ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА ИНФОРМАЦИОННЫХ И РОБОТОТЕХНИЧЕСКИХ СИСТЕМ

СИСТЕМА МОНИТОРИНГА СЕРВЕРНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И СИСТЕМ ХРАНЕНИЯ ДАННЫХ В ФИЛИАЛЕ ФКУ "НАЛОГ-СЕРВИС" ФНС РОССИИ В БЕЛГОРОДСКОЙ ОБЛАСТИ

Выпускная квалификационная работа
обучающегося по направлению подготовки
09.03.02 Информационные системы и технологии
очной формы обучения, группы 07001408

Бондаренко Дениса Андреевича

Научный руководитель
к.т.н., доцент
Титов. А.И.

БЕЛГОРОД 2018

РЕФЕРАТ

Система мониторинга серверного программного обеспечения и систем хранения данных в филиале ФКУ "Налог-сервис" ФНС России в Белгородской области – Бондаренко Денис Андреевич, выпускная квалификационная работа бакалавра, Белгород, Белгородский государственный национальный исследовательский университет (НИУ «БелГУ»), количество страниц 48, включая приложения 51, количество рисунков 23, количество таблиц 5, количество использованных источников 24.

КЛЮЧЕВЫЕ СЛОВА: мониторинг, JSON, NoSQL база данных, Python, визуализация данных.

ОБЪЕКТ ИССЛЕДОВАНИЯ: процесс автоматизации мониторинга серверного программного обеспечения и систем хранения данных в филиале ФКУ "Налог-сервис" ФНС России в Белгородской области

ПРЕДМЕТ ИССЛЕДОВАНИЯ: система мониторинга филиала ФКУ "Налог-сервис" ФНС России в Белгородской области.

ЦЕЛЬ РАБОТЫ: автоматизация мониторинга серверного программного обеспечения и систем хранения данных в филиале ФКУ "налог-сервис" ФНС России в Белгородской области.

ЗАДАЧИ ИССЛЕДОВАНИЯ: анализ существующих систем мониторинга, выявление требований к разрабатываемой системе, программная реализация системы, тестирование и отладка разработанной системы.

ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ: устранен основной недостаток существующих систем в программно реализованной системе.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современные ИТ-компании должны иметь актуальную информацию о доступности и состоянии своих сервисов. От того, насколько эффективно эта информация используется системными администраторами, проектировщиками, разработчиками, татуировщиками зависит качество и доступность предоставляемых сервисов компании, что отражается на финансовом положении компании. В случае недоступности своих сервисов компания может понести значительные убытки.

Мониторинг программного и аппаратного обеспечения играет важную роль в жизни ИТ-компании. Он необходим компаниям любых размеров для того, чтобы системные администраторы имели доступ к состоянию системы в разных временных интервалах. Такая информация поможет предугадать дальнейшее поведение системы.

Основной задачей мониторинга является оповещение о сбоях в программной или аппаратной части системы. Чем быстрее будет произведено оповещение – тем лучше для компании. Своевременной оповещение об ошибках в системе поможет оперативно отреагировать на неё – устранить причину возникновения ошибки и избежать возможных трат, связанных с появлением ошибки.

Второстепенной задачей мониторинга является сбора информации о состоянии аппаратной и программной части. Это достигается путем непрерывной работы систем сбора определенного количества ключевых параметров системы. Под объектом, в данном случае, понимается некоторый элемент аппаратного или программного обеспечения.

При отсутствии мониторинга системы компании, обнаружение даже самой мелкой уязвимости в системе может грозить катастрофой, особенно, если ошибку обнаружил опытный пользователь, который сможет эксплуатировать уязвимость в своих целях.

Актуальность работы заключается в необходимости системному администратору предприятия иметь возможность оперативно узнавать об ошибках, возникающих в системе.

Цель работы – автоматизация сбора и обработки информации о текущем состоянии серверного оборудования, ПО, и систем хранения данных в филиале ФКУ "Налог-сервис" ФНС России в Белгородской области.

Решаемые задачи:

– анализ положительных и отрицательных сторон существующих систем мониторинга;

– сбор требований к разрабатываемой системе;

– поиск оптимальных решений для реализации системы;

– разработка системы;

– тестирование и отладка разработанной системы;

Первый раздел ВКР посвящен сравнительному анализу существующих систем мониторинга.

Во втором разделе описано проектирование системы, включающее в себя разработку агента, сервера, способа хранения и визуализации данных.

В третьем разделе проведена программная реализация и тестирование разработанной системы мониторинга.

1 Анализ предметной области

Мониторинг информационных систем – основная часть управления информационной системой и её структурой. Необходимость мониторинга состоит в наблюдении, анализе и отслеживании происходящих в системе изменений. Выделяют 2 уровня мониторинга: мониторинг сервисов или приложений, и мониторинг самой инфраструктуры сети [1].

Системы мониторинга инфраструктуры контролируют работоспособность сетевого и серверного оборудования. Под контролем такой системы могут находится целые группы объектов, информация о которых необходима администраторам. Например, можно отслеживать количество занятой оперативной памяти, или можно отслеживать количество занятого места на жёстком диске.

Системы мониторинга приложений и сервисов предназначены для определения доступности приложений и сервисов на основе определённых показателей. Для каждого сервиса или приложения определяются свои показатели доступности. Система проводит мониторинг выбранных приложений и формирует показатели их работы. Мониторинг систем полезен тем, что помогает системным администраторам и DevOps-специалистам обнаруживать ошибки в работе системы [1].

Изначально, чтобы получить все необходимые данные о системе, администратору необходимо было иметь физический доступ к серверу. Он подходил к серверу, записывал необходимые ему данные на бумажке, а потом уже структурировал и анализировал полученные данные. С появлением SSH процедура значительно упростилась – администратор мог удалённо заходить на сервер, и записывать все метрики уже не на бумажку, а, например, в файл.

Развитие технологий способствовало упрощению процесса мониторинга. Теперь всю работу по сбору данных и их последующем анализе выполняет специальное ПО – система мониторинга [2].

1.1 Zabbix

Zabbix – система мониторинга с открытым исходным кодом. Она началась в 1998 году как внутренний проект в латвийском банке. 7 апреля 2001 года система была выпущена публично под лицензией GPL, первая стабильная версия 1.0 появилась спустя шесть лет 23 марта 2004 [3].

Zabbix предназначен для мониторинга многочисленных параметров сети, жизнеспособности и целостности серверов. Он использует гибкий механизм оповещений, что позволяет пользователям конфигурировать уведомления, основанные на e-mail практически для любого события. Данный подход позволяет быстро реагировать на проблемы с серверами [3].

1.1.1 Архитектура Zabbix

Документация Zabbix называет Zabbix-сервер как ядро системы, которое удалённо контролирует сетевые сервисы и в то же время является хранилищем, в котором содержатся все данные необходимые для функционирования системы [4].

Сам сервер предназначен для выполнения процесса снятия метрик, вычисления триггеров, отправления оповещения пользователям. Такой сервер является центральным компонентом, которому подключаемые Zabbix агенты и прокси сообщают данные метрики систем. Помимо агентов, сервер может самостоятельно проверять сетевые службы.

Функционал базового Zabbix сервера разделен на три отдельных компонента: сам Zabbix сервер, веб-интерфейс и хранилище в базе данных.

Все данные о конфигурации хранятся в базе данных, с которой взаимодействует сервер и веб-интерфейс. Например, когда создаётся новый элемент данных через API, запись об этом добавляется в таблицу элементов данных в базе данных. Далее Zabbix сервер опрашивает таблицу элементов данных для получения списка активных элементов данных, и сохраняет этот

список в кэш Zabbix сервера. Из-за такого подхода любые изменения в веб-интерфейсе Zabbix будут отображены в разделе последних данных с задержкой до двух минут.

Zabbix-агенты размещаются на хостах для активного мониторинга локальных ресурсов и приложений. Агент локально собирает необходимую информацию и отправляет данные Zabbix серверу для их дальнейшей обработки. В случае проблем, Zabbix сервер уведомляет администраторов конкретного сервера об ошибке. Zabbix агенты могут выполнять пассивные и активные проверки.

В случае пассивной проверки агент отвечает на запрос данных от сервера. При таком виде проверки Zabbix сервер запрашивает данные, например, загрузку CPU, и Zabbix агент возвращает ему результат [4].

При активной проверке агент сначала получает список элементов данных для независимой обработки от Zabbix сервера. Далее он будет периодически отправлять новые значения серверу.

Zabbix-прокси собирает данные о производительности и доступности от имени Zabbix-сервера. Собранные данные заносятся в буфер на локальном уровне, и после передаются серверу, к которому привязан прокси-сервер. Zabbix-прокси предназначен для дистанционного контроля узлов сети, не имеющих местных администраторов. Он может быть также использован для распределения нагрузки Zabbix-сервера. В этом случае, прокси только собирает данные, тем самым облегчая работу сервера [4].

1.1.2 Конфигурирование Zabbix

Для минимальной конфигурации Zabbix сперва необходимо добавить узел сети. Узел сети в Zabbix – это объект сети с которого будут получаться метрики. Узлом сети в Zabbix может быть физический сервер, сетевой коммутатор, виртуальная машина, сервис или приложение.

Узел сети представляет собой только машину в сети. Для снятия параметров с объекта сети необходимо определить элемент данных. Они лежат в основе сбора данных в Zabbix и определяют одну метрику или какие данные собираются с узла сети [4].

Элементы данных отвечают только за сбор данных. Для оценки этих самых элементов данных необходимо задать триггеры. Триггер содержит определяет порог, являющийся приемлемым уровнем для элемента данных. Если пришедшие данные будут превышать этот уровень, триггер будет срабатывать, давая понять, что пришедшие данные превышают заданный уровень. Если уровень станет снова приемлемым, то триггер вернется в нормальное состояние.

При наличии элементов данных, которые собирают данные, и триггеров, срабатывающих при проблемных ситуациях, необходимо ещё настроить механизм оповещений, который будет уведомлять администратора о важных событиях, когда администратор по каким-то причинам не имеет возможности посмотреть в веб-интерфейс Zabbix.

Для настройки e-mail оповещений необходимо настроить SMTP сервер. Доставка оповещений через e-mail – задача действий, которые делаются в Zabbix. В настраиваемом действии операции отвечают за то, что нужно сделать, в случае срабатывания действия.

1.2 Nagios

Nagios – система мониторинга с открытым исходным кодом, предназначенная для наблюдения, контроля состояния и оповещения администратора, если какие-то из служб прекращают свою работу. Первый выпуск состоялся 14 марта 1999 [5].

Nagios обладает следующими возможностями:

– мониторинг сетевых служб и состояния хостов;

– поддержка удаленного мониторинга через шифрованные туннели SSH или SSL;

– возможность создания расширений позволяет, используя любой язык программирования, разрабатывать свои собственные способы проверки служб;

– отправка оповещений в случае возникновения проблем со службой или хостом;

– возможность организации совместной работы нескольких систем мониторинга с целью повышения надёжности и создания распределенной системы мониторинга.

1.2.1 Архитектура Nagios

В состав Nagios входят следующие компоненты:

– хост – объект, описывающий физические или виртуальные устройства в сети, с которых будут получать метрики;

– служба – отдельная функциональность, выполняющаяся на хосте, либо одна из его характеристик загрузка процессора;

– команда – определяют каким именно образом будет осуществляться проверка того или иного сервиса;

Документация Nagios предполагает два варианта использования своей системы: централизованный и распределённый [6].

