МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский технологический университет»
МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения
(ИППО)

КУРСОВОЙ ПРОЕКТ

по дисциплине: Проектирование информационных систем
по профилю: Информационные системы и технологии
направления профессиональной подготовки: прикладной бакалавриат

Тема**:** Информационная система мониторинга кластера на базе Docker

Студент: Карих Дмитрий Степанович
Группа: ИСБОп-01-14
Работа представлена к защите _____ (дата) ____ / _____ /

Руководитель проекта: доцент Лобанов Александр Анатольевич
Работа допущена к защите _____ (дата) ____ / _____ /

Оценка по итогам защиты: _____
_____ / _____ /
_____ / _____ /
(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

М. МИРЭА. 2017г.

УДК 681.3.06

Карих Д.С. Информационная система мониторинга кластера на базе Docker / Курсовой проект по дисциплине «Проектирование информационных систем» направления профессиональной подготовки «Информационные системы и технологии (прикладной бакалавриат)» (7-й семестр) / руководитель доцент А.А. Лобанов / кафедра ИППО Института ИТ МИРЭА — с. 34, табл. 1, илл. 16, ист. 17 (в т.ч. 15 на англ. яз.).

Аннотация

Целью данного курсового проекта является проектирование и разработка информационной системы мониторинга кластера.

Результатом выполнения данного курсового проекта является набор Compose файлов и перечень конфигураций виджетов панели мониторинга.

Реферат

Целью курсового проекта является создание информационной системы мониторинга кластера серверов, построенного на базе Docker и Rancher.

В разделе «Введение» подробно формируется цель курсового проекта, а также рассматривается актуальность создаваемой системы.

В разделе «Анализ предметной области» осуществляется выбор целевого семейства операционных систем, производится сравнительный анализ существующих решений и формируется список задач и требований к создаваемой системе.

В разделе «Проектирование информационной системы» рассматривается и дорабатывается общая структура создаваемой системы, а также сравниваются и выбираются её составные части.

В разделе «Разработка информационной системы» создаются файлы Compose, необходимые для развёртывания системы на кластере, а также настраиваются алгоритмы мониторинга и аналитики.

В разделе «Заключение» дана оценка достигнутых результатов в разработке и проектировании системы.

Приложение 1 содержит примеры запросов к интерфейсу Graphite, формирующие выборки по нужным метрикам, а также критерии срабатывания тревоги по каждому показателю.

Список определений и сокращений

1. **Стек (stack)** — набор *сервисов*, связанных между собой какой-то определённой функцией;

2. **Сервис (service)** — описание окружения какого-либо программного продукта, запускаемого внутри контейнера. В сервис входят базовый образ и все параметры *контейнера*, включая метки для планировщика задач кластера;

3. **Контейнер (container)** — изолированная среда, содержащая в себе всё необходимое для запуска процесса окружение: библиотеки, системные исполняемые файлы, исполняемые файлы самого процесса и т.д.;

4. **Том (volume)** — область диска, используемая для хранения данных и переноса их между *контейнерами*;

5. **Кластер** — группа *хостов*, находящаяся под централизованным управлением *менеджера*;

6. **Хост** — физический сервер или виртуальная машина, входящие в состав *кластера*;

7. **Менеджер кластера** — программное обеспечение, распределяющее *контейнеры* между *хостами;*

8. **Docker** — технология, определяющая методику создания и управления *контейнерами*;

9. **Rancher** — *менеджер кластера*, используемый в рамках данного проекта.

Содержание

Введение **6**

1 Анализ предметной области **7**

1.1 Выбор семейства операционных систем 7

1.2 Сравнительный анализ существующих решений 7

1.3 Перечень собираемых метрик 9

1.4 Краткие итоги. Постановка задачи 10

2 Проектирование информационной системы **11**

2.1 Сборщики данных (агенты) 12

2.2 Хранилище данных 13

2.3 Обработчик данных 15

2.4 Функциональная схема системы 16

2.5 Структурная схема системы 19

3 Разработка информационной системы **20**

3.1 Файлы Compose 20

3.1.1 База данных InfuxDB 21

3.1.2 Прокси-сервер InfluxGraph 22

3.1.3 Агенты Netdata 23

3.1.4 Обработчик данных Grafana 24

3.1.5 Развёртывание системы на активном кластере 25

3.2 Настройка мониторинга в Grafana 26

3.2.1 Подключение источника данных 27

3.2.2 Создание и настройка панели индикаторов 27

3.2.3 Настройка каналов уведомлений 28

Заключение **29**

Приложение 1 **32**

Введение

В крупных информационных системах можно найти множество показателей, анализ которых помогает вовремя обнаружить и даже предотвратить возникновение неисправностей. Однако задача мониторинга часто воспринимается как очень сложная и затратная, из-за чего её используют только для наблюдения за особо критичными системами.

Цель данного проекта – показать, что базовый мониторинг доступен для всех, вне зависимости от размера системы. Не важно, состоит система из одного сервера или из сотен – набор бесплатных и свободных компонентов обеспечит наблюдение за тысячами показателей в реальном времени, а также предоставит доступ к истории наблюдений за любой необходимый временной промежуток.

В настоящий момент этот проект особенно актуален, так как происходит бурный рост популярности технологий контейнеризации программных процессов и построения кластеров на их базе. Информационные системы перестают зависеть от конкретного оборудования и получают возможность свободно мигрировать между физическими серверами. Такая свобода позволяет делать более устойчивые к внешним неполадкам системы, однако также усложняет поиск и устранение внутренних неисправностей.

В рамках данного проекта будет создана и рассмотрена свободно масштабируемая система мониторинга, предназначенная для работы на базе кластера Docker[1, 2] (см. рис. 1), находящегося под управлением системы Rancher.

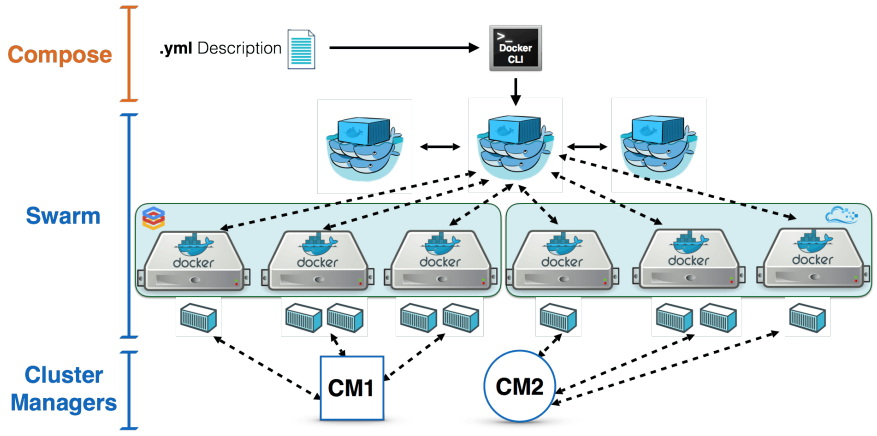

Рис. 1: *Swarm – одна из разновидностей кластера Docker*

1 Анализ предметной области

Главная особенность современных персональных компьютеров и серверов заключается в полной свободе выбора программного обеспечения от операционной системы до пользовательских программ. Однако такая свобода может стать проблемой при создании универсальной системы мониторинга.

В этом разделе будет аргументированно выбрано семейство операционных систем, которое станет основным объектом мониторинга. Также будет произведён анализ существующих решений и составлен список требований к разрабатываемой системе.

1.1 Выбор семейства операционных систем

Согласно исследованию W3Cook, проведённому в мае 2015 года, доля операционных систем семейства Unix и Unix-like на веб-серверах составляет 98.3% из 1 миллиона самых популярных серверов Интернета. При этом операционная система GNU/Linux (включая различные дистрибутивы) управляет подавляющим большинством веб-серверов — 96.6%[3].

Операционные системы из семейства GNU/Linux также являются родными для экосистемы Docker, на базе которого и работает кластер, использованный в рамках данного проекта. На ОС из других семейств (Windows и macOS) Docker запускается только внутри виртуальных машин.

Таким образом разрабатываемая система мониторинга будет применима, в основном, только для серверов на базе ОС GNU/Linux. Она сможет работать на остальных операционных системах, однако не будет настолько эффективно отслеживать параметры оборудования.

1.2 Сравнительный анализ существующих решений

В экосистеме Docker уже существуют решения, предназначенные для мониторинга информационных систем. Для получения общего представления о поставленной задаче и требованиях к системе было произведено сравнение трёх популярных систем мониторинга с последующим выявлением недостатков каждого из них.

1. Prometheus — бесплатный и свободный программный продукт, собирающий и анализирующий информацию о системе и других программах. Не имеет полноценного графического интерфейса, однако может интегрироваться с другими решениями для визуализации[4].

   Недостатки:

   (a) Объект мониторинга должен уметь предоставлять информацию по запросу от Prometheus;

   (b) Требуется ручная настройка всех объектов мониторинга внутри Prometheus, т.е. автоматическое обнаружение систем в кластере затруднено.

