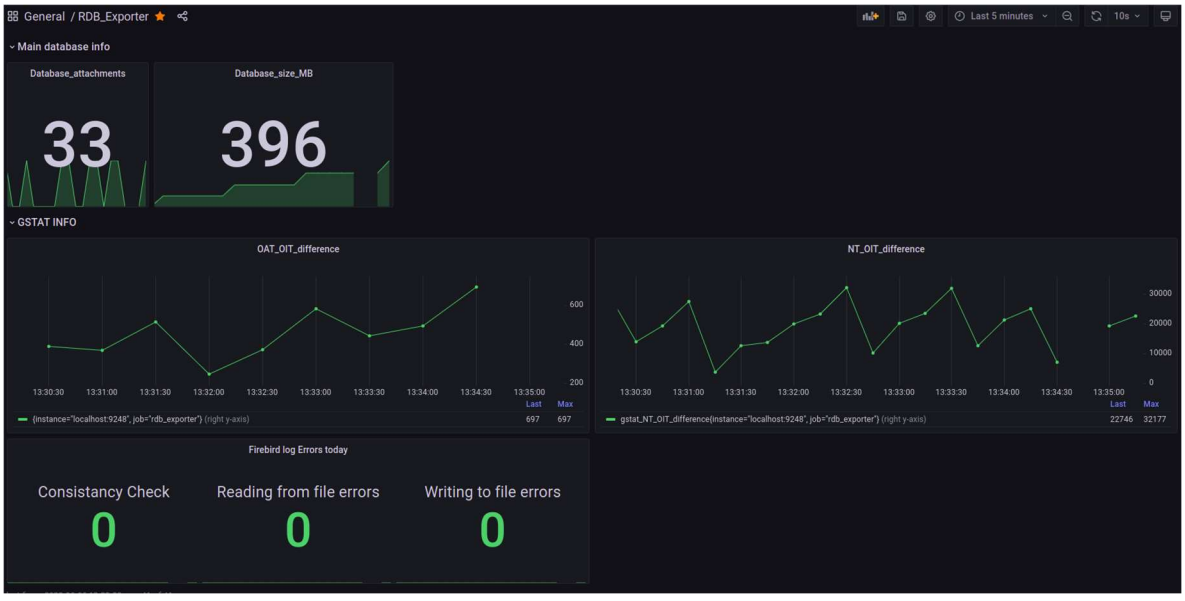

Рис. 3 – Отображение данных в Grafana

Как можно видеть на рисунке 3, собираются данные по всем параметрам, которые выдает экспортер и отображаются в «человеческом» графическом виде, для более удобного мониторинга. Сверху можно выбрать параметры отображения:

- Временной интервал данных;
- Частоту обновления;
- Добавить собственные элементы.

Все данные будут собираться постоянно и отображаться на графиках. Когда какие-либо значения будут заходить за некоторые конкретные значения (например количество ошибок firebird.log станет равно 1, 10, 20) они будут менять свой цвет. Также можно реализовать не только смену цвета, но и отправку письма на почту (например) с сообщением, содержащим время этого события и само событие.

Заключение

В данной пояснительной записке приведен процесс разработки системы мониторинга базы данных RedDatabase. Реализованы поставленные задачи. Код и вся информация по установке доступна для скачивания и установке на свое устройство по ссылке [6].

В первой главе приведена ознакомительная и объясняющая системы информация. Объяснен выбор технологий и их взаимосвязь.

Во второй главе приведен процесс разработки экспортера, который включает в себя код и настройку графического представления в Grafana.

В третьей главе было проведено тестирование работы системы мониторинга на базе данных tpcc.fdb и скрипте нагрузочного тестирования.

Таким образом, можно сделать вывод о том, что цель работы выполнена, все поставленные задачи реализованы в полном виде.

Список использованных источников

1. https://reddatabase.ru/ [Электронный ресурс]
2. https://prometheus.io/ [Электронный ресурс]
3. https://grafana.com/ [Электронный ресурс]
4. https://github.com/prometheus/client_python [Электронный ресурс]
5. https://firebirdsql.org/file/documentation/drivers_documentation/python/fdb/getting-started.html [Электронный ресурс]
6. https://github.com/KanVull/RDB_exporter_prometheus [Электронный ресурс]