Для централизованного варианта характерно размещение центрального сервера мониторинга Nagios на определённой машине. При такой конфигурации уведомление и отчёты обрабатываются центральным сервером Nagios. Центральный сервер может применять два вариант снятия метрик: активный и пассивный. При активном снятии сервер используется для мониторинга общедоступных веб-сайтов, или серверов электронной почты. Сервер сам с периодичностью проверяет доступность сервисов. Пассивный мониторинг используется для получения метрик с хостов и сервисов в закрытой сети. В данном случае, на хосте необходимо разместить пассивного

агента мониторинга, который будет снимать, и отправлять метрики на центральный сервис.

Второй вариант мониторинга, распределённый, предназначен для мониторинга сложной сетевой архитектуры, которая включает в себя большие удалённые сети, требующие мониторинг удалённых сетевых элементов. В данной модели удалённые сети и их элементы контролируется выделенными серверами Nagios.

Каждый выделенный сервер Nagios может управляться центральным сервером Nagios, так и быть полностью автономным [6]. Уведомления, отчёты и конфигурация обрабатывается каждым удалённым сервером Nagios самостоятельно. Для удалённых серверов существует возможность настроить передачу всей информации на центральный сервер. Такой подход позволяет системным администраторам снимать метрики со всех элементов сети и предоставляет им централизованную отчётность и оповещения.

По умолчанию Nagios предлагает оповещение по электронной почте и пейджерам, но расширения позволяют отправлять извещения и другими способами, которые могут быть удобны в различных обстоятельствах [6]. Имеется возможность группировки контактов. При таком подходе, вместо отдельных людей, которые должны быть извещены, Nagios будет оповещать соответствующую группу целиком.

1.2.2 Конфигурация Nagios

Главный конфигурационный файл Nagios называется nagios.cfg. Конфигурационный файл должен иметь следующий формат [6]:

define <имя_директивы> {

<имя_параметра> <значение_параметра>

}

Хосты определяются с помощью директивы host. Пример определения хоста представлен ниже:

```
define host {
host_name                    local
address                      127.0.0.1
check_command                check-host-alive
contact_groups               admins
notification_options         d,u,r
}
```

Используемые параметры конфигурации хоста:

– host_name – короткое уникальное имя хоста;

– address – IP адрес или полное доменное имя;

– contact_groups – определяют список для оповещения об изменениях;

– notification_options – опции оповещения:

– d (down) – хост не работает;

– u (unreachable) – хост недоступен;

– r (recovery) – хост восстанавливается.

Команды определяют каким именно образом будет проверяться та или иная служба. Пример определения команды:

```
define command {
command_name  check-host-alive
command_line  $USER1$/check_ping -H $HOSTADDRESS$
-w 3000.0,80% -c 5000.0,100% -p 5
}
```

Определение команды состоит из имени команды, и аргументов командной строки. Аргументы командной строки доступны через макроподстановки $ARG1$, $ARG2$ … $ARGN$ [6].

Контакты в Nagios определяют людей, задействованных в процессе мониторинга, которым в случае проблем будут рассылаться оповещения. Пример объявления контакта:

```
define contact {
contact_name                 denis
```

alias Denis Bondarenko

email db@somedomain.com

host_notification_options d,u,r

service_notification_options w,u,c,r

host_notification_commands host-notify-by-email

service_notification_commands notify-by-email

}

Используемые параметры конфигурации контакта:

– contact_name – уникальное имя контакта;

– alias – описание, как правило полное имя сотрудника;

– host_notification_period – определяет временные границы, в рамках которых будет происходить оповещение;

– host_notification_commands или service_notification_commands – команда, используемая для оповещения;

– host_notification_options или service_notification_options – опции оповещения;

– email – адрес электронной почты.

1.3 Prometheus

Prometheus – система мониторинга с открытым исходным кодом предназначенная для мониторинга микросервисных архитектур. С точки зрения микросервисов, приложение понимается не как монолитный компонент, а как набор мелких сервисов, каждый из которых работает в своём процессе. Такие отдельно работающие процессы никак не пересекаются друг с другом и взаимодействует с окружением при помощи простого механизма, чаще всего HTTP [7].

В отличие от мониторинга монолитных сервисов, при мониторинге микросервисов необходимо получать, как и состояние отдельных компонентов

системы, так и всей системы в целом. Prometheus представляет собой решение мониторинга микросервисной проблемы.

1.3.1 Архитектура Prometheus

В соответствии с документацией в состав Prometheus входят следующие компоненты [7]:

– сервер, который считывает метрики и сохраняет их в своей базе данных;

– клиентские библиотеки для языков программирования;

– дашборд для метрик;

– инструменты для экспорта данных из сторонних приложений;

– менеджер уведомлений AlertManager;

– клиент командной строки для выполнения запросов к данным.

Центральный компонент всей системы мониторинга – главный сервер Prometheus. Он работает автономно и сохраняет все данные в локальной базе данных. Сбор метрик в Prometheus осуществляется по запросу от главного сервера. Имеется также возможность снимать показатели с помощью механизма push, но для этого необходим специальный компонент pushgateway. Push может предназначен для случаев, когда сбор метрик с помощью pull невозможен. Также push предназначен для наблюдений за сервисами, подключающихся ко всей сети периодически.

Prometheus хранит данные в виде наборов значений, соотнесённых с временной меткой [7]. Элемент таких данных состоит из имени снятой метрики, временной метки, и пар ключ-значение. Например, снимаемая метрика с информацией о количестве HTTP-запросов к API имя может выглядеть следующим образом: api_req_total. В такой метрике может храниться вся информацию о GET-запросах на адрес, например, /api/ipa, на которые был отдан ответ с кодом 200. Такой временной ряд представляется в виде следующей нотации:

api_req_total{method="GET", endpoint="/api/ipa", status="200"}

Имя метрик не обязательно должно быть уникальным, оно может быть одним и тем же у нескольких рядов, но каждый временной ряд должен быть помечен хотя бы одним тэгом. Поддерживаются следующие типы метрик:

– счётчик – значения, увеличиваются с течением времени;

– шкала – значения, изменяемые со временем;

– гистограмма – содержит информацию об изменении метрики в течение определённого промежутка;

– сводка результатов – аналогично гистограмме, но также позволяет рассчитывать квантили для скользящих временных интервалов.

1.3.2 Конфигурирование Prometheus

Prometheus имеет два способа настройки параметров: через флаги командной строки и при помощи конфигурационного файла. Флаги командной строки настраивают неизменяемые системные параметры, а в файле конфигурации определяется всё, что связано со снятием метрик.

Prometheus имеет возможность перезагружать конфигурационный файл во время работы [7]. Если новая конфигурация была ошибочной – изменения не будут применены, и будет использоваться старая конфигурация. Обновить конфигурацию можно двумя способами: послать SIGHUP процессу Prometheus или отправить POST-запрос на /-/reload.

Конфигурационный файл описывается в формате YAML. Файл начинается с секции global:

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s
```

Секция global включает в себя следующие параметры

– scrape_interval – интервал сбора метрик;

– evaluation_interval – интервал сверки с правилами.

Следующая секция scrape_configs определяет базовые настройки сбора метрик на сервере:

```
scrape_configs:
  – job_name: "some_job_name"
  – scrape_interval: "15s"
target_groups:
    – targets:
      – "localhost:12345"
```

Секция включает в себя следующие обязательные параметры:

– job_name – имя задачи;

– scrape_interval – интервал сбора метрик;

– target_groups – сервисы или группы сервисов, для которых нужно собирать метрики.

Секция alerting предназначена для настройки оповещений:

```
alerting:
  alertmanagers:
    – "./alertmanager..conf"
```

Простой файл конфигурации менеджера оповещений выглядит так:

```
notification_config
  name: "some_manager_name"
  email_config
    email: "some_mail@bsu.edu.ru"
 aggregation_rule
  – "alertmanager_rule"
```

Простой файл конфигурации менеджера оповещений выглядит так:

```
notification_config
  name: "some_manager_name"
  email_config
    email: "some_mail@bsu.edu.ru"
 aggregation_rule
```

– "alertmanager_rule"

В секции norification_config указывается название оповещения [7]. В данном случае указано оповещение по e-mail. Секция agreagation_rule указывает на файлы правил для оповещений. Например, правило, описанное ниже, отправляет уведомление в том случае, если хост недоступен более 5 минут:

```
ALERT HostIsDown
 IF up == 0
 FOR 5m
 WITH {severity="page"}
DESCRIPTION "[$labels.instance] has been down for more than 5 minutes."
```

1.4 Сравнительный анализ существующих систем мониторинга

Собрав достаточное количество информации о существующих системах мониторинга, можно приступить к выявлению положительных и отрицательных сторон этих систем.

Таблица 1.1 позволяет оценить функционал Zabbix.

Таблица 1.1 – Функционал Zabbix

Плюсы	Минусы
Простой процесс снятия метрик с машины и приложений Внешняя SQL база данных Визуальная конфигурация Перезапуск приложений	Сложность мониторинга микросервисных архитектур Нет возможности экспорта данных через саму систему Сложность первичной настройки

Zabbix имеет интуитивно понятный процесс снятия метрик с машины и приложений, но его первичная настройка имеет множества проблем. У Zabbix отсутствует возможности экспорта данных через саму систему. Это связано с тем, что хранит она данные во внешней SQL базе данных. У Zabbix существует

возможность перезапуска неработающих приложений, что может облегчить задачу системного администратора, но в случае микросервисной архитектуры Zabbix не будет так эффективен, как при её отсутствии.

Таблица 1.2 позволяет оценить функционал Nagios.

Таблица 1.2 – Функционал Nagios

Плюсы	Минусы
Создание пользовательских скриптов Мониторинг по SNMP Поддержка плагинов Большое количество методов оповещения	Сложность установки Имеет большое количество конфигурационных файлов При обновлении возникают проблемы совместимости

Основное преимущество Nagios – возможность создания пользовательских плагинов сообществом, что позволяет сделать его многофункциональной системой мониторинга. Nagios полностью конфигурируется при помощи файлов, что создаёт проблему централизованного хранения его конфигурационных файлов.

Таблица 1.3 позволяет оценить функционал Prometheus.

Таблица 1.3 – Функционал Prometheus

Плюсы	Минусы
Хранение данных в собственной темпоральной базе данных Простой мониторинг микросервисной архитектуры Есть возможность контейнеризации сервера	Поверхностность мониторинга Большое количество конфигурационных файлов Отсутствие встроенной визуализации данных

Prometheus – система мониторинга, ориентированная на миркосервисную архитектуру. В связи с этим, Prometheus не подходит для глубокого

мониторинга. Из-за хранения данных в собственной темпоральной базе данных Prometheus имеет возможность импорта данных.

Разрабатываемая система, в первую очередь, предназначена для устранения отрицательных сторон свободно распространяемых систем, и, по возможности, улучшение их положительных сторон.

Основная проблема готовых решений – их ресурсоемкость. Каждое из описанных выше средств мониторинга имеет в комплекте большое количество плагинов. Каждый такой плагин потребляет ресурсы машины, чем сказывается на её работе. Разрабатываемая система решает данную проблему двумя способами: уменьшением размера потребляемых ресурсов, и сжатием передаваемых данных.

Уменьшение размера потребляемых ресурсов состоит в минимизации расхода системой процессорного времени и оперативной памяти.

Цель сжатия передаваемых данных – уменьшение сетевого трафика. Это полезно в случае размещения большого количества агентов в сети. Например, если большое количество агентов начнет отправлять метрики на центральный сервер – они могут серьезно снизить пропускную способность сети, что может сказаться на работе всей сети в целом.

Разрабатываемая система будет использовать пассивных агентов для сбора метрик. Агенту необходимо будет снимать следующие метрики, разбитые на две группы: метрики аппаратной и программной части.

Сбор метрик с программной части заключается в мониторинге работы Windows-сервисов (статус работы) и в мониторинге состояния процессов (расход времени ЦП и расход ОЗУ). Сбор метрик с аппаратной части заключается в получении информации о расходе ЦП, ПЗУ и ОЗУ машиной в целом. Также агенту необходимо контролировать состояние процесса резервного копирования. Для этого ему необходимо собрать следующие метрики: количество резервных копий, время создания последней копии и время выполнение последнего копирования.

Собранная агентами информация будет отправляться на центральный сервер для последующей обработки. Задача центрального сервера состоит в приёме данных от агентов. Получая данные от агента, сервер заносит их в базу, а после обрабатывает. В случае возникновение проблем – центральный сервер должен оповестить об ошибке в системе.

В качестве системы оповещение будет сделана отправка электронных писем. Такой подход позволит избежать проблем кроссплатформенности системы оповещения. Почтовые клиенты доступны на всех часто используемых платформах: Windows, Linux, OS X, iOS, Android. Такой подход к созданию системе оповещения позволяет администраторам получать уведомления вне зависимости от используемого устройства.

1.5 Вывод по первому разделу

В первом разделе выпускной квалификационной работы была рассмотрена архитектура и конфигурация следующих систем мониторинга: Zabbix, Nagios и Prometheus. Рассмотренные системы были проанализированы для выявления их преимуществ и недостатков. В результате проведения анализа систем мониторинга был выявлен общий недостаток для всех вышеописанных систем – ресурсоемкость. Разрабатываемая система нацелена на минимизацию эффекта от этого недостатка двумя способами: уменьшением потребления ресурсов, за счёт оптимизации сервера и агента, и сжатием передаваемых данных по сети.

2 Разработка системы мониторинга

Разрабатываемая система предполагает следующую схему работы: агенты, после сбора указанной информации системе, сжимают собранную информацию и отправляют её на центральный сервер по HTTP. Сервер, получив информацию, отправляет её в хранилище данных. В случае возникновении неполадок – сервер оповещает администратора об ошибке. Для обеспечения визуального интерфейса, в виде построения графиков, будет использована система для визуализации данных. Работа системы схематически показана на рисунке 2.1.

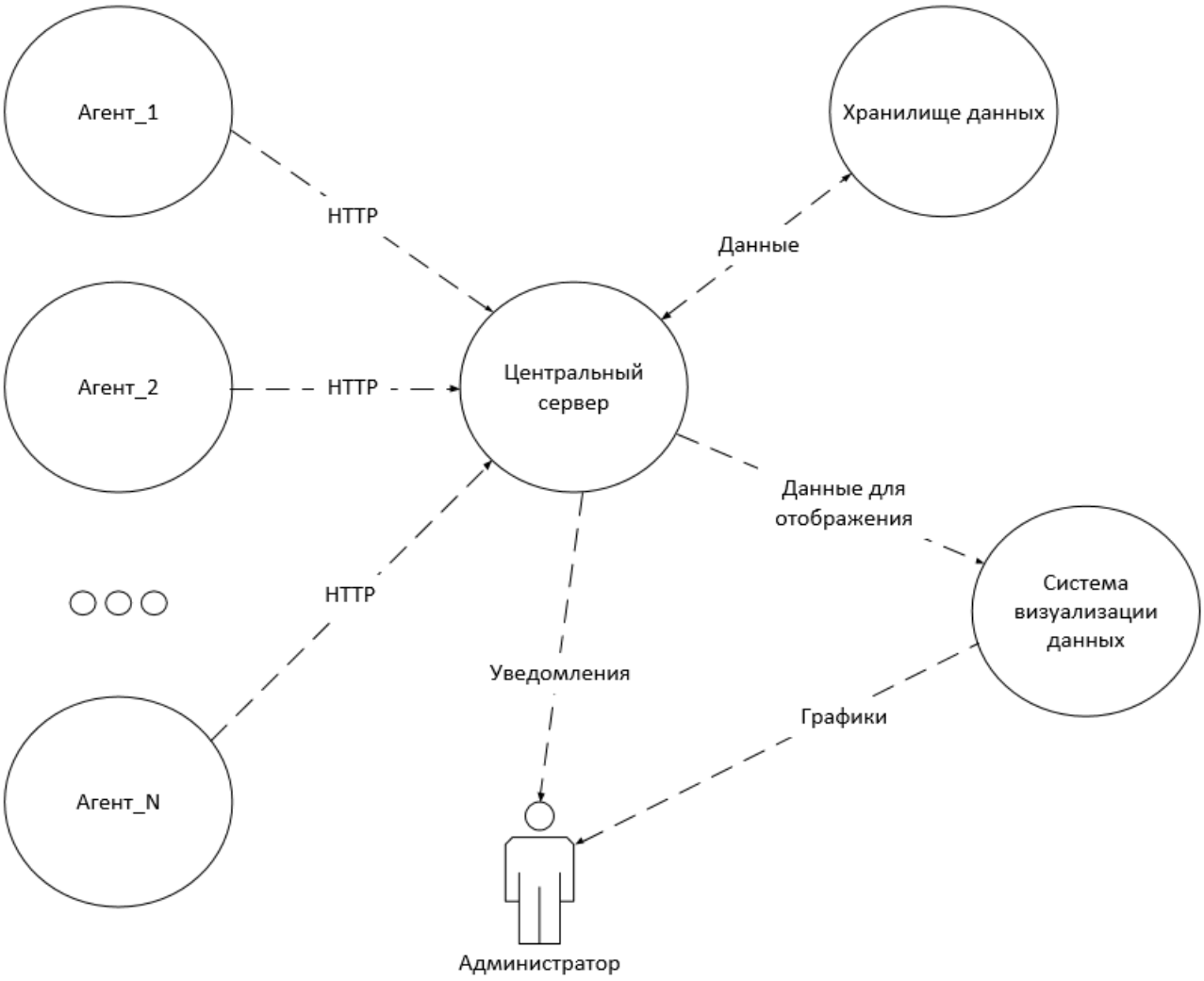

Рисунок 2.1 – Схематическое изображение работы системы

2.1 Разработка агента

При запуске агент считывает конфигурационный файл снимаемых метрик. Конфигурационный файл имеет JSON формат. Пример конфигурационного файла находится в приложении А. В конфигурационном файле указывается секция threshold (с англ. порог), которая содержит в себе пороговые значения снимаемых метрик в виде массива из двух элементов.

Первый элемент массива означает предупреждающий порог, за превышение которого производится одноразовое оповещение. Второй элемент означает ошибочный порог, в результате превышения которого оповещения будут отправляться до тех пор, пока процесс или использование дискового пространства не будет ниже этого значения. Например, если в секции threshold.hardware.ram указан массив [15.0, 30.0] – администратор будет одноразово оповещен, если на данной машине значение используемой оперативной памяти будет превышать 15 процентов. А в случае превышения значения в 30 процентов – оповещения будут отправляться, пока объем используемой оперативной памяти не будет ниже 30 процентов. Считанный порог отправляется на центральный сервер.

После отправки пороговых значений, программа входит в тело бесконечного цикла. В цикле происходит снятие метрик, отправки их на центральный сервер. Далее происходит ожидание в секундах, указанных в конфигурационном файле в поле heartbeat.

Сжатие передаваемых данных происходит при отправки данных на сервер. В качестве упаковщика данных был выбран MessagePack. Это библиотека для сериализации данных в двоичный формат [8]. Он позволяет сериализовать JSON таким образом, что он не потеряет свою структуру, но станет значительно меньшим по размеру [9].

Процесс сборки метрик агентом в нотации IDEF3 представлен на рисунке 2.2.

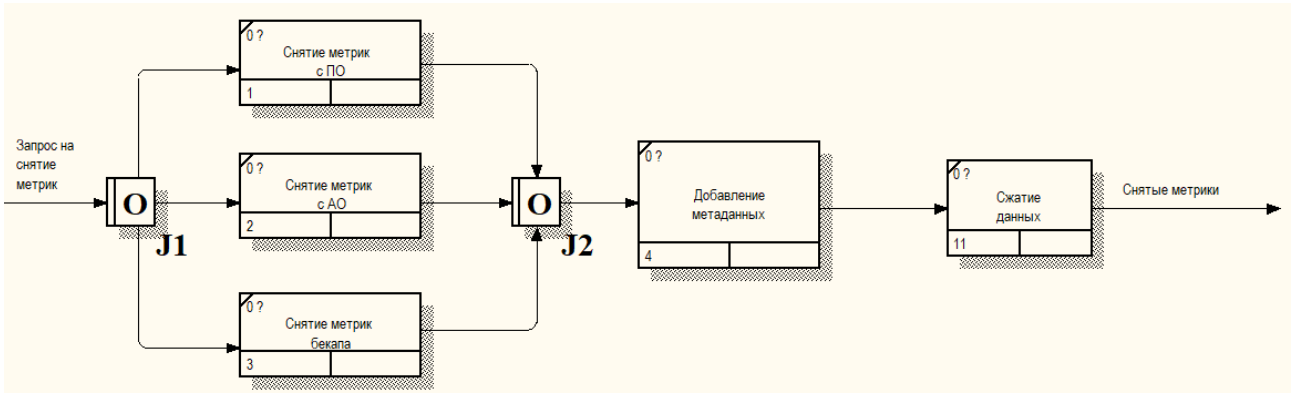

Рисунок 2.2 – IDEF3 диаграмма процесса работы агента

Таким образом, при прохождении одного цикла, в зависимости от конфигурационного файла, происходит снятие определённых метрик. Если в конфигурационном файле указан только блок снятия аппаратных метрик – будут сняты только метрики, соответственно, если указаны все метрики – будут сняты все метрики. Метаданные добавляются в любом случае. В качестве метаданных выступает временная метка снятия метрик, время снятие метрик, и имя агента.

В качестве языка программирования для программной реализации агента был выбран Python 3.5. Это высокоуровневый язык программирования, ориентированный на быстрое написание легко читаемого кода [11]. Основные архитектурные черты – интерпретируемый язык, динамическая типизация, автоматическое управление памятью.

Для получения информации из планировщика задач Windows был использован PowerShell скрипт. PowerShell – это расширяемое средство автоматизации от Microsoft, состоящее из оболочки с интерфейсом командной строки и сопутствующего языка сценариев [12]. Для интеграции PowerShell и Python был создана простая Python функция, которая выполняет PowerShell команды. Для выполнения этой задачи она создаёт объект Popen, который позволяет выполнять инструкции командной строки в Python. Для созданного объекта Popen вызывается функция communicate() которая отвечает за непосредственное выполнение процесса [13]. Полученный результат

выполнения инструкции командной строки необходимо декодировать в зависимости от кодировки ОС. Для Windows это cp866 [14].

Помимо этого, используется Python библиотека psutil – это кросс-платформенная библиотека для получения информации о запущенных процессах. Она используется для снятия всех метрик, кроме получения информации о задаче из планировщика задач.

В качестве основного механизма снятии метрик выступают классы сборщики. Они выполняют функцию обвёрток, предоставляющие более высокоуровневый интерфейс для сбора метрик, скрывая в себе механизмы снятия этих самых метрик. На рисунке 2.3 находится диаграмма классов UML для адаптеров в агенте.

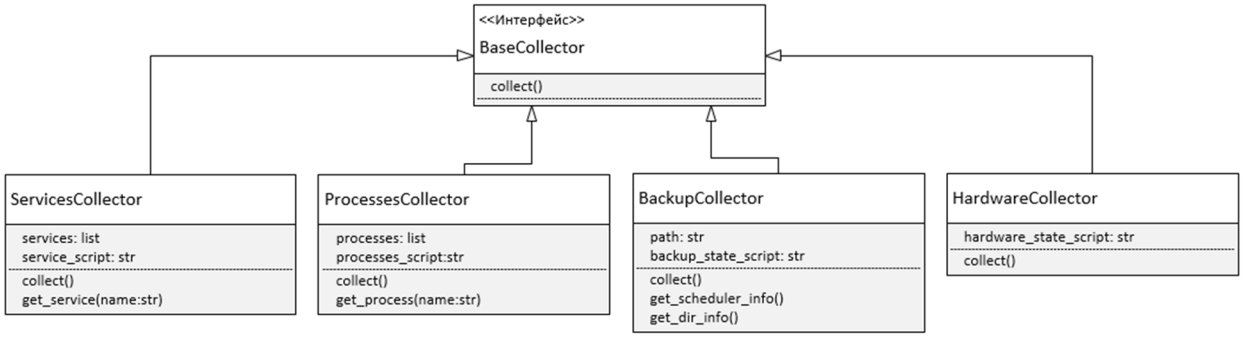

Рисунок 2.3 – Диаграмма классов UML

Класс ServiceCollector отвечает за сбор метрик с сервисов. Его метод collect() возвращает JSON-массив с информацией о собранных сервисах: [{“service_name”: “xbgm”, “display_name” : “Xbox Game Monitoring”, “status”: false }, ...]. Status принимает значение false, если сервер в момент снятия метрики не работает.

Класс ProcessesCollector отвечает за сбор метрик с процессов. По алгоритму работы метод сбора метрик аналогичен методу сбора метрик для ServiceCollector, за исключением того, что для получения информации о процессах используется другие функции библиотеки psutil. Возвращаемый JSON-массив имеет следующую структуру: [{“name”: “explorer”, “cpu” : 1.36,