2. Datadog — облачный сервис, принимающий и анализирующий любые метрики или логи. Является полностью самостоятельным средством мониторинга. Позволяет расширять функциональность с помощью открытых агентов, собирающих информацию о программных продуктах[5].

   Недостатки:

   (a) Платный облачный сервис с закрытым исходным кодом;

   (b) Система не может бть размещена на собственном оборудовании и/или в закрытой сети.

3. Netdata — бесплатный и свободный программный продукт, предназначенный для мониторинга показателей системы в реальном времени. Обладает собственным пользовательским интерфейсом и умеет рассылать уведомления о превышении показателей[6].

   Недостатки:

   (a) Не обладает собственной базой данных. В стандартной конфигурации данные хранятся только 24 часа;

   (b) Не является централизованным решением – запускается на каждом сервере в кластере и работает независимо от остальных.

Рассмотренные продукты являются крайне популярными в некоторых областях мониторинга. Однако в данном случае их использование может быть затруднено техническими, организационными или финансовыми ограничениями.

1.3 Перечень собираемых метрик

У всех Linux-систем есть базовый набор параметров, которые могут помочь в анализе поведения операционной системы и программного обеспечения. При планировании мониторинга необходимо учитывать, что любые из этих параметров могут пригодиться при оценке производительности системы.

- Параметры, характерные для любой вычислительной системы:
    1. Загрузка центрального процессора;
        (a) Ожидание ввода/вывода (iowait);
        (b) Системная нагрузка (sys);
        (c) Пользовательская нагрузка (user);
        (d) Простой системы (idle);
    2. Количество свободной оперативной памяти;
    3. Процент использования файла (или раздела) подкачки;
    4. Свободное пространство на дисковой подсистеме;
- Параметры, специфичные для Unix систем:
    1. Средняя загрузка (Load Average) — характеризует количество процессов, находящихся на очереди выполнения у центрального процессора или других устройств. В Linux представлена тремя параметрами: за 1, 5 и 15 минут соответственно[7];
    2. Сетевая активность — общее количество трафика, переданное через каждый сетевой интерфейс за единицу времени;
    3. Утилизация дисковой подсистемы — общее количество информации, записанное на или прочитанное из дисковой подсистемы;

- Параметры, относящиеся к отдельным приложениям:

    1. Количество неудавшихся входов в систему — показатель, позволяющий оценить угрозу от атак злоумышленников на сервер. Компонент fail2ban отслеживает множество возможных векторов атак, в том числе попытки входа по протоколам SSH, VNC, MySQL и т.д.;
    2. Статистика работы балансировщика нагрузки;

- Другие параметры (не всегда поддерживаются оборудованием):

    1. Температуры процессора, материнской платы, жёстких дисков и т.п.;
    2. Количество ошибок чтения/записи дисковой подсистемы и оперативной памяти.

Перечисленные выше метрики должны собираться со всех физических серверов для достижения наибольшего покрытия системы мониторинга. Большая часть этих параметров также применима и к виртуальным машинам, которые тоже могут использоваться в качестве подчинённых серверов.

Помимо этих параметров проектируемая система также должна обладать возможностью принимать и другие, специфичные для программных систем, метрики. Например, отслеживать посещаемость сайта.

1.4 Краткие итоги. Постановка задачи

Учитывая рассмотренные варианты, можно сформировать требования к системе, создаваемой в рамках этого проекта:

1. **Умеренная централизованность** — алгоритмы обработки данных и формулы для построения графиков должны настраиваться централизованно, а сам сбор метрик должен быть независимым для каждого компонента системы (нечто среднее между Prometheus и Netdata);

2. **Расширяемость** — у системы не должно быть ограничивающих факторов, которые не позволили бы подключить к ней новые проекты в

будущем. Этого можно достичь, используя общий протокол для сбора и обработки метрик;

3. **Обнаружение сервисов** — система должна автоматически находить и подключать все поддерживаемые объекты мониторинга, входящие в состав кластера. В данном случае объектами мониторинга могут быть как физические машины, на которых запускаются агенты сбора статистики, так и любые программы, обладающие возможностью сбора статистики;

4. **Эффективное хранение метрик** — несмотря на размеры системы и количество собираемой информации хранилище данных не должно бесконечно разрастаться. Это означает, что у системы должна быть возможность настройки необходимой длительности хранения данных и её автоматического удаления (или замещения).

2 Проектирование информационной системы

Из требований к системе можно сделать вывод, что она должна состоять из трёх логических частей: сборщиков данных, хранилища данных и модуля обработки и анализа (см. рис. 2).

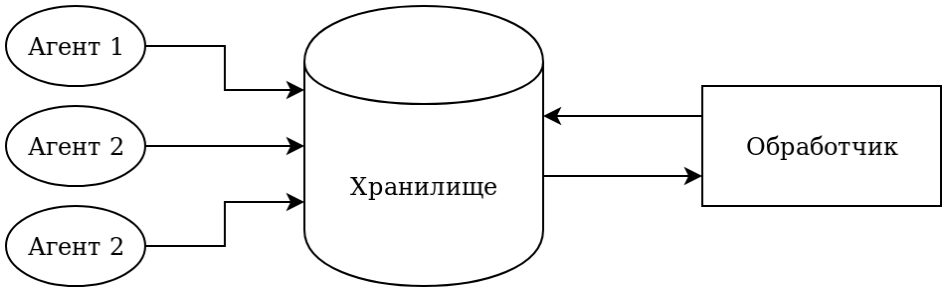

Рис. 2: *Предполагаемая схема системы*

Сборщики данных — упомянутые ранее агенты, которые запускаются на каждом объекте мониторинга, собирают всю необходимую информацию о системе и отправляют её в хранилище. Так как кластер построен на основе Docker, для распределения сборщиков логично использовать те же контейнеры. В таком случае можно расчитывать на то, что планировщик задач будет гарантировать размещение агента на каждом физическом сервере или виртуальной машине, входящих в состав кластера.

Хранилище данных — какой-либо программный продукт, способный хранить структурированные наборы данных. Для этих целей должны идеально подойти так называемые TSDB (time series databases, базы данных для временных рядов), так как метрики представляют собой ряды чисел с привязкой ко времени.

Модуль обработки и анализа данных — любой набор программного обеспечения, позволяющий вручную или автоматизированно проводить анализ данных, содержащихся в хранилище.

Для того, чтобы эти модули могли взаимодействовать между собой, они должны использовать один и тот же формат данных и протокол их передачи. Каких-либо особых требований к самому протоколу нет (кроме популярности и распространённости), поэтому в рамках этого проекта используется Graphite — очень популярный открытый протокол передачи метрик и их анализа.[8]

Более подробное представление связей между модулями можно увидеть на рис. 4 (декомпозиция блока «Система мониторинга» диаграммы IDEF0).

2.1 Сборщики данных (агенты)

Обычно агенты представляют из себя небольшие программы или набор скриптов, которые обращаются к различным ресурсам операционной системы, после чего отправляют эти данные посредством одного из нескольких возможных протоколов в любое хранилище.

Существует несколько открытых и свободных агентов, которые могут соответствовать требованиям к проекту:

1. collectd — демон, выполняющий все перечисленные выше задачи. Не имеет графического интерфейса, написан на C, открыт, бесплатен и работает на всех системах, поддерживающих стандарт POSIX[9];

   Недостатки:

   (a) Практически вся функциональность реализуется посредством расширений;

   (b) Для подключения каждого расширения необходимо вручную компилировать исполняемый файл демона;

2. StatsD — прокси-сервер для метрик, выполняющий роль посредника между приложениями и хранилищем данных. Является незаменимым для учёта большого количества быстро меняющихся показателей[10];

   Недостатки:

   (a) Не включает в себя инструменты для сбора информации о системах — каждое приложение должно самостоятельно генерировать и отправлять метрики;

3. Netdata (см. раздел 1.2, пункт 3) поддерживает не только мониторинг в реальном времени, но и экспорт собранных метрик в базу данных посредством протокола Graphite.

В ходе проектирования системы было решено использовать Netdata, так как эта программа не нуждается в настройке, поддерживает протокол Graphite, не является требовательной к ресурсам и допускает доступ к статистике в реальном времени при появлении такой необходимости.

В дальнейшем возможно будет также подключить один или несколько серверов StatsD, которые позволят собирать метрики не только из операционных систем, но и из приложений. Netdata и StatsD могут без проблем записывать различные данные в одно и то же хранилище.

Функциональная схема этого модуля в нотации IDEF0 представлена на рис. 5.

2.2 Хранилище данных

Учитывая возможное количество метрик (больше 1000 различных параметров с каждого агента) и ограниченную производительность дисковой подсистемы, необходимо очень внимательно выбирать программное обеспечение для хранения данных.

Протокол Graphite, выбранный в начале этой главы, поддерживается множеством различных баз данных и других хранилищ[8]. Рассмотрим несколько вариантов.