“mem” : 49.3} , ...]. Поле cpu показывает количество потребляемого процессорного времени и измеряется в процентах, mem показывает количество потребляемой памяти процессом и измеряется в мегабайтах.

Класс HardwareCollector отвечает за сбор аппаратных метрик. Его метод collect() возвращает JSON-объект с информацией о собранных аппаратных метрик. Метод collect() данного класса использует методы библиотеки psutil для получения расхода ЦПУ, ПЗУ и ОЗУ в процентах. Пример возвращаемой структуры: {“cpu” : 10.489, “ram”: 44.587 “rom” : [{“name”: “C”, “usage”: 49.3658}, ...], }. Поля cpu и ram показывают общую загруженность всех ядер процессора в процентах и количество занятой оперативной памяти в процентах соответственно. Поле ram содержит массив из JSON-объектов, которые показывают имя логического диска, и количество занятой памяти в процентах.

Класс BackupCollector отвечает за контроль над резервными копиями на машине. Его метод collect возвращает JSON-объект с информацией о времени выполнения резервного копирования, и общем размере директории. Перед расчетом времени выполнения резервного копирования агент получает информацию о задаче из планировщика задач, которая отвечает за резервное копирование. В качестве получаемой информации о задаче берётся временная метка её последнего и следующего выполнения. Полученные данные кэшируются в ОЗУ, чтобы каждый раз не выполнять запрос к планировщику задач. Это сделано для минимизации времени сбора.

Информация из планировщика задач получается при первом запуске агента, и когда агент определит, что задача уже должна была начать выполнение – когда текущее время системы больше следующего время выполнения задачи. Далее агент, используя встроенные функции языка, получает массив с краткой информацией о файле (имя файла, размер, время создания), и общем размере папки. Массив информации о файлах перед возвращением сортируется по времени создания файла. Таким образом, последний созданный файл будет в конце списка. Время создания резервной

копии вычисляется путем вычитания из времени последнего запуска задачи от времени создания файла.

После осуществления сбора всех метрик агент упаковывает полученные JSON-объекты с информацией по спецификации msgpack. В результате проведения тестирования было выявлено, что в случае сериализации данных в msgpack их размер сокращается на 60 процентов. Время, затраченное на упаковку, не превышает времени, затраченного на упаковку в JSON-объект. Это связано с малым объемом отправляемого JSON-объекта. Полный JSON-объект находится в приложении Б.

2.2 Разработка базы данных

Перед выбором оптимального хранилища для метрик был произведён сравнительный анализ трех СУБД, которые наиболее подходят для хранения метрик: SQLite, PostgreSQL, Elasticsearch. В таблице 2.1 расположены результаты данного анализа по следующим СУБД.

Таблица 2.1 – Сравнение функциональности СУБД.

Критерий	СУБД		
	SQLite	PostgreSQL	Elasticsearch
Построение схемы	Необходимо	Необходимо	Не требуется
Необходимые библиотеки	sqlalchemy	sqlalchemy	elasticsearch
Интеграция со средствами визуализации	Отсутствует	Присутствует	Присутствует
Язык запросов	SQL	SQL	Lucene

В качестве основных критериев используется: необходимость построения схемы таблиц в приложении (построение ORM), необходимые библиотеки, для

оптимальной работы с выбранной СУБД, наличие у СУБД встроенной интеграции со средствами визуализации данных, возможность хранения несжатых JSON-объектов, используемый язык запросов в СУБД для доступа к данным.

В результате проведенного анализа, в качестве СУБД выбор был сделан в пользу Elasticsearch. Такой выбор обусловлен, в первую очередь, способом хранения данных в ней. Данная СУБД представляет собой NoSQL и предоставляет возможность хранить целые JSON-объекты [19]. В разрабатываемой системе, агенты будут отправлять собранную информацию центральному серверу в виде JSON-объектов. Отправляемые объекты будут иметь сложную структуру – один объект будет состоять из одного или более других объектов. Чтобы центральный сервер не затрачивал время на нормализацию данных для формирования SQL-запроса к СУБД, рациональнее было бы хранить целый JSON-объект. Таким образом, можно минимизировать обработку запроса от каждого агента.

Во-вторых, Elasticsearch разработан таким образом, что работа с поступающими JSON-объектами идёт в параллельном режиме [19]. Помимо этого, можно осуществлять поиск по отдельным полям объектов, что позволит организовать поиск по параметрам в разрабатываемой системе.

В-третьих, для организации доступа к данным к другим БД в Python необходимо использовать библиотеку sqlalchemy, которая имеет большой вес. В результате добавления библиотеки увеличивается и вес всего приложения.

В-четвертых, Grafana, многофункциональная система для отображения данных в виде графиков, имеет встроенную поддержку Elasticsearch, что позволит импортировать данные из Elasticsearch, и отображать их в Grafana.

Из-за хранения целых JSON-объектов в Elasticsearch отсутствуют SQL-таблицы в привычном виде. Для доступа к данным Elasticsearch предоставляет веб-сервис, работа с которым происходит по HTTP [20]. Например, для создания записи в базе, необходимо сделать POST-запрос на /_index/_type/_id, а

в тело POST-запроса положить JSON-объект. Для используемой схемы доступа к данным можно провести аналогию с SQL, где:

– _index – имя базы данных;

– _type – таблица базы данных;

– _id – ключ.

Отличительная особенность в том, что по id мы получаем не строку из базы, как при выполнении SQL-запроса, а JSON объект. Получить объект по его ключу можно отправив GET-запрос на /_index/_type/_id.

Для разрабатываемой системы хранилище будет организовано следующим образом: /agents-YYYY.MM.DD/json/<хэш>, где:

– agents-YYYY.MM.DD – имя индекса будет зависит от дня снятия метрики;

– <хэш> – хэш автоматически создается Elasticsearch.

Таким образом, зная имя агента и временную метку можно сделать GET-запрос на, например: /agents-2017.12.25/json/some_object_hash, и получить метрики, снятые по этой временной метки и хэшу.

2.3 Разработка центрального сервера

При запуске центрального сервера, аналогично агенту, считывается конфигурационный файл, в котором указан хост и порт базы данных, и данные, необходимые для работы системы оповещений. Полный конфигурационный файл находится в приложении В.

Центральный сервер представляет собой REST веб-сервис. В качестве каркаса для центрально сервиса используется Flask. Это фреймворк для создания веб-приложений на языке программирования Python. Он относится к категории микрофреймворков – минималистичных каркасов веб-приложений, предоставляющих лишь самые базовые возможности [21]. Этим фактором обоснован выбор данного фреймворка как основного каркаса для разрабатываемого веб-сервиса.

Всего веб-сервис имеет 5 конечных точек, информация о которых расположена на таблице 2.2.

Таблица 2.2 – конечные точки веб-сервиса

Метод	Путь	Описание
POST	/agents/threshold	Принимает пороговые значения от агентов
POST	/agents	Принимает метрики от агента
GET, POST	/grafana	Используется Grafana для проверки доступности сервера
GET, POST	/grafana/search	Используется Grafana для получения доступных значений для автодополнения
GET, POST	/grafana/query	Используется Grafana для получения данных по запросу

Обработка метрик сервером происходит по POST запросу на /agents. Добавление пороговых значений агентов происходит по POST запросу на /agents/threshold. Конечные точки, связанные с Grafana, будут описаны в главе 2.5. Процесс обработки метрик в нотации IDEF3 представлен на рисунке 2.4.

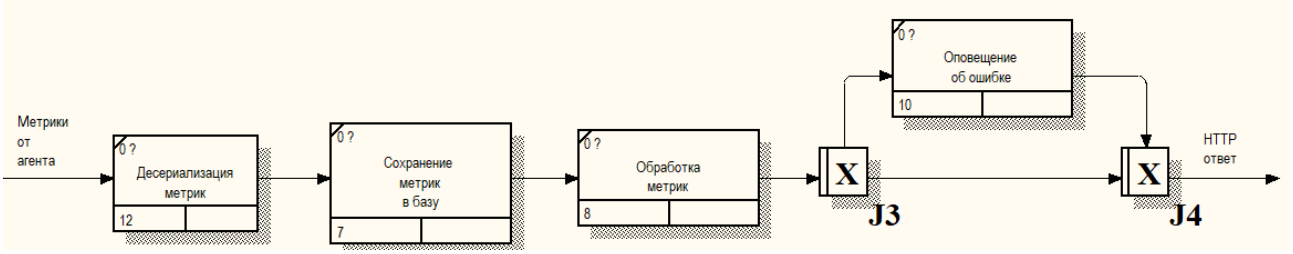

Рисунок 2.4 - IDEF3 диаграмма процесса обработки метрик агента сервером

За добавление метрик и пороговых значений в базу данных отвечают классы, UML диаграмма которых находиться на рисунке 2.5.

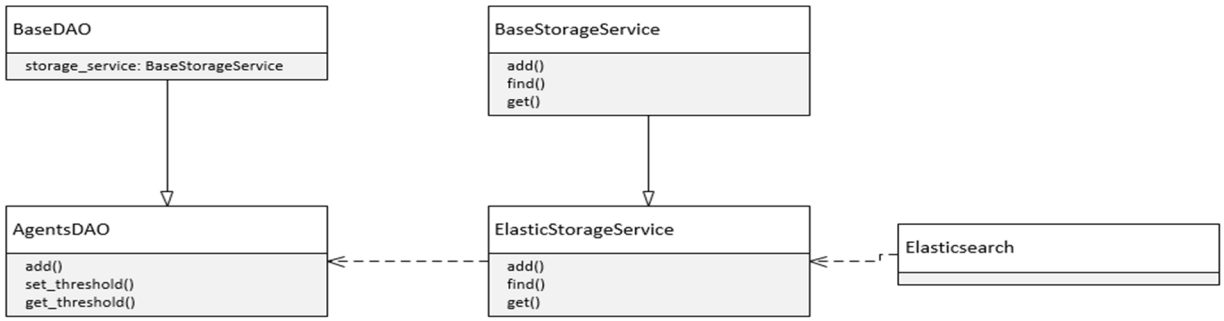

Рисунок 2.5 – UML диаграмма классов

Класс AgentsDAO наследуется от базового класса BaseDAO. Название этих двух классов говорит о том, что классы реализуют шаблон проектирования DAO (с англ. Data Access Object) – это объект, который предоставляет абстрактный интерфейс к какому-либо типу базы данных или механизму хранения данных [22].

BaseDAO, соответственно и все его потомки, имеют поле storage_service типа BaseStorageService, которое передается ему в конструкторе. BaseStorageService представляет собой абстрактный интерфейс над базой данных, который имеет абстрактные методы для добавления, получения и поиска объектов в базе. В разрабатываемой системе есть потомок BaseStorageService – ElasticStorageService, который отвечает за добавление, получение и поиску объектов по базе данных Elasticsearch.

Такое разбиение архитектуры обусловлено тем, что в случае необходимости изменения механизма хранения, можно было бы заменить его, создав программный код для работы с другим хранилищем [22]. При этом не придётся адаптировать старый код, например, в AgentsDAO, к новым изменениям. Единственное условие – новый класс должен быть унаследован от BaseStorageService, реализуя все его методы.

Проверка полученных метрик происходит после добавления их в базу. Метод handle_data класса ServerHandler обеспечивает эту проверку. На вход метод принимает полученные метрики, и пороговые значения агента из базы. По имени блока мы получаем функцию обработчик данного блока. Если такой функции не имеется, например, для блока meta такая функция отсутствует, продолжается обработка следующего блока. Если же обработчик имеется – мы вызываем его, передавая ему данные из блока снятых метрик, и аналогичный блок из пороговых значений агента.

Функции обработчики блоков занимаются сравнением пороговых и полученных значений. Например, функция проверки блока services проверяет, запущены ли все указанные сервисы. В случае выявления отключенного сервиса – возникает исключение, содержащее описание ошибки.

Возникшее исключение не выбрасывается программными средствами, а записывается в массив. Такой подход позволяет в случае возникновении нескольких ошибок не останавливаться на первой возникшей, а комплексно сообщить обо всех ошибках.

Если функция обработчик вернула пустой массив – иначе начинается обработка следующего блока. Если функция вернула массив с ошибкой – этот массив добавляется в ещё один массив, который возвращается функцией handle_data. После добавления начинается обработка следующего блока.

В дальнейшем, если были найдены какие-то ошибки, массив с исключениями возвращается функцией handle_data, иначе она возвращает пустой массив.

Перед преступлением непосредственно к оповещению ошибок проводится их фильтрация. Сообщения с типом error не проходят фильтрацию, они отправляются в любом случае. Сообщения с типом warning отправляются один раз и заносятся в базу, чтобы не было повторной отправки warning сообщений. В конце письма прикреплена ссылка для удаления возникших ошибок из базы.