1. Carbon — подсистема Graphite, использующаяся для приёма и сохранения данных на жёсткий диск;

   Недостатки:

   (a) Невысокая гибкость хранилища: место на диске выделяется при первом обращении к метрике и в дальнейшем не может быть изменено без удаления данных;

   (b) Высокая нагрузка на дисковую подсистему;

   (c) Огромное количество файлов. Возможно даже исчерпание ресурсов файловой системы.

2. InfluxData InfluxDB — база данных для числовых последовательностей. Работает аналогично MySQL — использует собственный формат хранения данных и схожий язык запросов. Поддерживает импорт данных как через собственный язык запросов, так и по протоколу Graphite[11];

   Недостатки:

   (a) Несмотря на то, что импорт данных происходит через Graphite, доступ к данным возможен только с помощью собственного языка запросов, аналогичного SQL. Это ограничение может затруднить использование обработчиков данных, не поддерживающих протокол InfluxDB;

3. Elasticsearch — база данных, совмещённая с поисковой системой и аналитикой. Тоже совместима с источниками данных, использующими протокол Graphite[12].

   Недостатки:

   (a) Хранение числовых рядов не является основной задачей этой базы данных, следовательно она не оптимизирована конкретно под них;

(b) Данный продукт является слишком мощным для задач мониторинга — поддержка кластеризации полезна, но избыточна;

В ходе выбора хранилища для проектируемой системы удалось устранить единственный существенный недостаток InfluxDB с помощью прокси-сервера InfluxGraph, размещаемого между хранилищем и обработчиком данных. Этот компонент преобразует запросы Graphite в аналогичные запросы для InfluxDB, что позволяет использовать эту базу данных как альтернативу более тяжёлому Graphite, не требуя при этом доработки остальных компонентов.

Процессы, происходящие с данными в этом модуле, представлены в виде Data Flow Diagram на рис. 6.

2.3 Обработчик данных

Данная часть является ядром системы и выполняет самую важную задачу с точки зрения конечных пользователей. Основными функциями обработчика являются удобный и наглядный пользовательский интерфейс с интерактивными графиками, а также уведомления о неполадках через множество каналов передачи сообщений.

В составе одной системы может быть множество независимых обработчиков, поэтому конкретный выбор в данном случае не так критичен. При проектировании этой системы выбор был сделан в пользу Grafana — открытой платформы для аналитики и мониторинга[13].

Доступ к графическому интерфейсу Grafana осуществляется через любой веб-браузер. Графики также строятся при помощи веб-браузера и языка программирования JavaScript.

Grafana также поддерживает множество различных баз данных, в том числе Graphite, Elasticsearch и InfluxDB. В данном случае для доступа к базе InfluxDB используется протокол Graphite, так как он обладает более гибкими и удобными математическими и аггрегационными функциями.

Дополнительной, но не менее важной функцией Grafana являются тревоги и соответствующие им уведомления. Для любого настроенного графика можно настроить предельные значения, выход за которые будет инициировать тревогу. Уведомления об этих тревогах могут рассылаться по

электронной почте или через мессенджеры (например Telegram). Также в случае тревоги график подсвечивается в интерфейсе, чтобы привлечь внимание наблюдателя к проблеме.

Упрощённая функциональная схема модуля Grafana в нотации IDEF0 представлена на рис. 7 (блок «Обработать данные»). Рассмотрение второстепенных функций Grafana (плагины, встраиваемые виждеты и т.д.) не входит в задачу этого курсового проекта, поэтому они отсутствуют на схеме.

2.4 Функциональная схема системы

На иллюстрациях ниже представлена модель схемы, разработанная по методологиям IDEF0[16] и DFD[17] с учётом выбранного программного обеспечения. На рис. (3) показан общий вид системы.

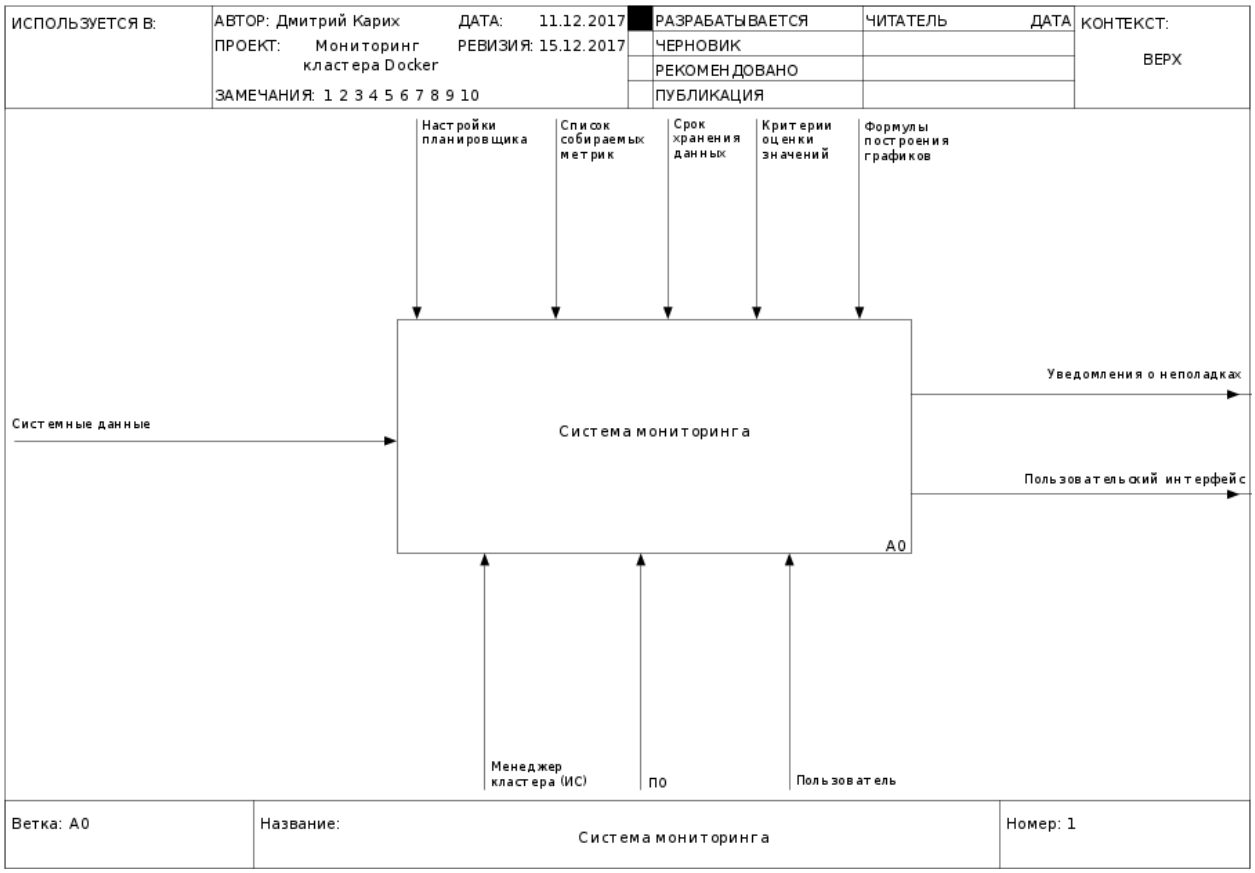

Рис. 3: *0-й уровень декомпозиции системы, «чёрный ящик» (IDEF0)*

На последующих иллюстрациях производится пошаговое разбиение системы до 2-го уровня декомпозиции.

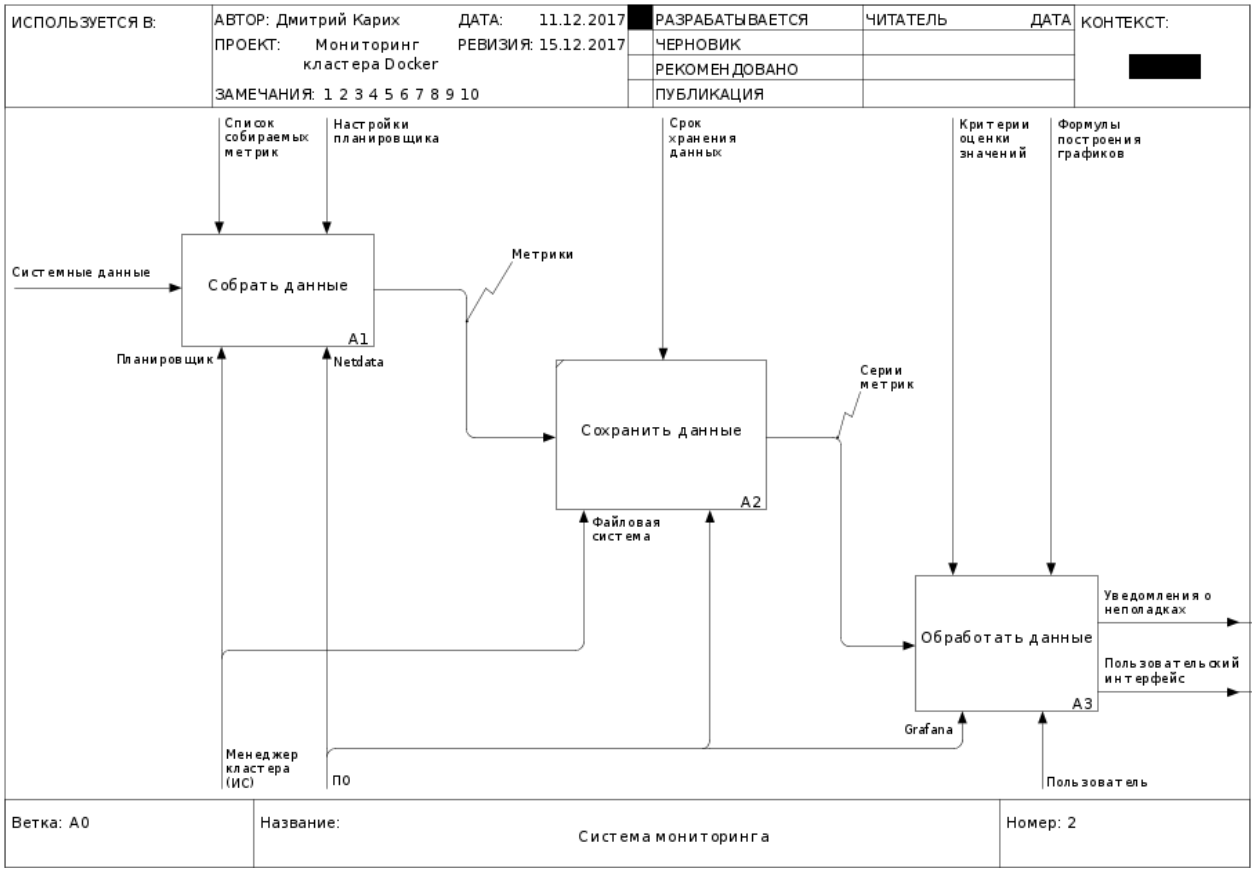

Рис. 4: *1-й уровень декомпозиции системы (IDEF0)*

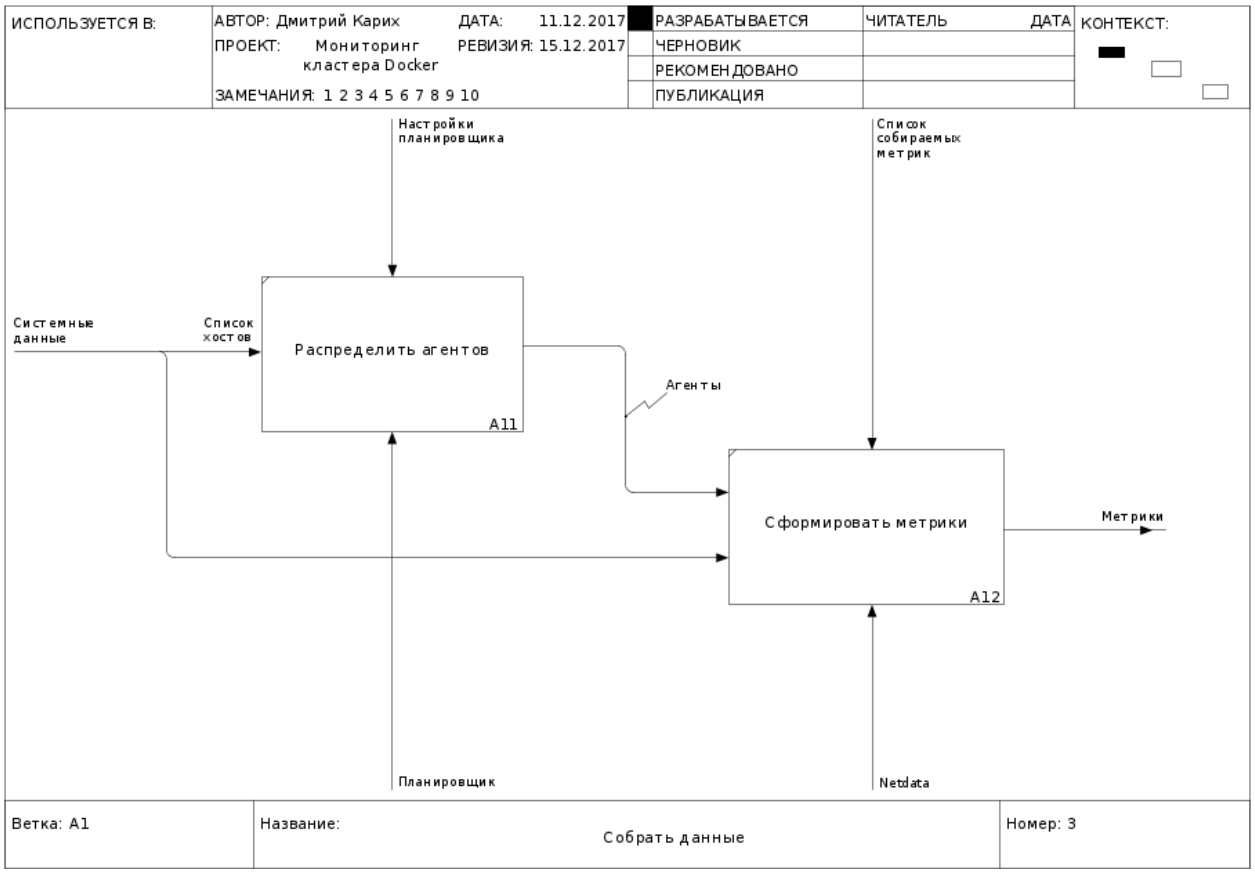

Рис. 5: *2-й уровень декомпозиции, блок «Собрать данные» (IDEF0)*

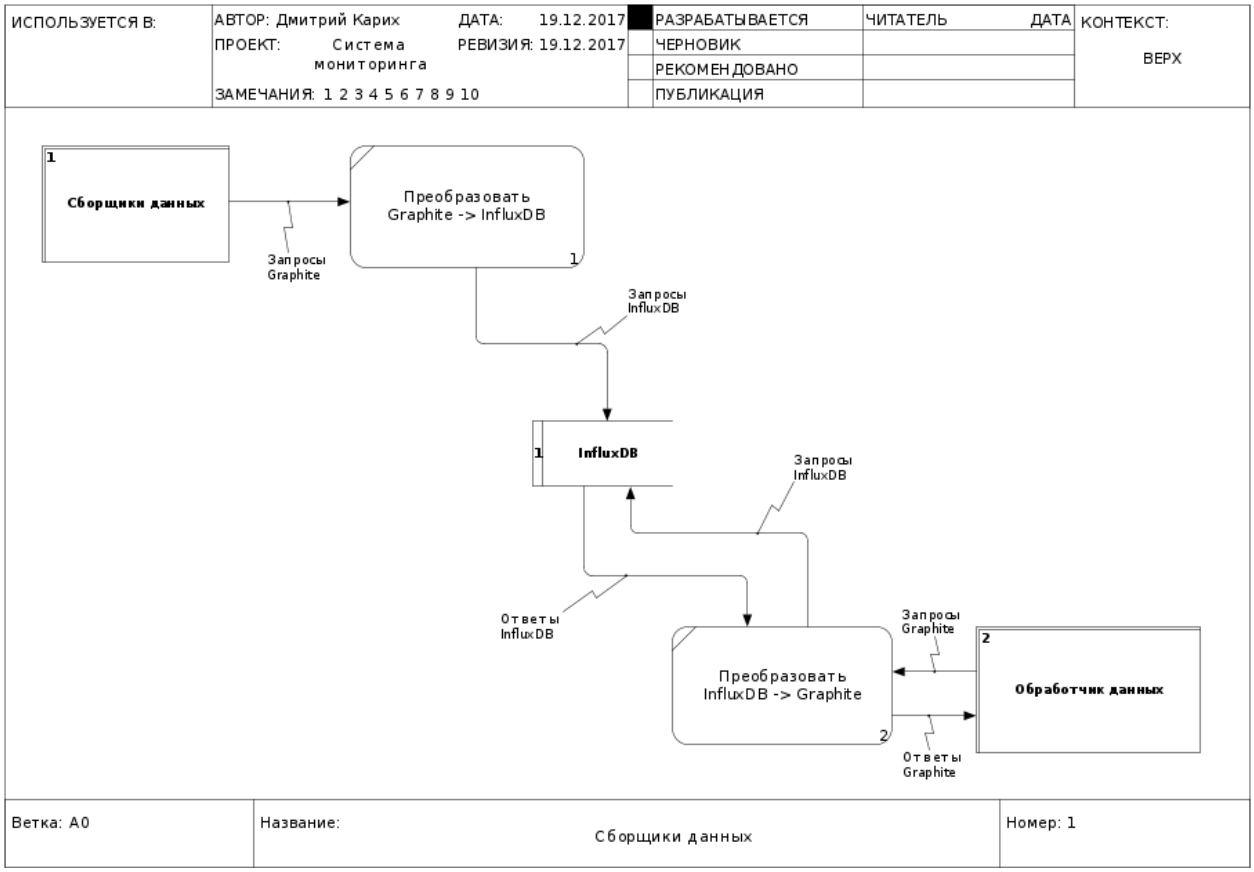

Рис. 6: *Описание процессов внутри блока «Сохранить данные» (DFD)*

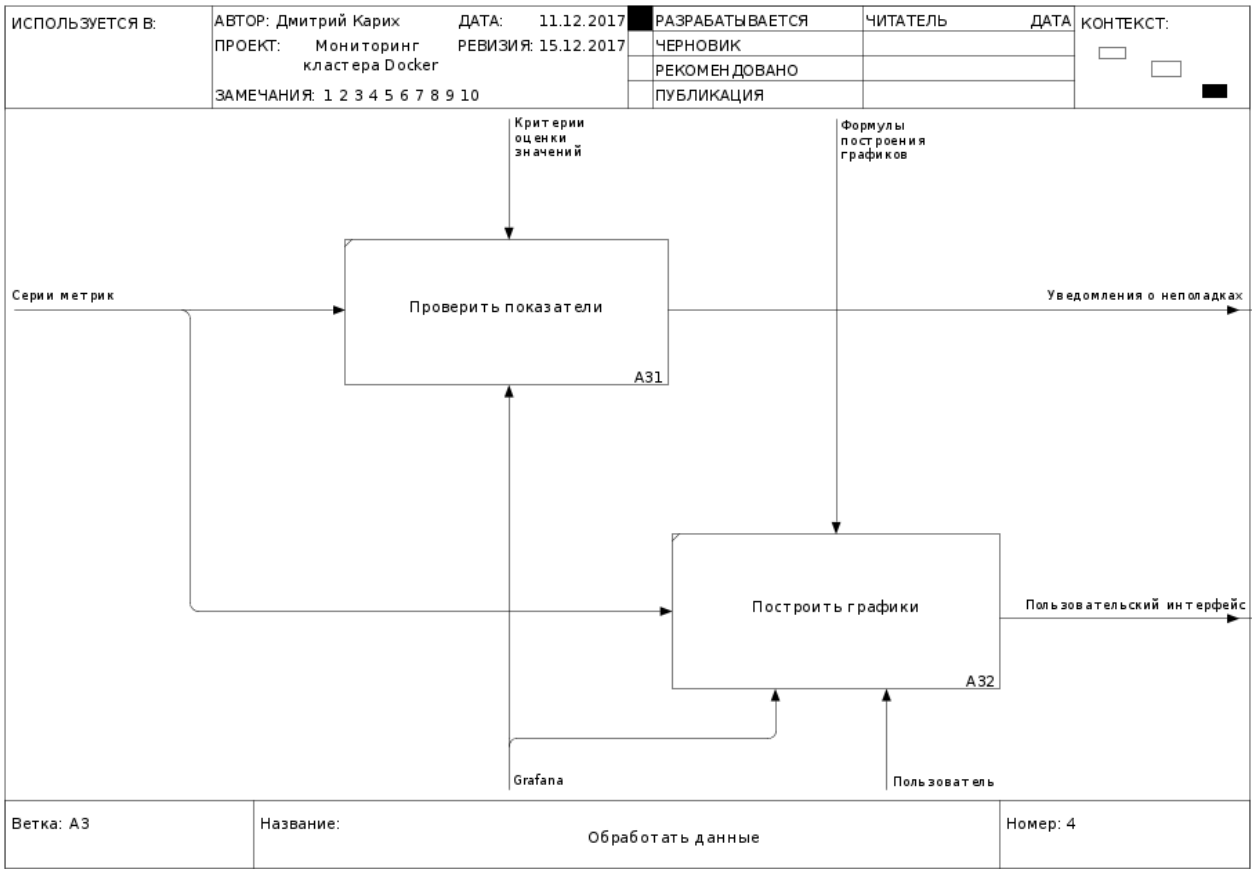

Рис. 7: *2-й уровень декомпозиции, блок «Обработать данные» (IDEF0)*

2.5 Структурная схема системы

С точки зрения надёжности, создаваемая информационная система довольно проста. От самих сборщиков данных работоспособность системы не зависит, так как отказ сборщика в итоге станет однозначным индикатором наличия проблем на стороне объекта мониторинга. Таким образом, у системы можно выделить два элемента надёжности, а также нужно учесть влияние инфраструктуры кластера на связность элементов. Полученная схема системы представлена на рис. 8.

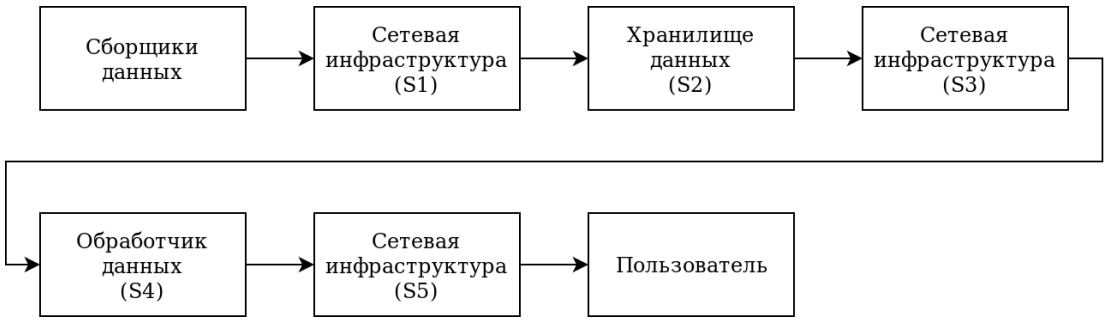

Рис. 8: Структурная схема системы

В схеме не учитываются возможные неполадки оборудования, так как обычно кластеры абстрагируются от таких неисправностей. Дисковая подсистема может быть децентрализованной и отказоустойчивой, что позволит без нарушения работоспособноти системы переносить её компоненты между физическими серверами.

Произведём расчёт надёжности информационной системы. Составим методом экспертных оценок матрицу надёжности элементов (см. табл. 1).

Таблица 1: *Матрица надёжности элементов*

	S_1	S_2	S_3	S_4	S_5
Экс. 1	0.995	0.999	0.995	0.998	0.995
Экс. 2	0.997	0.995	0.997	0.999	0.997
Экс. 3	0.994	0.998	0.994	0.998	0.994
Среднее	0.995	0.997	0.995	0.998	0.995

Исходя из линейной структуры системы можно расчитать надёжность по формуле (1).

$$P = \prod_{i=1}^{5} P_i \tag{1}$$

$$P = 0.995 \cdot 0.997 \cdot 0.995 \cdot 0.998 \cdot 0.995 \approx 98\% \quad (2)$$