2.4 Разработка системы оповещения

Система оповещения является частью центрального сервера. После создания класса, обрабатывающего полученные метрики, происходит инициализация менеджеров оповещений, которые указаны в конфигурационном в файле. В зависимости от метода оповещения, конфигурация в JSON-объекте может сильно отличаться. Например, для того, чтобы подключить вывод в консоль в конфигурационном файле в массиве alert_managers необходимо добавить лишь {"name": "console"}, а для конфигурации почты необходимо указать ещё и адрес и порт почтового сервера, логин, пароль и получателей писем.

Диаграмма классов UML для менеджеров оповещений находиться на рисунке 2.5.

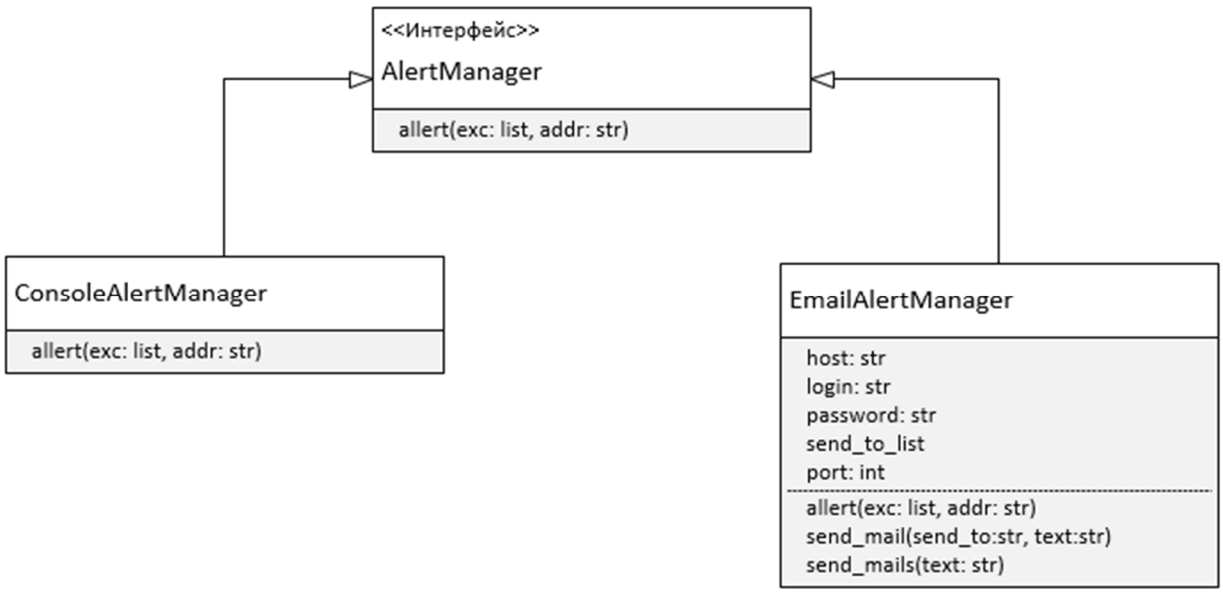

Рисунок 2.6 – Диаграмма классов UML

Для возможности последующего добавления новых менеджеров был создан базовый класс AlertManager. Класс EmailAlertManger в методе allert формирует письмо по следующему шаблону: “Error on {host} occurred.\n\n{errors_list}\n\n{reset_hash_link}”. В качестве host указывается

хост, на котором возникли ошибки. Список ошибок указывается в errors_list. Ссылка reset_hash_link овтечает за удаление ошибок из базы данных.

После формирования шаблона – менеджер приступает за отправку писем. Так как отправка письма может занимать длительное время, отправка писем получателем происходит в новом потоке, чтобы освободить главный поток для работы с другими агентами. Помимо этого, отправка писем происходит с задержкой, для того, чтобы почтовый сервер не определил отправляемые как спам.

Для отправки писем с Gmail ящика необходимо получить специальный ключ доступа, который используется для одноразового или многоразового входа в сервисы Google. При создании ключа необходимо указывать к каким сервисам можно получить доступ созданным ключом. Для отправки писем необходимо лишь открыть доступ по данному ключу для почтового сервиса Google. Это сделано в целях безопасности самой Google. Таким образом, приложение, использующее ключ, сможет отправлять только письма, и не будет иметь доступ к другим сервисам Google. В случае возникновения утечки в системе злоумышленник получит лишь возможность отправлять письма с почтового ящика, но доступ к другим возможностям ему будет закрыт. Сервис Яндекс.Почта имеет аналогичную защиту.

2.5 Визуализация данных

В качестве средства визуализации данных была выбрана Grafana. Данное средство предназначено для отображения циклических метрик [23]. Она даёт возможность сконфигурировать любой набор метрик, благодаря огромному количеству как встроенных, так и разрабатываемых сообществом источников данных. Помимо этого, при поддержки сообщества, существует возможность и создания своих панелей для отображения метрик.

Для подключения Grafana к системе мониторинга необходимо настроить источник данных. В качестве используемых источников данных будет

использован встроенный плагин интеграции с Elasticsearch, и разрабатываемый сообществом плагин для отображения JSON-объектов – Grafana Simple Json Datasource [23].

Перед настройкой источников данных, была создана переменная окружения agent_name. Она необходима для возможности динамического изменения имени агента, по которому делаются все запросы к Elasticsearch или к центральному серверу. Перенос данного значения в переменную даёт возможность изменения параметра имени агента в запросах к источникам данных. Таким образом, просто меняя это значение, Grafana будет отображать графики для разных агентов.

Для настройки интеграции с Elasticsearch, который имеет встроенную поддержку в Grafana [23]. В настройках необходимо указать адрес сервера, и паттерна для индекса. Существует возможность наложение паттерна для индекса. Это необходимо в тех случаях, когда индексы метрик имеют приставку в виде даты. В центральном сервисе метрики разбиты по дням, поэтому паттерн для индекса будет указан следующим образом [agents-]YYYY-MM-DD. Также можно сразу же указать стандартное поле временной метки.

Для отображения метрик необходимо создать дашборд, на котором будут располагаться все графики. На созданном дашборде размещаются панели, для отображения метрик. Для отображения данных в виде графика необходимо создать панель, и настроить запросы к источнику данных. Все запросы к источнику будут иметь фильтр meta.name:$agent_name, который будет получать все запросы только для определенного агента, имя которого указано в переменной $agent_name, описанной выше.

Помимо отображений в виде графиков, Grafana дает возможность выводить одиночную метрику. Данный тип отображения используется для демонстрации данных, которые не нуждаются в построении графика.

На рисунке 2.6 показан результат созданных панелей, для отображения данных с Elasticsearch.

Рисунок 2.7 – Результат настройки панелей для Elasticsearch

Одиночные метрики показывают последнюю запись из базы данных о расходе ЦПУ, ОЗУ и ПЗУ. Графики же показывают использование ЦПУ, ОЗУ, ПЗУ за определённый интервал времени.

Для настройки интеграции с центральным сервером был настроен соответствующий источник данных для отображения данных из JSON-объектов. Помимо этого, на центральном сервере, были созданы специальные конечные точки, которые будут выполнять запросы от плагина. Основная конечная точка для выполнения запросов – /grafana/query. Такой источник данных необходим для корректного отображения метрик из Elasticsearch, которые хранятся в виде массива. Например, данные о процессах и их расходах ЦПУ и ОЗУ. Grafana не имеет возможности обрабатывать их корректно, из-за этого возникла необходимость в разработке собственного источника данных.

Основными параметрами запроса от плагина к источнику данных являются временные метки, определяющие временной интервал, и параметры, передаваемые от плагина. По временным меткам центральный сервер делает запрос к Elasticsearch, получая необходимые ему данные. Из передаваемых параметров берётся имя агента и необходимый параметр для отображения.

После того как центральный сервер получил необработанные данные из базы, их необходимо нормализовать в схему, которую принимает плагин Grafana Simple Json Datasource. Схему представляет собой JSON-массив и выглядит следующим образом: [{'target': target_name, 'datapoints': [[value, timestamp_in_millisec], ...]}]. Построенные графики на таком источнике данных демонстрируются на рисунках 2.8.

Рисунок 2.8 – Отображение расхода ЦПУ и ОЗУ для процессов

Панель слева показывает количество расхода ЦПУ процессами. Панель слева показывает количество расхода ОЗУ процессами. Существует возможность фильтра графиков по выделению интересующих процессов.

2.6 Вывод по второму разделу

Во втором разделе выпускной квалификационной работы были спроектированы и разработаны следующие модули системы: агент – сборщик информации; сервер – обработчик информации; Elasticsearch – хранилище данных; E-mail – основной способ оповещения администратора; Grafana – средство визуализации данных.

3 Тестирование и отладка

Для запуска агента на машине необходимо установить интерпретатор Python 3.5. Чтобы избежать установки интерпретатора на каждую машину, было принято решение скомпилировать программу в исполняемый файл.

Компиляция программы на Python – это не компиляция в машинный код, как, например, программ на C, а лишь сборка в исполняемый файл программы вместе с частью интерпретатора Python. В качестве частей интерпретатора выступают базовые библиотеки и дополнительные библиотеки, необходимые для работы программы.

Компиляцию выполняет библиотека cx_Freeze. Перед созданием исполняемого файла библиотека делает копию интерпретатора при помощи модуля venv, который предназначен для создания виртуального окружения. Библиотека работает только со слепком интерпретатора, чтобы в случае возникновении ошибок при сборке не повредить основной интерпретатор.

В зависимости от конфигурации, библиотека может исключать или добавлять модули. Исключение модулей предназначено для минимизации веса исполняемого файла. Добавление модулей обеспечивает корректную работу компилируемой программы. После окончания работы с модулями библиотека делает выбранный модуль частью Python интерпретатора.

Созданный исполняемый файл вызывает портативный Python интерпретатор, который выполняет нужный скрипт для запуска программы.

После размещения необходимых файлов для работы Grafana, Elasticsearch и центрального сервера был произведён их запуск и проверена их доступность.

Убедившись, что все сервисы работают, был запущен агент на локальной машине. Данный агент будет снимать метрики каждые 10 секунд. В его задачи входит сбор доступности следующих сервисов: ZeroConfigService, WSearch, Dhcp. Помимо сервисов агент собирает информацию о процессах: java, com.docker.service, explorer.

Оповещение по почте будет произведено в следующих случаях:

– если процессы java и com.docker.service превысят потребление в 20 процентов ЦПУ, и в 150 мегабайтах ОЗУ;

– если ЦПУ на машине превышает 50 и ОЗУ в 40 процентов;

– если диск С будет занят на 75%.

После запуска агента получаем письмо, изображенное на рисунке 3.1.

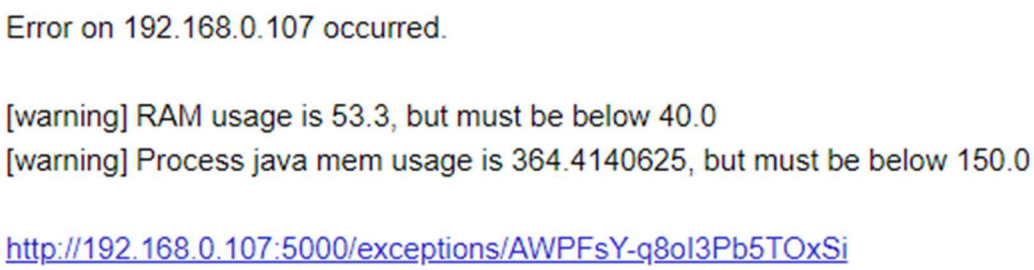

Рисунок 3.1 – Оповещение по электронной почте

В письме говорится о том, что процесс java использует 364.4140625 мегабайт ОЗУ, а общее потребление ОЗУ равняется 53.3 процентам. После создания нагрузки на ЦПУ, пришло письмо, показанное на рисунке 3.2.

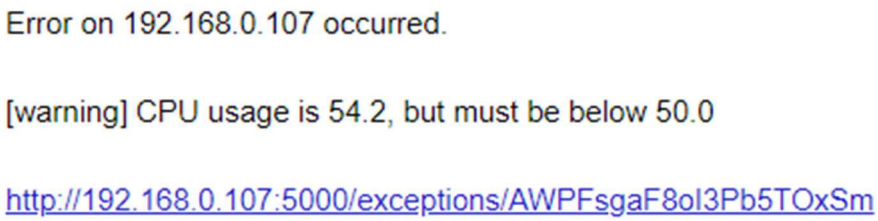

Рисунок 3.2 – Оповещение о загруженности ЦПУ

В письме говорится о том, что использование ЦПУ на машине превысило 54.2 процента (рис 3.2).

Все сообщения с типом warning приходят одноразово на почту. Это сделано для предотвращения спама. Ссылки в письмах позволяют удалить кэш ошибок, о которых было произведено оповещение через письмо. Перейдем по

ссылке с первого письма и через некоторое время получаем аналогичное письмо, но уже с другими значениями загруженности.

Error on 192.168.0.107 occurred.

[warning] RAM usage is 54.0, but must be below 40.0
[warning] Process java mem usage is 381.99609375, but must be below 150.0

http://192.168.0.107:5000/exceptions/AWPFutF08ol3Pb5TOxTj

Рисунок 3.3 – Повторное оповещение по электронной почте

Сообщения с типом error присылаются вне зависимости того, возникали ли они ранее или нет. Для проверки этого отключи сервис ZeroConfigService. Спустя некоторое время получаем письмо о том, что сервис не работает.

Error on 192.168.0.107 occurred.

[error] Service Intel(R) PROSet/Wireless Zero Configuration Service is down

Рисунок 3.4 – Оповещение об отключенном сервисе

Рисунок 3.5 демонстрирует графики и одиночные метрики для загруженности ЦПУ, ОЗУ и ПЗУ хоста за последние 10 минут.

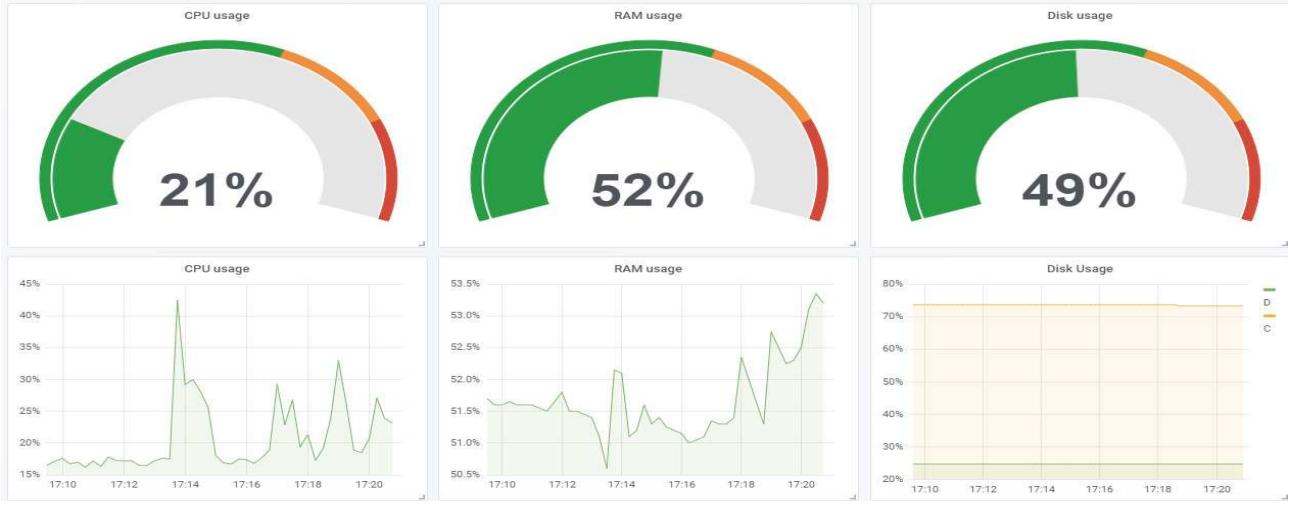

Рисунок 3.5 – Графики загруженности ЦПУ, ОЗУ и ПЗУ