По полученному результату можно заметить, что надёжность системы без учёта надёжности физического оборудования довольно высока. Такое значение достигается путём использования хорошо протестированного программного обеспечения в блоках хранения и обработки данных, а также независимостью общей работоспособности системы от состояния агентов.

3 Разработка информационной системы

Docker и Rancher предоставляют удобные инструменты для управления контейнерами и связями между ними. Инструмент Docker Compose позволяет документировать сервисы и развёртывать их на заранее созданный кластер Docker Swarm, либо на одну физическую машину с запущенным демоном Docker. Rancher Compose в свою очередь позволяет управлять стратегиями обновления каждого сервиса и другими специфичными для Rancher параметрами.

3.1 Файлы Compose

В первой части этого раздела будут созданы соответствующие файлы *docker-compose.yml* и *rancher-compose.yml*, подготовленные для развёртывания на любом кластере Rancher. Эти файлы используют синтаксис YAML[14], поэтому могут быть представлены в виде обычного текста.

Оба файла начинаются с такой структуры:

```
version: '2'
services:
  ...
```

Первая строка указывает версию формата Compose. В данном случае применяется формат версии 2, однако в некоторых проектах уже применяется формат версии 3.

Со второй строки начинается секция *services,* в которой описываются все контейнеры и связи между ними. В следующих 4-х подразделах будут созданы блоки кода, которые должны быть внесены в эту секцию.

3.1.1 База данных InfuxDB

В первую очередь необходимо выбрать базовый образ, на основе которого будет создан контейнер. В экосистеме Docker образы могут загружаться из множества источников, однако в рамках этого проекта будет использоваться только официальный сервис Docker Hub[15], на котором уже есть образ InfluxDB, созданный разработчиками этой базы данных. Базовый образ указывается как «*image: influxdb:версия*» и будет загружен автоматически. На момент написания этой работы последней версией InfluxDB была 1.4.2, однако была выбрана облегчённая версия 1.4.2-alpine, основанная на дистрибутиве Alpine Linux[1].

Так как контейнеры могут свободно удаляться и создаваться заново[18], важно предусмотреть отдельную область на системном диске, куда база данных сможет сохранить все метрики. В Docker это реализуется посредством томов (volumes). InfluxDB сохраняет все данные в директорию */var/lib/influxdb*, поэтому к ней необходимо подключить том. В Compose-файле это можно сделать с помощью строки «- *name:*/path» в блоке *volumes*, где *name* – название тома, */path* – путь внутри контейнера.

Наконец, нужно включить интерфейс Graphite в процессе InfluxDB. По умолчанию он выключен, так что нужно внести некоторые изменения в файл конфигурации */etc/influxdb/influxdb.conf*. Как уже было упомянуто ранее, контейнеры постоянно создаются из базового образа, поэтому простое изменение файла будет перезаписано. Вместо этого был использован небольшой скрипт, который вносит изменения автоматически при каждом запуске контейнера. Скрипт хранится в одной из переменных окружения (*environment*) и вызывается с помощью изменённой команды запуска контейнера (*entrypoint* и *command*).

Фрагмент файла *docker-compose.yml*

[1] Alpine Linux – небольшой дистрибутив Linux, созданный специально для использования в Docker-контейнерах.

```
influxdb:
  image: influxdb:1.4.2-alpine
  environment:
    EVAL: |-
      if ! grep -q graphite /etc/influxdb/influxdb.conf; then
        echo;
        echo [[graphite]];
        echo '  'enabled = true;
        echo '  'bind-address = \":2003\";
        echo '  'protocol = \"tcp\";
      fi
  volumes:
  - netdata-influxdb:/var/lib/influxdb
  entrypoint: /bin/sh
  command:
  - -c
  - eval "$$EVAL" >> /etc/influxdb/influxdb.conf; influxd
```

Фрагмент файла *rancher-compose.yml*

```
influxdb:
  scale: 1
  start_on_create: true
```

3.1.2 Прокси-сервер InfluxGraph

Теперь нужно подключить прокси-сервер InfluxGraph к InfluxDB. Данный сервис не нуждается в настройке – достаточно его запустить, чтобы он начал выполнять свою задачу.

Фрагмент файла *docker-compose.yml*

```
influxgraph:
  image: thedrhax/influxgraph
  volumes:
  - /dev/shm/influxgraph-index:/srv/graphite
  links:
  - influxdb:influxdb
```

Фрагмент файла *rancher-compose.yml*

```
influxgraph:
  scale: 1
  start_on_create: true
```

Здесь впервые используется параметр *links*, который указывает на то, что у этого контейнера должен быть свободный доступ к базе данных InfluxDB, настроенной ранее.

3.1.3 Агенты Netdata

Так как контейнеры являются изолированной средой, нужно отдельно разрешить все системные вызовы, которые требуются для нормальной работы Netdata. Также обязательно нужно предоставить доступ к системным разделам, содержащим актуальную информацию об оборудовании и процессах: */proc*, */sys* и */var/run/docker.sock*.

В приведённом фрагменте файла *docker-compose.yml* можно увидеть скрипт конфигурации, как и в случае с InfluxDB. Здесь это тоже необходимо, чтобы сообщить Netdata, куда и в каком формате отправлять собранные метрики. Главное отличие этого скрипта от прошлого заключается в том, что здесь скрипт отправляет запрос к Rancher, чтобы узнать имя сервера, на котором запустился контейнер. Это позволит системе отличать метрики с разных серверов друг от друга.

Фрагмент файла *docker-compose.yml*

```
netdata:
  image: titpetric/netdata:latest
  volumes:
  - /proc:/host/proc:ro
  - /sys:/host/sys:ro
  - /var/run/docker.sock:/var/run/docker.sock
  links:
  - influxdb:influxdb
  environment:
    NETDATA_IP: 127.0.0.1
    NETDATA_PORT: '19999'
    EVAL: |-
      echo [backend]; echo enabled = yes; echo type = graphite;
      echo destination = tcp:influxdb:2003; echo prefix = netdata;
```

```
      curl http://169.254.169.250/2016-07-29/self/host/name \
           > /tmp/name || exit 1;
      echo hostname = $$(cat /tmp/name);
  privileged: true
  cap_add:
  - SYS_PTRACE
  network_mode: host
  entrypoint: /bin/sh
  command:
  - -c
  - eval $$EVAL >> /etc/netdata/netdata.conf; /run.sh
  labels:
    io.rancher.scheduler.global: 'true'
    io.rancher.container.dns: 'true'
```

Фрагмент файла *rancher-compose.yml*

```
netdata:
  start_on_create: true
```

Параметры *privileged*, *cap_add* и *network_mode* снижают степень изоляции контейнера от операционной системы, чтобы можно было получить больше информации без вмешательства в саму систему. Параметр *labels* сообщает кластеру Rancher, что этот контейнер должен быть запущен на всех активных серверах.

3.1.4 Обработчик данных Grafana

Последний компонент разрабатываемой системы – пользовательский интерфейс, представленный программным обеспечением Grafana. Так как у Grafana есть графический интерфейс, файлы Compose содержат только основные настойки этой системы – URL и пароль администратора.

Фрагмент файла *docker-compose.yml*

```
grafana:
  image: grafana/grafana:4.5.2
  ports:
  - 3000:3000
  environment:
```

```
      GF_SERVER_ROOT_URL: http://grafana.example.org:3000
      GF_SECURITY_ADMIN_PASSWORD: password
    volumes:
    - grafana:/var/lib/grafana
    - /dev/shm/grafana-logs:/var/log/grafana
    - grafana-config:/etc/grafana
    links:
    - influxgraph:influxgraph
```

Фрагмент файла *rancher-compose.yml*

```
grafana:
  scale: 1
  start_on_create: true
```

Секция *ports* указывает демону Docker, что этот контейнер должен быть доступен из внешнего мира. Физический сервер, на который попадёт контейнер с Grafana, сможет принимать соединения через порт 3000, т.е. в пользовательский интерфейс можно зайти с помощью браузера, имея IP-адрес сервера и номер порта.

3.1.5 Развёртывание системы на активном кластере

При вводе содержимого файлов *docker-compose.yml* и *rancher-compose.yml* в интерфейс создания стека в Rancher, в кластере будут созданы все описанные сервисы, а контейнеры автоматически рапределятся по физическим серверам и виртуальным машинам.

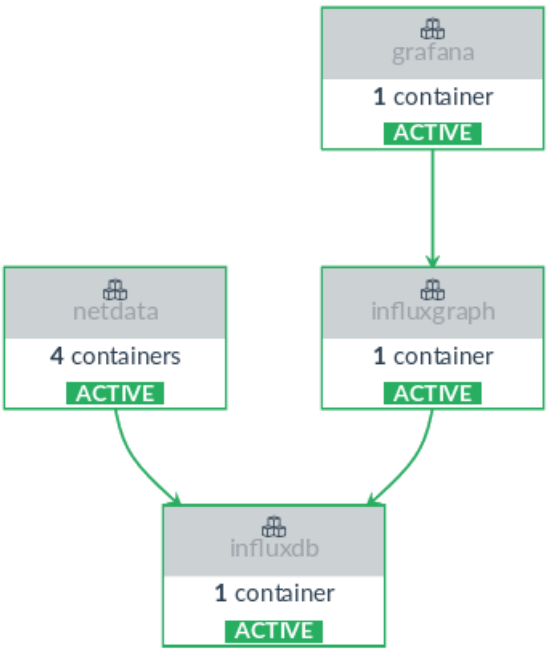

Рис. 9: *Схема стека, созданного по файлам Compose*

Если сравнить сгенерированную схему стека (рис. 9) и составленную ранее предполагаемую схему системы (рис. 2), то окажется, что они практически совпадают. Единственное отличие — направления стрелок. На схеме стека стрелки всегда идут от клиента к серверу и не отражают реальные направления потоков данных.

Также можно заметить, что Rancher создал 4 контейнера Netdata — по одному на каждый сервер. Это число совпадает с реальным количеством хостов в кластере, что и требовалось от системы.

3.2 Настройка мониторинга в Grafana

Единственный компонент системы, нуждающийся в ручной настройке после развёртывания — система аналитики и мониторинга Grafana. По умолчанию Grafana не умеет получать данные из хранилищ и не имеет формул и алгоритмов построения графиков. В данном подразделе будет произведена настройка Grafana от определения параметров мониторинга до настройки каналов уведомлений.

3.2.1 Подключение источника данных

Для начала необходимо научить Grafana загружать метрики из хранилища данных. Для этого в разделе «*Data Sources*» необходимо добавить новый источник данных. Для InfluxGraph эти параметры должны быть определены следующим образом:

- Name (имя): influxgraph
- Type (тип): Graphite
- Url (адрес): http://influxgraph
- Access (метод доступа): proxy
- Version (версия протокола Graphite): 1.0.x

После нажатия на кнопку «*Save & Test*» Grafana проверит возможность подключения и сообщит «*Data source is working*». Это означает, что можно приступать к следующему этапу настройки.