На графике расхода ЦПУ (левый нижний) можно наблюдать, что в период с 17:13 до 17:15 произошёл скачок в расходе ЦПУ на хосте. Аналогичные изменения можно наблюдать и в графике расхода ОЗУ (центральный нижний).

Рисунок 3.6 демонстрирует графики загруженности ЦПУ и ОЗУ по процессам.

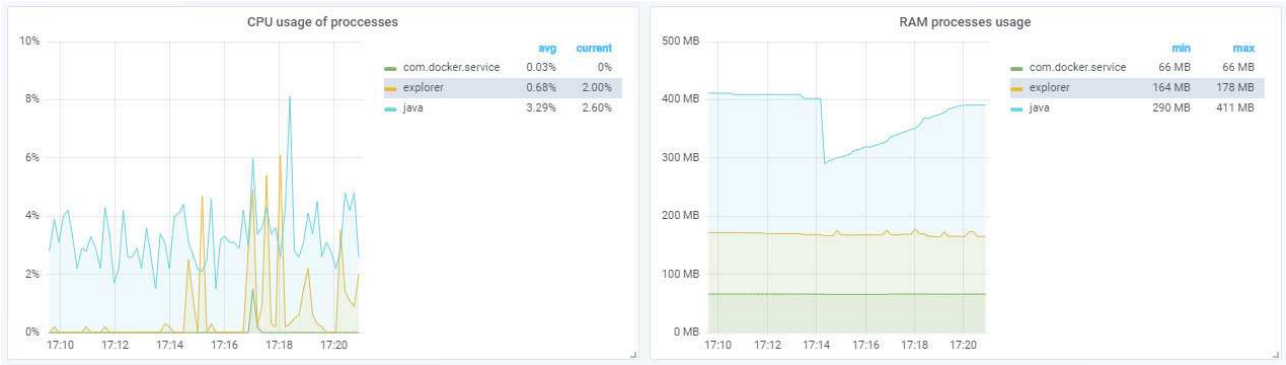

Рисунок 3.6 – Графики загруженности ЦПУ и ОЗУ процессами

На графике расхода ЦПУ по процессам отсутствует скачок расхода ЦПУ с 17:13 до 17:15. Это говорит о том, что отслеживаемые процессы в то время не повлияли на скачок расхода ЦПУ.

На рисунке 3.7 изображено время снятия метрик с агента.

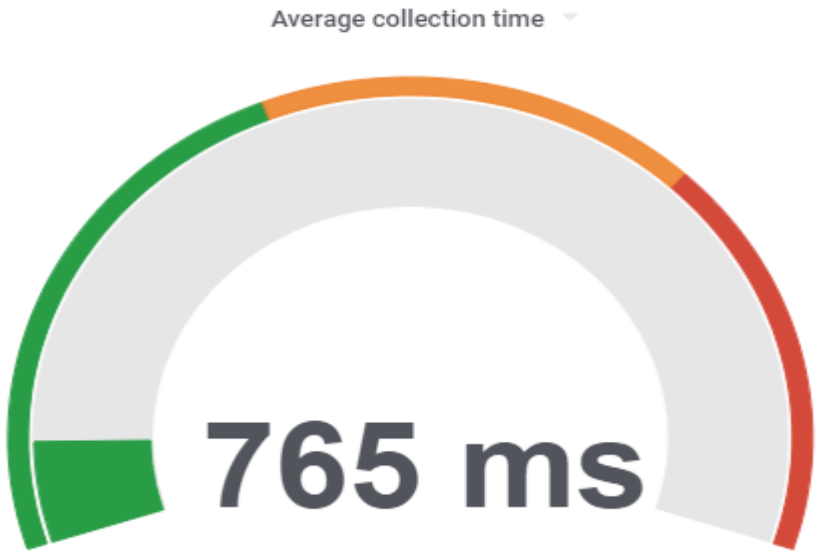

Рисунок 3.7 – Время снятия метрик с агента

Значение в 765 миллисекунд считается приемлемым. В случае превышения снятия метрик в 3 секунды, индикатор начнётся светиться желтым цветом. Если же время, затраченное на сбор метрик, превысит 6 секунд – индикатор засветиться красным цветом, сигнализируя о том, что агент тратит слишком много времени на сбор метрик.

После проверки работоспособности агента на хосте центрального сервера был произведён запуск агента на другом хосте. Второй агент будет снимать метрики каждые 15 секунд. В его задачи входит сбор доступности wscsvc и WSearch сервисов. Помимо сервисов агент собирает информацию о процессах: powershell, SearchUI, explorer.

Оповещение по почте будет произведено в следующих случаях:

– если процессы explorer и powershell превысят потребление в 20 процентов ЦПУ, и в 50 мегабайтах ОЗУ;

– если ЦПУ на машине превышает 50 и ОЗУ в 60 процентов.

После запуска агента никаких оповещений не пришло. В системе все процессы не превышали допустимые значения. Для проверки работоспособности отправки писем был нагружен процесс explorer. На рисунке 3.8 изображено электронное письмо о том, что использование ОЗУ в целом и ОЗУ процессом explorer на хосте превысило норму.

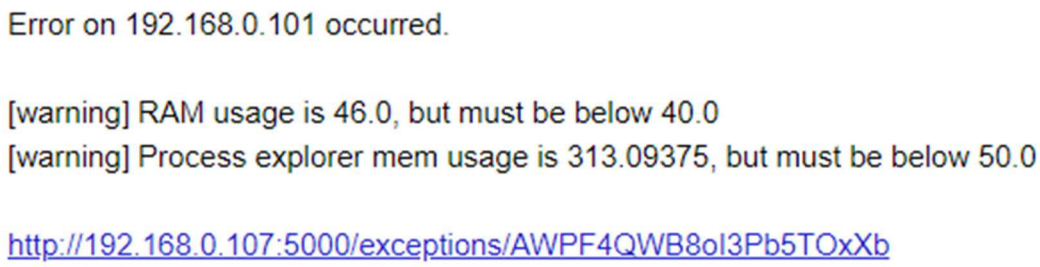

Рисунок 3.8 – Письмо о превышении нагрузки на ОЗУ процессом

Для проверки работоспособности отправки писем с ошибками об отключенных сервисах был отключен сервис Windows Search. На рисунке 3.9 изображено оповещение об отключенном сервере.

Error on 192.168.0.101 occurred.

[error] Service Windows Search is down

Рисунок 3.9 – Оповещение об отключенном сервисе Windows Search

Рисунок 3.10 демонстрирует графики загруженности ЦПУ по процессам.

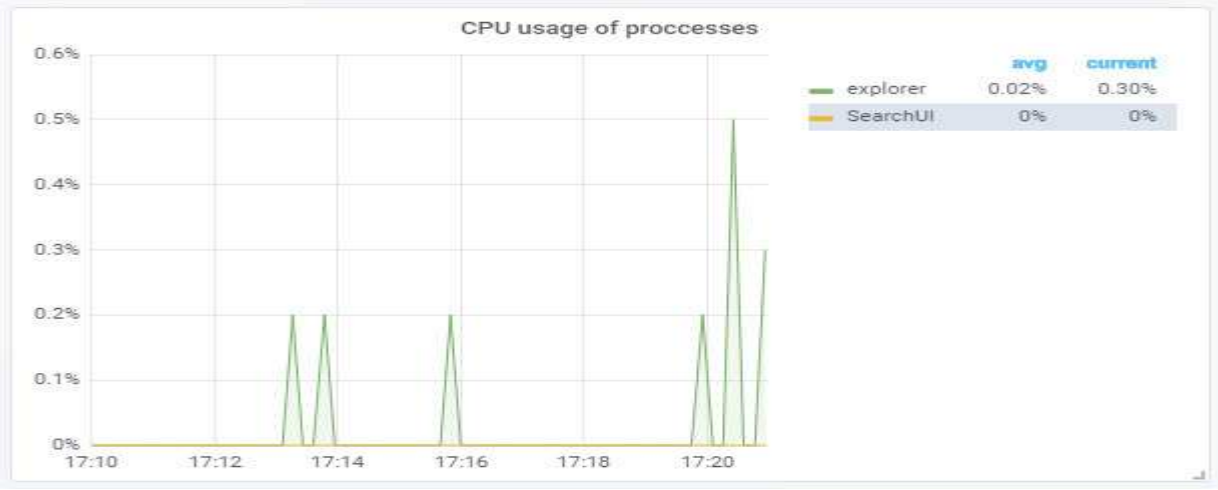

Рисунок 3.10 – График загруженности ЦПУ процессами

На графиках видно, что процесс SearchUI не потребляет ЦПУ, а процесс explorers лишь изредка потребляет менее одного процента ЦПУ. Рисунок 3.11 демонстрирует расход ОЗУ процессами.

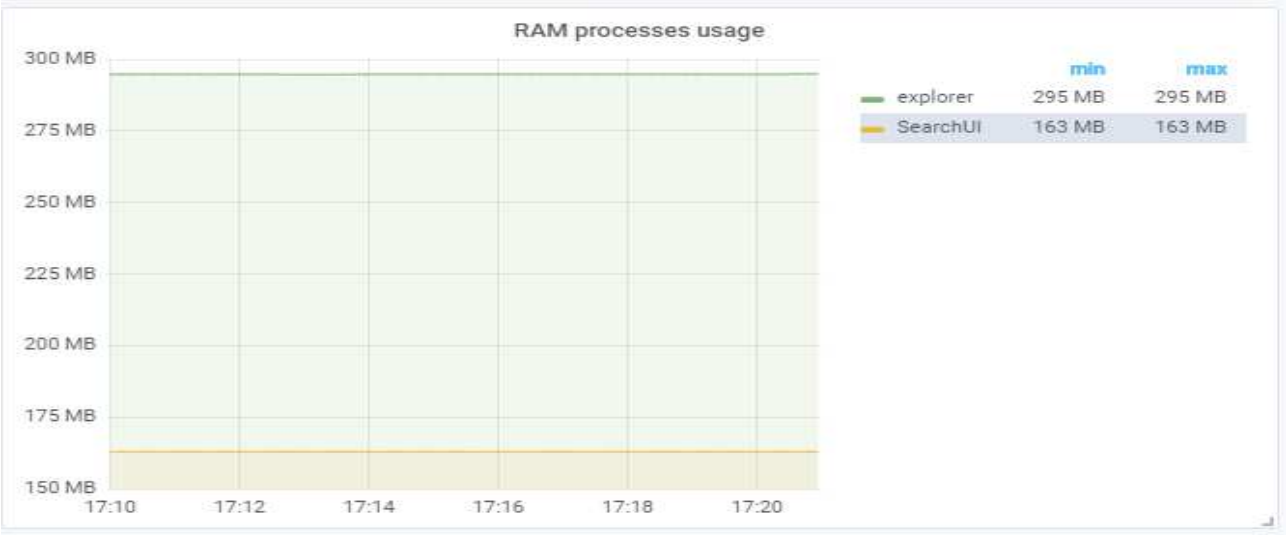

Рисунок 3.11 – Графики загруженности ОЗУ процессами

В конфигурационном файле, помимо процессов explorer и SearchUI был указан ещё и процесс powershell, который отсутствует на графиках. Это связано с тем, что процесс powershell в момент снятия метрик не работал.

Рисунок 3.12 демонстрирует графики и одиночные метрики для загруженности ЦПУ, ОЗУ и ПЗУ со второго агента за последние 10 минут.

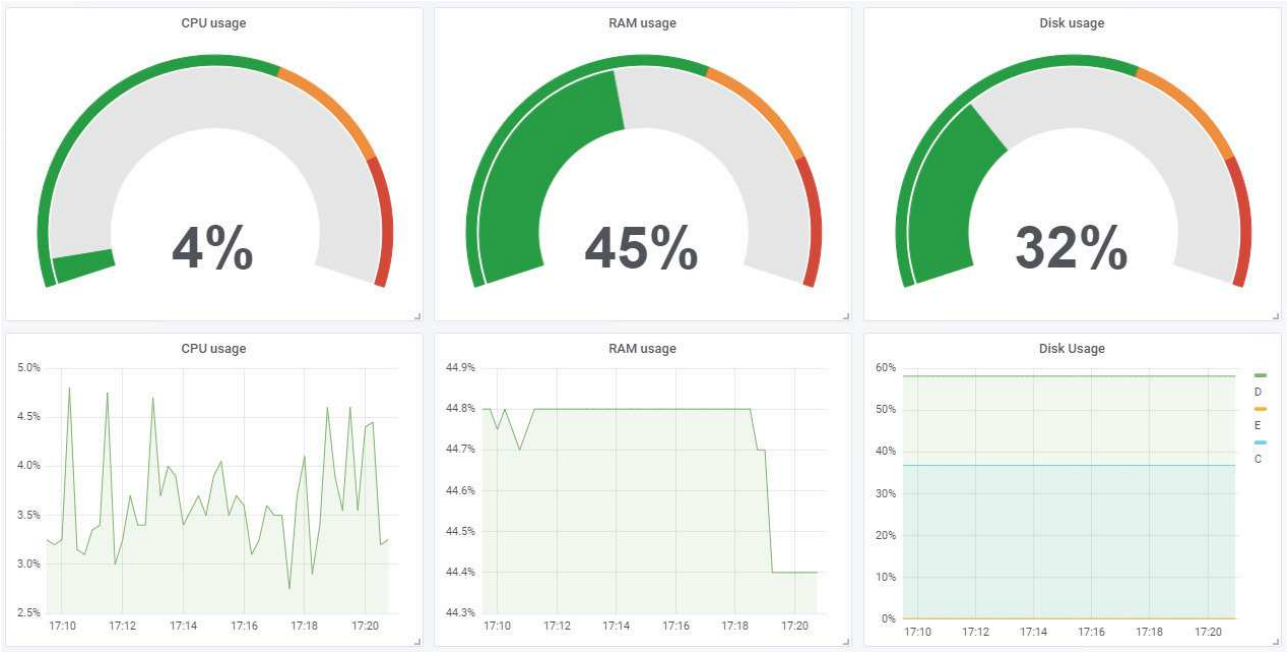

Рисунок 3.12 – Графики загруженности ЦПУ, ОЗУ и ПЗУ

По графикам можно сказать, что машина, в целом, простаивает. Процент потребляемого ЦПУ составляет всего лишь 4 процента, в то время как процент ОЗУ и ПЗУ составляют лишь 45 и 32 процентов соответственно.

3.1 Вывод по третьему разделу

В третьем разделе выпускной квалификационной работе была осуществлена программная реализация системы мониторинга серверного программного обеспечения и систем хранения данных в филиале ФКУ "Налог-сервис" ФНС России в Белгородской области.

ЗАКЛЮЧЕНИЕ

Современные компании в информационной сфере стараются предоставлять как можно более удобные сервисы для своих клиентов по всему миру. Чтобы выполнять эту функцию компаниям необходимо обеспечить доступность и работоспособность своей ИТ-инфраструктуры постоянно, чтобы предоставлять как можно более удобные сервисы для своих клиентов по всему миру. Чтобы обеспечить эту работоспособность, необходимо как можно раньше выявлять уязвимые места в своей сети, а также быстро узнавать о возникшей поломки и её причине. Для этих нужд принято использовать системы мониторинга.

Системы мониторинга бывают свободно распространяемые и платные, с открытым и закрытым исходным кодом, они могут различаться по своему встроенному функционалу: способу получения метрик, масштабируемости, требуемым ресурсам и уровнем знаний, необходимым для приемлемой настройки системы.

При начале работы с системой мониторинга сначала нужно определиться с объектами, за которыми будет вестись наблюдения. В зависимости от выбора объектов мониторинга следует выбирать и подходящую систему мониторинга. Например, Prometheus идеально подходит для мониторинга системы с микросервисной архитектуры, а Zabbix или Nagios могут справиться с этой задачей только при помощи плагинов.

Объекты мониторинга – не основополагающий фактор при выборе системы мониторинга. При выборе системы мониторинга следует обратить внимание на критические события и показатели, которые и определят количество оповещений при поломке, частоту сканирования. Оценивание данных показателей, в первую очередь, требуется осуществлять не с точки зрения разработчиков, а с точки зрения пользователя.

Ход выполнения работы полностью соответствовал поставленным во введении задачам. Для начала необходимо было провести анализ существующих систем мониторинга.

Проведенный обзор существующих систем мониторинга позволил выделить наиболее удобную, с точки зрения системного администратора, структуру приложения. Это стало возможным благодаря устранению недостатка существующих систем мониторинга – проблемы потребления ресурсов.

Сбор требований к системе и разработка технического задания определила основные функции, которая должна выполнять система. Программная реализация разработанных проектных моделей позволила получить систему мониторинга, которая упрощает процесс снятия метрик с хостов, визуализация данных позволяет провести анализ состояний машины в разные временные промежутки.

Разработанная система может быть усовершенствована путем добавления поддержки снятия метрик с серверов под управлением Linux.

В ходе выполнения выпускной квалификационный работы была достигнута поставленная цель автоматизации сбора и обработки информации о текущем состоянии серверного оборудования, ПО, и систем хранения данных в филиале ФКУ "Налог-сервис" ФНС России в Белгородской области.

Все поставленные для решения цели задачи, были выполнены:

– выполнен анализ положительных и отрицательных сторон существующих систем мониторинга;

– собраны требований к разрабатываемой системе;

– найдены оптимальные решения для реализации системы;

– произведена программная реализация системы;

– проведено тестирование и отладка разработанной системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Мониторинг ИТ систем – управление информационной инфраструктурой. Внедрение системы мониторинга. [Электронный ресурс] // ИТ-Аутсорсинг. – Режим доступа: https://www.alp.ru/itsm/sistemyi_monitoringa_it (дата обращения 08.01.2018).

2. Что такое мониторинг в IT или почему админы стали больше спать? [Электронный ресурс] // Хабрахабр. – Режим доступа: https://habr.com/company/croc/blog/144941/ (дата обращения 08.01.2018).

3. Вакке, А.Д. Zabbix. Практическое руководство / А. Д. Вакке; пер. с анг. А.Н. Кисилев. – Москва: ДМК Пресс, 2017. – 356 с.

4. Документация [Электронный ресурс] // Официальная документация Zabbix. – Режим доступа: https://www.zabbix.com/ru/manuals (дата обращения 21.01.2018).

5. Wojciech, K, Learning Nagios – Third Edition / W. Kocjan, P. Beltowski. - Packt Publishing, 2016. - 414 с.

6. Nagios Documentation – Nagios [Электронный ресурс] // Официальная документация Nagios. – Режим доступа: https://www.nagios.org/documentation/ (дата обращения 22.01.2018).

7. Overview | Prometheus [Электронный ресурс] // Официальная документация Prometheus. – Режим доступа: https://prometheus.io/docs/introduction/overview/ (дата обращения 24.01.2018).

8. MessagePack: It's like JSON. but fast and small. [Электронный ресурс] // Официальная документация msgpack. – Режим доступа: https://msgpack.org/#messagepack-for-python (дата обращения 19.01.2018).

9. msgpack/spec.md at master msgpack/msgpack [Электронный ресурс] // Спецификация msgpack. – Режим доступа: https://github.com/msgpack/msgpack/blob/master/spec.md (дата обращения 19.01.2018).

10. Черемных, С. Моделирование и анализ систем. IDEF-технологии: практикум / С. Черемных, И. Семенов, В. Ручкин. – Москва: Финансы и статистика, 2006. – 192 с.

11. Любанович, Б. Простой Python. Современный стиль программирования / Б. Любанович. – Питер, 2017. – 480 с.

12. Коробко, И. PowerShell как средство автоматического администрирования / И. Коробко. – Москва: ДМК Пресс, 2017. – 224 с.

13. 3.5.5 Documentation [Электронный ресурс] // Официальная документация Python 3.5.5. – Режим доступа: https://docs.python.org/3.5/ (дата обращения 13.01.2018).

14. Учебник по использованию Windows PowerShell [Электронный ресурс] // Официальная документация PowerShell. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/ee790872(v=azure.10).aspx (дата обращения 19.03.2018).

15. Буч, Г. Краткая история UML / Г. Буч, Д. Рамбо, И. Якобсон. – Москва: ДМК Пресс, 2006. – 496 с.

16. SQLite Documentation [Электронный ресурс] // Официальная документация SQLite. – Режим доступа: https://www.sqlite.org/docs.html (дата обращения 14.02.2018).

17. PostgreSQL: Documentation [Электронный ресурс] // Официальная документация PostgreSQL. – Режим доступа: https://www.postgresql.org/docs/ (дата обращения 10.02.2018).

18. Elastic Stack and Product Documentation | Elastic [Электронный ресурс] // Официальная документация Elasticsearch. – Режим доступа: https://www.elastic.co/guide/index.html (дата обращения 11.02.2018).

19. Редмонд, Э. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL / Э. Редмонд, Д. Р. Уилсон; пер. с. англ. А. Слинкин. – Москва: ДМК Пресс, 2018. – 384 с.

20. Берримен, Д. Релевантный поиск с использованием Elasticsearch и Solr / Д. Берримен, Д. Тарнбулл; пер. с анг. А.Н. Кисилев. – Москва: ДМК Пресс, 2018. – 408 с.

21. Гринберг, М. Разработка веб-приложений с использованием Flask на языке Python / М. Гринберг; пер. с. англ. А. Кисилев. – Москва: ДМК Пресс, 2016. – 272 с.

22. Фримен, Э. Паттерны проектирования. Обновленное юбилейное издание / Э. Фримен, Э. Робсон, К. Сиерра. – Питер, 2018. – 656 с.

23. Grafana documentation | Grafana Documentation [Электронный ресурс] // Официальная документация Grafana. – Режим доступа: http://docs.grafana.org/ (дата обращения 24.03.2018).

ПРИЛОЖЕНИЕ А

Конфигурационный файл агента

```
{
 "master": {
  "name": "master_agent",
  "url": "http://192.168.0.107:5000",
  "heartbeat": 10
 },
 "services": [
  "ZeroConfigService",
  "WSearch",
  "Dhcp"
 ],
 "processes": [
  "java",
  "com.docker.service",
  "explorer"
 ],
 "hardware": true,
 "threshold": {
  "processes": [
   {
    "name": "java",
    "cpu": [20.0, 40.0],
    "mem": [150.0, 700.0]
   },
   {
    "name": "com.docker.service",
    "cpu": [20.0, 40.0],
    "mem": [100.0, 400.0]
   }
  ],
  "hardware": {
   "cpu": [50.0, 70.0],
   "ram": [40.0, 80.0],
   "rom": [
    {
     "name": "C",
     "usage": [75.0, 90.0]
    }
   ]
  }
 }
}
```

ПРИЛОЖЕНИЕ Б

Снятые метрики


```
{"metrics": {
  "hardware": {
    "rom": [
      {
        "usage": 73.6,
        "name": "C"
      },
      {
        "usage": 24.8,
        "name": "D"
      }
    ],
    "cpu": 24.1,
    "ram": 53
  },
  "processes": [
    {
      "name": "java",
      "mem": 245.38671875,
      "cpu": 3.5
    },
    {
      "name": "com.docker.service",
      "mem": 84.1953125,
      "cpu": 0
    },
    {
      "name": "explorer",
      "mem": 131.22265625,
      "cpu": 1.4
    }
  ],
  "services": [
    {
      "name": "ZeroConfigService",
      "display_name": "Intel(R) PROSet/Wireless Zero Configuration Service",
      "status": true
    },
    {
      "name": "WSearch",
      "display_name": "Windows Search",
      "status": true
    },
    {
      "name": "Dhcp",
      "display_name": "DHCP-клиент",
      "status": true
    }
  ],
  "timestamp": "2018-06-03T15:40:08+03:00",
  "meta": {
    "name": "master_agent",
    "collection_time": 0.6760008335113525,
    "agent_addr": "192.168.0.107"
  }
 }
}
```

ПРИЛОЖЕНИЕ В

Конфигурационный файла сервера