3.2.2 Создание и настройка панели индикаторов

Основным компонентом Grafana является панель индикаторов. Каждая панель разделяется на строки (rows), в каждой из которых, в свою очередь, можно разместить несколько виджетов. Самым популярным виджетом является простой график, но также можно использовать различные диаграммы и даже графы. Эта функциональность расширяется при помощи дополнений.

При настройке каждого виджета необходимо указать несколько параметров: название метрики, аггрегационные функции и временной промежуток. Также можно указать критерии срабатывания тревоги. Примеры конфигурации виджетов приведены в Приложении 1.

На этом подготовку пользовательского интерфейса можно считать завершённой. Созданные виджеты способны генерировать графики и сообщения о тревогах.

3.2.3 Настройка каналов уведомлений

Рассмотрим настройку канала уведомлений на примере популярного мессенджера Telegram. Для этого необходимо получить токен бота и идентификатор чата между пользователем и ботом.

Для добавления канала уведомлений необходимо воспользоваться разделом «*Notification channels*» и заполнить следующие поля:

- Name (имя): Telegram
- Type (тип): Telegram
- Send on all alerts (оповещать о всех тревогах): Да
- BOT API Token[2]
- Chat ID[3]

После сохранения настроек можно отправить тестовое сообщение нажатием кнопки «*Send Test*». Сообщение должно сразу высветиться в мессенджере, что станет подтверждением правильности конфигурации. Теперь сообщения о возникновении и снятии тревоги будут доставляться в выбранный мессенджер. На рис. 10 представлен пример сообщения, созданного этой системой.

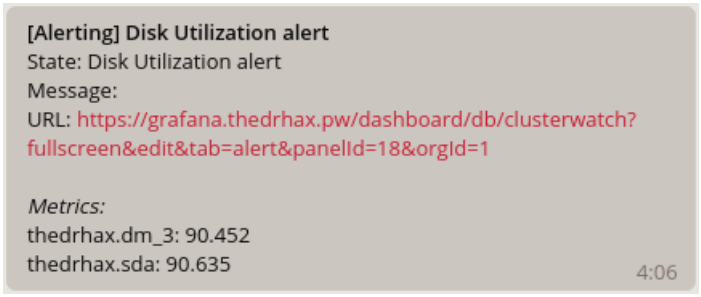

Рис. 10: *Уведомление о перегрузке дисковой подсистемы*

[2]Выдаётся при создании бота в Telegram при помощи @BotFather https://t.me/BotFather.

[3]Можно получить через BOT API с помощью токена, указанного ранее.

Заключение

В результате выполнения данного курсового проекта была создана легко расширяемая информационная система мониторинга кластера, построенного на базе Docker и Rancher.

Приведённая конфигурация системы является базовой и любой её компонент может быть заменён или продублирован:

- Вместо (или вместе с) Netdata можно использовать множество других сборщиков данных, работающих с протоколами Graphite или InfluxDB;
- Различные приложения могут экспортировать данные в систему без использования сборщиков данных;
- К уже существующей базе данных InfluxDB можно добавить и другие хранилища, поддерживающие другие протоколы и форматы данных;
- Для анализа данных можно применить не только систему мониторинга и аналитики Grafana, но и множество других инструментов, обладающих возможностью экспортировать данные из InfluxDB или сервера Graphite.

Концепция микросервисов, использованная в рамках этого проекта, отлично сочетается с технологиями контейнеризации: каждый компонент системы находится в своём изолированном окружении и влияет на другие компоненты только заранее определённым образом. Это позволяет развёртывать систему на любом оборудовании и любых операционных системах семейств Linux, Windows и macOS, а также делает её использование более безопасным для кластера.

Список литературы

[1] The Docker Book: Containerization is the new virtualization // James Turnbull; 17072 edition (July 12, 2014)

[2] Docker: Up & Running: Shipping Reliable Containers in Production // Karl Matthias; ISBN-10: 1491917571 ; ISBN-13: 978-1491917572

[3] Wikipedia: Usage share of operating systems — [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Usage_share_of_operating_systems (дата обращения: 11.12.2017)

[4] Prometheus — [Электронный ресурс]. URL: https://prometheus.io/ (дата обращения: 11.12.2017)

[5] Datadog — [Электронный ресурс]. URL: https://www.datadoghq.com/ (дата обращения: 11.12.2017)

[6] GitHub: Netdata — [Электронный ресурс]. URL: https://github.com/firehol/netdata (дата обращения: 11.12.2017)

[7] Википедия: Load Average — [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Load_Average (дата обращения: 12.12.2017)

[8] Graphite: Tools That Work With Graphite — [Электронный ресурс]. URL: http://graphite.readthedocs.io/en/latest/tools.html (дата обращения: 12.12.2017)

[9] collectd — [Электронный ресурс]. URL: https://collectd.org/index.shtml (дата обращения: 12.12.2017)

[10] GitHub: StatsD — [Электронный ресурс]. URL: https://github.com/etsy/statsd (дата обращения: 12.12.2017)

[11] InfuxData InfluxDB — [Электронный ресурс]. URL: https://www.influxdata.com/ (дата обращения: 12.12.2017)

[12] Elasticsearch — [Электронный ресурс]. URL: http://www.elastic.co/ (дата обращения: 12.12.2017)

[13] Grafana — [Электронный ресурс]. URL: https://grafana.com/ (дата обращения: 12.12.2017)

[14] Википедия: YAML — [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/YAML (дата обращения: 13.12.2017)

[15] Docker Hub — [Электронный ресурс]. URL: https://hub.docker.com (дата обращения: 13.12.2017)

[16] INTEGRATION DEFINITION FOR FUNCTION MODELING (IDEF0) // Processing Standards Publication 183 // Draft Federal Information (1993 December 21)

[17] Introduction to Systems Engineering Practices // John Azzolini (July 2001)

[18] Lifecycle of Docker Container, Nitin Agarwal — [Электронный ресурс] // https://medium.com/@nagarwal/lifecycle-of-docker-container-d2da9f85959 (дата обращения: 13.12.2017)

Приложение 1

Примеры конфигурации виджетов Grafana

- Загрузка процессора (в процентах)
  - Запрос: netdata.*.system.cpu.{user,system,iowait,steal,softirq}.movingAverage(20), groupByNode(1, sum)
  - Тревога: среднее значение за 5 минут превышает 80%

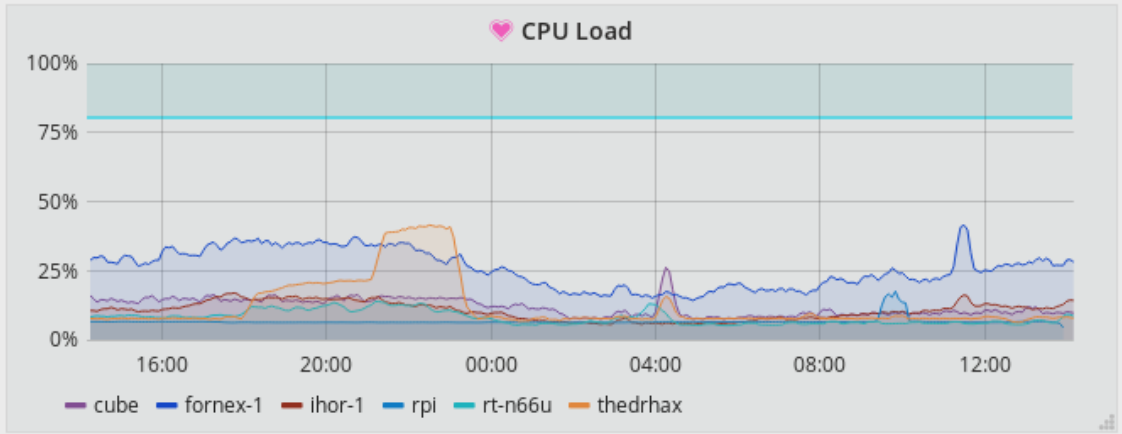

Рис. 11: *График загрузки процессора*

- Средняя загрузка системы (Load Average)
  - Запрос: netdata.*.system.load.load15.aliasByNode(1)
  - Тревога: среднее значение за 5 минут превышает 5