```
{
 "elastic": {
  "host": "http://192.168.107",
  "port": 9200
 },
 "alert_managers": [
  {
   "name": "console"
  },
  {
   "name": "email",
   "host": "smtp.gmail.com",
   "port": 587,
   "app_url": "http://192.168.0.107:5000",
   "login": "somegamaillogin",
   "password": "qwerasdfzxcv",
   "send_to": [
    "somemail@domain.com"
   ]
  }
 ]
}
```

Обозначение	Наименование	Дополнительные сведения
	Текстовые документы	
1. 110700026.09.03.02. 775.ПЗВКР	Пояснительная записка	48 с.
	Графические документы	
2. 110700026.09.03.02. 775.ДМВКР	Демонстрационные материалы (презентация)	14 сл.
	Демонстрационные материалы (пл. ф. А4)	14 лист.*5 экз
	Другие документы	
3.	Документы на компакт-диске	1 CD

Изм.	Лист.	Номер докум.	Подп.	Дата	110700026.09.03.02. 775.ПЗВКР			
Разработал		Бондаренко Д.А			Система мониторинга серверного программного обеспечения и систем хранения данных в филиале ФКУ "Налог-сервис" ФНС России в Белгородской области Ведомость ВКР	Лит.	Лист	Листов
Проверил		Титов А.И.				У	52	53
Н.контр.		Нестерова Е.В.				НИУ «БелГУ» гр.07001408		
Утвердил		Иващук О.А.						

Выпускная квалификационная работа выполнена мной совершенно самостоятельно. Все использованные в работе материалы и концепции из опубликованной научной литературы и других источников имеют ссылки на них.

«___» _____ _____ г.

_____ _____

(подпись) *(Ф.И.О.)*