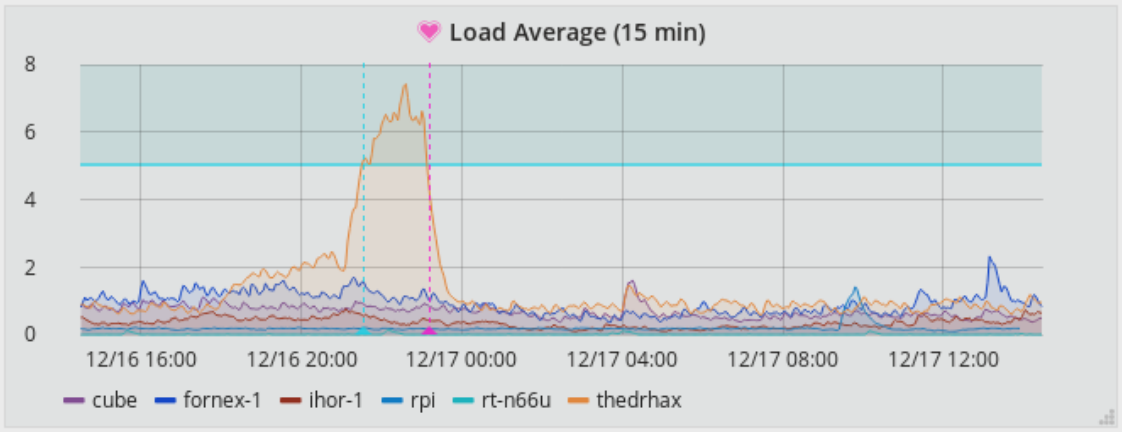

Рис. 12: *График средней загрузки за 15 минут*

- Занятое место на системном диске (в процентах)
  - Запросы:
    - A: netdata.*.disk_space._.*.groupByNode(1, sum) (скрыт)
    - B: netdata.*.disk_space._.used.asPercent(#A).aliasByNode(1)
  - Критерий тревоги: среднее значение за 10 минут превышает 80%

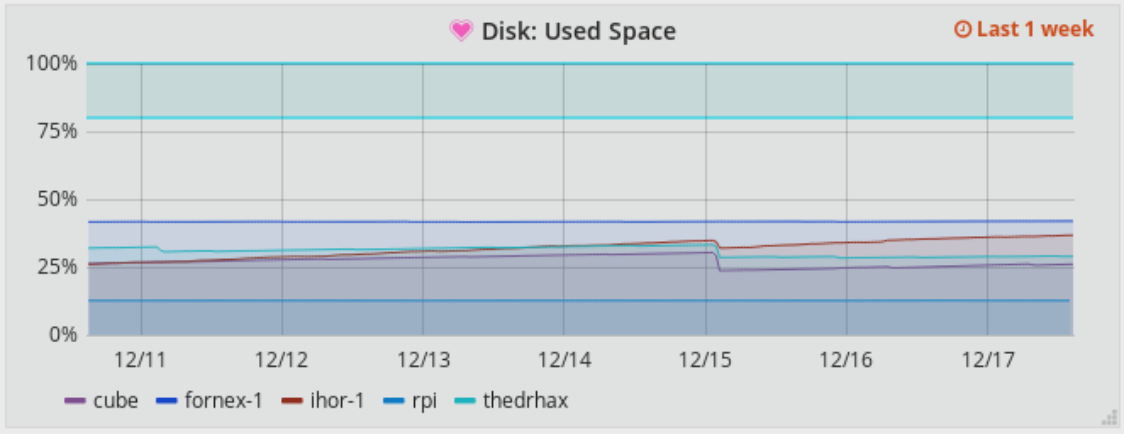

Рис. 13: *График занятого места на диске*

- Утилизация диска (в процентах)
  - Запрос: netdata.*.disk_util.*.utilization.averageAbove(5).movingAverage(10).aliasByNode(1, 3)
  - Тревога: среднее значение за 5 минут превышает 80%

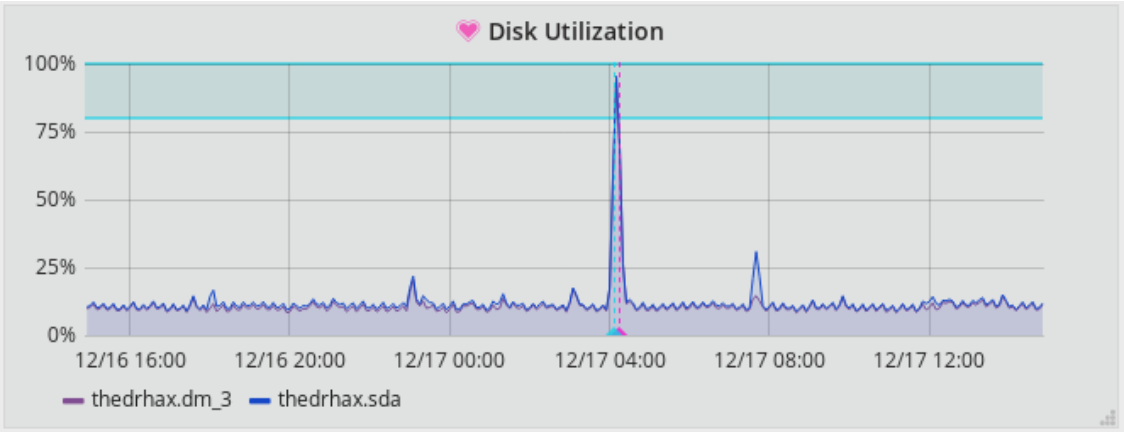

Рис. 14: *График утилизации диска*

- Использование оперативная памяти (в процентах)
  - Запросы:
    - A: netdata.*.system.ram.*.groupByNode(1, sum)
    - B: netdata.*.system.ram.used.asPercent(#A).aliasByNode(1)
  - Тревога: среднее значение за 10 минут превышает 80%

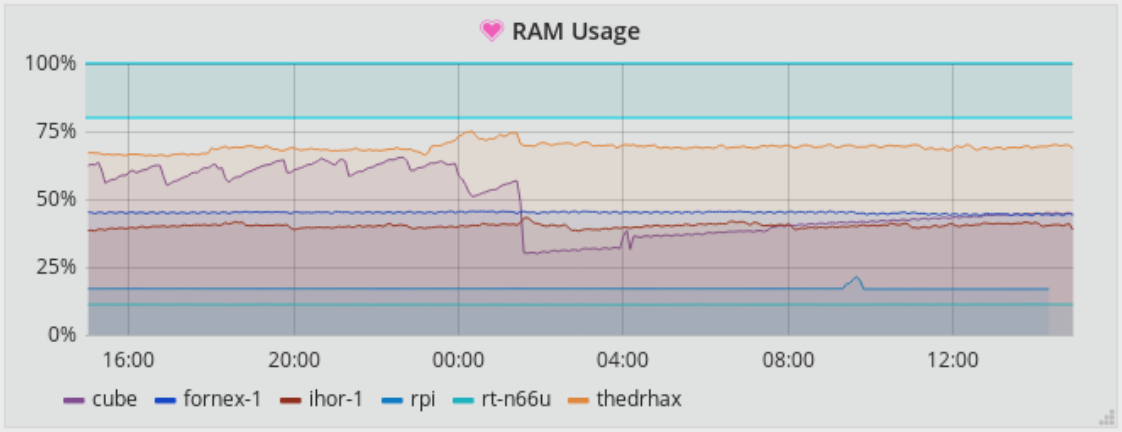

Рис. 15: *График использования оперативной памяти*

- Использование раздела подкачки (в процентах)
  - Запросы:
    - A: netdata.*.system.swap.*.groupByNode(1, sum)
    - B: netdata.*.system.swap.used.asPercent(#A).aliasByNode(1)
  - Тревога: среднее значение за 10 минут превышает 50%

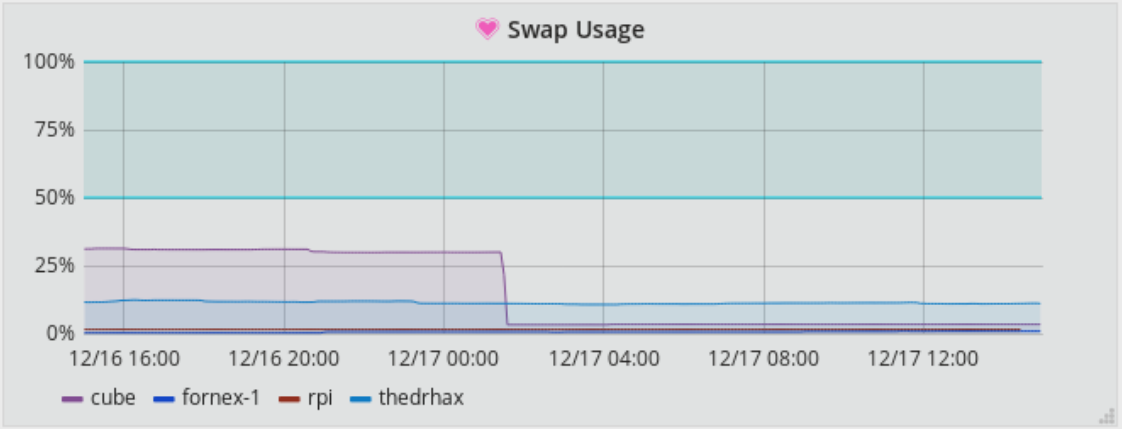

Рис. 16: *График использования файла подкачки*